Eine Abstrakte Implementierung der Low-Rank ADI Iteration für Lyapunovgleichungen in pyMOR

An der Fakultät für Mathematik
der Otto-von-Guericke-Universität Magdeburg
angefertigte

Bachelorarbeit

vorgelegt von
Linus Balicki
geboren am 02.07.1996 in Stade,
Studiengang Mathematik,
Studienrichtung Computermathematik.

21. Mai 2019

Betreut am Max-Planck-Institut für Dynamik komplexer technischer Systeme von
Dr. Jens Saak

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Symbolverzeichnis

$\mathbb{R}$	Die Menge der reelen Zahlen
$\mathbb{C}$	Die Menge der komplexen Zahlen
$\mathrm{Re}(x)$, $\mathrm{Im}(x)$	Der Real- beziehungsweise Imaginärteil von $x \in \mathbb{C}$
$\overline{x}$	Die komplex Konjugierte
$\lvert . \rvert$	Der Betrag eines Elements
i	Die imaginäre Einheit, wobei $i^2 = -1$
$\mathbb{C}_-$	Die Menge der komplexen Zahlen mit negativem Realteil
$\mathbb{R}^n$, $\mathbb{C}^n$	Die Mengen der Vektoren mit n Elementen aus $\mathbb{R}$ beziehungsweise $\mathbb{C}$
$\mathbb{R}^{m\times n}$, $\mathbb{C}^{m\times n}$	Die Mengen der Matrizen mit m Zeilen, n Spalten und Elementen aus $\mathbb{R}$ beziehungsweise $\mathbb{C}$
A^T	Die transponierte Matrix
A^{-1}	Die inverse Matrix
A^{-T}	Die inverse und transponierte Matrix $(A^T)^{-1} = (A^{-1})^T$
A^*	Die komplex konjugierte transponierte Matrix $\overline{A}^T$
A^{-*}	Die invertierte komplex konjugierte transponierte Matrix $(\overline{A}^{-1})^T$
$\lvert\lvert . \rvert\rvert$	Die euklidische Norm eines Vektors beziehungsweise die Spektralnorm einer Matrix
$\Lambda(A)$, $\Lambda(A, E)$	Das Spektrum der Matrix A beziehungsweise des Matrixpaars (A, E)
$\rho(A)$	Der Spektralradius der Matrix A
$\kappa(A)$	Die Konditionszahl der Matrix A
$A \otimes B$	Das Kroeneckerprodukt der Matrizen A und B
$\mathrm{vec}(A)$	Der Vektorisierungsoperator angewendet auf A
$\dot{x}(t) = \frac{\partial}{\partial t}x$	Die partielle Ableitung von $x(t)$ bezüglich t
$\ddot{x}(t) = \frac{\partial^2}{\partial t^2}x$	Die zweifache partielle Ableitung von $x(t)$ bezüglich t
$\mathrm{diag}(x_1, \ldots, x_n)$	Die Diagonalmatrix D mit n Zeilen und n Spalten, wobei $D_{ii} = x_i$
$\mathbb{1}$	Der Einsvektor

1 Einleitung

Die mathematische Modellierung und die Analyse des Verhaltens technischer und dynamischer Systeme ist ein zentraler Bestandteil vieler praktischer Anwendungen, sowie die Grundlage für Forschungsgebiete wie zum Beispiel der Regelungs- und Steuerungstheorie. Bei der Modellierung spielen Differentialgleichungen eine wichtige Rolle, welche mit der wachsenden Komplexität heutiger Problemstellungen immer höhere Dimensionen annehmen. Die resultierenden Differentialgleichungen sind oft zu groß, um Simulationen oder numerische Berechnungen in angemessener Zeit durchzuführen. Zielstellung der Modellreduktion ist es nun, die hochdimensionalen Systeme auf Kleinere zu projizieren, sodass diese effizient gelöst werden können und gleichzeitig das Eingangs- und Ausgangsverhalten des ursprünglichen Systems gut approximieren.

Ein System von Differentialgleichungen welches bei diesen Prozessen auftritt, sind sogenannte lineare zeitinvariante (LTI) Systeme

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{1.1}$$

mit $A, E \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$ und $D \in \mathbb{R}^{p\times m}$, wobei $u(t) \in \mathbb{R}^m$ die Eingangsgrößen, $y(t) \in \mathbb{R}^p$ die Ausgangsgrößen und $x(t) \in \mathbb{R}^n$ den Zustand des Systems beschreiben. Die in dieser Arbeit durchgeführten Betrachtungen beschränken sich dabei auf den Fall, dass die Matrix E invertierbar ist. Eine weitverbreitete Modellreduktionsmethode für diese Art von System stellt dabei das balancierte Abschneiden dar, bei welchem als zentrale Komponente verallgemeinerte Lyapunovgleichungen der Form

$$\begin{aligned} AXE^T + EXA^T + BB^T &= 0 \\ A^TXE + E^TXA + C^TC &= 0 \end{aligned} \tag{1.2}$$

gelöst werden müssen. Es existiert eine Vielzahl von Software, welche numerische Lösungen für die in der Modellreduktion auftretenden Probleme bereitstellt. Beispiele hierfür sind die Bibliotheken `SLICOT` [1], welche in `Fortran 77` implementierte Routinen für in der Regelungs- und Steuerungstheorie auftretende Probleme zur Verfügung stellt und `M.E.S.S.` [2], welche in erster Linie zum Lösen großer dünnbesetzter Matrixgleichungen in `MATLAB` entwickelt wurde. Eine für die `Python` Programmiersprache entwickelte Bibliothek stellt dabei `pyMOR` [3, 4] dar, wobei diese sich durch eine auf abstrakten Operatoren basierende Umsetzung profiliert. Das Ziel dieser Arbeit ist es die Low-Rank Alternating Direction Implicit (LR ADI) Iteration für das Lösen von Lyapunovgleichungen unter Verwendung der von `pyMOR` zur Verfügung gestellten Schnittstellen zu implementieren. Eine solche Implementierung kann anschließend in die Bibliothek integriert werden, sodass diese unabhängig von externer Software agieren kann. Des Weiteren bietet eine den Designvorgaben `pyMOR`s entsprechende Implementierung den Vorteil, dass verteilt paralleles Rechnen perspektivisch einfach zugelassen wird. Letztendlich soll untersucht werden, inwiefern sich die Laufzeit einer solchen Umsetzung zu Alternativen aus externen Bibliotheken unterscheidet. Dabei werden sich die numerischen Experimente auf die `C-M.E.S.S.` Bibliothek fokussieren, welche mit `Py-M.E.S.S.` eine optimierte, mit `Python` kompatible Implementierung der LR ADI

Iteration zur Verfügung stellt.

1.1 Überblick

Der Inhalt dieser Arbeit setzt sich aus dem Betrachten der LR ADI Iteration für Lyapunovgleichungen und den für die Implementierung dieses Verfahrens in `pyMOR` relevanten Aspekten zusammen.

Nach der Einleitung in Kapitel 1 wird sich Abschnitt 2 mit der Wiederholung von mathematischen Grundlagen und einer Motivation für das Lösen von Lyapunovgleichungen beschäftigen.

In Kapitel 3 wird die LR ADI Iteration hergeleitet und gesammelte Ergebnisse werden in Algorithmen zusammengefasst. Die Koeffizientenmatrizen von Lyapunovgleichungen sind in vielen Anwendungsfällen dünnbesetzt, wobei die Lösungen und Residuen dies im Regelfall nicht sind. Dieser Zusammenhang wird als Motivation dafür dienen, die Lösung und das Residuum als Produkt zweier Lösungsfaktoren niedrigen Ranges darzustellen. Darüber hinaus wird sich das Kapitel damit auseinandersetzen, auf welche Weise sich das Speichern komplexer Daten in der Iteration vermeiden lässt. Abschließend wird die Wahl von Shiftparametern diskutiert, welche einen wichtigen Bestandteil einer schnellen Konvergenz des Algorithmus darstellen werden.

Abschnitt 4 beschäftigt sich mit der verwendeten Software, wobei vorrangig praktische Aspekte der Implementierung des Algorithmus diskutiert werden.

