Bachelor scriptie

Het versimpelde topologische ε-algoritme

11 juni 2020

Auteur: Jeroen Kool

Begeleider: Prof. dr. ir. Jason Frank

Universiteit Utrecht

Inhoudsopgave

1 Inleiding

Als we een continue functie $f : \mathbb{R}^m \to \mathbb{R}^m$ hebben, kunnen we een rij definiëren als $\vec{x}_{n+1} = f(\vec{x}_n)$, als voor een zekere $\vec{x}_0$ geldt dat $\vec{x}_n \to \vec{x}$, oftewel de rij $(\vec{x}_n)_{n\in\mathbb{N}}$ convergeert naar $\vec{x}$, noemen we dit een vast punt iteratie. We weten met behulp van de contractiestelling van Banach dat er veel van deze vast punt iteraties bestaan. Voorbeelden van deze functies zijn Netwon's methode en de methode om stabiele punten in differentiaal vergelijkingen te vinden. In sommige gevallen convergeert een rij $(\vec{x}_n)_{n\in\mathbb{N}}$ heel langzaam naar een punt, in dit geval nemen we aan dat $\vec{x}_n \to \vec{x}_*$, bijvoorbeeld door gebruik te maken van een vast punt iteratie.

Om dit te versnellen kunnen we gebruik maken van een acceleratie methode. Er zijn verschillende acceleratie methodes, dit zijn meestal extrapolatie methodes. Er bestaan methodes voor lineaire en niet-lineaire systemen.

We gaan in dit verslag kijken naar een extrapolatie methode voor niet lineaire systemen, namelijk het versimpelde topologische ε-algoritme (STEA). We maken hier gebruik van de theorie die ondekt is door Claude Brezinski [2]. Het is alvast goed om te zeggen dat er twee varianten zijn, het eerste versimpelde topologische ε-algoritme (STEA1) en het tweede versimpelde topologische ε-algoritme (STEA2).

In hoofdstuk 2 gaan we kijken hoe we tot het versimpelde topologische ε-algoritme komen en het voordeel van het tweede ten opzichte van het eerste algoritme. Het versimpelde topologische ε-algoritme is gebaseerd op het scalaire ε-algoritme. We zullen daarom eerst behandelen hoe we tot het scalaire ε-algoritme komen. Hiervoor moeten we gaan kijken naar de Shanks transformatie. Het scalaire ε-algoritme is namelijk een efficiënte implementatie van de Shanks transformatie.

Daarna gaan we in hoofdstuk 3 twee numerieke experimenten behandelen waarbij we het tweede topologische ε-algoritme toepassen op verschillende problemen. We gaan kijken hoe het tweede versimpelde topologische ε-algoritme werkt op de rijen die we verkrijgen bij het oplossen van de problemen en of we het probleem sneller kunnen oplossen door STEA2 toe te passen.

Tot slot geven we in hoofdstuk 4 een conclusie van de gevonden resultaten bij de experimenten en enkele suggesties voor verder onderzoek.

2 Theorie achter het algoritme

We gaan in dit hoofdstuk de theorie achter het eerste en het tweede versimpelde topologische ε-algoritme bekijken. We gaan eerst naar de theorie achter het scalaire ε-algoritme kijken om zo makkelijker het topologische ε-algoritme te begrijpen.

2.1 Scalair ε-algoritme

Het scalaire ε-algoritme is een efficiënte implementatie van de Shanks transformatie. We gaan dus kijken naar de Shanks transformatie en de achtergrond hiervan. We zullen uiteindelijk bekijken hoe we hier het scalaire ε-algortime van maken.

2.1.1 Shanks transformatie

We nemen aan dat we een rij $(A_n)_{n\in\mathbb{N}}$ hebben die naar A convergeert, waarbij $A_n \in \mathbb{R}$. Met behulp van Shanks transformatie definiëren we een nieuwe rij $S(A_n)$ waarbij het doel is dat $S(A_n) \to A$, maar sneller dan de originele rij A_n. De Shanks transformatie is gedefinieerd als

$$S(A_n) := \frac{A_{n+2}A_n - A_{n+1}^2}{A_{n+2} - 2A_{n+1} + A_n}.$$

De Shanks transformatie is in essentie hetzelfde als Aitken's Δ^2 proces, waar we hier niet verder over gaan uitweiden.

In de volgende paragraaf gaan we verder op de achtergrond van de Shanks transformatie, maar om de formule nu vast te motiveren gaan we kijken naar het geval dat dat A_n van de vorm is

$$A_n = A + \alpha q^n$$

met $|q| < 1$. We zien dat in dit geval $A_n \to A$ en we kunnen zien dat de Shanks transformatie in één keer de oplossing geeft:

$$\begin{aligned} S(A_n) &= \frac{A_{n+2}A_n - A_{n+1}^2}{A_{n+2} - 2A_{n+1} + A_n} \\ &= \frac{(A + \alpha q^{n+2})(A + \alpha q^n) - (A + \alpha q^{n+1})^2}{(A + \alpha q^{n+2}) - 2(A + \alpha q^{n+1}) + (A + \alpha q^n)} \\ &= \frac{A\alpha q^n(1 - 2q + q^2)}{\alpha q^n(1 - 2q + q^2)} = A. \end{aligned}$$

2.1.2 Gegeneraliseerde Shanks transformatie

We gaan nu de gegeneraliseerde Shanks transformatie geven. We gaan deze transformatie bepalen door op een andere manier A, zoals in de vorige sectie omschreven, proberen te vinden.

We gaan nu c_i proberen te vinden zodat geldt dat

$$A = c_0 A_n + \cdots + c_k A_{n+k}.$$

Hierbij zijn c_i te bepalen constanten onafhankelijk van n met de eis dat $c_0 c_k \neq 0$ en $c_0 + \cdots + c_k \neq 0$, en waarbij $A_n \to A$ als het convergeert. Als we tevens eisen dat $c_0 + \cdots + c_k = 1$, dan kunnen we deze vergelijking opschrijven naar

$$c_0(A_n - A) + \cdots + c_k(A_{n+k} - A) = 0, \quad n = 0, 1, \ldots. \tag{1}$$

Om het schrijfwerk te vereenvoudigen definiëren we nu dat $\Delta A_n = A_{n+1} - A_n$.

Neem aan dat (1) waar is voor alle $n \in \mathbb{N}$. We willen nu A berekenen. We hebben $\forall n \in \mathbb{N}$

$$f(n) = c_0 \Delta A_n + \cdots + c_k \Delta A_{n+k} = 0. \tag{2}$$

Door $f(i)$ te nemen met $i = n, n+1, \ldots, n+k-1$ krijgen we k homogene lineaire vergelijkingen met $k+1$ onbekende $c_0, \ldots, c_k$.

