Technische Universität Berlin
Institut für Mathematik

Masterarbeit
im Studiengang Scientific Computing

Vergleich von Prolongations- und Restriktionsoperatoren für Deflationsmethoden zur Loesung von Gleichungssystemen

Alessandro Bartsch <ale.bartsch@gmail.com>
Betreut von Prof. Dr. Reinhard Nabben

Version vom
17. November 2018

Eidesstattliche Erklärung zur Masterarbeitarbeit

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

$Unterschrift:$ $Berlin, den$

Zusammenfassung

Die nachfolgende Arbeit befasst sich mit dem Vergleich von Deflations- und Prolongationsoperatoren. Diese werden zur Vorkonditionierung von Projektionsmethoden zum Lösen von linearen Gleichungssystemen genutzt. Als Referenz für die Projketionsmethoden wird die GMRES-Methode verwendet. Die betrachteten Deflationsoperatoren sind zwei simple Algorithmen, welche unabhängig von der Form der Matrix Einträge auswählen, einen Algorithmus, der sich auf die Umordnung durch Grob- und Feingitterpunkte bezieht und einen Algorithmus, der die Matrix als Graph interpretiert um die Einträge durch die Nachbarn zu gruppieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Anhangsverzeichnis

1 Einleitung

Eines der am längsten betrachteten Probleme in der Mathematik ist das Lösen des linearen Gleichungssystems

$$Ax = b, \quad A \in \mathbb{C}^{n\times n}, \quad x, b \in \mathbb{C}^n. \tag{1}$$

Häufig verbergen sich hinter A sehr große Matrizen, welche die Kapazitäten vieler Rechner übersteigen. Deswegen versucht man mittels eines Operators das Gleichungssystem auf ein Kleineres zu skalieren. Die Dimension $r \ll n$ des Zielsystems orientiert sich häufig an $\frac{n}{2}$.
Der gesuchte Operator bezieht sich jedoch auf sogenannte Prolongations- bzw. Restriktionsmatrizen $P \in \mathbb{C}^{n\times r}$ und $R \in \mathbb{C}^{r\times n}$.
In vielen Fällen ist die Restriktionsmatrix jedoch nur die Transponierte der Prolongationsmatrix, also

$$P = R^T.$$

Für die Wahl solcher Matrizen wären am Besten die Eigenvektoren von A als Spalten geeignet. Die Berechnung dieser ist in der Regel zu zeitaufwändig und somit keine praktikable Lösung. Deswegen wurden Algorithmen entwickelt um geeignete Matrizen zu konstruieren.
Für die Ermittlung der Matrizen kann man den Operator

$$P_N := I - AP(RAP)^{-1}R + P(RAP)^{-1}R$$

konstruieren. Mit diesem wird die Gleichung 1 überführt in

$$P_N Ax = P_N b. \tag{2}$$

Der Vorteil dieser Gleichung besteht darin, dass die linke Seite stets nicht singulär ist. Für die Lösung der Gleichung 2 nutzen wir den GMRES. Hier wird der Operator P_N als Vorkonditionierer verwendet.

In dieser Arbeit werden die theoretischen Grundlagen wiederholt und die einzelnen Methoden zur Ermittlung der Restriktionsmatrizen, sowie das Verfahren zum Lösen der Gleichung (2) und deren Auswertungen vorgestellt. Zu Beginn orientiert sich die Arbeit für die Grundlagen stark an dem Paper von Nabben[1] [Nab17] und dem Paper von Nabben, García[2] und Kehl[3] [NKR17], da die Erarbeitung der theoretischen Grundlagen nicht der Hauptbestandteil dieser Arbeit seien soll.

[1] Er beschäftigt sich hauptsächlich mit der numerischen Mathematik an der TU Berlin.
[2] Er forscht in der angewandten und numerischen Mathematik
[3] Sein Forschungsgebiet ist vor allem die numerische Mathematik

Anschließend werden die vier im Zuge dieser Arbeit verwendeten Algorithmen zur Restriktion bzw. Prolongation vorgestellt. Durch die simple Struktur der ersten beiden Algorithmen ist hier das Hauptaugenmerk auf die zwei komplexeren Algorithmen gelegt. Die genauen Definitionen sind dem Artikel [EN09] und dem Paper [GN12] entnommen.
Im nächsten Kapitel wird, das im Zuge dieser Arbeit entwickelte, Programm vorgestellt, welches durch einen Pseudocode verdeutlicht ist und auch im Anhang zu finden ist.
Das folgende und wohl auch wichtigste Kapitel dieser Arbeit bezieht sich auf die Auswertung des Programms. Hier wird unter anderem der Fehler auf die Iteration aufgetragen und die Geschwindigkeit gemessen. Anhand dieser Werte wird ein ausführlicher Vergleich aller vier Algorithmen vorgenommen. Für die Testdurchläufe wurden ausgewählte Matrizen aus dem Matrix-Markt [mm17] und Referenzprobleme aus anderen Arbeiten verwendet.
Zuletzt erfolgt eine Abschlussbemerkung. Diese umfasst eine kurze Zusammenfassung der ausgewerteten Daten, eine Wertung Dieser und ein Fazit.

2 Theoretische Grundlagen

Der Ansatz mittels Deflation ein Gleichungssystem zu lösen ist als erstes in den späten achtzieger Jahren aufgetreten und wurde zum Lösen von linearen Gleichungssystemen mit symmetrischen Matrizen verwendet. Diese Methode wurde zu einer neuen Alternative für das schnelle Lösen eines Gleichungssystems mittels KRYLOV-Unterräumen. Man betrachtet auch hier das allgemeine Gleichungssystem (1) und möchte dieses mittels Restriktion in ein singuläres lineares Gleichungssystem mit geringerer Dimension überführen. Das Ziel ist es ein für die KRYLOV-Unterräume geeigneteres Spektrum zu erreichen. Wie in der Einleitung bereits erwähnt werden die erzeugten Deflations-Unterräume idealerweise durch die Eigenvektoren von A aufgespannt. In der Praxis verwendet man jedoch häufig Approximationen dieser Eigenvektoren.
Im Allgemeinen werden Projektionen eingesetzt um das initiale Gleichungssystem zu überführen. In Anlehnung an [NKR17] werden die Grundzüge der allgemeinen Deflation aufgezeigt.

Definition 2.1.
Eine Matrix $P \in \mathbb{C}^{n\times n}$ ist eine Projektion, falls gilt

$$P = P^2.$$

Der $\mathbb{C}^n$ kann also als direkte Summe der Range ($\mathcal{R}(P)$) und des Null-Raum ($\mathcal{N}(P)$) einer Projektion dargestellt werden. Damit lässt sich eine weitere Definition von Projektionen darstellen. Diese beruht auf eben zwei solchen Unterräumen von $\mathbb{C}^n$.

Definition 2.2.
Für zwei Unterräume $\mathcal{V}, \mathcal{W} \subseteq \mathbb{C}^n$ mit $\mathcal{V} \oplus \mathcal{W} = \mathbb{C}^n$ ist der Operator $P_{\mathcal{V},\mathcal{W}}$ als eindeutige Projektion mit den Eigenschaften

$$\mathcal{R}(P) = \mathcal{V} \text{ und } \mathcal{N}(P) = \mathcal{W},$$

definiert. Die Projektion $P_{\mathcal{V},\mathcal{W}}$ wird Projektion auf $\mathcal{V}$ entlang $\mathcal{W}$ genannt.

Deflationen im Allgemeinen werden durch Projektionen der Form $P_{\mathcal{Y}^\perp, A\mathcal{Z}}$ dargestellt. In vielen Fällen wird hier $\mathcal{Y} = A\mathcal{Z}$ verwendet. Für die Wohldefiniertheit müssen die Unterräume $\mathcal{Y}, \mathcal{Z}$ die Eigenschaften des folgenden Theorems erfüllen.

Theorem 2.3.
Seien $\mathcal{Y}, \mathcal{Z}$ zwei Unterräume von $\mathbb{C}^n$ der Dimension r, Y,Z zwei Matrizen, dessen Spalten die Basis der Räume $\mathcal{Y}, \mathcal{Z}$ bilden und A eine nicht-singuläre Matrix aus dem Raum $\mathbb{C}^{n\times n}$. Dann sind die folgenden Aussagen äquivalent:

1. $(C)^n = A\mathcal{Z} \oplus \mathcal{Y}^\perp$

2. *$Y^H AZ \in (C)^{r\times r}$ ist nicht-singulär und die Projektion $P_{A\mathcal{Z},\mathcal{Y}^\perp}$ kann durch*

$$P_{A\mathcal{Z},\mathcal{Y}^\perp} = AZ(Y^H AZ)^{-1}Y^H,$$

dargestellt werden.

3. *$(C)^n = A^{-1}\mathcal{Y}^\perp \oplus \mathcal{Z}$*

4. *$Y^H AZ \in (C)^{r\times r}$ ist nicht-singulär und die Projektion $P_{A^{-1}\mathcal{Y}^\perp,\mathcal{Z}}$ kann durch*

$$P_{A^{-1}\mathcal{Y}^\perp,\mathcal{Z}} = I - Z(Y^H AZ)^{-1}Y^H A,$$

dargestellt werden.

Damit folgt auch die Wohlefiniertheit der von der nicht-symmetrischen Deflation verwendeten Operatoren $P_D := P_{\mathcal{Y}^\perp,A\mathcal{Z}}$ und $Q_D := P_{A^{-1}\mathcal{Y}^\perp,\mathcal{Z}}$. Wählt man $\mathcal{Y} = A\mathcal{Z}$, dann erhält man außerdem die Eigenschaft, dass $Y^H AZ$ nicht singulär ist, wenn A nicht singulär ist.
Mit diesen Operatoren kann nun der Deflationsprozess beschrieben werden. Um das Gleichungssystem $Ax = b$ zu lösen wird zunächst das System

$$P_D Ax = P_D b,$$

gelöst. Die Lösung x' dieses Systems kann nun verwendet werden um das eigentliche System zu lösen. Durch Umformungen erhält man:

$$A(Q_D x' + Qb) = b.$$

Damit ergibt sich, dass die Lösung des ursprünglichen Gleichungssystem gleich $Q_D x * +Qb$ ist, also $x = Q_D x * +Qb$. Man kann diesen Prozess, so wie das in den hier aufgezeigten Versuchen praktiziert wurde, um einen Glätter $M^{-1} \approx A^{-1}$ erweitern.
Der folgende und zentrale Satz stellt sicher, dass der GMRES auch für das vorkonditionierte projezierte System

$$M^{-1}P_D Ax = M^{-1}P_D b,$$

konvergiert und die Lösung auch zu der gewünschten Lösung des Ursprungssystems führt.

Theorem 2.4.
Seien $A, M \in \mathbb{C}^{n\times n}$ nicht-singulär. Die Räume $\mathcal{Z}$ und $\mathcal{Y}$ sind Unterräume des $\mathbb{C}^n$ und die Matritzen Z und Y sind Basen von $\mathcal{Z}$ und $\mathcal{Y}$. Falls gilt

$$A\mathcal{Z} \oplus \mathcal{Y}^\perp = M\mathcal{Z} \oplus \mathcal{Y}^\perp = \mathbb{C}^n,$$

dann folgt

1. *Die Projektionen P_D und Q_D sind wohldefiniert.*

2. *Der GMRES konvergiert für Startvektor bei dem Gleichungssystem*

$$M^{-1}P_DAx = M^{-1}P_Db. \tag{3}$$

3. *Falls x_i' die approximierte Lösung des GMRES auf die Gleichung (3) ist, dann gelten für*

$$x_i = Q_Dx_i' + Qb,$$

folgende Eigenschaften:

- *Falls $r_i := b - Ax_i$ und $r_i' = P_D(b - Ax_i')$, dann gilt $r_i = r_i'$. Die Fehler sind also gleich. Damit folgt, dass spätestens $X_n = x$ gelten muss.*
- $P_Dr_i' = r_i'$
- $P_Dr_i = r_i$

Damit haben wir die nötigen Grundlagen für diese Arbeit geschaffen, nämlich die Wohldefiniertheit und die Konvergenz des GMRES-Verfahrens für die Vorkoditionierung mittels Deflations-Methoden.

3 Deflationsmethoden

In diesem Kapitel werden die vier verschiedenen Methoden aufgezeigt, die zur Konstruktion der Prolongations- und Restriktionsmatrizen verwendet werden.

3.1 diagonale Restriktion

Diese Methode ist simpelste und auch schnellste der vier vorgestellten Methoden. Sie ist vollkommen unabhängig von den Einträgen der ursprünglichen Matrix A.
Zur Erstellung benötigt man nur die Dimension n und erstellt mit Dieser eine Matrix deren Einträgen zunächst alle null betragen. Dann geht man jede Zeile i durch und setzt den Eintrag in der Spalte $j = 2i - 1$ gleich eins.
Man erhält für ein gerades n die Form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}.$$

Und für ein ungerades n erhält man die Form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \end{bmatrix}.$$

Diese Restriktion beschränkt sich demzufolge auf jeden zweiten Eintrag der Ursprungsmatrix A.

3.2 gewichtete diagonale Restriktion

Die zweite Restriktion ist, genau wie die erste, ebenfalls unabhängig von den Einträgen der vorliegenden Matrix A. Zum Aufbau wird ebenso zunächst eine Matrix mit den Einträgen null erstellt um dann in jeder Zeile i in den Spalten $2i - 1$-ten und $2i + 1$-ten den Wert $0,25$ und in der Spalte $2i$-ten den Wert $0,5$ zu setzen.

Man erhält also Matrizen der Form:

$$\begin{bmatrix} 0,25 & 0,5 & 0,25 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0,25 & 0,5 & 0,25 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0,25 & 0,5 & 0,25 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0,25 & 0,5 & 0,25 \end{bmatrix}.$$

Man erkennt, dass hier eine einfache Gewichtung der Einträge aus A vorliegt.

3.3 häufungsbasierte Restriktion

Der Algorithmus basiert auf dem *Algorithmus 4* aus dem Paper [EN09].
Die Matrix A wird hier als Graph interpretiert und auf ihre Häufungspunkte analysiert.

Definition 3.3.1.
Für eine dünn-besetzte, nicht-symmetrische Matrix $A = [a_{ij}] \in \mathbb{R}^{n \times n}$ sei $\mathcal{G}_A(\mathcal{U}, \mathcal{E})$ der Graph. Dieser besteht aus den n Knoten $\mathcal{U} = \{u_1, u_2, ..., u_n\}$ und den Kanten $\mathcal{E}$, sodass alle Kanten e_{ij} existieren. Eine Kante e_{ij} verbindet zwei Knoten u_i und u_j genau dann, wenn $a_ij \neq 0$ und $i \neq j$ gilt.

Es wird versucht eine Vergröberung mit einem Faktor von $\frac{1}{4}$ zu erreichen. Dabei konzentriert man sich zunächst auf die nahen Nachbarn, der Knoten.

Definition 3.3.2.
Die nahen Nachbarn $\mathcal{N}_i^C$ eines Knoten u_i sind durch

$$\mathcal{N}_i^C := \{u_j \in \mathcal{U} \mid e_{ij} \in \mathcal{E}\},$$

definiert. Die Untermenge der τ-starken nahen Nachbarn ist definiert durch

$$\mathcal{N}_i^{C,\tau} := \left\{ u_j \in \mathcal{N}_i^C \;\middle|\; \sqrt{\frac{a_{ij}^2}{|a_{ii}a_{jj}|}} \geq \tau \right\}.$$

Damit erreicht man zumeist eine Gruppierung von maximal drei Knotenpunkten. Man erreicht dementsprechend eine $\frac{1}{3}$-Vergröberung, die noch nicht genügt. Es sollte jeweils ein weiterer Knoten zu den Gruppen gefügt werden. Hierfür wird der distante Nachbar einbezogen.

Definition 3.3.3.
Ein Knoten u_k ist in der Gruppe der distanten Nachbarn $\mathcal{N}_i^D$ von u_i, falls dieser ein gemeinsamer nahen Nachbar der nahen Nachbarn $u_j \in \mathcal{N}_i^{C,\tau}$. Die τ-starken distanten Nachbarn sind eine Untergruppe der distanten Nachbarn, welche zusätzlich die τ-Bedingung aus der Definition der nahen Nachbarn erfüllen.