In Kapitel 5 werden Laufzeiten verschiedener Implementierungen der LR ADI Iteration verglichen und analysiert.

Abschließend fasst Kapitel 6 die Resultate dieser Arbeit zusammen, wobei auf weitere Funktionen welche in `pyMOR` implementiert werden könnten, hingewiesen wird. Die Anhänge bestehen aus den Testergebnissen der numerischen Experimente und einem Datenträger mit den dafür verwendeten Implementierungen.

2 Grundlagen und Motivation

In den folgenden Paragraphen werden die grundlegenden mathematischen Bestandteile dieser Arbeit betrachtet. Dafür soll zunächst eine Motivation für das Lösen von Lyapunovgleichungen über das Einführen des balancierten Abschneidens geschaffen werden, welches anschließend mit einem Beispiel aus der Physik verknüpft wird. Wie in (1.1) eingeführt fokussiert sich diese Arbeit auf LTI Systeme, welche zeitkontinuierlich und asymptotisch stabil sind, also $\Lambda(A, E) \subset \mathbb{C}_-$ erfüllen. Diese Systeme werden im Folgenden kurz als *stabil* bezeichnet.

2.1 Balanciertes Abschneiden

Das balancierte Abschneiden stellt eine etablierte Modellreduktionsmethode dar, welche im Folgenden basierend auf den Auslegungen in [5] kurz diskutiert wird. In diesem Verfahren wird versucht die Dimension eines vorliegenden LTI Systems zu reduzieren, indem Zustandsvariablen eliminiert werden, welche sowohl schwierig zu erreichen als auch schwierig zu beobachten sind. Als erreichbar soll hierbei ein Zustand gelten, welcher ausgehend vom Anfangszustand $x(0)$ eines LTI Systems in endlicher Zeit und mit endlich viel eingehender Energie konstruierbar ist. Das Konzept der Beobachtbarkeit beschäftigt sich damit, ob durch das ausschließliche Betrachten der Ausgangsvariablen $y(t)$ und Eingangsvariablen $u(t)$ in einem Zeitintervall $t \in [T, T + \tau]$ der Zustand $x(T)$ ermittelt werden kann. Es stellt sich natürlich die Frage wie genau *schwierig* erreichbar und beobachtbar verstanden werden soll. Eine Möglichkeit ein Maß für Erreichbar- und Beobachtbarkeit zu erhalten bieten die Steuerbarkeits-Gramsche

$$\mathcal{P} = \int_0^\infty e^{[E^{-1}A]t} E^{-1} B B^T E e^{[E^{-1}A]^T t} dt,$$

und Beobachtbarkeits-Gramsche

$$E^T \mathcal{Q} E = E^T \int_0^\infty e^{[E^{-1}A]^T t} C C^T e^{[E^{-1}A]t} dt E.$$

Kleine Eigenwerte von $\mathcal{P}$ können nun mit Zuständen assoziiert werden deren Erreichbarkeit ein hohes Maß an eingehender Energie erfordert, also schwierig zu erreichen sind, wobei kleine Eigenwerte von $\mathcal{Q}$ auf gewisse Weise mit Zuständen korrespondieren, welche sich nur wenig vom ursprünglichen Zustand unterscheiden und somit schwierig zu beobachten sind. Da nun beim balancierten Abschneiden Zustände eliminiert werden sollen, welche beide dieser Eigenschaften gleichzeitig erfüllen, wird eine invertierbare Matrix $T \in \mathbb{R}^{n \times n}$ gesucht, sodass für die transformierten Gramsche mit $\Sigma_1 \in \mathbb{R}^{r \times r}$ und $\Sigma_2 \in \mathbb{R}^{(n-r)\times(n-r)}$ die folgende Transformation gilt

$$T \mathcal{P} T^T = T^{-1} E^T \mathcal{Q} E T^{-T} = \Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n).$$

Das hier betrachtete LTI System wurde bisher durch das 4-Tupel $(E^{-1}A, E^{-1}B, C, D) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m} \times \mathbb{R}^{p\times n} \times \mathbb{R}^{p\times m}$ beschrieben, wobei ein solches Tupel als *Realisierung* bezeichnet wird. Mithilfe der Transformation T kann die bisherige Realisierung des Sys-

tem in eine neue, sogenannte *balancierte Realisierung* $(TE^{-1}AT^{-1}, TE^{-1}B, CT^{-1}, D) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m} \times \mathbb{R}^{p\times n} \times \mathbb{R}^{p\times m}$ überführt werden, welche das selbe physikalische System beschreibt wie die ursprüngliche Realisierung. Bei der balancierten Realisierung liegen nun schwierig erreichbare und beobachtbare Zustände im Spann der Eigenvektoren von Σ, welche mit kleinen σ_i korrespondieren. Um ein solches T zu ermitteln ist es zunächst notwendig die beiden Gramsche zu berechnen. Es lässt sich leicht zeigen [6], dass diese sich aus den Lösungen der verallgemeinerten Lyapunovgleichungen

$$A\mathcal{P}E^T + E\mathcal{P}A^T + BB^T = 0$$
$$A^T\mathcal{Q}E + E^T\mathcal{Q}A + C^TC = 0$$

ergeben, deren Eigenschaften im nächsten Unterkapitel genauer untersucht werden. Anschließend müssen Choleskyartige Faktorisierungen der Form $\mathcal{P} = Z_\mathcal{P} Z_\mathcal{P}^T$ und $\mathcal{Q} = Z_\mathcal{Q} Z_\mathcal{Q}^T$ ermittelt werden, welche über die Singulärwertzerlegung

$$Z_\mathcal{Q}^T E Z_\mathcal{P} = L\Sigma R = \begin{bmatrix} L_1 & L_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$$

letztendlich das Bilden der Matrix $T := Z_\mathcal{P} R \Sigma^{-\frac{1}{2}}$ mit der gewünschten Eigenschaft zulassen. Hierbei gilt $\Sigma^{-\frac{1}{2}} = \operatorname{diag}(\frac{1}{\sqrt{\sigma_1}}, \ldots, \frac{1}{\sqrt{\sigma_n}})$, wobei offensichtlich $T^{-1} = Z_\mathcal{Q} L \Sigma^{-\frac{1}{2}}$. Nach dem Ermitteln einer solchen balancierenden Transformation fehlt nur noch das Abschneiden, also das Entfernen der unerwünschten Zustände. Seien hierfür $T_l := Z_\mathcal{P} R_1 \Sigma_1^{-\frac{1}{2}}$ und $T_r := Z_\mathcal{Q} L_1 \Sigma_1^{-\frac{1}{2}}$, welche mit $\tilde{E} := T_l^T E T_r$, $\tilde{A} := T_l^T A T_r$, $\tilde{B} := T_l^T B$, $\tilde{C} := C T_r$ und $\tilde{D} = D$ das reduzierte System

$$\tilde{E}\dot{\tilde{x}}(t) = \tilde{A}\tilde{x}(t) + \tilde{B}u(t)$$
$$\tilde{y}(t) = \tilde{C}\tilde{x}(t) + \tilde{D}u(t)$$

der Ordnung r bilden, wobei $\tilde{A}, \tilde{E} \in \mathbb{R}^{r\times r}$, $B \in \mathbb{R}^{r\times m}$, $\tilde{C} \in \mathbb{R}^{p\times r}$ und $\tilde{D} \in \mathbb{R}^{p\times m}$.