Als we nu de eis $c_0 + \cdots + c_k = 1$ toevoegen hebben we een lineair stelsel dat we kunnen oplossen en dit levert $c_0, \cdots, c_k$ op. We hebben dan coëfficiënten die afhangen van n en k. Schrijf $c_i^{(n,k)}$ en $e_k(A_n)$ met

$$e_k(A_n) = c_0^{(n,k)} A_n + \cdots + c_k^{(n,k)} A_{n+k} \qquad k, n = 0, 1, \ldots. \tag{3}$$

Waarbij we dus hebben dat $c_i^{(n,k)}$ de oplossing is van het stelsel

$$\begin{cases} c_0^{(n,k)} + \cdots + c_k^{(n,k)} & = 1 \\ c_0^{(n,k)} \Delta A_n + \cdots + c_k^{(n,k)} \Delta A_{n+k} & = 0 \\ \quad \vdots \qquad\qquad\qquad\quad \vdots & \quad \vdots \\ c_0^{(n,k)} \Delta A_{n+k-1} + \cdots + c_k^{(n,k)} \Delta A_{n+2k-1} & = 0. \end{cases} \tag{4}$$

Met het oplossen van het lineaire stelsel hopen we een oplossing te vinden voor vergelijking (1). In dat geval hebben we in één stap de oplossing voor ons probleem gevonden. Dit zal in de meeste gevallen niet het geval zijn, maar we komen wel dichter in de buurt van de oplossing.

We kunnen het oplossen van stelsel (4) ook schrijven als $Ax = b$ waarbij we x zoeken en A en b gegeven worden door

$$A = \begin{pmatrix} 1 & \cdots & 1 \\ \Delta A_n & \cdots & \Delta A_{n+k} \\ \vdots & & \vdots \\ \Delta A_{n+k-1} & \cdots & \Delta A_{n+2k-1} \end{pmatrix} \qquad b = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Als we dit stelsel oplossen met behulp van de Regel van Cramer, en de generalisatie van deze regel gevonden door Sylvan Burgtahler [3], vinden we de volgende vergelijking:

$$e_k(A_n) = \frac{\begin{vmatrix} A_n & \cdots & A_{n+k} \\ \Delta A_n & \cdots & \Delta A_{n+k} \\ \vdots & & \vdots \\ \Delta A_{n+k-1} & \cdots & \Delta A_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \Delta A_n & \cdots & \Delta A_{n+k} \\ \vdots & & \vdots \\ \Delta A_{n+k-1} & \cdots & \Delta A_{n+2k-1} \end{vmatrix}}. \tag{5}$$

Deze vergelijking noemen we de gegeneraliseerde Shanks transformatie.

Er geldt dat $e_1(A_n) = S(A_n)$. Dit kunnen we eenvoudig zien, want

$$\begin{aligned} e_1(A_n) &= \frac{\begin{vmatrix} A_n & A_{n+1} \\ \Delta A_n & \Delta A_{n+1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \Delta A_n & \Delta A_{n+1} \end{vmatrix}} \\ &= \frac{A_n \Delta A_{n+1} - A_{n+1} \Delta A_n}{\Delta A_{n+1} - \Delta A_n} \\ &= \frac{A_n(A_{n+2} - A_{n+1}) - A_{n+1}(A_{n+1} - A_n)}{(A_{n+2} - A_{n+1}) - (A_{n+1} - A_n)} \\ &= \frac{A_{n+2}A_n - A_{n+1}^2}{A_{n+2} - 2A_{n+1} + A_n} \\ &= S(A_n). \end{aligned}$$

Door middel van numerieke experimenten is eenvoudig in te zien dat Shanks transformatie en de gegeneraliseerde Shanks transformatie bij numerieke experimenten iets van elkaar afwijken. Dit is bijvoorbeeld te zien als we de rij definiëren als $A_{n+1} = \frac{1}{4}(A_n^2 + 2)$ met $A_0 = 0$ [5]. Er geldt hier niet dat $S_k(A_n) = e_k(A_n)$ er zit hier een kleine afwijking in door de verschillende ma-

nieren van berekenen.

2.1.3 Het algoritme

Met behulp van de gegeneraliseerde Shanks transformatie kunnen we het scalaire ε-algoritme afleiden. Hiervoor maken we gebruik van een stelling die eerder door Peter Wynn is gevonden en bewezen [5].

Stelling 2.1 (Wynn [5])

Als we definiëren

$$\varepsilon_{2m}(A_n) := e_m(A_n)$$

en

$$\varepsilon_{2m+1}(A_n) := \frac{1}{e_m(\Delta A_n)},$$

dan

$$\varepsilon_{k+1}(A_n) = \varepsilon_{k-1}(A_{n+1}) + \frac{1}{\varepsilon_k(A_{k+1}) - \varepsilon_k(A_k)}.$$

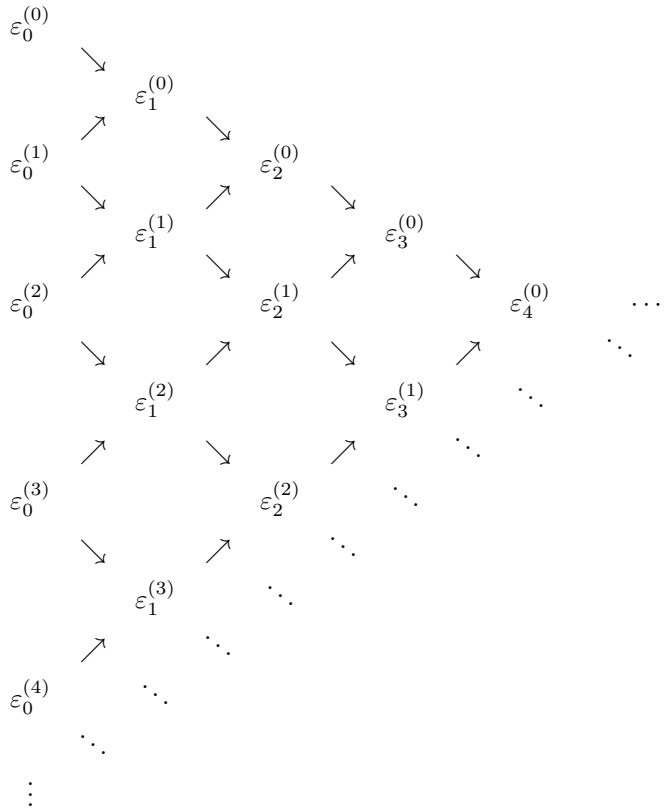
We schrijven vanaf nu $\varepsilon_k^{(n)} := \varepsilon_k(A_n)$ om het schrijfwerk iets te versimpelen en we leiden het volgende algoritme af:

$$\begin{cases} \varepsilon_{-1}^{(n)} = 0, & n = 0, 1, \ldots \\ \varepsilon_0^{(n)} = A_n, & n = 0, 1, \ldots \\ \varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + (\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)})^{-1}, & k, n = 0, 1, \ldots . \end{cases}$$

Dit noemen we het scalaire ε-algoritme.