Mit dieser Einteilung kann man die einzelnen Knotenpunkte durchgehen und durch ihre Nachbarn gruppieren um eine gewünschte Vergröberung zu erzielen. Mit dieser Einteilung schafft man sich disjunkte Untermengen Ω_q mit den Indexsets $\mathcal{I}_q$. Mit diesen kann die Restriktionsmatrix R mit den Einträgen r_{iq} konstruiert werden:

$$r_{iq} = \begin{cases} 1, & i \in \mathcal{I}_q, \\ 0, & i \notin \mathcal{I}_q. \end{cases}.$$

3.4 reduktionsbasierende Restriktion

Zum Aufbau der Reduktionsmatrix muss zunächst die Matrix A zunächst umsortiert werden, sodass die Form

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

entsteht. Hierbei muss die Matrix $A_{11} \in \mathbb{R}^{n_1 \times n_1}$ alle Feingitterpunkte und $A_{22} \in \mathbb{R}^{n_2 \times n_2}$ alle Grobgitterpunkte der Matrix A enthalten. Diese Umsortierung erreichen wir durch den "Greedy Coarser" von Scott MacLachlan aus dem Anhang des Paper [GN12]. Nimmt man den F-Glätter[4], so wäre die optimale Interpolation die Matrix

$$R_{opt} := \begin{bmatrix} -A_{11}^{-1} A_{12} \\ I \end{bmatrix}.$$

Die Berechnung der Inversen $-A_{11}^{-1}$ ist jedoch widerum sehr rechenintensiv. Deswegen ersetzt man A_{11} durch eine Approximation $D \approx A_{11}$, welche leicht zu invertieren ist und in einer dünn besetzten Inversen resultiert.
Für diese Arbeit wurde D als Diagonalmatrix von A_{11} gewählt:

$$D = \operatorname{diag}(A_{11}).$$

Somit erhält man als Reduktionsmatrix

$$R = \begin{bmatrix} -\operatorname{diag}(A_{11})^{-1} A_{12} \\ I \end{bmatrix}.$$

[4]Der F-Glätter arbeitet nur auf dem $\mathbb{R}^{n_1}$

4 Verfahren

Um die Gleichungssysteme zu lösen wird zunächst eine Restriktionsmatrix R benötigt. Hierzu wird eine der in Kapitel 3 vorgestellten Methoden benutzt.
Für den Vorkonditionierer definieret man den Prolongationsoperator $P := R^T$. Mit Hilfe dieser Operatoren kann man nun das erste transformierte Gleichungssystem aufstellen.

$$P_D A\tilde{x} = P_D b,$$

mit dem Operator

$$P_D := I - AP\left(RAP\right)^{-1} R.$$

Um eine optimale Lösung zu erhalten wird dieser Operator verschoben um $Q := P\left(RAP\right)^{-1} R$. Man erhält den aus der Literatur bekannten Operator

$$\begin{aligned} P_N : &= P_D + Q, && (4)\\ &= I - AQ + Q. && (5) \end{aligned}$$

Mit diesem Operator wird die Hälfte der Berechnungen durchgeführt.
Als weitere Variante glättet man den ersten Teil des Operators P_N um $M = \mathrm{diag}(A)$ und erhält den Operator

$$P'_N := M^{-1} P_D + Q. \quad (6)$$

Diese beiden Operatoren werden zum Lösen den Gleichungssystems als Vorkonditionierer in den neustartenden GMRES übergeben.
Da zum Teil sehr kleine Zahlen in den Testmatrizen zu finden sind, kann es vorkommen, dass die Inverse von RAP nicht berechnet werden konnte. War dies der Fall, so wurde hierfür die Pseudo-Inverse der entsprechenden Matrix verwendet. Der Algorithmus verwendet den GMRES aus dem Julia-internen "LinearAlgebra"-Paket. Er wurde so konfiguriert, dass die Toleranz bei 10^{-22} liegt und die maximale Iterationszahl mindestens 10000 beträgt. In der Standardkonfiguration sind bei diesem Algorithmus automatische Restarts konfiguriert. Diese wurden für genauere Ergebnisse auskonfiguriert. Dadurch erhält man natürlich eine deutlich höheren Speicherbelegung.

Data: Matrix $A \in \mathbb{R}^{n \times n}$, Array $b \in \mathbb{R}^{n \times 1}$, Restriktionsmatrix $R \in \mathbb{R}^{n \times m}$ mit $m \ll n$, Boolean $useM3$

Result: Approximierte Lösung x, Historie h

$n = \text{size}(A, 1)$;

$P = R^T$;

try:

| $Q = P(RAP)^{-1}R$

catch *SingularException*:

| $Q = P * \text{pseudoinverse}(RAP) * R$

end

`/* Ermittle den Vorkonditionierer */`

if *useM3* **then**

| **try:**
| | $InvDiagA = \text{diag}(A)^{-1}$
| **catch** *SingularException*:
| | $InvDiagA = \text{pseudoinverse}(\text{diag}(A))$
| **end**
| $P_N = InvDiagA(I - AQ) + Q$

else

| $P_N = I - AQ + Q$

end

`/* maximale Iteration auf` $\max(10000, 2n)$ `und Toleranz auf` 10^{-22} `*/`

return $\text{gmres}(A, b, P_N)$

Algorithm 1: Algorithmus zum Lösen des Gleichungssystems $Ax = b$ mit GMRES

5 Auswertung mit Glättung

In diesem Kapitel erfolgt die Auswertung, des in Kapitel 4 vorgestellten Verfahrens. Zunächst wird jeder Algorithmus aus Kapitel 3 einzeln untersucht. Danach erfolgen dann die Vergleiche der Algorithmen untereinander. Abschließend werden alle Ergebnisse mit denen verglichen, welche mit dem Vorkonditionierer entstanden, der keine Vorglättung erhielt.
Alle Versuche werden auf dem gleichen Computer ausgeführt. Dieser ist mit dem Betriebssystem "Windows 10" ausgestattet und wird 16 RAM und dem Prozessor "Intel® Core™ i7-4771 CPU @ 3,5GHz" angetrieben.
Das Programm wurde vollständig in Julia[5] v.1.0.1 geschrieben und verwendet ausschließlich Matrizen aus dem Matrix-Market ([mm17]).
Im folgenden werden die Auswertungen mit dem vorkonditionierten Operator (6) durchgeführt. Falls es zu großen Abweichungen zu dem in der Literatur häufiger zu findenden Operator (4) vorkommt, so werden diese am Ende dieses Kapitels vorgestellt.
Für die Auswertungen werden Matrizen aus der Harwell-Boeing-Kollektion verwendet, da man hier ein breites Spektrum finden kann. Die Spanne reicht von kleinen Matrizen, welche nur zu Widerlegung bestimmter Thesen für Sparse-Matrizen konstruiert wurden, bis hin zu großen Matrizen, die man direkt aus realen Testfällen abgeleitet hat.

Matrix	Größe	Konditionszahl
BCSSTK25	15439 x 15439	65
GEMAT11	4929 x 4929	3.74e+08
BCSSTK24	3562 x 3562	65
BCSSTK23	3134 x 3134	6.9e+12
BCSSTM26	1922 x 1922	2.6e+05
WATT1	1856 x 1856	5.38e+09
BCSPWR09	1723 x 1723	-
BCSPWR07	1612 x 1612	-
SAYLR3	1000 x 1000	1e+02
DWT992	992 x 992	-
GR3030	900 x 900	3.8e+02
NOS7	729 x 729	4.1e+09
NNC666	666 x 666	1.8e+11
MBEAUSE	496 x 496	-
DWT234	234 x 234	-
JGL009	9 x 9	-

Tabelle 1: Testmatrizen aus der Harwell-Boeing-Kollektion

[5] siehe https://julialang.org/

5.1 Diagonale Restriktion

Dieses Kapitel befasst sich mit der Auswertung der im Kapitel 3.1 vorgestellten Restriktionsmethode. Da die Restriktionsmatrix einzig anhand der Dimension der Matrix A erstellt wird, werden komplexe Strukturen nicht beachtet. Es wird lediglich jeder zweite Eintrag der Matrix betrachtet und somit bezieht dieser Algorithmus am wenigstens Einträge der eigentlichen Matrix mit in die Berechnung des Vorkonditionierers mit ein. Im gröbsten können die Ergebnisse unterteilt werden in konvergiert und nicht-konvergiert.

nicht-konvergiert	konvergiert
GEMAT11	GR3030
BCSSTK25	JGL009
MBEAUSE	WATT1
NNC666	BCSPWR07
	BCSPWR09
	BCSSTK23
	BCSSTK24
	DWT234
	DWT992
	BCSSTM26
	NOS7
	SAYLR3

Tabelle 2: Übersicht zur Konvergenz für die diagonale Restriktion

Eine Eigenschaft, die drei der vier Matrizen auf der nicht-konvergenten Seite eint, ist die hohe Konditionszahl. Für die vierte Matrix liegen hier keine Informationen vor. Drei andere Matrizen dieser Seite werden dadurch geeint, dass man bei der Berechnung des Glätters für den Vorkonditionierer auf die Pseudoinverse ausweichen musste. Dies mag auch der Grund sein, weswegen das Residuum dieser Matrizen für den gesamten Durchlauf bei positiv Unendlich lag. Ausgenommen von dieser Eigenschaft ist die Matrix BCSSTK25. Aber auch bei anderen Matrizen kam es zur Berechnung der Inversen von A mittels Pseudoinversität, so zum Beispiel die Matrix JGL009. Hier konvergierte das Verfahren aber sehr überzeugend.

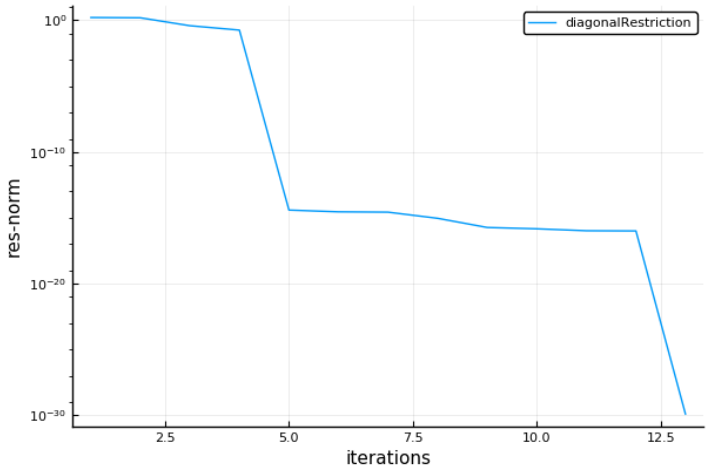

Abbildung 1: Konvergenz der Matrix JGL009 mit der diagonalen Restriktion

Dieser typische Verlauf der konvergenten Durchläufe sieht durch seine geringe Iterationszahl natürlich sehr glatt aus. Doch auch für Durchläufe mit deutlich höheren Iterationszahl ist das Muster wiederzuerkennen. In einer Anfangsphase, welche in einigen Fällen deutlich länger oder kürzer ausfällt, fällt das Residuum nur sehr schwach um dann abrupt in einen deutlich stärkerer Fall überzugehen. Im Anschluss wird dieser Verlauf ein zweites mal aufgezeigt und dies auch mit einem abrupten Übergang in ein schwaches Gefälle.
Es gibt jedoch auch verschiedene Formen dieses Musters. So zum Beispiel erkennt man für die Matrix NOS7 eine höhere Wiederholungsrate des Musters und etwas abgeglättete Übergänge.

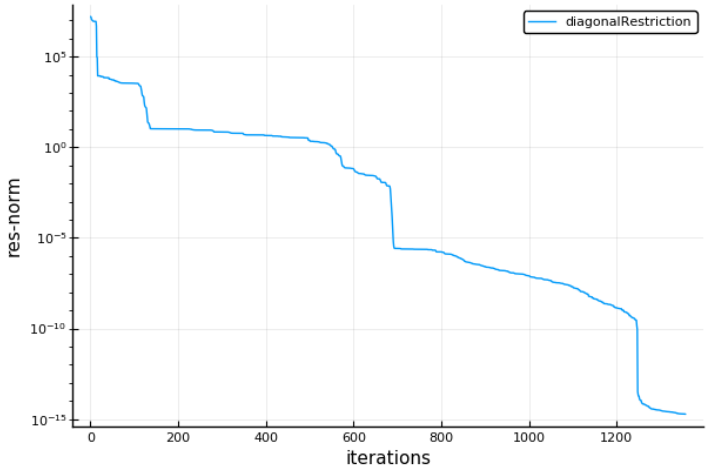

Abbildung 2: Konvergenz der Matrix NOS7 mit der diagonalen Restriktion

Die mit Abstand am meisten abgerundeten Übergänge sind jedoch bei den Matrizen der Reihe BCSPWR zu finden. Man kann auch hier deutlich das Muster des Durchlaufs mit der Matrix JGL009 wiedererkennen, obwohl die anfängliche Phase sich hier deutlich länger aufzeigt. Die Anzahl an Wechseln ist bei diesem Durchlauf aber identisch.

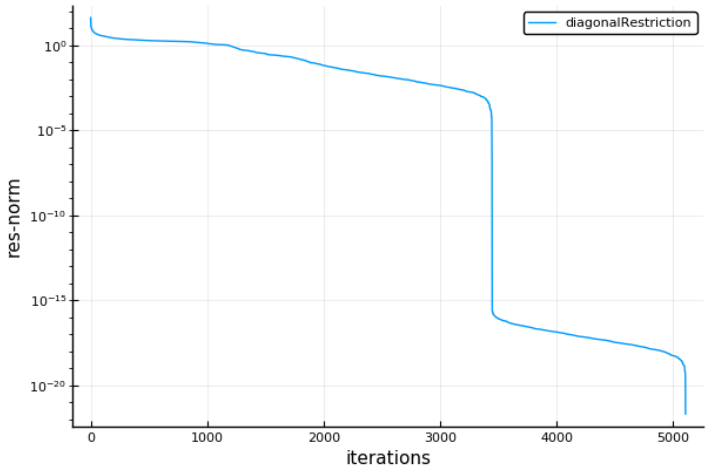

Abbildung 3: Konvergenz der Matrix BCSPWR09 mit der diagonalen Restriktion

Die Matrizen SAYLR3 und WATT1 weichen am meisten von dem Beschriebenen ab. Hier kann man fast einen teilweisen Übergang in eine Linearität erkennen. Im Detail sind dies aber nur sehr schnelle Wiederholungen des oben Beschriebenen Wechsels zwischen starkem und schwachen Fall des Residuums.

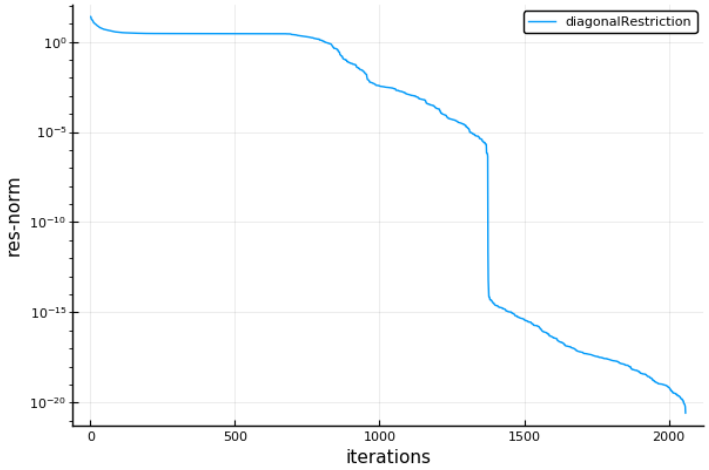

Abbildung 4: Konvergenz der Matrix SAYLR3 mit der diagonalen Restriktion

Schaut man sich die Laufzeit der Durchläufe an, so korreliert diese stark mit der Größe und Komplexität der Matrizen.