2.2 Lyapunovgleichungen

Die zeitkontinuierliche Lyapunovgleichung ist eine lineare algebraische Matrixgleichung der Form

$$AX + XA^T + BB^T = 0, \tag{2.1}$$

wobei $X = X^T \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{n\times n}$ und $B \in \mathbb{R}^{n\times m}$. In vielen Anwendungen ist es jedoch notwendig verallgemeinerte Lyapunovgleichungen zu betrachten. Entsprechend wird sich die Formulierung des Algorithmus auf den Fall

$$AXE^T + EXA^T + BB^T = 0 \tag{2.2}$$

fokussieren, wobei stets $E \in \mathbb{R}^{n\times n}$ invertierbar sein soll. Es stellt sich die Frage, ob und wann Gleichungen dieser Form eindeutig lösbar sind. Um die entsprechenden Betrachtungen durchzuführen sei zunächst das Kronecker-Produkt $\otimes : \mathbb{R}^{d\times q} \times \mathbb{R}^{v\times w} \to \mathbb{R}^{dv\times qw}$

für beliebige Matrizen $M = (m_{ij})_{\substack{1\leq i\leq d\\1\leq j\leq q}} \in \mathbb{R}^{d\times q}$ und $N = [n_1, \ldots, n_w] \in \mathbb{R}^{v\times w}$ definiert als

$$M \otimes N = \begin{bmatrix} m_{11}N & \cdots & m_{1q}N \\ \vdots & \ddots & \vdots \\ m_{d1}N & \cdots & m_{dq}N \end{bmatrix} \in \mathbb{R}^{dv\times qw}$$

und der Vektorisierungsoperator $\mathrm{vec} : \mathbb{R}^{v\times w} \to \mathbb{R}^{vw}$ definiert als

$$\mathrm{vec}(N) = \begin{bmatrix} n_1 \\ \vdots \\ n_w \end{bmatrix} \in \mathbb{R}^{vw},$$

welcher somit die Spaltenvektoren einer Matrix in einem Vektor anordnet. Mithilfe dieser Definitionen kann nun die verallgemeinerte Lyapunovgleichung (2.2) in ein lineares Gleichungssystem überführt werden. Es kann leicht gezeigt werden [7], dass Gleichung (2.2) äquivalent ist zu

$$(A \otimes E^T + E \otimes A^T)\mathrm{vec}(X) = -\mathrm{vec}(BB^T).$$

Dieses System ist nun eindeutig lösbar, sofern die Matrix $\mathcal{M} = A \otimes E^T + E \otimes A^T$ invertierbar ist, also $\Lambda(\mathcal{M}) \cap \{0\} = \emptyset$ gilt. Dieser Fall tritt genau dann ein, wenn für alle $\lambda_i, \lambda_j \in \Lambda(A, E)$ gilt $\lambda_i \neq \lambda_j$. Insbesondere tritt dies ein sofern $\Lambda(A, E) \subset \mathbb{C}_-$, wobei ein Matrixpaar (A, E) mit einer solchen Eigenschaft oder eine Matrix A mit $\Lambda(A) \subset \mathbb{C}_-$ als *Hurwitz* bezeichnet wird. In diesem Fall kann, wie bereits im letzten Unterkapitel erwähnt, gezeigt werden [6], dass die eindeutige Lösung durch

$$X = \int_0^\infty [e^{E^{-1}A}]E^{-1}BB^TE[e^{E^{-1}A}]^T dt$$

gegeben ist. Dabei übernimmt X die Definitheitseigenschaften der Matrix BB^T.
Die folgenden Betrachtungen beschränken sich im Sinne der Übersichtlichkeit und Wohldefiniertheit stets auf den Fall, dass (A, E) in Gleichung (2.2) und A in Gleichung (2.1) Hurwitz sind. Des Weiteren werden ausschließlich reelle Matrizen A, E und B betrachtet, da dies der für die Anwendung relevanteste Fall ist.

2.3 Anwendungsbeispiel

Die Anwendung von Lyapunovgleichungen in der Physik lässt sich durch ein Beispiel mit Wärmeleitungsgleichungen zeigen [3, 8]. Es wird dafür die Wärmeausbreitung $T(s, t)$ auf einem 1-dimensionalen Segment im Intervall $[0, 1]$ betrachtet, welche sich folgendermaßen modellieren lässt:

$$\frac{\partial}{\partial t}T(s, t) = \frac{\partial^2}{\partial s^2}T(s, t), \quad t > 0, s \in (0, 1).$$

In diesem Beispiel wird der Fall untersucht, dass die linke Seite des Segments erhitzt wird und sind daran interessiert, inwiefern sich die Temperatur auf der rechten Seite in zeitlicher Abhängigkeit verhält. Beschreibe nun $u(t)$ die eingehende und $y(t)$ die ausgehende Wärme, dann lassen sich die Rand- und Anfangsbedingungen formulieren als

$$\begin{aligned}\frac{\partial}{\partial s}T(0,t) &= T(0,t) - u(t),\\ \frac{\partial}{\partial s}T(1,t) &= -T(1,t),\\ y(t) &= T(1,t),\end{aligned}$$

wobei stets $t > 0$. Um eine approximative Lösung dieses Systems von Differentialgleichungen ermitteln zu können, bietet es sich zunächst an, das System über beispielsweise die Finite-Differenzen-Methode [9] zu diskretisieren. Die Gitterpunkte $0 = s_0 < \ldots < s_n = 1$ seien hierfür definiert als äquidistante Zerlegung des Intervalls $[0, 1]$. Entsprechend ergibt sich durch das Betrachten zentraler Differenzenquotienten für $i = 1, \ldots, n$ die semi-diskretisierte Formulierung

$$\begin{aligned}\dot{x}_i(t) &= \frac{x_{i-1}(t) - 2x_i(t) + x_{i+1}(t)}{(\Delta s)^2},\\ \frac{x_2(t) - x_0(t)}{2\Delta s} &= x_1(t) - u(t),\\ \frac{x_{n+1}(t) - x_{n-1}(t)}{2\Delta s} &= -x_n(t),\\ \hat{y}(t) &= x_n(t),\end{aligned}$$

wobei $\Delta s = \frac{1}{n-1}$. In dieser Darstellung liefert $x_i(t)$ eine Approximation für $T(s_i, t)$ und $\hat{y}(t) \approx y(t)$. Durch das Umstellen der zweiten und dritten Gleichung der Semi-Diskretisierung nach x_0 beziehungsweise x_{n+1} und dem anschließenden Einsetzen der resultierenden Terme in die erste Gleichung für $i = 1, n$ ergeben sich die folgenden Darstellungen für $i = 2, \ldots, n-1$

$$\begin{aligned}\dot{x}_1(t) &= -2n(n-1)x_1(t) + 2(n-1)^2x_2(t) + 2(n-1)u(t),\\ \dot{x}_i(t) &= (n-1)^2x_{i-1}(t) - 2(n-1)^2x_i(t) + (n-1)^2x_{i+1}(t),\\ \dot{x}_n(t) &= 2(n-1)^2x_{n-1}(t) - 2n(n-1)x_n(t).\end{aligned}$$

Mithilfe dieser Gleichungen lassen sich die Matrizen

$$A = \begin{bmatrix} -2n(n-1) & 2(n-1)^2 & & & \\ (n-1)^2 & -2(n-1)^2 & (n-1)^2 & & \\ & \ddots & \ddots & \ddots & \\ & & (n-1)^2 & -2(n-1)^2 & (n-1)^2 \\ & & & 2(n-1)^2 & -2n(n-1) \end{bmatrix} \in \mathbb{R}^{n\times n},$$

$$B = \begin{bmatrix} 2(n-1) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{n \times 1} \quad \text{und} \quad C = \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix} \in \mathbb{R}^{1 \times n}$$

definieren, welche das LTI System

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ \hat{y}(t) &= Cx(t) \end{aligned}$$

bilden. Dieses System hat an dieser Stelle gegebenenfalls eine zu große Dimension, um damit weitere Berechnungen oder Simulationen durchzuführen. Wie in Abschnitt 2.1 beschrieben, kann das System mithilfe des balancierten Abschneidens reduziert werden, wobei das hierfür notwendige Lösen der Lyapunovgleichungen wie in den folgenden Kapiteln thematisiert durchgeführt werden kann.