Met behulp van dit algoritme kunnen we de Shanks transformatie bepalen zonder dat we hiervoor de determinanten van de matrices in vergelijking (5) hoeven te bepalen.

Hieronder is schematisch weergegeven welke elementen nodig zijn om de volgende elementen te maken met behulp van scalaire ε-algoritme.



(6)

We zien dus dat we de elementen $\varepsilon_0^{(n)}, \varepsilon_0^{(n+1)}, \ldots, \varepsilon_0^{(n+k)}$ nodig hebben om het element $\varepsilon_k^{(n)}$ te vormen. We zullen later, bij het topologische ε-algoritme, nog terugkomen op dit overzicht om de implementatie van het algoritme te verduidelijken.

2.2 Topologisch ε-algoritme

Als we het ε-algoritme willen uitvoeren op iets anders dan een scalair, bijvoorbeeld vectoren of matrices, komen we bij het topologische ε-algoritme. We kunnen dit algoritme uitvoeren op topologische vector ruimtes, oftewel vector ruimtes welke ook topologische ruimtes zijn. Bekende voorbeelden van topologische vector ruimtes zijn Hilbertruimtes en Banachruimtes. Dit is een voordeel ten opzichte van andere acceleratie methodes die meestal alleen werken op scalaire- of vectorruimtes.

We gaan eerste kijken hoe we de topologisch gegeneraliseerde Shanks transformatie kunnen maken, dit gaan we op vergelijkbare wijze doen als dat we de scalaire variant van de gegeneraliseerde Shanks transformatie gemaakt hebben. We gaan van de topologisch gegeneraliseerde Shanks transformatie twee varian-

ten maken. Deze twee varianten kunnen we daarna gebruiken, om net als bij het scalaire ε-algoritme, de twee topologische ε-algoritmes te vormen.

2.2.1 Topologisch gegeneraliseerde Shanks transformatie

Neem nu een willekeurige topologische vector ruimte E. Stel we hebben een rij elementen uit E, $\left((\vec{A}_n)_{n\in\mathbb{N}} \subset E\right)$ waarbij $\vec{A}_n \to \vec{A}$. Om de gegeneraliseerde Shanks transformatie hier op uit te kunnen voeren gaan we kijken naar de duale ruimte E^*, oftewel de ruimte van alle lineaire functies op E.

Vergelijkbaar met vergelijking (1) kunnen we nu de volgende vergelijking opstellen:

$$c_0(\vec{A}_n - \vec{A}) + \cdots + c_k(\vec{A}_{n+k} - \vec{A}) = 0 \in E. \tag{7}$$

We stellen hier weer de eisen dat $c_0 c_k \neq 0$ en $c_0 + \cdots + c_k = 1$. Verwant met vergelijking (2) krijgen we

$$c_0\Delta\vec{A}_n + \cdots + c_k\Delta\vec{A}_{n+k} = 0. \tag{8}$$

We nemen een $y \in E^*$, we kunnen y zelf kiezen en willen dit zo kiezen dat we bij het stelsel (10) een unieke oplossing krijgen. In hoofdstuk 3 zullen we twee voorbeelden van keuzes van y zien.

We willen nu y op $\Delta\vec{A}_n$ laten werken zodat we een scalair element krijgen. We krijgen dan

$$c_0\langle y, \Delta\vec{A}_n\rangle + c_1\langle y, \Delta\vec{A}_{n+1}\rangle + \cdots + c_k\langle y, \Delta\vec{A}_{n+k}\rangle = 0. \tag{9}$$

Het voordeel hiervan is dat we dit weer als het volgende stelsel kunnen schrijven dat we op gaan lossen om $c_0, \ldots, c_n$ te vinden.

$$\begin{cases} c_0^{(n,k)} + \cdots + c_k^{(n,k)} & = 1 \\ c_0^{(n,k)}\langle y, \Delta\vec{A}_n\rangle + \cdots + c_k^{(n,k)}\langle y, \Delta\vec{A}_{n+k}\rangle & = 0 \\ \quad\vdots \qquad\qquad\qquad\qquad \vdots & \quad\vdots \\ c_0^{(n,k)}\langle y, \Delta\vec{A}_{n+k-1}\rangle + \cdots + c_k^{(n,k)}\langle y, \Delta\vec{A}_{n+2k-1}\rangle & = 0. \end{cases} \tag{10}$$

We verkrijgen hiermee de eerste topologische Shanks transformatie

$$\hat{e}_k(\vec{A}_n) = c_0^{(n,k)}\vec{A}_n + \cdots c_k^{(n,k)}\vec{A}_{n+k} \qquad k, n = 0, 1, \ldots \tag{11}$$

en zo ook

$$\hat{e}_k(\vec{A}_n) = \frac{\begin{vmatrix} \vec{A}_n & \cdots & \vec{A}_{n+k} \\ \langle y, \Delta\vec{A}_n\rangle & \cdots & \langle y, \Delta\vec{A}_{n+k}\rangle \\ \vdots & & \vdots \\ \langle y, \Delta\vec{A}_{n+k-1}\rangle & \cdots & \langle y, \Delta\vec{A}_{n+2k-1}\rangle \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \langle y, \Delta\vec{A}_n\rangle & \cdots & \langle y, \Delta\vec{A}_{n+k}\rangle \\ \vdots & & \vdots \\ \langle y, \Delta\vec{A}_{n+k-1}\rangle & \cdots & \langle y, \Delta\vec{A}_{n+2k-1}\rangle \end{vmatrix}}. \tag{12}$$

We definiëren de tweede topologische Shanks transformatie

$$\tilde{e}_k(\vec{A}_n) = c_0^{(n,k)}\vec{A}_{n+k} + \cdots c_k^{(n,k)}\vec{A}_{n+2k} \qquad k, n = 0, 1, \ldots, \tag{13}$$

door een verschuiving van de rij $\vec{A}_n$. Hierbij zijn de waarde van $c_i^{(n,k)}$ hetzelfde als we gevonden hebben na het oplossen van stelsel (10). We krijgen dan dat

$$\tilde{e}_k(\vec{A}_n) = \frac{\begin{vmatrix} \vec{A}_{n+k} & \cdots & \vec{A}_{n+2k} \\ \langle y, \Delta\vec{A}_n\rangle & \cdots & \langle y, \Delta\vec{A}_{n+k}\rangle \\ \vdots & & \vdots \\ \langle y, \Delta\vec{A}_{n+k-1}\rangle & \cdots & \langle y, \Delta\vec{A}_{n+2k-1}\rangle \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \langle y, \Delta\vec{A}_n\rangle & \cdots & \langle y, \Delta\vec{A}_{n+k}\rangle \\ \vdots & & \vdots \\ \langle y, \Delta\vec{A}_{n+k-1}\rangle & \cdots & \langle y, \Delta\vec{A}_{n+2k-1}\rangle \end{vmatrix}}. \tag{14}$$

Een logische vraag zou zijn waarom we de tweede transformatie ook willen formuleren. Deze vraag gaan we aan het eind van dit hoofdstuk behandelen in sectie 2.2.4 als we de algoritmes hebben afgeleid.