Matrix	Zeit	Speicher
BCSSTK25	22183s	45.3GB
GEMAT11	381s	5.21GB
BCSSTK24	141s	2.84GB
BCSSTK23	101s	2.41GB
NNC666	40.6s	3.07GB
BCSPWR09	33.0s	1.69GB
MBEAUSE	30.9s	3.04GB
BCSPWR07	26.7s	1.62GB
WATT1	17.9s	1.35GB
BCSSTM26	12.6s	1.59GB
SAYLR3	3.80s	0.97GB
GR3030	2.79s	0.93GB
DWT992	2.46s	0.94GB
NOS7	1.34s	892MB
JGL009	413ms	764MB
DWT234	365ms	791MB

Tabelle 3: Übersicht zu Zeit und Speicher für die diagonale Restriktion

Man kann deutlich erkennen, dass die Matrix BCSSTK25 die mit Abstand längste Zeit und den meisten Speicher benötigte. Die Tabelle spiegelt außerdem eine grobe Einteilung nach Größe wieder. So sind etwa die ersten drei Matrizen auch in der exakten Anordnung nach Größe. Durch die nicht-Konvergenz fallen Matrizen wie die NNC666 aus dieser Reihenfolge heraus. Dies liegt auch daran, dass die maximale Iterationszahl erreicht wurde. Den Abschluss bildet wider Erwartung die Matrix DWT234, trotz der 9×9 großen Matrix JGL009. Der verwendete Speicher korreliert sehr stark mit der Größe der Matrizen im Zusammenhang mit der Iterationszahl.
Die zeitliche Übersicht und der verbrauchte Speicher zur Erstellung der Restriktionsmatrix bedarf an dieser Stelle keiner Tabelle, da man hier aus der Theorie und und in den Ergebnissen eine direkte Verbindung zwischen Größe der Matrix und dem Ergebnis deuten kann.

5.2 Gewichtete diagonale Restriktion

Die im Kapitel 3.2 vorgestellte Restriktion erzielt ähnliche Ergebnisse, wie die im vorherigen Kapitel Vorgestellten. Bei dieser Berechnung werden alle Einträge zur Berechnung des Vorkonditionierers mit Hilfe einer Gewichtung einbezogen. Es ist auch hier die Struktur der Matrix unerheblich und nimmt somit keinen Einfluss auf die Berechnung der Restriktion ein. Die Übersicht zur Konvergenz zeigt, dass es nur für drei Matrizen bei einer nicht-Konvergenz blieb.

nicht-konvergiert	konvergiert
GEMAT11	GR3030
BCSSTK25	JGL009
MBEAUSE	WATT1
	BCSPWR07
	BCSPWR09
	BCSSTK23
	BCSSTK24
	DWT234
	DWT992
	BCSSTM26
	NOS7
	SAYLR3
	NNC666

Tabelle 4: Übersicht zur Konvergenz für die diagonale gewichtete Restriktion

Für die Berechnung des Vorkonditionierers wurde für die Matrix MBEAUSE die Glättung durch die Pseudoinverse von A dargestellt. Daher wurde hier ein konstantes Residuum von positiv Unendlich verzeichnet. Die Matrizen BCSSTK25 und GEMAT11 weisen überdies eine hohe Konditionszahl auf. Für die Matrix MBEAUSE liegt leider kein Wert für die Konditionszahl vor. Schaut man sich den Verlauf von GEMAT11 und BCSSTK25 an, erkennt man, dass auch für diese Matrizen eine Konvergenz erreicht worden wäre, wenn man die maximale Iterationszahl erhöht hätte.

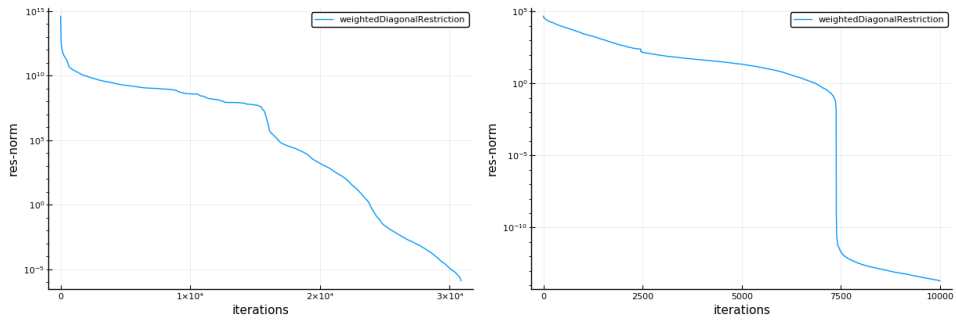

Abbildung 5: Konvergenz der Matrizen BCSSTK25 und GEMAT11 (v.l.) mit der diagonalen gewichteten Restriktion

Der Verlauf von BCSSTK25 ist sehr ähnlich zu den anderen Verläufen der BCSST-Matrizen. Die anderen Matrizen der BCSSTK-Reihe konvergiertem sogar bei ähnlichen Residuums-Werten um 10^{-5}. Dies ist relativ ungewöhnlich da die restlichen Matrizen alle bei Residuums-Werten von $< 10^{-22}$ konvergierten, so wie es auch als relative Toleranz angegeben war. So zum Beispiel war dies der Fall bei der Matrix BCSPWR07.

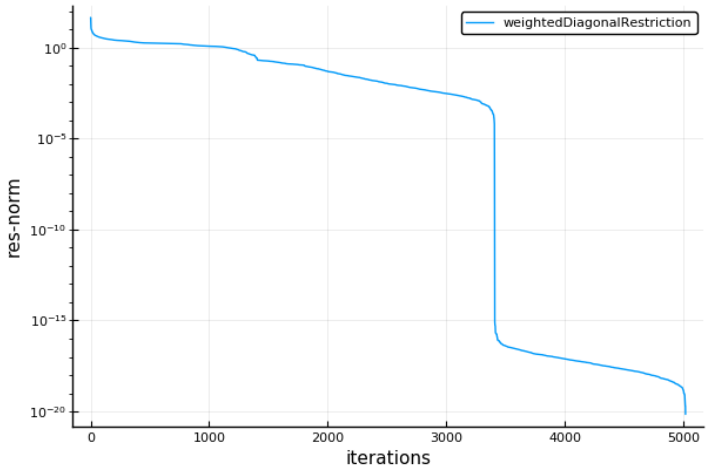

Abbildung 6: Konvergenz der Matrix BCSPWR07 mit der diagonalen gewichteten Restriktion

An diesem Verlauf ist auch sehr gut das typische Konvergenzverhalten zu erkennen. Es gibt eine beliebige Anzahl von Umbrüchen zwischen sehr schwachen und sehr starken Fall des Residuums. Ein Beispiel, welches dies in mehreren Facetten demonstriert ist der Verlauf für den Durchlauf der Matrix GR3030. Man hat hier einen zunächst sehr schnellen Wechsel zwischen den beiden Fallarten um dann in einen nahezu konstanten Verlauf überzugehen.

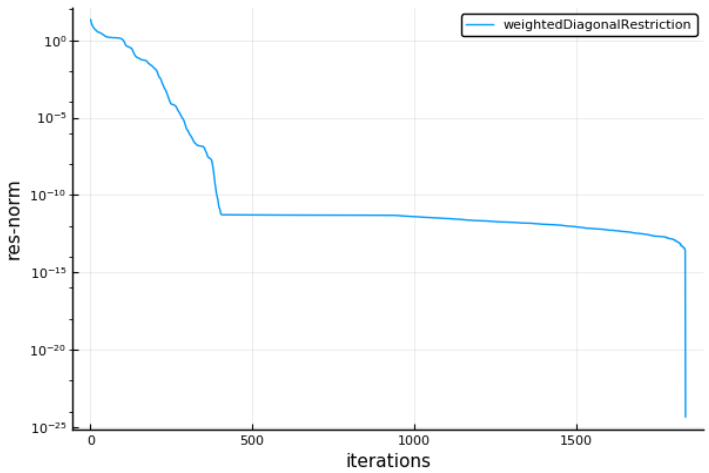

Abbildung 7: Konvergenz der Matrix GR3030 mit der diagonalen gewichteten Restriktion

Den Abschluss bildet jedoch wieder ein sehr starker Fall des Residuums um darin zu konvergieren. Diesen stetigen Wechsel am Anfang des Verlaufes kann man bei vielen anderen Durchläufen ebenfalls beobachten und er lässt den Graphen in einigen Fällen nahezu linear wirken.
Vergleicht man die Geschwindigkeiten der Matrizen, so erkennt man, dass umso größer die Matrix war, umso länger hat die Berechnung gebraucht. Daher entspricht die Anordnung nach Geschwindigkeit fast der Anordnung nach Größe der Matrizen.

Matrix	Zeit	Speicher
BCSSTK25	22188s	45.3GB
GEMAT11	365s	4.71GB
BCSSTK24	138s	2.84GB
BCSSTK23	91.5s	2.37GB
BCSSTM26	37.3s	1.45GB
BCSPWR09	33.7s	1.68GB
MBEAUSE	31.0s	3.04GB
BCSPWR07	27.7s	1.58GB
WATT1	17.9s	1.35GB
SAYLR3	3.90s	0.97GB
GR3030	2.79s	0.93GB
DWT992	2.46s	0.94GB
NOS7	1.71s	905MB
NNC666	1.01s	869MB
JGL009	401ms	764MB
DWT234	337ms	790MB

Tabelle 5: Übersicht zu Zeit und Speicher für die diagonale gewichtete Restriktion

Die Matrix BCSSTK25 benötigt eindeutig die meiste Zeit geschuldet durch ihre mit Abstand größten Dimension, der damit hohen maximalen Iterationszahl von über 30000 und dem Umstand, dass der Durchlauf nicht konvergierte. Etwas anders als eine Reihenfolge, welche durch die Dimension vorgegeben wäre, ordnet sich WATT1 ein, welche eine hohe Konditionszahl hat und eine größere Dimension als die Matrizen BCSPWR07 und BCSPWR07. Trotzdem ist sie erst nach diesen Matrizen zu finden. Außerdem ist die recht kleine Matrix MBEAUSE, geschuldet durch ihre nicht-Konvergenz, auch recht weit oben in der Tabelle zu finden. Und auch wenn die kleinste Matrix den geringsten Speicher in Anspruch genommen hat, so benötigte der Durchlauf für die deutlich größere Matrix DWT234 etwas weniger Zeit. Es handelt sich hier jedoch um einige Millisekunden.
Der Blick auf die Tabelle zur Berechnung der Restriktionsmatrix lohnt sich nicht, da man hier den direkten Zusammenhang zu der Dimension der Matrix sieht.

5.3 Häufungsbasierte Restriktion

In diesem Teilkapitel werden die Ergebnisse des geglätteten Vorkonditionierers mit der ermittelten Restriktion aus Kapitel 3.3 vorgestellt. Der Algorithmus ist einer der komplexeren in dieser Arbeit. Man kann erkennen, dass auch hier der Durchlauf für den überwiegenden Teil der Matrizen konvergierte.

nicht-konvergiert	konvergiert
GEMAT11	GR3030
BCSSTK25	JGL009
MBEAUSE	WATT1
NNC666	BCSPWR07
	BCSPWR09
	BCSSTK23
	BCSSTK24
	DWT234
	DWT992
	BCSSTM26
	NOS7
	SAYLR3

Tabelle 6: Übersicht zur Konvergenz für die häufungsbasierte Restriktion

Es fällt besonders auf, dass bei der Berechnung des Vorkonditionierers von drei der nicht-konvergenten Matrizen -alle außer BCSSTK25- es durch eine erhöhte Verwendung der Pseudoinversen zu Einträgen von positiv Unendlich kam. Dies scheint der Grund für ein konstantes Residuum von positiv Unendlich und damit auch für die nicht-Konvergenz. Einzig für die Matrix BCSSTK25 kann man erkennen, dass es vermutlich zu einer Konvergenz gekommen wäre, hätte man die maximale Iterationszahl erhöht.
Der typische Verlauf des Residuums aus den vorangehenden Kapiteln findet sich auch bei der häufungsbasierten Restriktion wieder. So kann man zum Beispiel an der Matrix BCSPWR07 sehen, wie sich der starke und sehr schwache Fall des Residuums abrupt abwechseln.

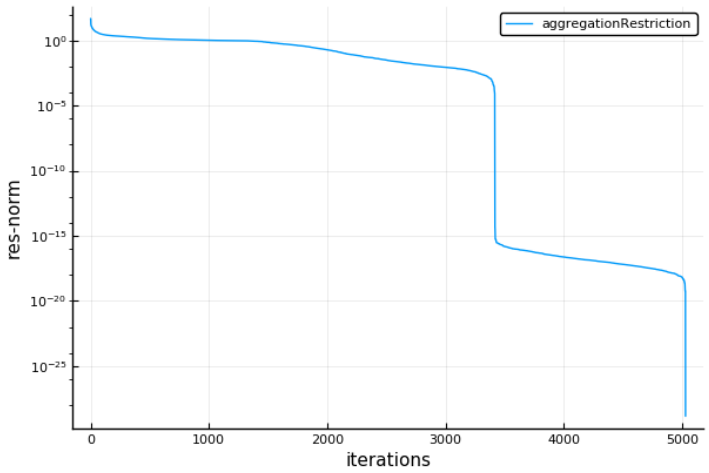

Abbildung 8: Konvergenz der Matrix BCSPWR07 mit der häufungsbasierten Restriktion

Bei genauem Betrachten fällt schnell auf, dass das Residuum fast aller Plots im letzten Iterationsschritt sehr gering ist. Bis auf die Matrizen BCSSTK23, BCSSTK24 und NO7 liegt das Residuum bei $< 10^{-25}$ und zum Teil sogar $< 10^{-30}$.
Die Verläufe der BCSSTK-Matrizen ähneln sich auch hier untereinander und denen aus

den vorherigen Kapiteln und für die Matrix NOS7 kann eine sehr schneller Wechsel der oben beschriebenen typischen Struktur der Graphen beobachtet werden.

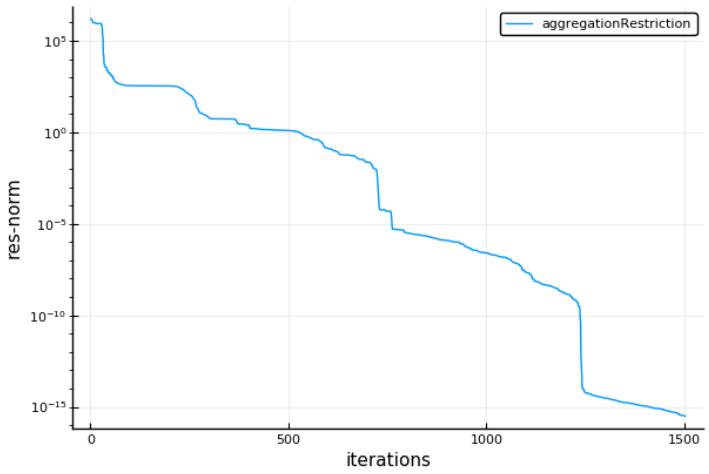

Abbildung 9: Konvergenz der Matrix NOS7 mit der häufungsbasierten Restriktion

Insgesamt kann man kein unerwartetes Verhalten in den Verläufen erkennen. Die Matrix BCSSTK25 ist mit hoher Wahrscheinlichkeit durch ihre hohe Dimension nicht in der angegebenen maximalen Iterationszahl konvergiert, aber der Verlauf lässt eine Konvergenz vermuten. Somit sind auch hier nur die fehlerhaften Berechnungen durch die Pseudoinversen Schuld an der nicht-Konvergenz einiger Matrizen.
Betrachtet man die Geschwindigkeit der Berechnungen, so muss man eine Unterscheidung machen. Durch den komplexeren Aufbau zu der Berechnung der Restriktion wird die Betrachtung unterteilt auf die Zeit die benötigt wurde um die Restriktionsmatrix zu berechnen und die, die benötigt wurde um das Gleichungssystem zu lösen beziehungsweise bis zum letzten Iterationsschritt die Berechnungen durchzuführen.
Bei dem Vergleich zum Aufbau der Restriktionsmatrix fällt auf, dass die Zeit nicht direkt von der Größe der Matrizen abhängt.