3 Numerische Lösungsansätze für Lyapunovgleichungen

Für das Lösen von Lyapunovgleichungen existiert eine Vielzahl von Algorithmen, wobei der Bartels-Stewart [10] und Hammarlings Algorithmus [11] die etabliertesten direkten Verfahren darstellen. Die Laufzeit dieser Methoden ist dabei $\mathcal{O}(n^3)$, wobei bei ihnen unter anderem Schur-Zerlegungen von Matrizen ermittelt werden müssen . Entsprechend lassen sich mit ihnen in angemessener Zeit nur Gleichungen lösen bei welchen die Matrizen A und E aus Gleichung (2.2) kleine Dimensionen haben. Jedoch treten bei diversen in der Modellreduktion benutzten Methoden, beispielsweise bei der Diskretisierung von Differentialgleichungen [9], große, dafür aber dünnbesetzte Koeffizientenmatrizen auf. Die daraus resultierenden Lyapunovgleichungen können nicht effizient mit den genannten direkten Verfahren gelöst werden.

Es kann jedoch angenommen werden, dass sich Matrixvektorprodukte und lineare Gleichungssysteme bei diesen Matrizen effizient lösen lassen [6], weshalb Algorithmen für große dünnbesetzte Matrizen weitestgehend auf solche Matrixoperationen zurückgreifen. Des Weiteren tritt es in der Praxis häufig auf, dass der Rang der Matrix B und somit auch BB^T sehr niedrig ist. Ein iteratives Verfahren welches die genannten Eigenschaften der Koeffizientenmatrizen ausnutzen kann und auch für größere Problemstellungen in der Lage ist in angemessener Zeit Lösungen zu berechnen, stellt dabei die LR ADI Iteration dar. In den folgenden Paragraphen wird diese betrachtet, wobei an mehreren Stellen die Resultate in Algorithmen zusammengefasst werden.

3.1 Herleitung der ADI Iteration

Einen klassischen Ansatz für das Herleiten der ADI Iteration für Lyapunovgleichungen stellt dabei die Formulierung der zweistufigen Iteration [12, 6]

$$\begin{aligned}(A+\alpha_i I)X_{i-\frac{1}{2}} &= -BB^T - X_{i-1}(A^T-\alpha_i I)\\ (A+\alpha_i I)X_i^T &= -BB^T - X_{i-\frac{1}{2}}^T(A^T-\alpha_i I).\end{aligned}$$

für die Standard Lyapunovgleichung (2.1) dar. Dabei ist $\mathcal{S} = \{\alpha_1, \ldots, \alpha_s\} \subset \mathbb{C}_-$ die Menge der *Shiftparameter*, deren sinnvolle Auswahl in Abschnitt 3.4 besprochen wird, und $X_0 = X_0^T \in \mathbb{R}^{n\times n}$ die beliebig gewählte Startmatrix der Iteration. Dieser Ausdruck lässt sich durch das Umstellen der zweiten Gleichung nach $X_{i-\frac{1}{2}}$ und dem Einsetzen des resultierenden Ausdrucks in die erste Gleichung umformulieren, sodass X_i explizit geschrieben werden kann als

$$\begin{aligned}X_i = {} & (A+\alpha_i I)^{-1}(A-\overline{\alpha_i}I)X_{i-1}(A-\overline{\alpha_i}I)^*((A+\alpha_i I)^{-1})^*\\ & - 2\,\mathrm{Re}(\alpha_i)(A+\alpha_i I)^{-1}BB^T((A+\alpha_i I)^{-1})^*.\end{aligned} \tag{3.1}$$

Des Weiteren lässt sich eine ähnliche Formulierung für die verallgemeinerte Lyapunovgleichung (2.2) herleiten [6]. Es wird dafür Gleichung (2.1) mit $\tilde{A} = E^{-1}A$, sowie $\tilde{B} = E^{-1}B$ betrachtet:

$$\tilde{A}X + X\tilde{A}^T + \tilde{B}\tilde{B}^T = 0. \tag{3.2}$$

Offensichtlich lässt sich somit also eine verallgemeinerte Lyapunovgleichung mit invertierbarem E stets als Standard Lyapunovgleichung formulieren. Durch das Einsetzen der Matrizen $\tilde{A}$ und $\tilde{B}$ in Gleichung (3.1), ergibt sich der folgende Ausdruck für X_i:

$$\begin{aligned} X_i = &(A + \alpha_i E)^{-1}(A - \overline{\alpha_i}E)X_{i-1}(A - \overline{\alpha_i}E)^*((A + \alpha_i E)^{-1})^* \\ &- 2\,\mathrm{Re}(\alpha_i)(A + \alpha_i E)^{-1}BB^T((A + \alpha_i E)^{-1})^*. \end{aligned} \tag{3.3}$$

Die hergeleitete Iteration nutzt in dieser Form jedoch noch nicht den niedrigen Rang der Matrix BB^T aus. Inwiefern sich diese Eigenschaft in den Algorithmus integrieren lässt, wird in dem nächsten Abschnitt betrachtet.

3.2 Das Low-Rank Phänomen

Eines der relevantesten Resultate für die Formulierung der LR ADI Iteration für große Lyapunovgleichungen mit einem $B \in \mathbb{R}^{n\times m}$ von niedrigem Rang wird beispielsweise in [13, 14] thematisiert. Die für dieses Kapitel wichtigsten Aussagen dieser Arbeiten sind dabei, dass $m \ll n$ auch einen niedrigen numerischen Rang für die Lösung X der Lyapunovgleichung impliziert. Genauer gesagt, nähern sich die Singulärwerte von X oft exponentiell 0 an. Dieser Zusammenhang motiviert dazu, die Lösung als Produkt von zwei Lösungsfaktoren niedrigen Ranges, also $X = ZZ^*$, darzustellen, wobei $Z \in \mathbb{C}^{n\times l}$ und $l \ll n$.

3.2.1 Lösungsfaktoren

Ziel dieses Kapitels soll es nun sein den Ausdruck in Gleichung (3.3) auf einen einzelnen Lösungsfaktor Z_i der Zerlegung $X_i = Z_i Z_i^*$ zu übertragen. Als Startmatrix der Iteration wird hierfür $X_0 = Z_0 Z_0^*$ betrachtet, wobei $Z_0 = [\,] \in \mathbb{C}^{n\times 0}$. Aus der folgenden Umformulierung von Gleichung (3.3) wird direkt ein Ausdruck für die Lösungsfaktoren ersichtlich:

$$\begin{aligned} Z_i Z_i^* = &\big[(A + \alpha_i E)^{-1}(A - \overline{\alpha_i}E)Z_{i-1}\big]\big[(A + \alpha_i E)^{-1}(A - \overline{\alpha_i}E)Z_{i-1}\big]^* \\ &+ \big[\sqrt{-2\,\mathrm{Re}(\alpha_i)}(A + \alpha_i E)^{-1}B\big]\big[\sqrt{-2\,\mathrm{Re}(\alpha_i)}(A + \alpha_i E)^{-1}B\big]^*. \end{aligned}$$

Für Z_i gilt dann entsprechend

$$Z_i = \big[\sqrt{-2\,\mathrm{Re}(\alpha_i)}(A + \alpha_i E)^{-1}B, (A + \alpha_i E)^{-1}(A - \overline{\alpha_i}E)Z_{i-1}\big]$$

und schließlich durch das wiederholte Einsetzen der Z_j für $j = i-1, \ldots, 0$:

$$Z_i = \big[\sqrt{-2\,\mathrm{Re}(\alpha_i)}(A + \alpha_i E)^{-1}B, \ldots, \sqrt{-2\,\mathrm{Re}(\alpha_1)}T_i \cdots T_1(A + \alpha_1 E)^{-1}B\big], \tag{3.4}$$

wobei $T_j := (A + \alpha_j E)^{-1}(A - \overline{\alpha_j}E)$. In dieser Form kann die Iteration noch nicht sinnvoll genutzt werden, da mit wachsendem i in jedem Berechnungsschritt eine Vielzahl linearer Gleichungssysteme gelöst werden müsste. Um dies zu vermeiden sollen die einzelnen in den T_j vorkommenden Faktoren, also $(A - \beta E)$ und $(A + \gamma E)^{-1}$ mit $\beta, \gamma \in \mathcal{S}$, umsortiert werden. Die folgenden Umformungen zeigen, dass dies sogar für allgemeine $\beta, \gamma \in \mathbb{C}$ und $A, E \in \mathbb{R}^{n\times n}$ mit $\tilde{A} = E^{-1}A$ möglich ist, sofern die auftretenden