2.2.2 Het algoritme

Vergelijkbaar met stelling 2.1 kunnen we de volgende twee stellingen formuleren.

Stelling 2.2 ([1])

Als we definiëren

$$\hat{\varepsilon}_{2k}(\vec{A}_n) := \hat{e}_k(\vec{A}_n)$$

en

$$\hat{\varepsilon}_{2k+1}(\vec{A}_n) := \frac{y}{\langle y, \hat{e}(\Delta\vec{A}_n)\rangle},$$

dan

$$\hat{\varepsilon}_{2k+1}(\vec{A}_n) = \hat{\varepsilon}_{2k-1}(\vec{A}_{n+1}) + \frac{y}{\langle y, \hat{\varepsilon}_{2k}(\vec{A}_{n+1}) - \hat{\varepsilon}_{2k}(\vec{A}_n)\rangle} \quad (\in E^*)$$

en

$$\hat{\varepsilon}_{2k+2}(\vec{A}_n) = \hat{\varepsilon}_{2k}(\vec{A}_{n+1}) + \frac{\hat{\varepsilon}_{2k}(\vec{A}_{n+1}) - \hat{\varepsilon}_{2k}(\vec{A}_n)}{\langle \hat{\varepsilon}_{2k+1}(\vec{A}_{n+1}) - \hat{\varepsilon}_{2k+1}(\vec{A}_n), \hat{\varepsilon}_{2k}(\vec{A}_{n+1}) - \hat{\varepsilon}_{2k}(\vec{A}_n)\rangle} \quad (\in E).$$

Stelling 2.3 ([1])

Als we definiëren

$$\tilde{\varepsilon}_{2k}(\vec{A}_n) := \tilde{e}_k(\vec{A}_n)$$

en

$$\tilde{\varepsilon}_{2k+1}(\vec{A}_n) := \frac{y}{\langle y, \tilde{e}(\Delta\vec{A}_n)\rangle},$$

dan

$$\tilde{\varepsilon}_{2k+1}(\vec{A}_n) = \tilde{\varepsilon}_{2k-1}(\vec{A}_{n+2}) + \frac{y}{\langle y, \tilde{\varepsilon}_{2k}(\vec{A}_{n+1}) - \tilde{\varepsilon}_{2k}(\vec{A}_n)\rangle} \quad (\in E^*)$$

en

$$\tilde{\varepsilon}_{2k+2}(\vec{A}_n) = \tilde{\varepsilon}_{2k}(\vec{A}_{n+1}) + \frac{\tilde{\varepsilon}_{2k}(\vec{A}_{n+2}) - \tilde{\varepsilon}_{2k}(\vec{A}_{n+1})}{\langle \tilde{\varepsilon}_{2k+1}(\vec{A}_{n+1}) - \tilde{\varepsilon}_{2k+1}(\vec{A}_n), \tilde{\varepsilon}_{2k}(\vec{A}_{n+2}) - \tilde{\varepsilon}_{2k}(\vec{A}_{n+1})\rangle} \quad (\in E).$$

Met behulp van stelling 2.2 en 2.3 kunnen we twee algoritmes opstellen.

Ten eerste met de stelling 2.2 kunnen we het eerste topologische ε-algoritme (TEA1) opstellen, waarbij we definiëren dat $\hat{\varepsilon}_k(\vec{A}_n) := \hat{\varepsilon}_k^{(n)}$. Het algoritme wordt gegeven door:

$$\begin{cases} \hat{\varepsilon}_{-1}^{(n)} = \vec{0}, & n = 0, 1, \ldots \\ \hat{\varepsilon}_{0}^{(n)} = \vec{A}_n, & n = 0, 1, \ldots \\ \hat{\varepsilon}_{2k+1}^{(n)} = \hat{\varepsilon}_{2k-1}^{(n+1)} + \frac{y}{\langle y, \hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)} \rangle} \quad (\in E^*), & k, n = 0, 1, \ldots \\ \hat{\varepsilon}_{2k+2}^{(n)} = \hat{\varepsilon}_{2k}^{(n+1)} + \frac{\hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)}}{\langle \hat{\varepsilon}_{2k+1}^{(n+1)} - \hat{\varepsilon}_{2k+1}^{(n)}, \hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)} \rangle} \quad (\in E). & k, n = 0, 1, \ldots \end{cases} \tag{15}$$

met de eigenschappen dat

$$\langle y, \hat{\varepsilon}_{2k}^{(n)} \rangle = \hat{e}_k(\langle y, \vec{A}_n \rangle),$$

$$\hat{\varepsilon}_{2k+1}^{(n)} = \frac{y}{\hat{e}(\langle y, \Delta \vec{A}_n \rangle)},$$

en

$$\hat{\varepsilon}_{2k+1}^{(n)} = \frac{y}{e_k(\langle y, \Delta \vec{A}_n \rangle)}.$$