Matrix	Zeit
BCSSTK25	65.0s
BCSSTK24	4.47s
BCSSTK23	2.92s
BCSSTM26	1.48s
WATT1	781ms
BCSPWR07	720ms
BCSPWR09	688ms
GEMAT11	561ms
GR3030	373ms
DWT234	300ms
SAYLR3	287ms
DWT992	246ms
NOS7	149ms
MBEAUSE	68.1ms
NNC666	49.8ms
JGL009	71.4μs

Tabelle 7: Zeitliche Übersicht zur Erstellung der Restriktionsmatrizen für die häufungsbasierte Restriktion

Man erkennt, dass für die Matrix BCSSTK25 die längste Zeit in Anspruch genommen werden musste und im Gegensatz zu ihr die Matrix JGL009 innerhalb weniger Mikrosekunden behandelt werden konnte. Dies liegt aber vor allem daran, dass diese beiden Matrizen in der Größenordnung einen wirklich großen Abstand haben.
Jedoch findet man die zweitgrößte Matrix, welche ungefähr ein Drittel der Dimension von BCSSTK25 besitzt, erst im mittleren Teil der Tabelle. Mehrere Matrizen mit weniger als der Hälfte der Dimension haben mehr als das doppelte der Zeit benötigt um die Restriktionsmatrix zu ermitteln. Und für die zweit-kleinste Matrix DWT234 werden nur wenige Millisekunden weniger zur Berechnung benötigt.
Betrachtet man hingegen die benötigte Zeit für die Berechnung der Lösung des Systems, so erhält man eine Anordnung, welche nahezu der Anordnung nach Größe entspricht.

Matrix	Zeit	Speicher
BCSSTK25	22217s	45.4GB
GEMAT11	375s	4.44GB
BCSSTK24	140s	2.88GB
BCSSTK23	107s	2.56GB
NNC666	40.2s	3.06GB
BCSPWR09	33.9s	1.68GB
MBEAUSE	30.8s	3.03GB
BCSPWR07	27.8s	1.58GB
WATT1	18.0s	1.36GB
BCSSTM26	9.40s	1.26GB
SAYLR3	4.15s	1.02GB
GR3030	2.85s	0.95GB
DWT992	2.18s	937MB
NOS7	1.60s	907MB
DWT234	310ms	791MB
JGL009	260ms	764MB

Tabelle 8: Übersicht zu Zeit und Speicher für das Lösen mit der häufungsbasierten Restriktion

Man kann erkennen, dass die größte Matrix auch die meiste Zeit in Anspruch genommen hat und die kleinste am wenigsten. Die Verschiebung der Matrizen MBEAUSE und NNC666 liegt an der nicht-Konvergenz und der damit verbundenen hohen Iterationszahl. Interessant ist, dass die Matrizen WATT1 und BCSSTM26 schneller als die deutlich kleinere Matrix BCSPWR07 waren. Auch mit der Speicherauslastung verhält es sich ähnlich. Man bekommt hier ein durch zum Teil nicht-konvergierte Durchläufe leichte Abwandlung der Anordnung nach der Dimension der Matrizen. Die Matrix mit einer deutlich höheren Konditionszahl und geringfügig niedrigeren Dimension NOS7 hat im Gegensatz zu GR3030 zeitlich und speicherplatztechnisch weniger als die Hälfte benötigt. Dies lässt darauf schließen, dass diese Variablen eher mit der Dimension als der Konditionszahl korrelieren.

5.4 Reduktionsbasierende Restriktion

Für den Algorithmus aus 3.4 hält sich die Übersicht der konvergierten und nicht-konvergierten Durchläufe ungefähr die Waage. Man kann feststellen, dass die Seite der Konvergenz deutlich überwiegt. Die Matrix BCSSTK25, welche eindeutig den höchsten Aufwand benötigte konvergierte ebenso. Besonders ist auf der Seite der nicht-Konvergenz, dass die Matrix SAYLR3 erscheint. Wie auch später in dem Vergleich mit dem ungeglätteten Vorkonditionierer zu sehen ist, liegt dies wohl an der reduktionsbasierten Methode. Wir erhalten hier ein konstantes Residuum von positiv Unendlich.

nicht-konvergiert	konvergiert
GEMAT11	GR3030
SAYLR3	JGL009
MBEAUSE	WATT1
NNC666	BCSPWR07
	BCSPWR09
	BCSSTK23
	BCSSTK24
	BCSSTK25
	DWT234
	DWT992
	BCSSTM26
	NOS7

Tabelle 9: Übersicht zur Konvergenz für die reduktionsbasierte Restriktion

Auch auffällig sind die wieder durch die Pseudoinverse gestörten Berechnungen für die restlichen drei Matrizen. Ebenso wie für die Matrix SAYLR3 konnte kein Plot erstellt werden, da man hier dauerhaft einen Wert von positiv Unendlich für das Residuum erhält.
Schaut man sich den Verlauf der Matrix BCSSTK25 an, so erkennt man, dass das Residuum wahrscheinlich auch weiterhin gefallen wäre, hätte man eine geringere Toleranz angesetzt. Es konvergierte jedoch unter Vorgaben bei einem Wert von $\approx 10^{-7}$ in der Iteration 30251.

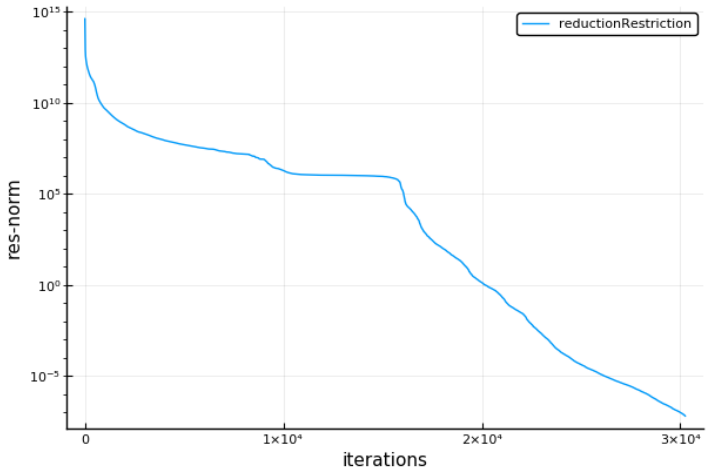

Abbildung 10: Konvergenz der Matrix BCSSTK25 mit der reduktionsbasierten Restriktion

Bei der Betrachtung fällt die relativ lange Konstanz in dem Bereich von 10^6 des Residuums. Bei höherer Toleranz wäre der Durchlauf sehr wahrscheinlich hier beendet worden und man hätte eine nicht zufrieden stellende Lösung. Dieses Verhalten lässt sich bei allen Matrizen der BCSSTK-Reihe erkennen.
Auch bei der Matrix WATT1 ist dieses Verhalten zu erkennen. Bei diesem Durch-

lauf geht jedoch ein steiler Fall des Residuums voraus, welcher dann in einen deutlich Geringeren übergeht.

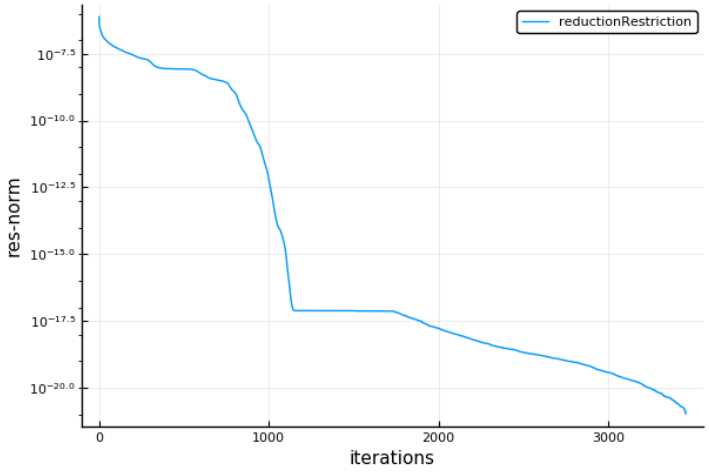

Abbildung 11: Konvergenz der Matrix WATT1 mit der reduktionsbasierten Restriktion

Man kann bei diesem und auch vielen weiteren Verläufen auch erkennen, dass das Residuum meist unter einem Wert von 10^{-20} konvergiert und zum Teil sogar bis auf 10^{-30} sinkt. Dies ist auch der Fall bei den Matrizen DWT234 und DWT992.

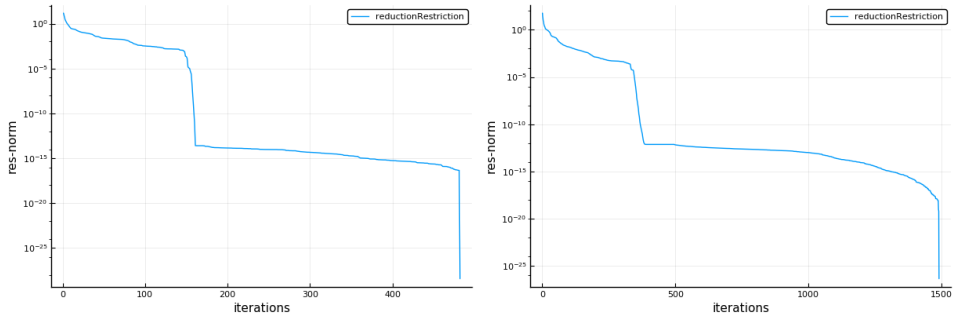

Abbildung 12: Konvergenz der Matrizen DWT234 und DWT992 (v.l.) mit der reduktionsbasierten Restriktion

Es kann wieder der typische Verlauf aufgezeigt werden, der eine Abwechslung zwischen starken und schwachen Fall des Residuums. Man kann an diesem Beispiel sehr gut erkennen, dass mit wachsender Dimension der Matrizen auch die Übergänge zwischen den verschieden starken Fällen des Residuums glatter werden. Außerdem steigt mit wachsender Dimension auch der Punkt, an dem der Verlauf konvergiert.
Die restlichen Verläufe ähneln stark denen, die schon in den vorherigen Kapiteln aufgezeigt wurden und werden deswegen hier nicht erneut aufgeführt.
Bei der Betrachtung der benötigten Zeit zur Berechnung wird unterschieden in Berechnungzeit der Restriktionsmatrix und die benötigte Zeit, sodass der Algorithmus zum Lösen der linearen Gleichung endet.
Zunächst wird die Erstellung der Restriktionsmatrix betrachtet. Man erkennt schnell,

dass sich hier die Anordnung stark an der Größe der vorgegebenen Matrizen orientiert. Zwar gibt einige Matrizen, die vor einer größeren kommen, jedoch handelt es sich hauptsächlich um nur kleine Unterschiede in der Dimension.Einzig die Matrix MBEAUSE und DWT234 zeigen ein anderes Verhalten. Es handelt sich bei beiden um, im Vergleich zu den restlichen Matrizen, recht kleine Matrizen.

Matrix	Zeit
BCSSTK25	167s
BCSSTK24	44.3s
GEMAT11	5.57s
BCSSTK23	5.19s
MBEAUSE	1.13s
BCSSTM26	949ms
WATT1	754ms
BCSPWR07	626ms
BCSPWR09	497ms
DWT992	400ms
DWT234	355ms
GR3030	215ms
SAYLR3	181ms
NOS7	112ms
NNC666	98.0ms
JGL009	81.3μs

Tabelle 10: Zeitliche Übersicht zur Ermittlungszeit für die reduktionsbasierte Restriktion

Die beiden deutlich kleineren Matrizen benötigen zum Teil deutlich länger als viele der größeren Matrizen. Vor allem die Matrix MBEAUSE ist überraschender Weise im oberen Drittel der Tabelle zu finden. Die wahrscheinlichste Erklärung hierfür ist, dass sie im Vergleich zu vielen der größeren Matrizen deutlich dichter besetzt ist.
Bei der Betrachtung der Übersicht zu Zeit und Speicher für das Lösen des Gleichungssystems erkennt man schnell einige Strukturen wieder. Die Reihenfolge ähnelt auch hier stark der Reihenfolge, welche durch die Anordnung nach Dimension entsteht. Die mit Abstand größte Matrix benötigt auch hier die meiste Zeit, nämlich gute sechs Stunden und den größten Speicher. Der große Abstand zu den folgenden Matrizen ist wohl mit dem großen Abstand der Dimensionen zu erklären, da die erste folgende Matrix eine Dimension von etwas weniger als einem Drittel von BCSSTK25 besitzt.

Matrix	Zeit	Speicher
BCSSTK25	22183s	45.3GB
GEMAT11	381s	5.21GB
BCSSTK24	141s	2.84GB
BCSSTK23	101s	2.41GB
NNC666	40.6s	3.07GB
BCSPWR09	33.0s	1.69GB
MBEAUSE	30.9s	3.04GB
BCSPWR07	26.7s	1.62GB
WATT1	17.9s	1.35GB
BCSSTM26	12.6s	1.59GB
SAYLR3	3.80s	0.97GB
GR3030	2.79s	0.93GB
DWT992	2.46s	0.94GB
NOS7	1.34s	892MB
JGL009	413ms	764MB
DWT234	365ms	791MB

Tabelle 11: Zeitliche Übersicht zur Konvergenz für die reduktionsbasierte Restriktion

Natürlich reihen sich die Matrizen nicht ein, für die der Algorithmus nicht konvergierte, da hier eine Verzerrung der Zeit und des Speichers stattfand durch die hohen Iterationszahlen, die diese Durchläufe erreichten. Aber auch einige der Matrizen für die eine Konvergenz erreicht wurde, zeigen ein interessantes Verhalten auf. So ist am Ende der Tabelle nicht die mit Abstand kleinste Matrix JGL009 zu finden, sondern die beachtlich größere Matrix DWT234. Die Matrix BCSSTM26 hingegen konnte eine für ihre Dimension vergleichsweise kurze Zeit erreichen und ist somit weiter unten in der Tabelle zu finden.

5.5 Vergleich simpler Restriktionen

In diesem Kapitel werden die bereits in Kapitel 5.1 und 5.2 vorgestellten Ergebnisse miteinander verglichen. Schaut man sich die Übersichten zu Konvergenz und nicht-Konvergenz an, so stellt man fest, dass für die gewichtete diagonale Restriktion eine Matrix mehr konvergierte. Außerdem konnten gute Ergebnisse für die Matrix GEMAT11 erzielt werden. Für diese und die Matrix BCSSTK25 wäre bei höheren maximalen Iterationszahlen wahrscheinlich ebenfalls eine Konvergenz erreicht worden. Für die Matrix BCSSTK25 scheint dies auch der Fall bei der diagonalen Restriktion zu sein. Bezieht man jedoch diese Matrizen ebenso als konvergent in den Vergleich mit ein, so existiert nur noch eine Matrix für die der Algorithmus aus Kapitel 3.2 nicht konvergierte und dies aus dem gleichen Grund, für den auch der andere Algorithmus mit drei Matrizen auf der nicht-konvergenten Seite verbleibt. Bei der Berechnung kam es zu Werten von positiv unendlich, welche somit kein Ergebnis produzierten.
Für die Matrizen, die konvergierten, ähneln sich die Graphen zumeist sehr. Und auch

bei kleinen Abweichungen konvergieren viele im nahezu identischen Iterationsschritt und bei einem sehr nahen Residuum. Gut zu erkennen ist ein solches Verhalten bei der Matrix SAYLR3.

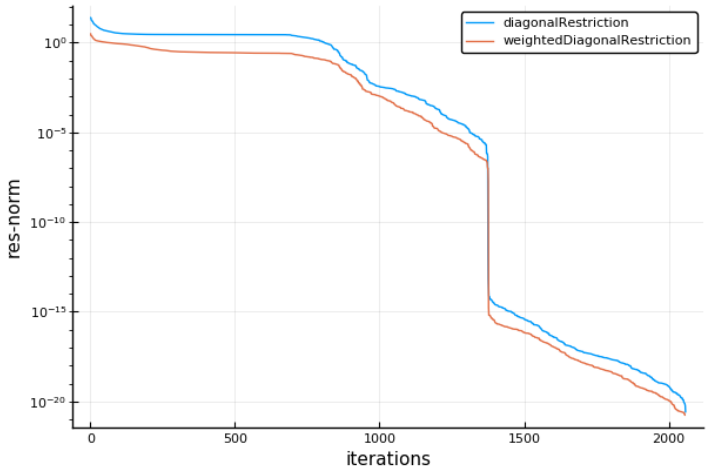

Abbildung 13: Vergleich der Matrix SAYLR3 mit den simplen Restriktionen

Man kann bei dieser Matrix die große Ähnlichkeit zwischen den Verläufen erkennen, wobei der Graph der gewichteten diagonalen Restriktion fast in jedem Iterationschritt ein geringeres Residuum annimmt. Die Eigenschaften der Matrix SAYLR3 findet man bei den Matrizen BCSSTK23, BCSSTK24, BCSSTK25, GR3030, WATT1 und JGL009 wieder.
Bei den restlichen Matrizen unterschieden sich entweder der Itertionsschritt in dem sie konvergierten stark oder das schlussendliche Residuum wich zu sehr voneinander ab. In jedem Fall ist es jedoch der Einfachste der vier Algorithmen welcher früher konvergierte und zumeist auch ein geringeres Residuum erzielte. Dies war zum Beispiel der Fall für die Matrix BCSPWR07.