Matrixinverse existieren:

$$\begin{aligned}(A+\gamma E)^{-1}(A-\beta E) &= (I+\gamma\tilde{A}^{-1})^{-1}A^{-1}A(I-\beta\tilde{A}^{-1}) \\ &= (I+\gamma\tilde{A}^{-1})^{-1}(I-\beta\tilde{A}^{-1}) \\ &= (I+\gamma\tilde{A}^{-1})^{-1} - \beta(I+\gamma\tilde{A}^{-1})^{-1}\tilde{A}^{-1} \\ &= \tilde{A}(\tilde{A}+\gamma I)^{-1} - \beta(\tilde{A}+\gamma I)^{-1} \\ &= (\tilde{A}-\beta I)(\tilde{A}+\gamma I)^{-1} \\ &= E^{-1}(A-\beta E)(A+\gamma E)^{-1}E\end{aligned} \tag{3.5}$$

Dies ermöglicht es einen zu Gleichung (3.4) äquivalenten Ausdruck zu formulieren

$$Z_i = \left[Z_{i-1}, \sqrt{-2\,\mathrm{Re}(\alpha_i)}V_i\right], \tag{3.6}$$

welcher durch entsprechendes Vertauschen der in den T_j auftretenden Faktoren hergeleitet werden kann. Hierbei wird die folgende Notation eingeführt:

$$\begin{aligned}V_1 &= (A+\alpha_1 E)^{-1}B, \\ V_i &= E^{-1}(A-\overline{\alpha_{i-1}}E)(A+\alpha_i E)^{-1}EV_{i-1} \\ &= V_{i-1} - (\alpha_i+\overline{\alpha_{i-1}})(A+\alpha_i E)^{-1}EV_{i-1}, \\ Z_1 &= \sqrt{-2\,\mathrm{Re}(\alpha_1)}V_1.\end{aligned} \tag{3.7}$$

Bei dieser Formulierung der Iteration muss in jedem Schritt ein lineares Gleichungssystem mit der Koeffizientenmatrix $A+\alpha_i E$ und mehreren durch EV_{i-1} definierten rechten Seiten gelöst werden. Vor allem wird nun nicht mehr die gesamte wachsende Matrix Z neu berechnet sondern nur der letzte durch $\sqrt{-2\,\mathrm{Re}(\alpha_i)}V_i$ gegebene Block. Die Effizienz des Algorithmus wird dadurch drastisch verbessert. Somit lässt sich basierend auf den bisher gesammelten Resultaten eine erste Version der LR ADI Iteration formulieren.

Algorithmus 1 – LR ADI Iteration Version 1

Input: $A, E \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $\{\alpha_1, \ldots, \alpha_s\} \subset \mathbb{C}_-$ (siehe Abschnitt 3.4)
Output: $Z \in \mathbb{C}^{n\times sm}$

1: **solve** $(A+\alpha_1 E)V_1 = B$ **for** V_1
2: $Z_1 = \sqrt{-2\,\mathrm{Re}(\alpha_1)}V_1$
3: **for** $i = 2, \ldots, s$ **do**
4: **solve** $(A+\alpha_i E)\hat{V} = EV_{i-1}$ **for** $\hat{V}$
5: $V_i = V_{i-1} - (\alpha_i+\overline{\alpha_{i-1}})\hat{V}$
6: $Z_i = \left[Z_{i-1}, \sqrt{-2\,\mathrm{Re}(\alpha_i)}V_i\right]$

Ein grundlegendes Problem dieser Variante stellt die Ungewissheit über die Qualität der approximierten Lösung dar. Ein heuristischer Ansatz für die Überprüfung der Genauigkeit der Approximation ist beispielsweise zu prüfen, ob sich die Lösung in einem Berechnungsschritt um mehr als einen festgelegten Schwellenwert $0 < \epsilon \ll 1$ ändert.

Dies würde in einem Abbruchkriterium der Form

$$||Z_i Z_i^* - Z_{i-1} Z_{i-1}^*|| < \epsilon$$

resultieren. Jedoch hat sich auch hierbei gezeigt, dass der Algorithmus gegebenenfalls bei zu ungenauen Lösungen terminiert [6], wenn aufgrund schlechter Shiftparameter die Stagnation als Konvergenz fehlinterpretiert wird. Eine wesentlich bessere Methode ist es, zu prüfen ob das *Lyapunov Residuum*

$$\mathcal{R}_i = AX_iE^T + EX_iA^T + BB^T$$

kleiner ist als eine vorgegebene Schranke. In der angegebenen Form wäre es jedoch sehr ineffizient das Residuum zu ermitteln, da in jedem Schritt der Iteration mehrere aufwendig zu berechnende Matrixmatrixprodukte gebildet werden müssten. Das nächste Kapitel widmet sich entsprechend einer praktikableren Formulierung des Lyapunov Residuums.

3.2.2 Residuumsfaktoren

Basierend auf Resultaten aus [15] wird in diesem Abschnitt gezeigt, dass neben der Lösung der Lyapunovgleichung auch ihr Residuum einen niedrigen Rang besitzt und somit die Zerlegung

$$\mathcal{R}_i = AX_iE^T + EX_iA^T + BB^T = W_iW_i^* \tag{3.8}$$

naheliegt, wobei $W_i \in \mathbb{C}^{n\times m}$ mit $m \ll n$. Sind die W_i bekannt, lässt sich wie im Verlauf des Abschnitts ersichtlich wird $||\mathcal{R}_i||$ beispielsweise in der Spektralnorm leicht berechnen.

Um einen handhabbaren Ausdruck für die W_i herzuleiten wird wie in Gleichung (3.2) die zur verallgemeinerten Lyapunovgleichung äquivalente Standardgleichung mit dem entsprechenden Residuum

$$\tilde{\mathcal{R}}_i = \tilde{A}X_i + X_i\tilde{A}^T + \tilde{B}\tilde{B}^T$$

betrachtet. Für das transformierte Residuum $E\tilde{\mathcal{R}}_iE^T = \mathcal{R}_i$ ergibt sich die Darstellung

$$\tilde{\mathcal{R}}_i = \tilde{A}(X_i - X) + (X_i - X)\tilde{A}^T = \tilde{W}_i\tilde{W}_i^*. \tag{3.9}$$

Um einen Ausdruck für $\tilde{W}_i$ herzuleiten, wird zunächst $X_i - X$ betrachtet. Dabei ergeben sich die folgenden Umformungen aus der Differenz von Gleichung (3.1) und X:

$$\begin{aligned}
X_i - X &= (\tilde{A} + \alpha_i I)^{-1}\big[(\tilde{A} - \overline{\alpha_i} I)X_{i-1}(\tilde{A} - \overline{\alpha_i} I)^* - 2\,\mathrm{Re}(\alpha_i)\tilde{B}\tilde{B}^T\big]((\tilde{A} + \alpha_i I)^{-1})^* - X \\
&= (\tilde{A} + \alpha_i I)^{-1}\big[(\tilde{A} - \overline{\alpha_i} I)X_{i-1}(\tilde{A} - \overline{\alpha_i} I)^* + (\alpha_i + \overline{\alpha_i})(\tilde{A}X + X\tilde{A}^T) \\
&\quad - (\tilde{A} + \alpha_i I)X(\tilde{A} + \alpha_i I)^*\big]((\tilde{A} + \alpha_i I)^{-1})^* \\
&= (\tilde{A} + \alpha_i I)^{-1}(\tilde{A} - \overline{\alpha_i} I)(X_{i-1} - X)(\tilde{A} - \overline{\alpha_i} I)^*((\tilde{A} + \alpha_i I)^{-1})^*.
\end{aligned}$$