Daarnaast kunnen we met stelling 2.3 het tweede topologische ε-algoritme (TEA2) opstellen, waarbij we definiëren dat $\tilde{\varepsilon}_k(\vec{A}_n) := \tilde{\varepsilon}_k^{(n)}$. Dit algoritme wordt gegeven door:

$$\begin{cases} \tilde{\varepsilon}_{-1}^{(n)} = \vec{0}, & n = 0, 1, \ldots \\ \tilde{\varepsilon}_{0}^{(n)} = \vec{A}_n, & n = 0, 1, \ldots \\ \tilde{\varepsilon}_{2k+1}^{(n)} = \tilde{\varepsilon}_{2k-1}^{(n+2)} + \frac{y}{\langle y, \tilde{\varepsilon}_{2k}^{(n+1)} - \tilde{\varepsilon}_{2k}^{(n)} \rangle} \quad (\in E^*), & k, n = 0, 1, \ldots \\ \tilde{\varepsilon}_{2k+2}^{(n)} = \tilde{\varepsilon}_{2k}^{(n+1)} + \frac{\tilde{\varepsilon}_{2k}^{(n+2)} - \tilde{\varepsilon}_{2k}^{(n+1)}}{\langle \tilde{\varepsilon}_{2k+1}^{(n+1)} - \tilde{\varepsilon}_{2k+1}^{(n)}, \tilde{\varepsilon}_{2k}^{(n+2)} - \tilde{\varepsilon}_{2k}^{(n+1)} \rangle} \quad (\in E). & k, n = 0, 1, \ldots \end{cases} \tag{16}$$

met de eigenschappen dat

$$\langle y, \tilde{\varepsilon}_{2k}^{(n)} \rangle = \tilde{e}_k(\langle y, \vec{A}_n \rangle),$$

$$\tilde{\varepsilon}_{2k+1}^{(n)} = \frac{y}{\tilde{e}(\langle y, \Delta \vec{A}_n \rangle)},$$

en

$$\tilde{\varepsilon}_{2k+1}^{(n)} = \frac{y}{e_k(\langle y, \Delta \vec{A}_n \rangle)}.$$

Door het formuleren van deze twee algoritmes kunnen we weer de eerste en tweede topologisch gegeneraliseerde Shanks transformatie bepalen zonder dat we de determinanten in de vergelijkingen (12) en (14) hoeven te bepalen.

2.2.3 Versimpelde algoritme

Bij de topologische ε-algoritmes berekenen we elementen uit de duale ruimte. Het berekenen van een element uit de duale ruimte kan lastig zijn, omdat dit een functie is. We gaan daarom de algoritmes die we gevonden hebben versimpelen zodat we geen elementen uit de duale ruimte ($\hat{\varepsilon}_{2k+1}$ en $\tilde{\varepsilon}_{2k+1}$ met $k \in \mathbb{N}$) meer hoeven te berekenen. Een bijkomend voordeel hiervan is dat we ook geen elementen uit de duale ruimte op hoeven te slaan. Wel gaan we nog, het door ons zelf gekozen element, $y \in E^*$ nodig hebben.

We nemen hiervoor de rij $A_n = \langle y, \vec{A}_n \rangle$, waardoor we met behulp van het scalaire ε-algoritme krijgen dat $\varepsilon_{2k}^{(n)} = e_k(\langle y, \vec{A}_n \rangle)$ en $\varepsilon_{2k+1}^{(n)} = \frac{1}{e_k(\langle y, \Delta \vec{A}_n \rangle)}$.

Als we nu de volgende eigenschap bekijken die gegeven is bij TEA1

$$\hat{\varepsilon}_{2k+1}^{(n)} = \frac{y}{e_k(\langle y, \Delta \vec{A}_n \rangle)},$$

dan zien we dat

$$\begin{aligned} \hat{\varepsilon}_{2k+1}^{(n+1)} - \hat{\varepsilon}_{2k+1}^{(n)} &= \frac{y}{e_k(\langle y, \Delta \vec{A}_{n+1} \rangle)} - \frac{y}{e_k(\langle y, \Delta \vec{A}_n \rangle)} \\ &= y \left(\frac{1}{e_k(\Delta A_{n+1})} - \frac{1}{e_k(\Delta A_n)} \right) \\ &= y(\varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}). \end{aligned} \tag{17}$$

Verder kunnen we met behulp van de volgende eigenschap, $\langle y, \hat{\varepsilon}_{2k}^{(n)} \rangle = \hat{e}_k(\langle y, \vec{A}_n \rangle)$, van TEA1 zien dat

$$\begin{aligned} \langle y, \hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)} \rangle &= \langle y, \hat{\varepsilon}_{2k}^{(n+1)} \rangle - \langle y, \hat{\varepsilon}_{2k}^{(n)} \rangle \\ &= \hat{e}_k(\langle y, \vec{A}_{n+1} \rangle) - \hat{e}_k(\langle y, \vec{A}_n \rangle) \\ &= \hat{e}_k(A_{n+1}) - \hat{e}_k(A_n) \\ &= e_k(A_{n+1}) - e_k(A_n) \\ &= \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}. \end{aligned} \tag{18}$$

We gaan nu vergelijking (17) en (18) gebruiken om vergelijking (15) om te schrijven en krijgen:

$$\begin{aligned}\hat{\varepsilon}_{2k+2}^{(n)} &= \hat{\varepsilon}_{2k}^{(n+1)} + \frac{\hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)}}{\langle \hat{\varepsilon}_{2k+1}^{(n+1)} - \hat{\varepsilon}_{2k+1}^{(n)}, \hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)} \rangle} \\ &= \hat{\varepsilon}_{2k}^{(n+1)} + \frac{\hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)}}{\langle y, \hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)} \rangle (\varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)})} \\ &= \hat{\varepsilon}_{2k}^{(n+1)} + \frac{\hat{\varepsilon}_{2k}^{(n+1)} - \hat{\varepsilon}_{2k}^{(n)}}{(\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)})(\varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)})}.\end{aligned} \tag{19}$$

We verkrijgen hierdoor het eerste versimpelde topologische ε-algoritme (STEA1).

Als we hetzelfde voor het tweede topologische ε-algoritme doen krijgen we dat

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

en vinden we dus het tweede versimpelde topologische ε-algoritme (STEA2).

2.2.4 Verschil tussen het eerste en tweede versimpelde algoritme

Om het voordeel van STEA2 ten opzichte van STEA1 te bekijken moeten we eerst weten hoe we deze algoritmes gaan implementeren. Dit zullen we uitleggen aan de hand van het scalaire ε-algoritme. De implementatie van STEA1 en STEA2 is verwant hieraan.

Voor een gekozen k is het ons doel om $\varepsilon_{2k}^{(0)}$ te bepalen. Als we kijken naar vergelijking (6) zien we welke elementen we nodig hebben om dit element te bepalen. We zouden dit kunnen doen door alle elementen uit te rekenen en op te slaan. Dit zou in het scalaire geval misschien nog goed te doen zijn, maar bij een vector, of iets anders wat we willen benaderen met het topologische ε-algoritme, kan dit een probleem worden. Een matrix opslaan kost meer geheugen dan een scalair. Het cache geheugen van een computer zal dus sneller vol zitten en de berekening kan hierdoor langer duren. We gebruiken daarom een andere manier om $\varepsilon_{2k}^{(0)}$ te bepalen waarbij we minder op hoeven te slaan en het cache geheugen minder snel vol zit. We kijken hiervoor naar de oplopende diagonalen.