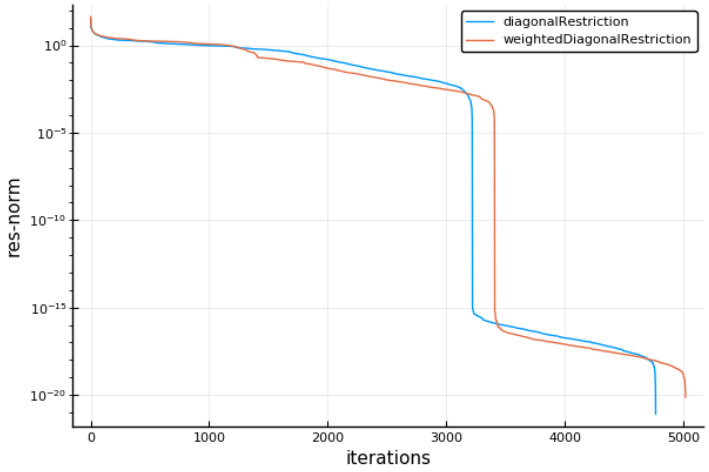

Abbildung 14: Vergleich der Matrix BCSPWR07 mit den simplen Restriktionen

Der Verlauf ähnelte sich sehr stark und auch im Abschluss sehen die Verläufe noch recht ähnlich aus. Jedoch konvergierte wie bereits erwähnt der Verlauf der diagonalen

Restriktion über 200 Iterationsschritte eher in ein geringeres Residuum. Ein ähnliches Verhalten ist bei den Matrizen BCSPWR09, DWT234 und DWT992 zu finden.
Die Matrix BCSSTM26 zeigt den wohl größten Unterschied der angenommenen Residuumswerte aller Matrizen, da hier ein deutlich steilerer Fall des Residuums für diagonale Matrizen zu erkennen ist.

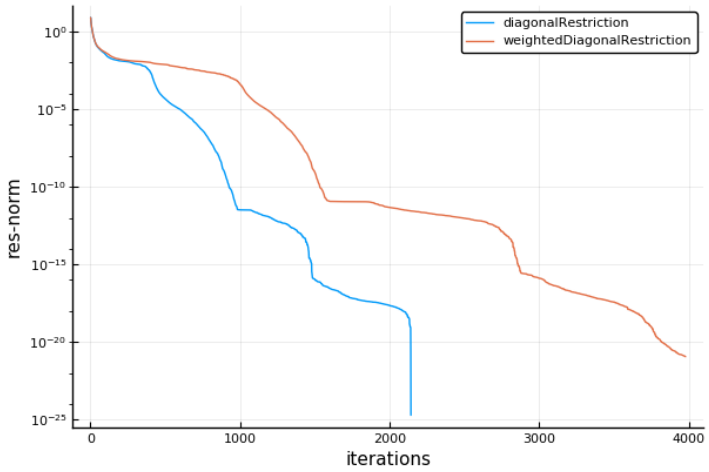

Abbildung 15: Vergleich der Matrix BCSSTM26 mit den simplen Restriktionen

Man kann hier trotzdem eine große Ähnlichkeit im Verlauf des Residuums erkennen. Der Durchlauf der diagonalen Restriktion konvergierte bei der Hälfte der Itertionsschritten des anderen Algorithmus und auch bei einem deutlich geringeren Residuum. Der Verlauf der Matrix NOS7 ist der einzige der einen Vorteil für die gewichtete diagonale Restriktion aufzeigt. Auch hier konvergierte der Algorithmus deutlich nach dem der diagonalen Restriktion, jedoch konnte ein bedeutend geringeres Residuum erzielt werden.

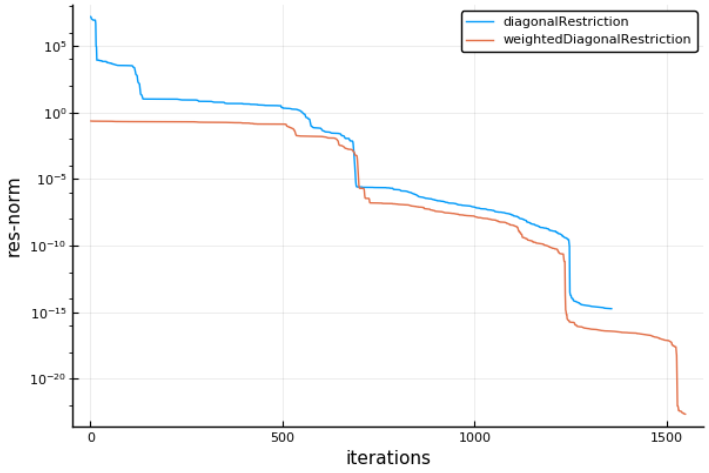

Abbildung 16: Vergleich der Matrix NOS7 mit den simplen Restriktionen

Abschließend kann man feststellen, dass die diagonale Restriktion in den meisten Fällen die besseren und schnelleren Ergebnisse liefert, jedoch ist die diagonale gewichtete Restriktion in diesen Fällen deutlich stabiler, da für mehr Matrizen Ergebnisse

die Konvergenz erreicht wurde beziehungsweise ein Vorkonditionierer erzeugt wurde, welcher nicht zu Fehlern führte.

5.6 Vergleich komplexer Restriktionen

Dieses Teilkapitel beschäftigt sich mit dem Vergleich der komplexen Restriktionen in dieser Arbeit und auch hier werden nur die Ergebnisse mit dem geglätteten Vorkonditionierer betrachtet.
Die Matrizen GEMAT11, NNC666 und MBEAUSE stehen für beide Algorithmen auf der nicht-konvergenten Seite, da es hier bei der Berechnung zu Fehlern kam. Für diese Durchläufe konnten keine auswertbaren Daten gewonnen werden, da hier Werte von positiv Unendlich während der Glättung des Vorkonditionierers entstanden. Diese Matrizen werden daher in diesem Kapitel nicht weiter betrachtet. Anders verhält es sich bei den Matrizen SAYLR3 und BCSSTK25. Für die Matrix SAYLR3 konnte eine Konvergenz für den häufungsbasierten Algorithmus verzeichnet werden, jedoch scheint der Reduktionsbasierte hier einen Fehler eingebaut zu haben, welcher dazu führt, dass auch hier Werte von positiv unendlich auftreten. Bei der Matrix BCSSTK25 gab es bei keinem der Algorithmen ein Problem, jedoch erreicht der häufungsbasierte durch die Beschränkung der maximalen Iteration hier nicht die Konvergenz.

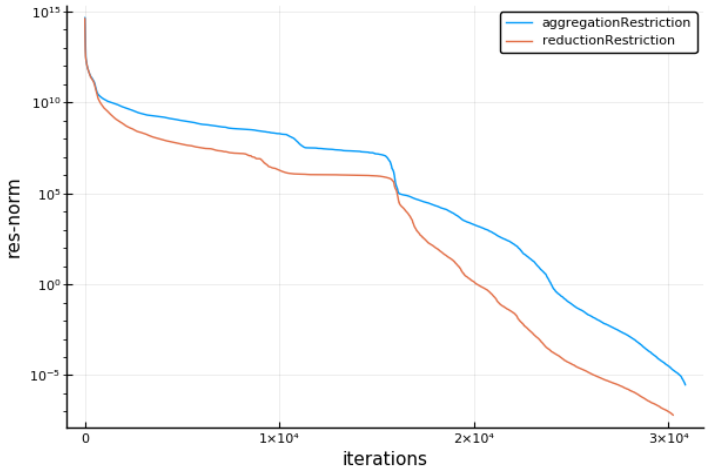

Abbildung 17: Vergleich der Matrix BCSSTK25 mit den komplexen Restriktionen

Hier kann man bereits erkennen, dass der reduktionsbasierte Algorithmus, wenn ein Unterschied besteht, meist früher konvergiert. Außerdem erkennt man, dass der Wert des Residuums fast den gesamten Durchlauf über einen bereits deutlich niedrigeren Wert hat. Diese Beispiel spiegelt sich vor allem in den Matrizen BCSSTK23 und BCSSTK24 wieder aber ist auch bei dem Vergleich für die Matrix DWT234 wiederzuerkennen.
Bei der Matrix DWT992 hingegen ist nahezu identischer Iterationsschritt in dem der Verlauf konvergierte zu erkennen, aber das Residuum ist deutlich geringer.

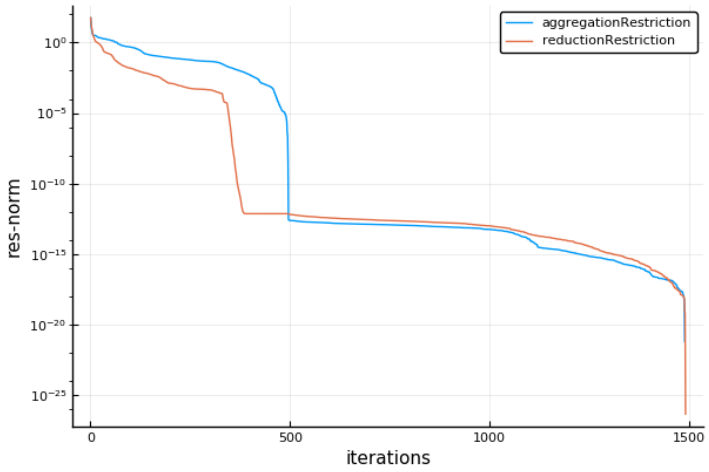

Abbildung 18: Vergleich der Matrix DWT992 mit den komplexen Restriktionen

Interessant ist an diesem Vergleich auch, dass der Graph für reduktionsbasierte Methode nur im ersten Drittel die niedrigeren Residuumswerte aufweist. Danach halten sich beide immer bei recht ähnlichen Werten auf und haben somit auch einen sehr ähnlichen Verlauf, wobei der reduktionsbasierte Verlauf in einem deutlich geringeren Residuum konvergiert. Dies ist aber die einzige Ausnahme dieser Art.
Dies ist für einige Matrizen sogar während des kompletten Verlaufs der Fall. Man könnte hier fast davon ausgehen, dass die Verläufe komplett identisch sind. Bei der Matrix GR3030 sieht man dies sehr stark. Es gibt nur einige wenige Stellen an denen sich die Graphen minimal unterscheiden.

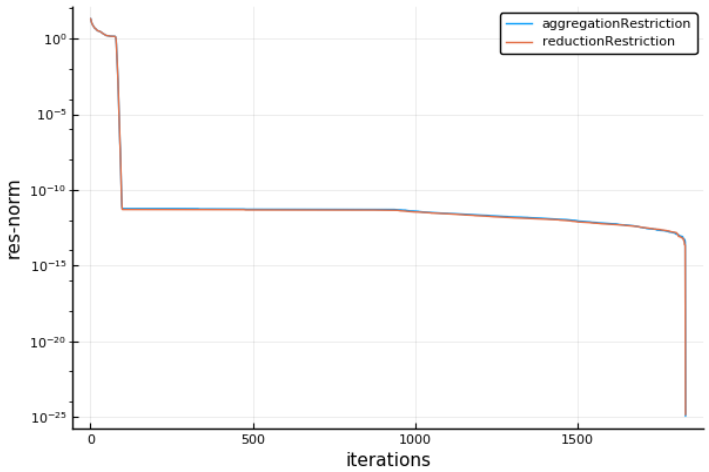

Abbildung 19: Vergleich der Matrix GR3030 mit den komplexen Restriktionen

Die feinen Unterschiede die hier entdeckt werden können spielen aber bei dem Verlauf, der Konvergenzzeit und dem endgültigen Residuum kein Rolle. Dies trifft mit in manchen Fällen etwas stärkeren Abweichungen auch auf die Matrizen BCSPWR07, BCSTM26, JGL009, WATT1.
Zwar gibt es keinen Fall, bei dem die häufungsbasierte Methode schneller beim Lösen

des Systems war, aber für einige Matrizen hat sie ein zum Teil in ein deutlich genaueres Resultat konvergiert. So zum Beispiel bei der Matrix NOS7.

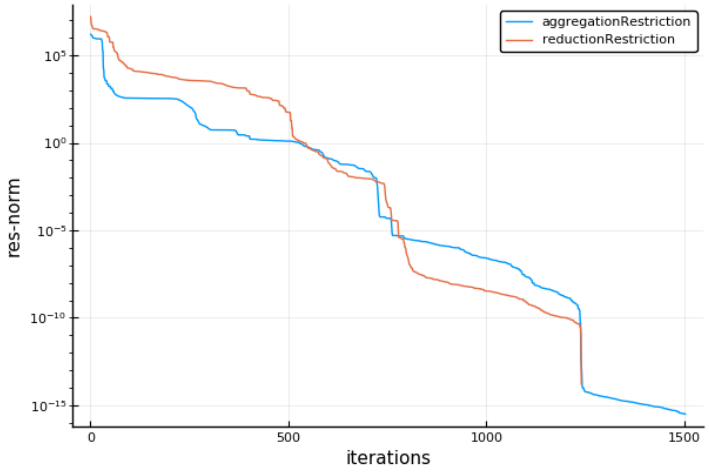

Abbildung 20: Vergleich der Matrix NOS7 mit den komplexen Restriktionen

Dieses Verhalten ist auch bei den restlichen noch nicht genannten Matrizen zu beobachten. Vor allem die Matrix BCSPWR09 sticht hier hervor, da das Residuum für die häufungsbasierte Methode bei 10^{-30} lag und für die Reduktionsbasierte nur bei 10^{-20}. Zusammenfassend kann man sagen, dass die reduktionsbasierte Methode in jedem dieser Fälle schneller konvergierte und dies zur Hälfte mit geringerem oder gleichen Residuum. Da es für die Matrix SALR3 vor dem Glätten des Vorkonditionierers bereits zu Einträgen führte, welche in einem konstanten Residuum von positiv unendlich resultierten, kann die häufungsbasierte Methode als sicherere Variante zum Lösen des Systems mit den vorgegebenen Matrizen bezeichnet werden. Außerdem ist sie für alle bis auf eine Matrix entweder genauso genau oder eben genauer als die andere Methode.

5.7 Vergleich aller Restriktionen

In dem folgenden Kapitel werden alle vier Algorithmen verglichen, da der Übersicht halber in den vorherigen Kapiteln maximal zwei Algorithmen verglichen wurden. Es werden sowohl die Verläufe als auch die Konvergenzeigenschaften der jeweiligen Methoden miteinander verglichen.
Betrachtet man die Übersichten zu Konvergenz und nicht-Konvergenz, so erkennt man, dass für die gewichtete Diagonale Restriktion am meisten Matrizen konvergierten. Bezieht man zusätzlich die Tatsache mit ein, dass einige Matrizen, wie zum Beispiel die Matrix BCSSTK25, bei höheren maximalen Iterationszahlen ebenfalls konvergiert wären, so verbessert sich die Position des Algorithmus nur weiterhin. Im Gegensatz zu den anderen Algorithmen kann dann der Durchlauf für nur noch eine Matrix nicht konvergieren. Bei der diagonalen und häufungsbasierten sind es drei Matrizen und bei der reduktionsbasierten sogar vier, die nicht zu einer Konvergenz führen können. Unter

der Berücksichtigung der Vorglättung ist die simple Restriktion, welche eine Gewichtungsmatrix anhand der Dimension aufbaut die Stabilste. Auffällig ist, dass für keine der Methoden die Matrix MBEAUSE eine Möglichkeit zur Konvergenz besitzt. Dies ist für die Matrizen GEMAT11 und NNC666 für drei der vier Algorithmen ebenfalls der Fall. Nur die Matrix SAYLR3 ist das Alleinstellungsmerkmal der reduktionsbasierten Restriktion, da nur bei dieser Methode keine Konvergenz erreicht werden kann. Jedoch ist diese Restriktion auch die einzige, für die die Matrix BCSSTK25 innerhalb der angegebenen Iterationsschritten konvergierte.
Für die Matrizen, bei denen alle Verfahren konvergierten, gibt es sehr unterschiedliche Ergebnisse, was Genauigkeit und Geschwindigkeit angeht. In wenigen Fällen ähneln sich die Ergebnisse ist den punkten letzter Iterationsschritt und Residuum. Dies ist zum Beispiel der Fall für die Matrix WATT1, denn hier kann man sehen, dass die Graphen in den späteren Iterationsschritte alle einen nahezu identischen Verlauf aufweisen.