Durch wiederholtes Anwenden obiger Gleichung auf $X_j - X$ für $j = i-1, \ldots, 0$ resultiert der Ausdruck

$$X_i - X = [\prod_{k=1}^{i} \tilde{T}_k](X_0 - X)[\prod_{k=1}^{i} \tilde{T}_k^*], \tag{3.10}$$

wobei $\tilde{T}_k = (\tilde{A} + \alpha_k I)^{-1}(\tilde{A} - \overline{\alpha_k} I)$. Wenn $X_0 = 0$ betrachtet wird, folgt aus den Gleichungen (3.9) und (3.10)

$$\begin{aligned}\tilde{W}_i\tilde{W}_i^* &= \tilde{A}[\prod_{k=1}^{i} \tilde{T}_k](-X)[\prod_{k=1}^{i} \tilde{T}_k^*] + [\prod_{k=1}^{i} \tilde{T}_k](-X)[\prod_{k=1}^{i} \tilde{T}_k^*]\tilde{A}^T \\ &= [\prod_{k=1}^{i} \tilde{T}_k]\tilde{B}\tilde{B}^T[\prod_{k=1}^{i} \tilde{T}_k^*].\end{aligned}$$

Schließlich wurde damit die gewünschte Zerlegung hergeleitet und es gilt

$$\tilde{W}_i = [\prod_{k=1}^{i} \tilde{T}_k]\tilde{B}.$$

Als nächstes wird erneut das in (3.7) eingeführte V_i betrachtet:

$$V_i = (\tilde{A} - \overline{\alpha_{i-1}} I)(\tilde{A} + \alpha_i I)^{-1} V_{i-1}.$$

Durch wiederholtes Einsetzen der V_j für $j = i-1, \ldots, 0$ und das Umsortieren der auftretenden Faktoren, ergibt sich

$$V_i = (\tilde{A} + \alpha_i I)^{-1}[\prod_{k=1}^{i-1} \tilde{T}_k]\tilde{B} = (\tilde{A} + \alpha_i I)^{-1}\tilde{W}_{i-1} = (\tilde{A} - \overline{\alpha_i} I)^{-1}\tilde{W}_i.$$

Daraus folgt direkt die folgende Formulierung

$$\tilde{W}_i = (\tilde{A} + \alpha_{i+1} I)V_{i+1} = (\tilde{A} - \overline{\alpha_i} I)V_i = \tilde{W}_{i-1} - 2\,\mathrm{Re}(\alpha_i)V_i.$$

Dieser Ausdruck lässt sich nun unter Betrachtung von $E\tilde{W}_j = W_j$ zurücktransformieren, sodass letztendlich gilt

$$W_i = (A + \alpha_{i+1} E)V_{i+1} = (A - \overline{\alpha_i} E)V_i = W_{i-1} - 2\,\mathrm{Re}(\alpha_i)EV_i. \tag{3.11}$$

Eine solche Darstellung von W_i macht es nun möglich beispielsweise in der Spektral- oder Frobeniusnorm das Lyapunov Residuum effizient zu bestimmen. Dies basiert vor allem darauf, dass die genannten Normen selbst-adjungiert sind, also

$$||\mathcal{R}_i|| = ||W_i W_i^*|| = ||W_i^* W_i||$$

gilt, welches sich über die Singulärwertzerlegung $W_i = U\Sigma V^*$ zeigen lässt:

$$||W_i W_i^*|| = ||U\Sigma V^* V \Sigma^* U^*|| = ||\Sigma^2|| = ||V\Sigma U^* U \Sigma V^*|| = ||W_i^* W_i||.$$

Hierbei ist $W_i^* W_i \in \mathbb{R}^{m \times m}$ eine kleine quadratische Matrix deren Eigenwerte sich durch ein direktes Verfahren in $\mathcal{O}(m^3)$ bestimmen lassen. Da $m \ll n$ vorausgesetzt wurde, hat das Berechnen der Eigenwerte keinen nennenswerten Einfluss auf die Laufzeit des Algorithmus. Dies ermöglicht es eine zweite Version der Low-Rank ADI Iteration zu formulieren, wobei hier der Algorithmus im i-ten Schritt terminiert sofern für das relative Lyapunovresiduum gilt

$$\frac{||\mathcal{R}_i||}{||\mathcal{R}_0||} \leq \epsilon,$$

wobei $0 < \epsilon \ll 1$. Wenn $X_0 = 0$ und Gleichung (3.8) betrachtet wird ist dies äquivalent zu

$$\frac{||W_i^* W_i||}{||B^T B||} \leq \epsilon.$$

Algorithmus 2 – LR ADI Iteration Version 2

Input: $A, E \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $\{\alpha_1, \dots \alpha_s\} \subset \mathbb{C}_-$ (siehe Abschnitt 3.4)
Output: $Z \in \mathbb{C}^{n \times sm}$ (unter der Annahme, dass genau s Iterationen ausgeführt werden)

1: $Z_0 = []$, $W_0 = B$, $i = 1$
2: **while** $||W_i^* W_i|| > \epsilon ||B^T B||$ **do**
3: **solve** $(A + \alpha_i E)V_i = W_{i-1}$ **for** V_i
4: $W_i = W_{i-1} - 2\,\mathrm{Re}(\alpha_i) E V_i$
5: $Z_i = \left[Z_{i-1}, \sqrt{-2\,\mathrm{Re}(\alpha_i)} V_i\right]$
6: $i = i + 1$

3.3 Komplexe Arithmetik in der LR ADI Iteration

Da $\alpha_i \in \mathbb{C}_-$ wurde bisher davon ausgegangen, dass die in der Iteration auftretenden Matrizen Z_i und W_i stets komplex sind. Dieses Kapitel basiert nun auf Resultaten aus [16] welche sich damit auseinandersetzen, in welchem Fall und wie sich Z_i und W_i als reelle Matrizen darstellen lassen. Die Grundlage dieser Aussagen ist dabei, dass für die Shiftparameter $\mathcal{S} = \{\alpha_1, \dots \alpha_s\} \subset \mathbb{C}_-$ gilt, dass $\alpha_i \in \mathcal{S}$ auch $\overline{\alpha_i} \in \mathcal{S}$ impliziert. Insbesondere sollen komplexe Shiftparameter nur in der Form von komplex konjugierten Paaren $\{\alpha_i, \alpha_{i+1} = \overline{\alpha_i}\}$ direkt hintereinander auftreten.

3.3.1 Reelle Residuumsfaktoren

Im Folgenden wird angenommen, dass sich die LR ADI Iteration im k-ten Schritt befindet. Es werden nun reelle Ausdrücke für die Residuumsfaktoren unter Berücksichtigung der in der Iteration möglichen Situationen hergeleitet.

Fall 1, $\{\alpha_\mathbf{1}, \dots, \alpha_\mathbf{k}\} \subset \mathbb{R}_-$: In dem trivialen Fall dass alle Shiftparameter bis zum k-ten Schritt reell sind, gilt offensichtlich unter Berücksichtigung von (3.7) und (3.11), dass $V_k, W_k \in \mathbb{R}^{n \times m}$.

Fall 2, $\{\alpha_\mathbf{1}, \dots, \alpha_\mathbf{k-1}\} \subset \mathbb{R}_-, \{\alpha_\mathbf{k}, \alpha_\mathbf{k+1} = \overline{\alpha_\mathbf{k}}\} \subset \mathbb{C}_-$: In dieser Situation wird zunächst

die Zerlegung von Gleichung (3.11) in Imaginär- und Realteil betrachtet:

$$\begin{aligned} W_{k-1} &= (A+\alpha_k E)V_k = (A+(\operatorname{Re}(\alpha_k)+i\operatorname{Im}(\alpha_k))E)(\operatorname{Re}(V_k)+i\operatorname{Im}(V_k)) \\ &= A\operatorname{Re}(V_k)+\operatorname{Re}(\alpha_k)E\operatorname{Re}(V_k)-\operatorname{Im}(\alpha_k)E\operatorname{Im}(V_k) \\ &\quad + i\big[\operatorname{Im}(\alpha_k)E\operatorname{Re}(V_k)+A\operatorname{Im}(V_k)+\operatorname{Re}(\alpha_k)E\operatorname{Im}(V_k)\big]. \end{aligned}$$