Een oplopende diagonaal is een rij van elementen $\varepsilon_0^{(n)}, \varepsilon_1^{(n-1)}, \ldots, \varepsilon_n^{(0)}$. Uit zo'n oplopende diagonaal kunnen we, bij het scalaire ε-algoritme, de volgende oplopende diagonaal bepalen door het element $\varepsilon_0^{(n+1)}$ toe te voegen. Elke volgende oplopende diagonaal is zo 1 element langer dan zijn voorganger. In het geval van het scalaire ε-algoritme hoeven we zo, in plaats van $\frac{(2k+1)(2k+2)}{2}$ elementen, maar $(2k+1)$ elementen op te slaan om $\varepsilon_{2k}^{(0)}$ te berekenen en zo zal het cache geheugen minder snel vol raken.

Als we nu kijken naar STEA1, dan zijn er twee opeenvolgende oplopende dia-

gonalen nodig om het volgende oplopende diagonaal te bereken. Om $\hat{\varepsilon}_{2k+2}^{(n)}$ te bepalen hebben we $\hat{\varepsilon}_{2k}^{(n+1)}$, een element uit de vorige oplopende diagonaal, en $\hat{\varepsilon}_{2k}^{(n)}$, een element uit een-na-laatste oplopende diagonaal, nodig.

Bij STEA2 is het genoeg om enkel de vorige oplopende diagonaal op te slaan. Om $\tilde{\varepsilon}_{2k+2}^{(n)}$ te bepalen hebben we $\tilde{\varepsilon}_{2k}^{(n+2)}$, een element uit dezelfde oplopende diagonaal (het voorgaande bepaalde element), en $\tilde{\varepsilon}_{2k+2}^{(n+1)}$, een element uit het vorige oplopende diagonaal, nodig.

Het grote voordeel van STEA2 ten opzichte van STEA1 is dus dat we minder op hoeven te slaan om het algoritme uit te kunnen voeren. Hierdoor zal het cache geheugen, van een computer die dit algoritme uitvoert, minder snel vol raken en zal de berekening sneller kunnen verlopen.

3 Numerieke experimenten

We gaan twee numerieke experimenten uitvoeren op numerieke problemen. Per probleem krijgen we een rij van vectoren of matrices. We gaan op elk stukje van de verkregen rij het tweede versimpelde topologische ε-algoritme uitvoeren.

We kunnen kiezen hoe vaak we het algoritme uitvoeren, en definiëren hiervoor het volgende begrip *k-de ε-diepte* als volgt: Als we zeggen dat we kijken naar de k-de ε-diepte van een rij $(x_n)_{n\in\mathbb{N}}$, bedoelen we dat we kijken naar $(\varepsilon_{2k}(x_n))_{n\in\mathbb{N}}$.

Volgens [2] mag het verschil tussen twee stappen niet te klein zijn om een betere benadering te vinden. Dit komt bij veel meer acceleratie methode voor. Er is een p waarvoor geld dat als

$$\frac{\langle y, \tilde{\varepsilon}_k^{(n+1)}\rangle - \langle y, \tilde{\varepsilon}_k^{(n+1)}\rangle}{\langle y, \tilde{\varepsilon}_k^{(n+1)}\rangle} < 10^{-p}$$

dat het algoritme niet meer altijd stabiel is voor het berekenen met een computer. De p is afhankelijk van de computer precisie. Dit komt sneller voor als het verschil tussen de stappen in de originele rij klein zijn.

Een oplossing hiervoor zou kunnen zijn om een deelrij te nemen van de originele rij en hierop het tweede versimpelde topologische ε-algoritme uit te voeren. We kunnen bijvoorbeeld elk j-de element in de rij nemen. Door van de rij $(x_n)_{n\in\mathbb{N}}$ elk j-de element te nemen krijgen we de deelrij $(x_{j\cdot n})_{n\in\mathbb{N}}$

Als we naar de k-de ε-diepte van een rij kijken zien we hoe het algoritme zich gedraagt op verschillende punten in de rij. Dit is niet handig om uiteindelijk een verbetering mee te vinden in de praktijk. We weten namelijk niet wanneer we het algoritme moeten uitvoeren. Het is pas handig als we het punt dat we vinden met STEA2 weer kunnen gebruiken in om in de iteratie verder mee te werken en we zo uiteindelijk sneller bij de oplossing komen dan zonder het algoritme.

Als we het punt dat we vinden met het algoritme weer gebruiken doen we dat als volgt, gegeven dat de iteratie gegeven wordt door $x_{n+1} = f(x_n)$. We gaan hiervoor een nieuwe iteratie definiëren. We doen eerst 3 iteraties. Hierop voeren we STEA2 uit en verkrijgen dus $\varepsilon_2(x_n)$. Op het antwoord dat we gevonden hebben voeren we weer 3 iteraties uit, en op deze manier gaan we door.

Als we dit wiskundig uitdrukken kunnen we de volgende rij $\left(y_k^{(n)}\right)_{k\in\mathbb{N}}$ definiëren met $n = 1, 2$ waarbij we zeggen dat $y_0^{(0)} = x_0$ en we geven dat $y_k^{(n+1)} = f(y_k^{(n)})$ en als laatste gaan geven we $y_{k+1}^{(0)} = \varepsilon_2(y_k^{(0)})$. Door nu steeds $y_k^{(0)}, y_k^{(1)}$ en $y_k^{(2)}$ te bepalen, kunnen we $y_{k+1}^{(0)}$ bepalen.

We gaan in beide experimenten eerst de rij horende bij het originele probleem berekenen om zo een goede benadering te vinden voor het echte antwoord. We gaan de best gevonden benadering gebruiken om hiermee het verschil met overige

elementen uit de rij te bekijken. We noemen deze beste benadering S_* en u_*, voor respectievelijk het experiment met de matrix vergelijking en het experiment met de diffusie vergelijking. We bekijken het verschil met de rij en de best gevonden oplossing in een norm op logaritmische schaal.

Met de blauwe lijn wordt telkens het resultaat van de originele rij weergegeven. De rode lijn geeft het het resultaat van de rij weer dat met het algoritme werkt.

3.1 Matrix vergelijking

In het eerste probleem gaan we de vergelijking

$$S + A^* S^{-1} A - I = 0$$

oplossen, waarbij $S, A \in \mathbb{R}^{3\times 3}$, I de identiteitsmatrix is en A een gegeven inverteerbare matrix is. We gaan dit oplossen met behulp van de methode zoals beschreven in [4]. We krijgen hieruit dat

$$S_{n+1} = 2S_n - S_n A^{-*}(I - S_n)A^{-1}S_n,$$

met $S_0 = A^*A$.