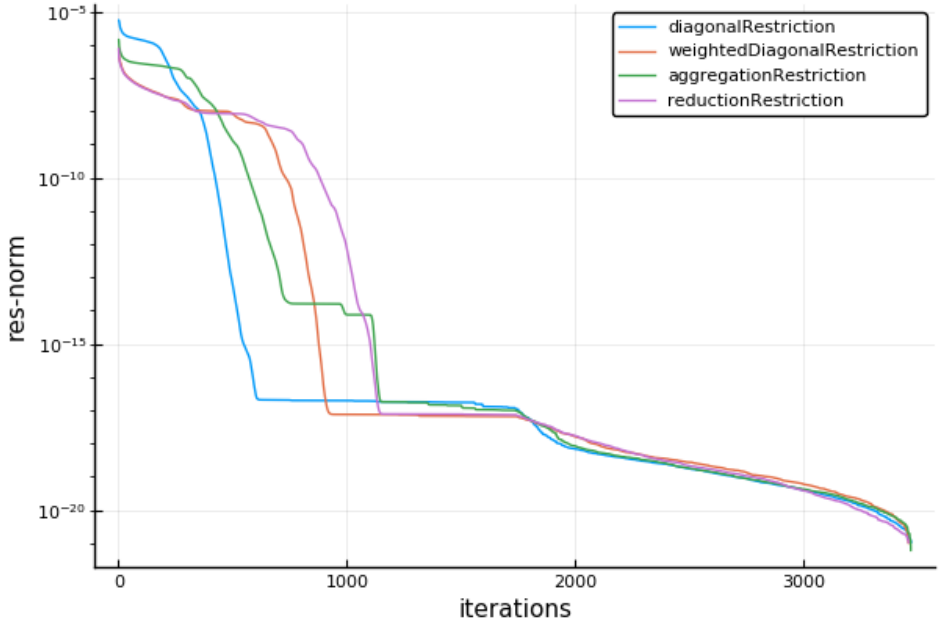

Abbildung 21: Vergleich der Matrix WATT1 mit allen Restriktionen

Hätte man jedoch eine niedrigere Toleranz angesetzt, so wäre in diesem Fall der Algorithmus mit der diagonalen Restriktion der schnellste gewesen, da die Verläufe sich sehr unterscheiden in der ersten Hälfte der Iterationen. Ein ähnliches Verhalten lässt sich auch bei GR3030 und JGL009 feststellen, wobei hier nicht die diagonale Restriktion die schnellste gewesen wäre. Für die JGL009 lassen durch die geringe Dimension keine bemerkenswerte Unterschiede feststellen, aber für GR3030 wäre bei einer Toleranz von 10^{-10} die reduktionsbasierte und die häufungsbasierte Methoden die ersten gewesen, welche ein solches Residuum erreichen.
Für mehrere der Matrizen konnte der reduktionsbasierte Algorithmus die schnellste Konvergenz, gemessen an der geringsten Iterationszahl in der Konvergenz, erzielen.

Trotzdem konnte ein sehr ähnliches Residuum erreicht werden. Die Durchläufe für die Matrix BCSSTK24 zeigen dieses Verhalten.

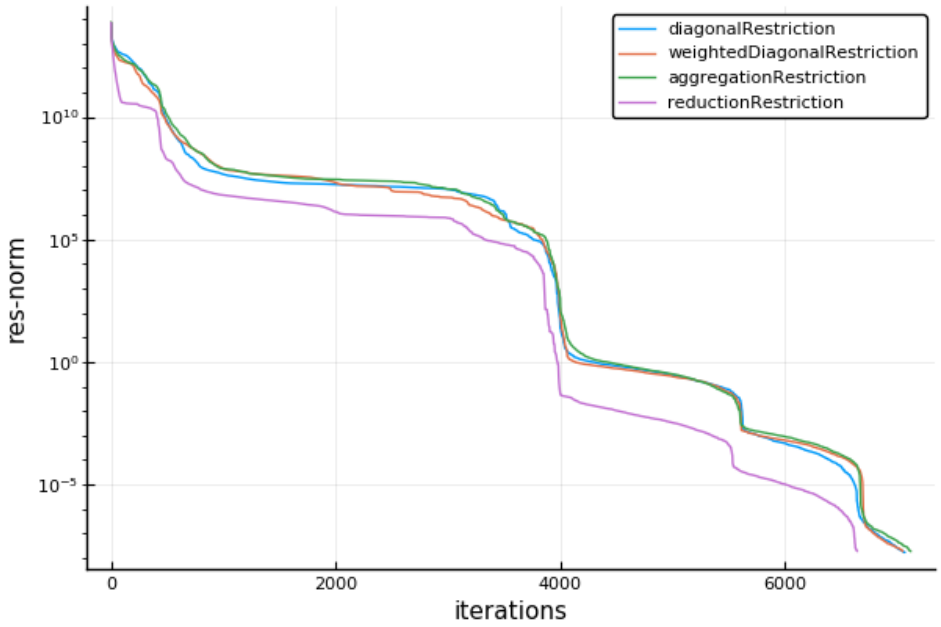

Abbildung 22: Vergleich der Matrix BCSSTK24 mit allen Restriktionen

Es kann ein sehr ähnlicher Verlauf aller Durchläufe erkannt werden, obwohl sich die reduktionsbasierte Methode durch ein durchgängig geringeres Residuum absetzt. Nahezu identisch sehen die Verläufe der Matrizen BCSSTK23 und BCSSTK25 aus. Auch bei den Verläufen für DWT234 kann man ein solches Verhalten erkennen. Aber bei dieser und auch bei allen anderen noch nicht benannten Matrizen weisen sich zum Teil größere Unterschiede in dem abschließenden Residuumswert auf. So kann man zum Beispiel bei der Matrix BCSPWR07 erkennen, dass die diagonale Restriktion die schnellste Konvergenz erzielt, aber ebenso bereits bei $10-22$ konvergiert, wobei die restlichen drei Verläufe bis auf 10^{-30} absinken. Dieses Bild lässt sich ähnlich auch für die Matrix BCSPWR09 wiedererkennen, wobei hier einzig der häufungsbasierte Algorithmus einen großen Abstand im Residuum erreichen kann. Die Verläufe der vier Algorithmen ähneln sich während des gesamten Verlaufs für beide Matrizen sehr stark.

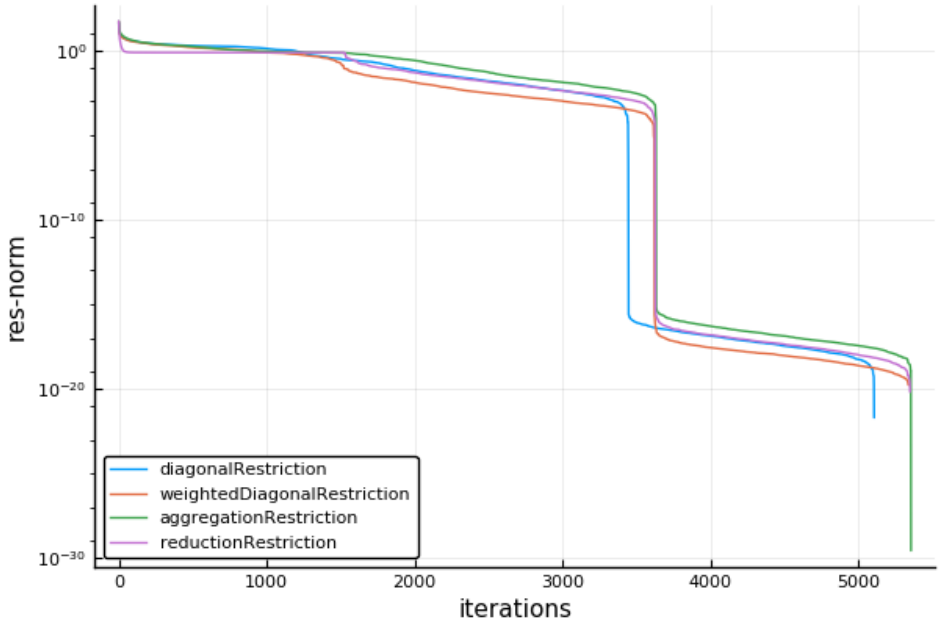

Abbildung 23: Vergleich der Matrix BCSPWR09 mit allen Restriktionen

Wie in dem Kapitel 5.5 aufgezeigt gibt es bei dem Vergleich aller Restriktionen ein Beispiel dafür, dass die gewichtete diagonale Restriktion ein deutlich kleineres Residuum erreicht als der Rest und ein Beispiel dafür, dass die Konvergenz viel später erreicht wurde und das Residuum höher war als für die anderen Restriktionen. Die Matrix BCSSTM26 zeigt auch hier zweiteres auf.

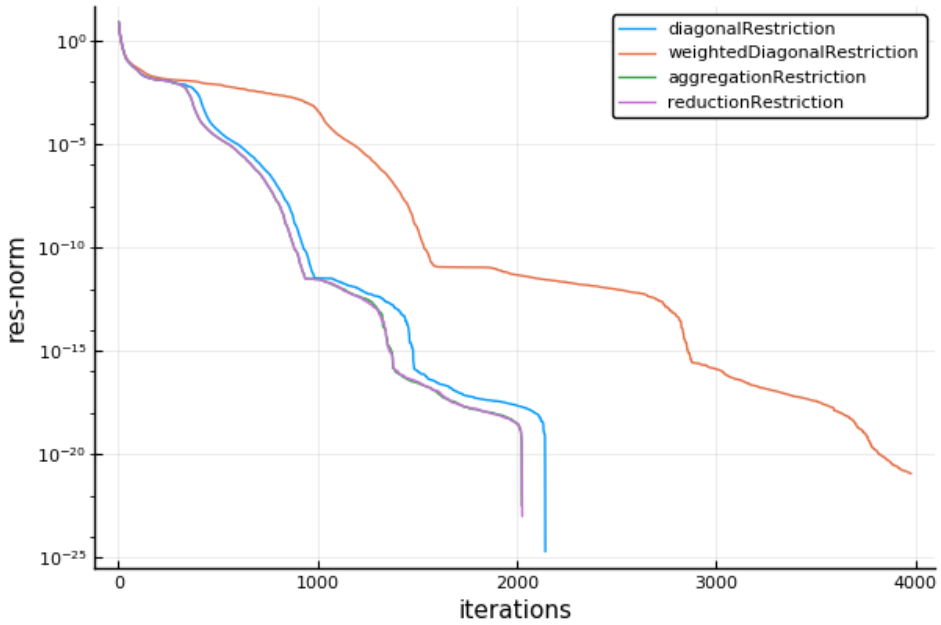

Abbildung 24: Vergleich der Matrix BCSSTM26 mit allen Restriktionen

Die restlichen Restriktionen zeigen hier einen sehr ähnlichen Verlauf auf und trotzdem die simpelste Methode etwas später als die komplexen Methoden konvergierte erreicht sie hier den geringsten Residuumswert. Im Vergleich konnten die komplexen

Algorithmen im Gegensatz zu der gewichteten diagonalen Restriktion sogar in der Hälfte der Iterationsschritten mit einem niedrigeren Residuum konvergieren.
In den Verläufen für NOS7 ist davon nichts wiederzuerkennen. Hier verlaufen die Graphen annähernd ähnlich bis hin zur Konvergenz.

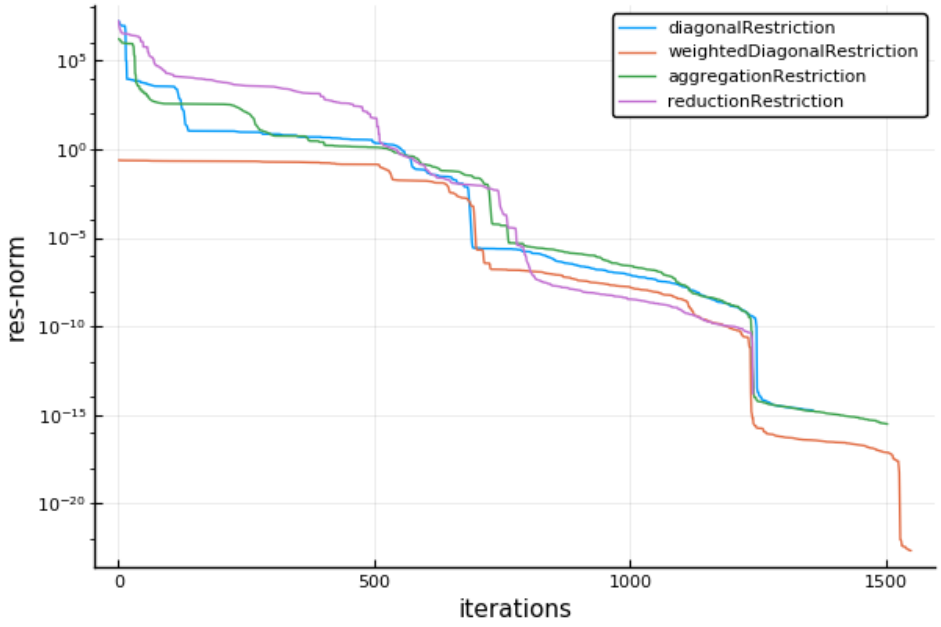

Abbildung 25: Vergleich der Matrix NOS7 mit allen Restriktionen

Die Verlauf der gewichteten diagonalen Restriktion konvergiert jedoch nicht bei einem Residuum zwischen 10^{-14} und 10^{-16} wie die restlichen Restriktionen. Es gibt einen weiteren abrupten Wechsel zu einem starken Fall und konvergiert somit bei einem Wert von ungefähr 10^{-25} und ist auch schon während der Iterationsschritte in denen die restlichen Graphen konvergieren bei einem niedrigeren Wert.
Das letzte Beispiel in diesem Kapitel ist die Matrix DWT992. Auch hier sieht man einen ähnlichen Verlauf über die gesamten Iterationen, jedoch enden die Graphen bei sehr unterschiedlichen Residuumswerten. Auf den ersten Blick kann man bereits erkennen, dass der simpelste Algorithmus den genausten und schnellsten Wert geliefert hat. Für die Betrachtung der restlichen Verläufe muss man etwas genauer hinschauen, da sich die Graphen hier stark überlappen. Tut man dies, so erkennt man, dass der häufungsbasierte Durchlauf bei dem höchsten Residuum von 10^{-22} endete und der Graph von der gewichteten diagonalen Restriktion mit einem minimal schlechteren Ergebnis als das des reduktionsbasierten Verlaufs bei 10^{-25} konvergierte.