Wie bereits im ersten Fall festgestellt, gilt

$$\begin{aligned} 0=\operatorname{Im}(W_{k-1}) &= \operatorname{Im}(\alpha_k)E\operatorname{Re}(V_k)+A\operatorname{Im}(V_k)+\operatorname{Re}(\alpha_k)E\operatorname{Im}(V_k) \\ &= (A+\overline{\alpha_k}E)\operatorname{Im}(V_k)+\operatorname{Im}(\alpha_k)EV_k \\ &= \frac{1}{\operatorname{Im}(\alpha_k)}\operatorname{Im}(V_k)+(A+\overline{\alpha_k}E)^{-1}EV_k. \end{aligned}$$

Aus dieser Umformung folgt nun direkt mit $\alpha_{k+1}=\overline{\alpha_k}$ und (3.7)

$$\begin{aligned} V_{k+1} &= V_k - 2\overline{\alpha_k}(A+\overline{\alpha_k}E)^{-1}EV_k \\ &= V_k + 2(\operatorname{Re}(\alpha_k)-i\operatorname{Im}(\alpha_k))\frac{1}{\operatorname{Im}(\alpha_k)}\operatorname{Im}(V_k) \\ &= \overline{V_k}+2\frac{\operatorname{Re}(\alpha_k)}{\operatorname{Im}(\alpha_k)}\operatorname{Im}(V_k). \end{aligned} \tag{3.12}$$

Basierend auf (3.11) und der hergeleiteten Darstellung für V_{k+1} ergibt sich der Ausdruck

$$\begin{aligned} W_{k+1} &= W_k - 2\operatorname{Re}(\alpha_k)EV_{k+1} \\ &= W_{k-1} - 2\operatorname{Re}(\alpha_k)EV_k - 2\operatorname{Re}(\alpha_k)EV_{k+1} \\ &= W_{k-1} - 2\operatorname{Re}(\alpha_k)E\big[V_k+\overline{V_k}+2\frac{\operatorname{Re}(\alpha_k)}{\operatorname{Im}(\alpha_k)}\operatorname{Im}(V_k)\big] \\ &= W_{k-1} - 4\operatorname{Re}(\alpha_k)E\big[\operatorname{Re}(V_k)+\frac{\operatorname{Re}(\alpha_k)}{\operatorname{Im}(\alpha_k)}\operatorname{Im}(V_k)\big]. \end{aligned} \tag{3.13}$$

Offensichtlich handelt es sich bei W_{k+1} wieder um eine reelle Matrix.

Fall 3, $\{\alpha_{\mathbf{1}},\dots,\alpha_{\mathbf{k-3}}\}\subset\mathbb{R}_-, \{\alpha_{\mathbf{k-2}},\alpha_{\mathbf{k-1}}=\overline{\alpha_{\mathbf{k-2}}}\}\subset\mathbb{C}_-, \{\alpha_{\mathbf{k}},\alpha_{\mathbf{k+1}}=\overline{\alpha_{\mathbf{k}}}\}\subset\mathbb{C}_-$**:** Im zweiten Fall wurde gezeigt, dass das hier betrachtete W_{k-1} reell ist. Entsprechend lässt sich wieder Gleichung (3.13) herleiten, sodass erneut gilt $W_{k+1}\in\mathbb{R}^{n\times m}$.

Fall 4, $\{\alpha_{\mathbf{1}},\dots,\alpha_{\mathbf{k-3}}\}\subset\mathbb{R}_-, \{\alpha_{\mathbf{k-2}},\alpha_{\mathbf{k-1}}=\overline{\alpha_{\mathbf{k-2}}}\}\subset\mathbb{C}_-, \{\alpha_{\mathbf{k}}\}\subset\mathbb{R}_-$**:** Es wird wieder (3.11) und die entsprechende Zerlegung in Imaginär- und Realteil betrachtet

$$W_{k-1}=(A+\alpha_k E)V_k=(A+\alpha_k E)\operatorname{Re}(V_k)+i(A+\alpha_k E)\operatorname{Im}(V_k).$$

In Fall 2 wurde bereits gezeigt, dass $\operatorname{Im}(W_{k-1})=0$. Dementsprechend gilt auch $\operatorname{Im}(V_k)=0$, wodurch sich analog zum ersten Fall ergibt, dass

$$W_k = W_{k-1} - 2\operatorname{Re}(\alpha_k)EV_k$$

eine reelle Matrix ist.

Ein wichtiges Resultat dieser Fallunterscheidung ist, dass reelle Shiftparameter stets in reellen Residuumsfaktoren resultieren. Des Weiteren lässt sich bei komplexen Shiftparameterpaaren $\{\alpha_k, \alpha_{k+1}\}$ direkt der reelle Residuumsfaktor W_{k+1} bestimmen, wobei W_k in diesem Fall komplex sein kann. Dieser Zusammenhang motiviert dazu bei dem Auftreten von komplexen Shiftparametern einen doppelten Iterationsschritt auszuführen. Entsprechend widmet sich der nächste Abschnitt einer Formulierung reeller Lösungsfaktoren Z_{k+1} im k-ten Schritt des Algorithmus.

3.3.2 Reelle Lösungsfaktoren

Angenommen die LR ADI Iteration befindet sich im k-ten Schritt mit $\{\alpha_k, \alpha_{k+1} = \overline{\alpha_k}\} \subset \mathbb{C}_-$ und die bisherigen Lösungsfaktoren $Z_1, \ldots, Z_{k-1}$ haben ausschließlich reelle Einträge. Es wird zunächst der aus Gleichung (3.6) resultierende Ausdruck

$$Z_{k+1} = \big[Z_{k-1},\ \sqrt{-2\operatorname{Re}(\alpha_k)}V_k,\ \sqrt{-2\operatorname{Re}(\alpha_{k+1})}V_{k+1}\big]$$

betrachtet. Daraus folgt mit der im Folgenden verwendeten Notation $\delta_k := \frac{\operatorname{Re}(\alpha_k)}{\operatorname{Im}(\alpha_k)}$, der Tatsache dass $V_k\overline{V_k}^T + \overline{V_k}V_k^T = 2\operatorname{Re}(V_k)\operatorname{Re}(V_k)^T + 2\operatorname{Im}(V_k)\operatorname{Im}(V_k)^T$ und (3.12):

$$\begin{aligned}
X_{k+1} = Z_{k+1}Z_{k+1}^* &= Z_{k-1}Z_{k-1}^T - 2\operatorname{Re}(\alpha_k)\big[V_k\overline{V}_k^T + V_{k+1}\overline{V}_{k+1}^T\big] \\
&= Z_{k-1}Z_{k-1}^T - 4\operatorname{Re}(\alpha_k)\big[\operatorname{Re}(V_k)\operatorname{Re}(V_k)^T + \operatorname{Im}(V_k)\operatorname{Im}(V_k)^T + \delta_k\operatorname{Im}(V_k)V_k^T \\
&\quad + \delta_k\overline{V_k}\operatorname{Im}(V_k)^T + 2\delta_k^2\operatorname{Im}(V_k)\operatorname{Im}(V_k)^T\big] \\
&= Z_{k-1}Z_{k-1}^T - 4\operatorname{Re}(\alpha_k)\big[(\operatorname{Re}(V_k) + \delta_k\operatorname{Im}(V_k))(\operatorname{Re}(V_k) + \delta_k\operatorname{Im}(V_k))^T \\
&\quad + (\delta_k^2+1)\operatorname{Im}(V_k)\operatorname{Im}(V_k)^T\big].
\end{aligned}$$

Diese Darstellung von X_{k+1} liefert einen neuen reellen Ausdruck für die Lösungsfaktoren

$$Z_{k+1} = \big[Z_{k-1},\ 2\sqrt{-\operatorname{Re}(\alpha_k)}(\operatorname{Re}(V_k) + \delta_k\operatorname{Im}(V_k)),\ 2\sqrt{-\operatorname{Re}(\alpha_k)(\delta_k^2+1)}\operatorname{Im}(V_k)\big].$$

Basierend auf den Herleitungen der reellen Residuums- und Lösungsfaktoren kann die finale Version der LR ADI Iteration formuliert werden. Hierbei sollte beachtet werden, dass die reellen Darstellungen zwei wesentliche Vorteile mit sich bringen. Zum Einen wird für das Speichern reeller Matrizen etwa halb so viel Speicherplatz benötigt wie für das Speichern der komplexen Faktoren. Des Weiteren muss bei dem Auftreten eines komplexen Shiftpaares nur eines anstatt zweier großer linearer Gleichungssysteme gelöst werden, welche den Hauptbestandteil des Rechenaufwands darstellen.