We kiezen in ons probleem

$$A = \begin{pmatrix} 0.37 & 0.13 & 0.12 \\ -0.30 & 0.34 & 0.12 \\ 0.11 & -0.17 & 0.29 \end{pmatrix}.$$

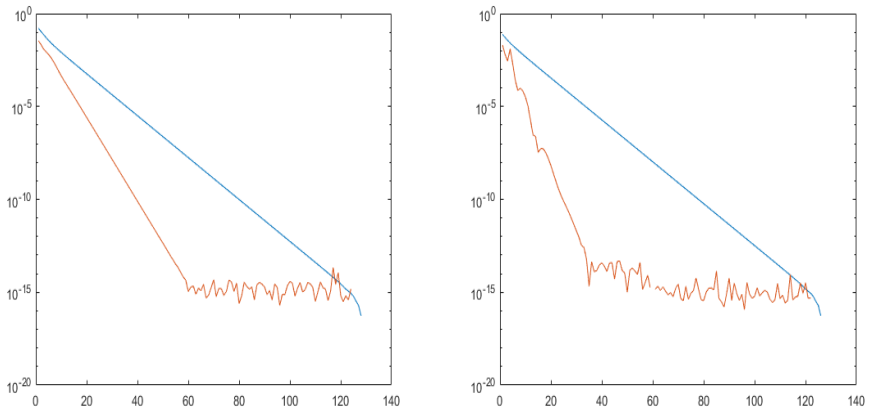
We gebruiken hier bij het toepassen van het tweede versimpelde topologische ε-algoritme dat $\langle y, S_n\rangle := \text{trace}(S_n)$. We maken bij het kijken naar de verschillen in de rij gebruikt van de Frobenius norm. De Frobenius norm van matrix S is gedefinieerd als

$$||S||_F = \sqrt{\text{trace}\, S^*S}.$$

We zien dus dat $\langle y, S_n\rangle = ||I \cdot S_n||_F = \text{trace}(S_n)$.

3.1.1 Resultaten

We gaan kijken naar het verschil tussen de gevonden benadering in de rij en S_*. We gaan kijken naar de k-de ε-diepte van de rij $(S_n)_{n\in\mathbb{N}}$. We doen dit voor $k = 1$ en $k = 2$ in figuur 1.



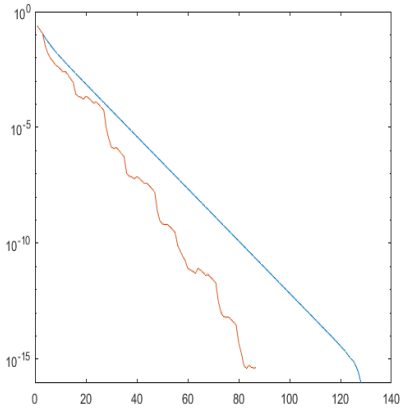
(a) $k = 1$ (b) $k = 2$

Figuur 1: k-de ε-diepte van $(S_n)_{n\in\mathbb{N}}$

We zien dat we maar de helft van de iteraties nodig hebben in de originele rij om een goede benadering te kunnen vinden met het tweede topologische ε-algoritme in het geval van $k = 1$. In het geval $k = 2$ hebben we zelfs maar $1/3$ van de iteraties nodig.

We zien hier dat de grafiek van de 2-de ε-diepte op de rij meer oscilleert dan de grafiek van 1-e ε-diepte op de rij. Wel is het hier het geval dat ook bij $k = 2$ alsnog een goede benadering wordt gevonden en sneller. Dit komt waarschijnlijk omdat de stappen in de originele rij groter zijn.

Echter, zoals eerder aangegeven, is dit in de praktijk niet handig. We gaan daarom kijken naar of we met het gevonden resultaat uit STEA2 verder kunnen werken in de iteratie. Het resultaat hiervan is terug te vinden in figuur 2.



Figuur 2: STEA2 uitgevoerd op S_n

We zien dat we hier maar 2/3e van de originele iteraties nodig hebben om tot een goede benadering te komen. Na een aantal iteraties wordt er geen verbetering meer gevonden. Dit is vermoedelijk omdat het verschil tussen de stappen te klein wordt en het versimpelde topologische ε-algoritme hier niet meer op werkt. We krijgen het antwoord 'NaN'.

3.2 Stabiel rustpunt van diffusie vergelijking

In het tweede probleem gaan we een stabiel rustpunt proberen te vinden van een diffusie vergelijking. Hierbij kijken we naar de volgende differentiaalvergelijking

$$\frac{\partial u(x,t)}{\partial t} = c \cdot \frac{\partial^2 u(x,t)}{\partial x^2} + u(x,t) - u(x,t)^3$$

met de randvoorwaarde dat $u(0,t) = 0$, $u(1,t) = 1$ en $u(x,0) = 0$ voor $0 \leq x \leq 1$.

We gaan dit oplossen met behulp van Euler's methode. We discretiseren de ruimte met $M = 100$ rooster punten. We nemen $c = 0.01$, $\Delta x = 1/M$ en $\Delta t = \frac{0.05}{c \cdot (\Delta x)^2}$ voor dit probleem.

We vinden de iteratie

$$u_{n+1} = u_n + \Delta t F(u_n), \quad u_0 = \vec{0}.$$

Waarbij geldt dat

$$F(u) = c(Lu + b) + u - u^3$$

met

$$L = \begin{pmatrix} -2 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -2 \end{pmatrix}, \quad b = \frac{1}{(\Delta x)^2} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

In het tweede versimpelde topologische ε-algoritme moeten we een y kiezen uit de duale ruimte. We nemen hier dat $\langle y, u_n \rangle$ dus som van de elementen in u_n is.

Om het tweede versimpelde topologische ε-algoritme goed te laten werken moet het verschil tussen de stappen in de rij niet te groot zijn, zoals te lezen in [2].

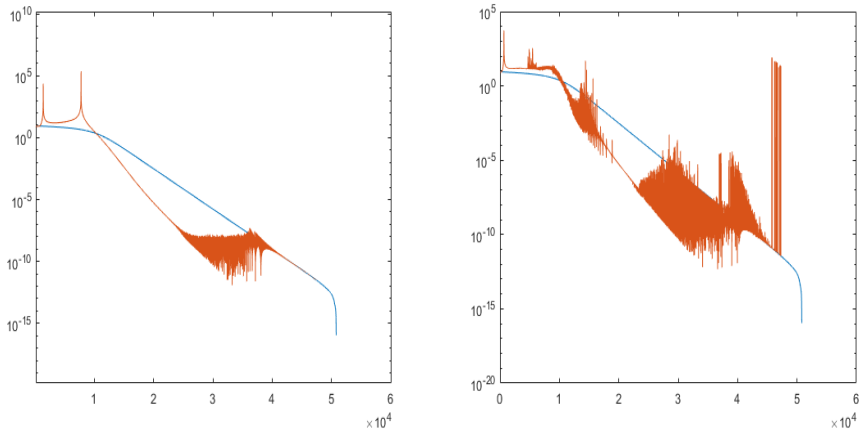
De stappen in de rij zijn in de rij die we verkrijgen bij dit probleem best klein. Om een beter resultaat te kijken hebben we dus, naast de originele rij, ook gekeken naar een deelrij, namelijk $(u_{10n})_{n\in\mathbb{N}}$.