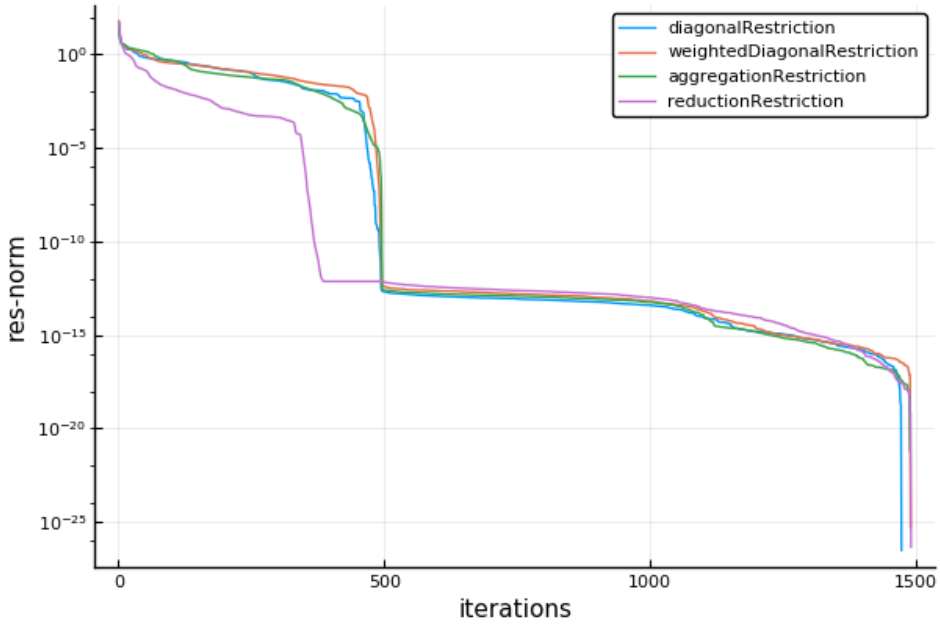

Abbildung 26: Vergleich der Matrix DWT992 mit allen Restriktionen

Im nachfolgenden werden die zeitlichen Unterschiede betrachtet. Diese Betrachtung wird mittels der Matrix BCSSTK24 durchgeführt, da hier jeder Algorithmus konvergierte und die Matrix eine der größeren ist und somit die zeitlichen Unterschiede besser wahrnehmbar sein sollten. Schaut man sich zunächst die benötigte Zeit für den Bau der Restriktionsmatrx an, so erkennt man, dass die Abstände zwischen den Verfahren sehr groß sind. Die abgebildete Tabelle zeigt die Werte der reinen Berechnungszeit für Restriktion. Es sind alle Schritte zu Erstellen des Vorkonditionierers und dessen Glättung ausgeschlossen.

Algorithmus	Zeit
diagonale Restriktion	18.4ms
gewichtete diagonale Restriktion	37.4ms
häufungsbasierte Restriktion	4.32s
reduktionsbasierte Restriktion	43.8s

Tabelle 12: Zeitliche Übersicht zur Erstellung der Restriktionsmatrix für BCSSTK24

Zu erklären sind diese Zahlen durch die Komplexität der eingesetzten Verfahren. Die diagonale Restriktion ist zeitlich logischerweise am schnellsten, da hier keine Rücksicht auf die Beschaffenheit einer Matrix genommen wird und nur jeder zweite Eintrag ausgewählt werden soll, weswegen die resultierende Matrix auch nur sehr schwach besetzt ist. Bei der gewichteten diagonalen Restriktion ist es ähnlich. Da hier aber nicht nur jeder zweite Eintrag gewählt, sondern einer Gewichtung der Einträge mit ihren Nachbarn stattfindet ist die entstehende Matrix minimal stärker besetzt. Dies wird etwas mehr als doppelten zeitlichen Aufwand erklären, welcher sich jedoch bei sehr viel größeren Matrizen wie der BCSSTK25 auswirkt. Hier ist der Unterschied von 294ms zu

819ms menschlich zwar immer noch nicht wirklich wahrnehmbar, aber deutet an, dass es bei Matrizen mit noch größeren Dimensionen ins Gewicht fällt. Bei den Vergleichen zu den komplexeren Methoden fällt der zeitliche Unterschied noch viel deutlicher ins Gewicht. Man hat hier mehr als ein zweihundertfaches an zeitlichem Aufwand zwischen der simpelsten Methode und der häufungsbasierten Restriktion. Dies ist durch ihren komplexeren Aufbau zu erklären, welcher die Matrix auf nahen und fernen Nachbarn untersucht um Häufungen zu identifizieren und zu nutzen. Dieser Abstand vervielfacht sich aber nochmal deutlich, wenn man die Berechnungszeit der reduktionsbasierten Restriktionsmatrix mit einbezieht. Der Abstand zwischen dem schnellsten und langsamsten Aufbau der Restriktionsmatrix beträgt mehr als 40 Sekunden und damit ist der Wert des Langsamsten bei mehr als dem Zweitausendfachen des Schnellsten. Zieht man hier wieder das Beispiel BCSSTK25 mit heran, so ist das ein Unterschied von 294ms zu 165s. Hier relativiert sich der Abstand langsam. Dies ist auch bei anderen Matrizen zu erkennen. So hat man zum Beispiel bei der kleinsten Matrix ein Verhältnis von 585ns zu 2.86ms. Mit steigender Größe der Matrix relativiert sich also der Abstand zwischen den Berechnungen der Restriktionsmatrizen. Der Grund für den doch immer deutlich höheren zeitlichen Aufwand wird das Vertauschen der Spalten und Zeilen in der Matrix sein. Dies ist immer ein hoher rechentechnischer Aufwand des Algorithmus, wenn wirklich die entsprechenden Zeilen und Spalten umkopiert werden.
Die anschließende Zeit um das System zu lösen, sieht bei allen Restriktionsmethoden sehr ähnlich aus.

Algorithmus	Zeit
diagonale Restriktion	141s
gewichtete diagonale Restriktion	138s
häufungsbasierte Restriktion	140s
reduktionsbasierte Restriktion	127s

Tabelle 13: Zeitliche Übersicht zur Lösung des Systems für BCSSTK24

Für die Matrix BCSSTK24 lohnt sich jedenfalls der zusätzliche Aufwand nicht, welcher in die Berechnung einer komplexeren Restriktionsmatrix gesteckt wurde. Dies ist schon anders für deutlich größere Matrizen. Bei der Matrix BCSSTK25 liegt der Abstand von der Berechnung für die reduktionsbasierte Restriktion und der zweitschnellsten Methode bei über 600 Sekunden und macht damit die längere Rechenzeit von 165 Sekunden mehr als wett.
Abschließend kann man sagen, dass der sicherste Algorithmus für den vorgeglätteten Vorkonditionierer die diagonale gewichtete Restriktion ist, welche aber in den Punkte Geschwindigkeit und Genauigkeit nicht die besten Ergebnisse erzielen konnte. Für die Geschwindigkeit lohnt es sich, den reduktionsbasierten Algorithmus zu benutzen, wenn man ein sehr großes System lösen muss.

6 Abweichungen zu dem nicht geglätteten Vorkonditionierer

In diesem Kapitel werden die erreichten Ergebnisse und Erkenntnisse aus Kapitel 5 (blaue Graphen) mit den Testdurchläufen ohne die Glättung des Vorkonditionierers durch diag(A) beziehungsweise diag(A)$^{-1}$ (orange Graphen) .
Bei dem Blick auf die grobe Einteilung nach Konvergenz und nicht-Konvergenz sieht man bereits große Unterschiede. Denn es konnte bei jeder Restriktionsart für alle Matrizen eine Konvergenz erzielt werden. Die einzige Ausnahme bildet auch hier die reduktionsbasierte Restriktion mit der Matrix SAYLR3. Allein diese Aussage deutet schon den großen Unterschied an, den die Glättung verursacht. Dies lässt auch darauf schließen, dass die häufige nicht-Konvergenz der Matrizen GEMAT11, MBEAUSE und NNC666 durch die Glättung entstand. Wie bereits angedeutet musste hier auch in allen drei Fällen zur Berechnung auch immer die Pseudoinverse benutzt werden, was wahrscheinlich die Ursache des Problems ist. Daraus lässt sich nicht schließen, dass bei der Verwendung der Pseudoinverse das Verfahren nicht mehr konvergiert, jedoch ist Wahrscheinlichkeit deutlich höher. Ebenso bemerkenswert ist, dass auch für die Matrix BCSSTK25 immer eine Konvergenz erreicht wurde. Bei den geglätteten Varianten konnte dies nur der reduktionsbasierte Algorithmus erreichen und dies auch nur kurz vor dem maximalen Iterationsschritt.
Damit wurde bereits eine schnellere Konvergenz der Graphen angedeutet, welche sich auch bei dem Blick auf die restlichen Matrizen bestätigt. Die Anzahl der Iterationen war ohne Glättung entweder exakt identisch oder kleiner.
Bei dem Vergleich der wirklichen Graphen kommt es zu zum Teil sehr verschiedenen Ergebnissen. Aus dem Einführungstext dieses Kapitels kann man bereits erkennen, dass im Gegensatz zu dem Durchlauf mit Glättung fast alle Matrizen konvergierten. Bei den zusätzlich konvergierten Matrizen ist jedoch keine Besonderheit zu erkennen. Sie ähneln sehr stark den bereits aufgezeigten Verläufen.
Bei vielen der Verläufen konnte kein Unterschied zwischen den beiden betrachteten Verfahren erkannt werden. Die Matrix DWT992 ist ein sehr gutes Beispiel hierfür, da für keine Restriktion hier ein Unterschied ersichtlich ist.

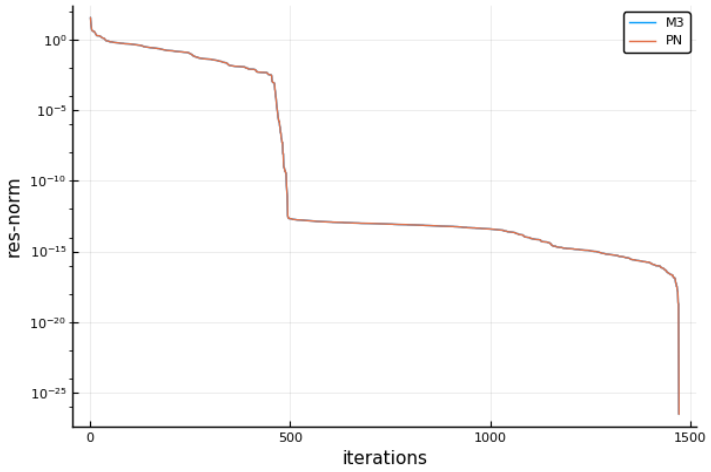

Abbildung 27: Konvergenz der Matrix DWT992 mit den diagonalen Restriktionen

Dieses Verhalten findet man außerdem bei den Matrizen DWT234, BCSPWR07 und BCSPWR09 wieder. Bei der reduktionsbasierten Restriktion gilt dies sogar noch für die beiden Matrizen WATT1 und BCSSTM26. Bei den Matrizen GR3030 und JGL09 kam es bei allen Restriktionen ebenfalls zu starken Ähnlichkeiten, man kann aber nicht von identischen Verläufen sprechen, da an einigen Stellen die Graphen etwas von einander abweichen. Sie konvergieren jedoch im gleichen Iterationsschritt und bei einem annähernd identischen Residuumswert.
Für die diagonale, diagonale gewichtete und die häufungsbasierte Restriktionen sind die Unterschiede zwischen geglättetem und nicht-geglättetem Vorkonditionierer für die Matrizen der BCSSTK-Reihe ähnlich. Bei allen Vergleichen konvergierte der nicht-geglättete Verlauf deutlich früher und bei einem nur leicht abweichendem Residuumswert.

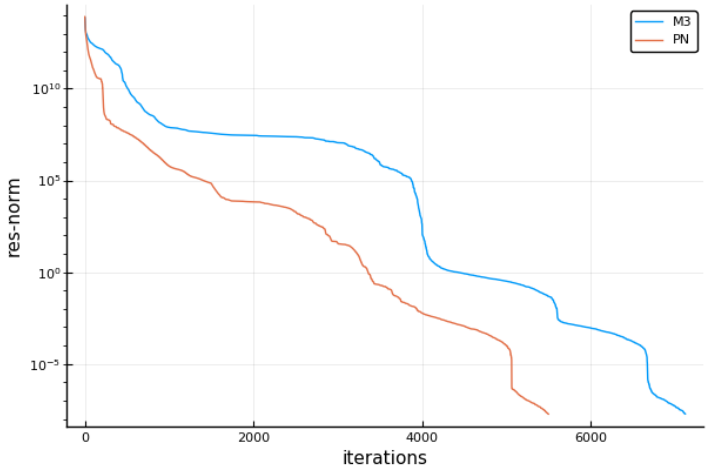

Abbildung 28: Konvergenz der Matrix BCSSTK24 mit den häufungsbasierten Restriktionen

Bei diesem Verlauf unterscheidet sich der Residuumswert nur minimalst und ist mit dem Auge nicht erkennbar. Bei anderen Verläufen, hauptsächlich für BCSSTK23 und

BCSSTK25, kann man deutlichere Unterschiede in beide Richtungen feststellen.
Für die reduktionsbasierte Restriktion ergab sich hier ein anderes Bild. In jedem der Verläufe konvergierte der geglättete Verlauf deutlich später aber mit einem niedrigeren Residuum. Bei der Matrix BCSSTK23 konvergierte der nicht-geglättete Verlauf schon bei einem Residuum von 10^3 und liefert somit kein zufriedenstellendes Ergebnis.

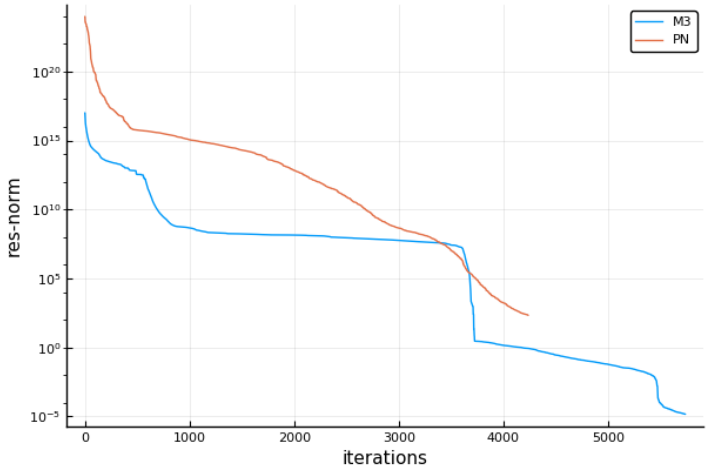

Abbildung 29: Konvergenz der Matrix BCSSTK23 mit den reduktionsbasierten Restriktionen

Auch für die beiden anderen Matrizen ist das Residuum bei den nicht-geglätteten Verläufen vergleichsweise hoch nach der Konvergenz.
Für die Matrix BCSSTM26 sehen die Vergleiche bei allen Restriktionen verschieden aus. Für die reduktionsbasierte Methode sind die Verläufe wie bereits erwähnt nahezu identisch. Auch bei der häufungsbasierten Restriktion kann man einer hohe Ähnlichkeit entdecken. Der einzige Unterschied ist hier, dass der nicht-geglättete Verlauf in der Konvergenz auf ein niedrigeren Wert des Residuums fällt. Für die beiden simplen Restriktionen kann ebenfalls eine Ähnlichkeit festgestellt werden, jedoch überlagern sich die Verläufe keinesfalls.

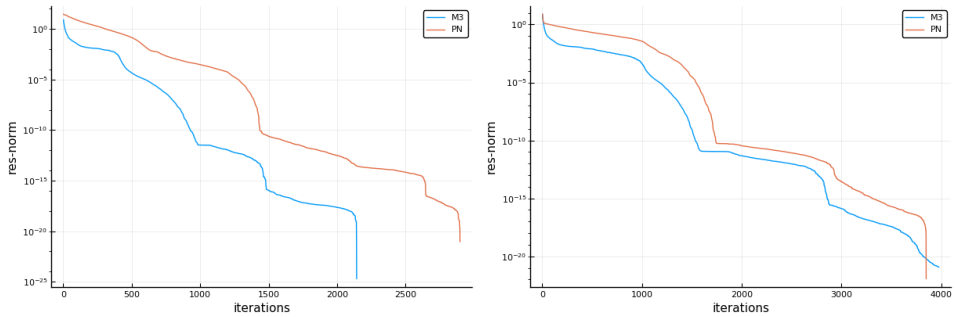

Abbildung 30: Konvergenz der Matrix BCSSTM26 mit der diagonalen und diagonalen gewichteten (v.l.) geglätteten und ungeglätteten Vorkonditionierern

Bei der diagonalen Restriktion ist der geglättete Vorkonditionierer schneller und präziser. Den ganzen Verlauf über ist er bei niedrigeren Residuumswerten als der nicht-

geglättete Verlauf während des selben Iterationsschrittes. Für die diagonale gewichtete Restriktion scheint sich dieses Verhalten mit etwas kleineren Abstand zu wiederholen, jedoch sorgt ein sehr starker Fall des nicht geglätteten Verlaufs kurz vor dem Ende für eine Drehung der Ergebnisse, sodass dieser im Endeffekt schneller und präziser ist.
Für die Matrix NOS7 verhält es sich ähnlich den Matrizen GR3030 und JGl09. Die Verläufen sind für jede Restriktion immer sehr nah beieinander, konvergieren in einem fast identischen Residuum und bei sehr nah aneinander liegenden Iterationsschritten. Nur bei der diagonalen Restriktion ist hier ein kleiner Versatz in dem konvergierenden Iterationsschritt zu erkennen.

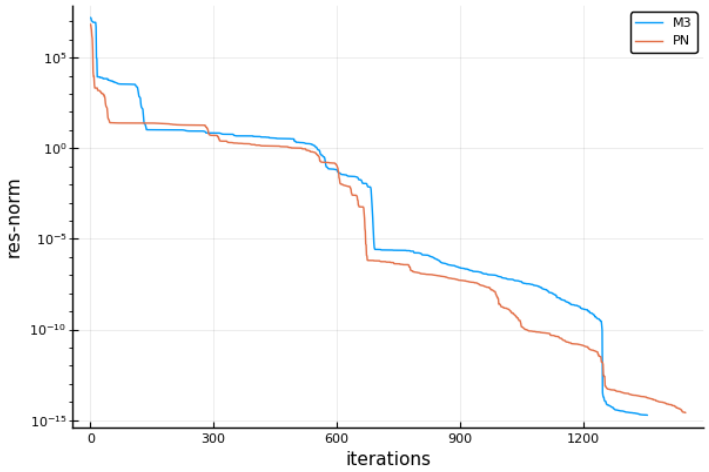

Abbildung 31: Konvergenz der Matrix NOS7 mit den diagonalen Restriktionen

Aber der restliche Verlauf lässt sich sehr gut auch als Beispiel für die anderen Restriktionsmethoden verwenden.
Auch für den Verlauf der restlichen Restriktionen bei der Matrix WATT1 sollte ein Beispiel reichen, da sich hier alle Verläufe sehr ähneln.

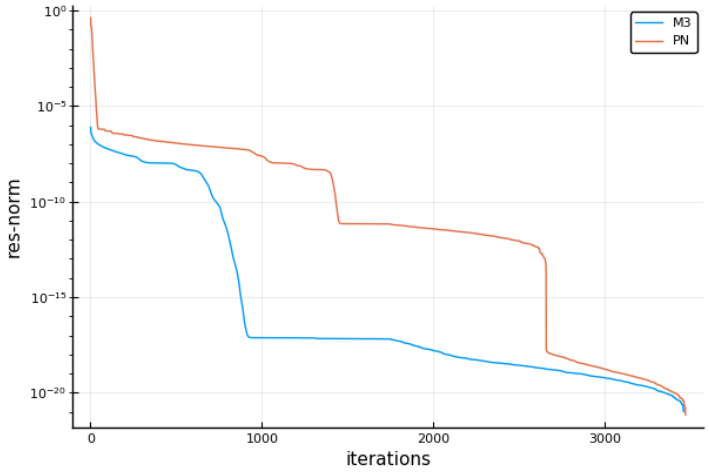

Abbildung 32: Konvergenz der Matrix WATT1 mit den diagonalen gewichteten Restriktionen

Die beiden Graphen beginnen mit sehr Unterschiedlichen Verläufen und es entsteht schnell eine große Lücke zwischen den Residuumswerten, jedoch wird diese zum Ende hin verschwindend gering bis nicht mehr existent.
Der Vergleich für die Matrix SAYLR3 kann hier nur für die simplen Restriktionen und die häufungsbasierte Restriktion vollzogen werden, da die reduktionsbasierte Methode in beiden Fällen keine Ergebnisse lieferte. Im Allgemeinen kann man auch hier erkennen, dass der geglättete Vorkonditionierer die besseren Ergebnisse erzielen konnte. Für die diagonale gewichtete und häufungsbasierte Restriktion konnte der Verlauf in der Hälfte der Iterationsschritte des ungeglätteten Durchlaufs konvergieren, aber dies mit einem leicht größeren Residuum.

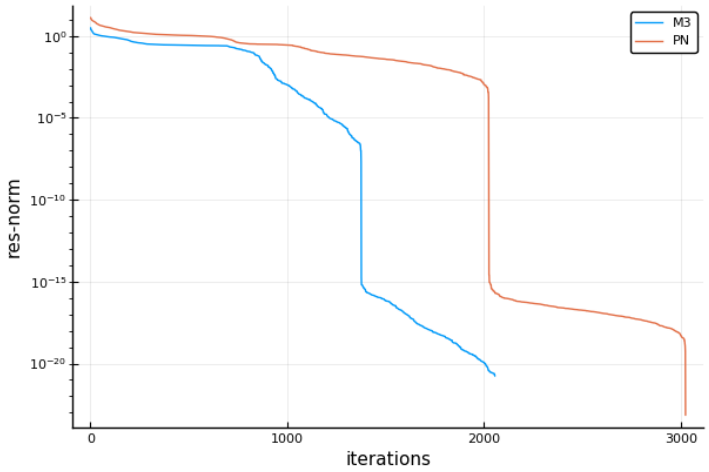

Abbildung 33: Konvergenz der Matrix SAYLR3 mit den diagonalen gewichteten Restriktionen

Für die diagonale Restriktion ähneln beide Verläufe dem Verlauf des Ungeglätteten aus dem obigen Beispiel für die diagonale gewichtete Restriktion. Im Abschluss erreicht der ungegelätte Durchlauf ein bedeutend geringeres Residuum.
Einzig für die diagonale gewichtete Restriktion lassen sich Aussagen für die Matrizen NNC666 und GEMAT11 treffen. Beide Vergleiche sind recht eindeutig ausgefallen. Bei der Matrix GEMAT11 sieht man, dass der ungeglättete Graph in einer nahezu identischen Anzahl an Iterationsschritten mit einem deutlich geringeren Wert konvergierte.

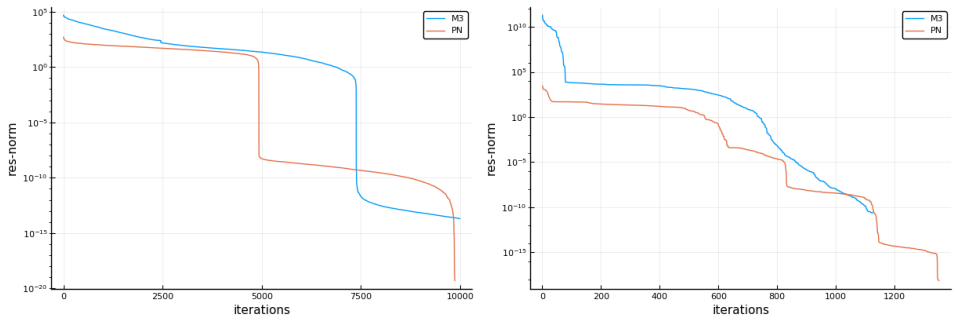

Abbildung 34: Konvergenz der Matrizen GEMAT11 und NNC666 (v.l.) mit den diagonalen gewichteten Restriktionen

Auch bei dem Vergleich für die Matrix NNC66 schneidet der ungegelättete Vorkonditionierer besser ab, da dieser trotz höheren Iterationszahlen auch hier ein deutlich besseres Residuum erreicht.
Im abschließenden Vergleich gibt es also viele Beispiele für die der ungeglättete Vorkonditionierer genauere Ergebnisse liefert, jedoch auch Fälle in denen ein zu ungenaues Resdiuum erreicht wurde. Im allgemeinen sind alle ungeglätteten Vorkonditionierer deutlich sicherer, da hier fast immer eine Konvergenz erreicht werden konnte und durch Anpassung der Toleranz können auch die gewünschten Ergebnisse erzielt werden.

7 Schlussbemerkung

Literaturverzeichnis

[EN09] Erlangga, Yogi A.; Nabben, Reinhard: *SIAM Journal on Scientific Computing*Algebraic Multilevel Krylov Methods 31 (2009), Nr. 5, S. 3417–3437

[GN12] Gosller, Florian; Nabben, Reinhard: *Electronic Transactions on Numerical Analysis*On AMG methods with F-smoothing based on Chebyshev polynomials and their Relation to AMGr 45 (2012), S. 146–159

[mm17] *Matrix Market*. `https://math.nist.gov/MatrixMarket/`. Version: 2017, Abruf: 04.05.2018

[Nab17] Nabben, Reinhard: *Multilevel methods for solving linear systems of equations*. 7 2017

[NKR17] Nabben, Reinhard; Kehl, René ; Ramos, Luis G.: Projections, deflation and multigrid for non-symmetric matrices. (2017)

Anhang

Listing 1: simple Restriktionen

```
module SimpleProlongations
export prolongation1
export prolongation2

    function  prolongation1(n)
        r = floor(Int, n/2)
        R = zeros(r,n)
        i = 1
        j = 1
        while i ≤ r
            R[i,j] = 1
            i = i + 1
            j = j + 2
        end
        return R
    end

    function  prolongation2(n)
        r = floor(Int, n/2)
        R = zeros(r,n)
        i = 1
        j = 1
        while i ≤ r
            R[i,j] = 1
            if(j+1 ≤ n)
                R[i,j+1] = 2
            end
            if(j+2 ≤ n)
                R[i,j+2] = 1
            end
            i = i + 1
            j = j + 2
        end
        return 1/4*R
    end

end
```

Listing 2: häufungsbasierte Restriktion

```
module AggregationbasedProlongation
export prolongationAggregation

    function prolongationAggregation(A)
        # Init
        n = size(A,1)
        U = Set(collect(1:n))
        W = Set()
        V = Set()
        tau = 0.25
        q = 0

        # Start Computing
        while true
            q = q + 1

            # find u
            u = -1
            for localU ∈ U
                if(!in(localU,W))
                    u = localU
                    #println("Knoten $u in Iteration $q")
                    break
                end
            end
            # Break if no u can be found
            if(u == -1)
                break
            end


            # Compute NiCTau
            NiCTau = Set()
            for j = 1:n
                if( (A[u,j] != 0) & ( sqrt(A[u,j]^2/abs(A[u,u]*A[j,j])) ≥ tau ))
                    push!(NiCTau, j)
                end
            end
            NiCTau = setdiff(NiCTau, W)
            #println("Nahe Nachbarn $NiCTau in Iteration $q")

            # Compute Vq
            Vq = Set()
            if(length(NiCTau) == 2)
                #println("Nur nahe Nachbarn in Iteration $q")
                push!(NiCTau, u)
                Vq = NiCTau
                push!(V, Vq)
            else
                # Compute NiDTau
                NiDTau = Set()
                for i = 1:n

                    isDistantNeighbor = 1
                    for j in NiCTau
                        if( (A[i,j] == 0) |
                            ( sqrt(A[i,j]^2/abs(A[i,i]*A[j,j])) < tau ))
                            isDistantNeighbor = 0
                        end
```

```
                    end
                    if(isDistantNeighbor == 1)
                        push!(NiDTau, i)
                    end
                end
                #println("Distante Nachbarn $NiCTau in Iteration $q")
                push!(NiCTau, u)
                Vq = ∪(NiCTau, NiDTau)
                push!(V, Vq)
            end

            # Set W and U
            W = ∪(W, Vq)
            #println("W in Iteration $q ist $W")
            U = setdiff(U,Vq)
            #println("U in Iteration $q ist $U")

            # Break if there are no more entries in U
            if(length(U) == 0)
                break
            end
        end

        #println("V Vektoren $V")

        Z = zeros(q,n)
        # Compute Interpolation-Matrix
        for i = 1:q
            Vq = pop!(V)
            #println("Vq in Iteration $j ist $Vq")
            for j = 1:n
                if(in(j, Vq))
                    Z[i,j] = 1
                end
            end
        end
        return Z

    end

end
```

Listing 3: reduktionsbasierte Restriktion

```
module ReductionbasedProlongation
export prolongationReduction

    using LinearAlgebra

    function computeTauI(A, F, i)
        aij = 0
        for j ∈ F
            aij = aij + abs(A[i,j])
        end

        if(isempty(F))
            return abs(A[i,i])
        end

        return abs(A[i,i]) / aij
    end

    function Adj(A, j, n)
        Adj = Set()
        for k = 1:n
            if(A[j,k] != 0)
                push!(Adj,k)
            end
        end
        return Adj
    end

    function greedyCoarsing(A)
        tau = 1

        n = size(A,1)
        U = Set(collect(1:n))
        F = Set()
        C = Set()
        TauI = zeros(n)

        for i = 1:n
            TauI[i] = computeTauI(A,F,i)
        end

        for i = 1:n
            if(TauI[i] ≥ tau)
                F = ∪(F, i)
                U = setdiff(U, i)
            end
        end

        while(!isempty(U))
            min = Inf
            j = -1
            for i = 1:n
                if(!in(i,U))
                    continue
                end
                if(TauI[i] ≤ min)
                    min = TauI[i]
                    j = i
                end
```

```
            end
            C = ∪(C, j)
            U = setdiff(U,j)
            for i ∈ ∩(U, Adj(A,j,n))
                TauI[i] = computeTauI(A,F,i)
                F = ∪(F, i)
                U = setdiff(U, i)
            end
        end
        return F, C, TauI
    end

    function swapColumns(A, i, j)
        if(i == j)
            return A
        end

        m, n = size(A)
        if (1 <= i <= n) && (1 <= j <= n)
            for k = 1:m
                  @inbounds A[k,i],A[k,j] = A[k,j],A[k,i]
            end
            return A
        else
            throw(BoundsError())
        end
    end

    function swapRows(A, i, j)
        if(i == j)
            return A
        end

        m, n = size(A)
        if (1 <= i <= n) && (1 <= j <= n)
            for k = 1:n
                  @inbounds A[i,k],A[j,k] = A[j,k],A[i,k]
            end
            return A
        else
            throw(BoundsError())
        end
    end

    function prolongationReduction(origA)
        A = copy(origA)
        n = size(A,1)

        FSet, CSet, TauI = greedyCoarsing(A)
        F = collect(FSet)
        C = collect(CSet)
        # Swaps for F
        f = length(F)
        for i in 1:f
            swapRows(A,i,F[i])
            swapColumns(A,i,F[i])
        end

        # Swaps for C
        for i in f+1:n
```

```
            # was already swaped
            if in(C[i-f],1:f)
                if(in(i,F) && findfirst(x->x==i,F) == F[C[i-f]])
                    continue
                end
                swapRows(A,i,F[C[i-f]])
                swapColumns(A,i,F[C[i-f]])
            else
                #normal swap
                swapRows(A,i,C[i-f])
                swapColumns(A,i,C[i-f])
            end
        end


        D1 = []
        try
            D1 = -1*inv(Diagonal(A[1:f,1:f]))
        catch
            D1 = -1*pinv(Diagonal(A[1:f,1:f]))
        end
        TopMatrix = hcat(D1, A[1:f,(f+1):n])
        return Matrix(transpose(vcat(TopMatrix, hcat(zeros(n-f,f),
        Matrix{Int64}(I, n-f,n-f)))))
    end

end
```

Listing 4: Aufbau des Vorkonditionierers

```
module Solver
export solve

using LinearAlgebra
using IterativeSolvers

    # Solves the System Ax=b with de Deflation-Operator R
    function solve(A, b, R, useM3, verb = false)
        n = size(A,1)

        P = transpose(R)

        Q = []
        try
            Q = P*inv(R*A*P)*R
        catch
            Q = P*pinv(R*A*P)*R
        end

        if(useM3)
            InvDiagA = []
            try
                InvDiagA = inv(Diagonal(A))
            catch
                InvDiagA = pinv(Diagonal(A))
            end
            PN = InvDiagA*(I - A * Q) + Q
        else
            PN = I - A * Q + Q
        end


        IterativeSolvers.gmres(A, b; tol = 1/10000000000000000000000,
        Pl = LinearAlgebra.lu(PN, check = false), restart=max(10000, 2*n),
        maxiter=max(10000, 2*n), log = true, verbose=verb)
    end
end
```