Er is op meerdere manieren geprobeerd om het punt dat we vinden met het algoritme weer te gebruiken in onze benadering. Bijvoorbeeld door het eerste stuk van de rij weg te nemen of grotere stappen te nemen. Niks heeft hier tot een bruikbaar resultaat geleid.

3.2.1 Resultaten

We kijken hier steeds naar het verschil tussen de gevonden oplossing en u_* op de ℓ^2 norm.

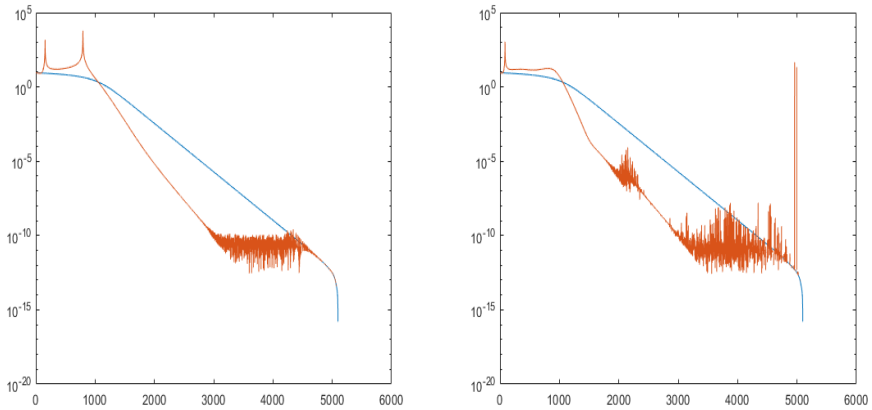
Eerst kijken we naar de k-de ε-diepte van de rij $(u_n)_{n\in\mathbb{N}}$. We doen dit voor $k = 1$ en $k = 2$. Het resultaat hiervan is te vinden in figuur 3.



(a) $k = 1$ (b) $k = 2$

Figuur 3: k-de ε-diepte van $(u_n)_{n\in\mathbb{N}}$

We zien dat de benadering erg oscilleert. We gaan nu kijken naar de k-de ε-diepte van de rij deelrij waarbij we elk 10e element nemen, oftewel $(u_{10n})_{n\in\mathbb{N}}$. We doen dit weer voor $k = 1$ en $k = 2$ in figuur 4.



(a) $k = 1$ (b) $k = 2$

Figuur 4: k-de ε-diepte van $(u_{10n})_{n\in\mathbb{N}}$

We zien dat de oscillatie al enorm is afgenomen door het nemen van een deelrij, zodat de stapgrootte groter wordt. Als we nog grotere stappen gaan nemen, door een nog kleinere deelrij van de originele rij te nemen, gaan we zien dat dit steeds minder zal oscilleren.

In het geval van $k = 1$ is te zien dat de benadering, met behulp van het algoritme, door het nemen van de grotere stappen nauwkeuriger wordt. In de 1-e ε-diepte van $(u_n)_{n\in\mathbb{N}}$ wordt het verschil tussen de benadering en u_* niet veel kleiner dan 10^{-8}, terwijl dit verschil in 1-e ε-diepte van $(u_{10n})_{n\in\mathbb{N}}$ zeker 10^{-10} wordt.

Als we kijken naar het verschil tussen de k-de ε-deelrij met $k = 1$ en $k = 2$ zien we dat bij grotere k, de grafiek weer meer gaat oscilleren. Dit komt vermoedelijk omdat voor $k = 2$ het tweede topologische ε-algoritme uitgevoerd wordt op de 1-de ε-diepte van de rij. Deze rij heeft ook weer kleinere stapjes, waardoor er meer oscillatie optreed.

4 Conclusie en suggesties voor verder onderzoek

Een groot voordeel van de versimpelde ε-methode ten opzichte van andere acceleratie methodes is dat deze methode kan werken in op iteraties in topologische vectorruimtes, terwijl ander dit vaak alleen kunnen op vector ruimtes. We hebben ook gezien dat we bij STEA2 minder op hoeven te slaan dan bij STEA1 om het te implementeren, waardoor het in sommige gevallen sneller zal werken.

De versimpelde ε-methode werkt in experiment 1, de diffusie vergelijking die we bekeken hebben, niet heel erg goed. Wel werkt de methode goed bij de matrix vergelijking die we hebben proberen op te lossen.

We hebben hier gezien dat de tweede versimpelde topologische ε-methode erg oscilleert als er te kleinen stappen in de rij genomen worden. Dit gebeurt vermoedelijk ook als we het algoritme vaak uitvoeren, en dus als de k groot wordt. We hebben gezien dat het algoritme beter werkt als er grotere stappen worden genomen in de rij die we willen verbeteren.

In verder onderzoek zou het interessant zijn om te kijken of en hoe het eerste topologische ε-algoritme andere resultaten op levert dan het tweede topologische ε-algoritme. Ook zouden we deze algoritmes kunnen vergelijken met andere extrapolatie methodes.

Daarnaast is het voor de bruikbaarheid van het algoritme handig om te onderzoeken in welke gevallen er doorgewerkt kan worden met het resultaat dat uit het algoritme komt om het algoritme zo in de praktijk goed in te kunnen zetten.

Referenties

[1] C Brezinski. Généralisations de la transformation de shanks, de la table de padé et de l'ε-algorithme. *Calcolo*, 12(4):317–360, 1975.

[2] Claude Brezinski and Michela Redivo-Zaglia. The simplified topological ε-algorithms for accelerating sequences in a vector space. *SIAM Journal on Scientific Computing*, 36(5):A2227–A2247, 2014.

[3] Sylvan Burgstahler. A generalization of cramer's rule. *The Two-Year College Mathematics Journal*, 14(3):203–205, 1983.

[4] Marlliny Monsalve and Marcos Raydan. A new inversion-free method for a rational matrix equation. *Linear algebra and its applications*, 433(1):64–71, 2010.

[5] Peter Wynn. On a device for computing the e m (s n) transformation. *Mathematical Tables and Other Aids to Computation*, pages 91–96, 1956.