Algorithmus 3 – LR ADI Iteration Version 3

Input: $A, E \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $\{\alpha_1, \dots \alpha_s\} \subset \mathbb{C}_-$ (siehe Abschnitt 3.4)
Output: $Z \in \mathbb{C}^{n\times sm}$ (unter der Annahme, dass genau s Iterationen ausgeführt werden)

1: $Z_0 = []$, $W_0 = B$, $i = 1$
2: **while** $||W_i^T W_i|| > \epsilon ||B^T B||$ **do**
3: **solve** $(A + \alpha_i E)V_i = W_{i-1}$ **for** V_i
4: **if** $\mathrm{Im}(\alpha_i) = 0$ **then**
5: $W_i = W_{i-1} - 2\,\mathrm{Re}(\alpha_i) E V_i$
6: $Z_i = \left[Z_{i-1}, \sqrt{-2\,\mathrm{Re}(\alpha_i)} V_i\right]$
7: $i = i + 1$
8: **else**
9: $\delta_i = \frac{\mathrm{Re}(\alpha_i)}{\mathrm{Im}(\alpha_i)}$, $\beta_i = 2\sqrt{-\mathrm{Re}(\alpha_i)}$
10: $W_{i+1} = W_{i-1} - \beta_i^2 E\left[\mathrm{Re}(V_i) + \delta_i \,\mathrm{Im}(V_i)\right]$
11: $Z_{i+1} = \left[Z_{i-1},\ \beta_i(\mathrm{Re}(V_i) + \delta_i \,\mathrm{Im}(V_i)),\ \beta_i\sqrt{(\delta_i^2 + 1)}\,\mathrm{Im}(V_i)\right]$
12: $i = i + 2$

3.4 Wahl der Shiftparameter und Konvergenz der Iteration

Es stellt sich nach wie vor die Frage, ob der angegebene Algorithmus überhaupt terminiert. Es wird zunächst die Norm von Gleichung (3.10) betrachtet, wodurch sich mit $T_k = (A + \alpha_k E)^{-1}(A - \overline{\alpha_k} E)$ die folgende Abschätzung ergibt

$$\frac{||X_i - X||}{||X_0 - X||} \leq ||\prod_{k=1}^{i} T_k||^2. \tag{3.14}$$

Offensichtlich hängt die Konvergenz der LR ADI Iteration maßgeblich von $||\prod_{k=1}^{i} T_k||$, also entsprechend vom Spektralradius $\rho(\prod_{k=1}^{i} T_k)$ ab. Als nächstes wird das verallgemeinerte Eigenwertproblem bezüglich des Matrixpaars (A, E) betrachtet, welches mit $D = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$ die Zerlegung

$$A = EVDV^{-1}$$

liefert, wobei V die Rechtseigenvektoren des Matrixpaars enthält. Daraus folgt der Ausdruck

$$\begin{aligned} T_k &= (A + \alpha_k E)^{-1}(A - \overline{\alpha_k} E) \\ &= \left[E(VDV^{-1} + \alpha_k I)\right]^{-1}\left[E(VDV^{-1} - \overline{\alpha_k} I)\right] \\ &= V\,\mathrm{diag}\left(\frac{\lambda_1 - \overline{\alpha_k}}{\lambda_1 + \alpha_k}, \dots, \frac{\lambda_n - \overline{\alpha_k}}{\lambda_n + \alpha_k}\right) V^{-1}. \end{aligned} \tag{3.15}$$

Aus der Tatsache, dass $\left|\frac{\lambda - \overline{\alpha}}{\lambda + \alpha}\right|^2 = \frac{1 - 4\,\mathrm{Re}(\alpha)\,\mathrm{Re}(\lambda)}{|\lambda + \alpha|^2}$ ergibt sich dann direkt der Zusammenhang, dass aus $\alpha_k \in \mathbb{C}_-$ umgehend $\rho(T_k) < 1$ folgt. Wie beispielsweise in [6, 8] näher erläutert folgt entsprechend eine superlineare Konvergenz der LR ADI Iteration.

Insbesondere kann eine schnelle Konvergenz des Algorithmus durch die Wahl von Shiftparametern $\{\alpha_1, \dots \alpha_i\} \subset \mathbb{C}_-$, welche $\rho(\prod_{k=1}^{i} T_k)$ minimieren, erzwungen wer-

den. Durch das Betrachten von (3.14) und (3.15) ergibt sich die folgende Abschätzung für optimale Shiftparameter:

$$\min_{\{\alpha_1,\ldots,\alpha_i\}\subset\mathbb{C}_-} ||\prod_{k=1}^{i} T_k||^2 \leq \kappa(V)^2 \min_{\{\alpha_1,\ldots,\alpha_i\}\subset\mathbb{C}_-} \left(\max_{\lambda\in\Lambda(A,E)} \left| \prod_{k=1}^{i} \frac{\lambda-\overline{\alpha_k}}{\lambda+\alpha_k} \right|^2 \right).$$

Obwohl hierbei das in der verallgemeinerten Lyapunovgleichung (2.2) auftretende B und sein niedriger Rang nicht berücksichtigt werden, können Approximationen der α_j aus dem als *ADI Shiftparameter Problem* [17] bekannten Optimierungsproblem

$$\{\alpha_1,\ldots,\alpha_i\} = \operatorname*{arg\,min}_{\{\mu_1,\ldots,\mu_i\}\subset\mathbb{C}_-} \left(\max_{\lambda\in\Lambda(A,E)} \left| \prod_{k=1}^{i} \frac{\lambda-\overline{\mu_k}}{\lambda+\mu_k} \right|^2 \right) \tag{3.16}$$

brauchbare Shifts liefern. Entsprechend beschäftigt sich das nächste Unterkapitel kurz mit Verfahren, welche basierend auf obigem Optimierungsproblem Shiftparameter generieren.

3.4.1 Vorberechnete Shifts

Ein weitverbreiteter Ansatz stellt dabei das Konstruieren eines Gebiets $\Omega \subset \mathbb{C}_-$ mit $\Lambda(A,E) \subset \Omega$, über welchem dann (3.16) gelöst wird, dar [8, 6]. Ein in [17] vorgestelltes Verfahren basiert beispielsweise darauf das Spektrum von (A,E) in den Definitionsbereich elliptischer Funktionen einzubetten. Dabei sind die einzigen vom Spektrum abhängigen Parameter

$$a := \min_{\lambda\in\Lambda(A,E)} \mathrm{Re}(\lambda), \quad b := \max_{\lambda\in\Lambda(A,E)} \mathrm{Re}(\lambda), \quad \text{und} \quad \gamma := \max_{\lambda\in\Lambda(A,E)} \arctan\left(\left| \frac{\mathrm{Im}(\lambda)}{\mathrm{Re}(\lambda)} \right| \right),$$

welche sich beispielsweise über die Ritzwerte von $E^{-1}A$ und $A^{-1}E$ aus dem Arnoldi- beziehnungsweise Lanczos-Prozess approximieren lassen [6].
Eine heuristische Vorgehensweise, welche nicht auf dem Bestimmen eines Ω basiert, wird unter anderem in [12] thematisiert. Zunächst wird dabei $\Lambda(A,E)$ über $R := R_+ \cup \frac{1}{R_-}$ approximiert. Dabei ist R_+ die Menge von k_+ Ritzwerten der Matrix $E^{-1}A$ und R_- die Menge von k_- Ritzwerten der Matrix $A^{-1}E$. Anschließend wird eine Teilmenge $\mathcal{S} \subset R$ festgelegter Größe l, welche (3.16) über die in Algorithmus 4 vorgestellte Heuristik approximativ löst, ausgewählt.

Algorithmus 4 – Heuristische Penzl Shifts

1: Finde $p \in R$ mit $\max_{t\in R} \left|\frac{t-p}{t+p}\right| = \min_{r\in R} \max_{s\in R} \left|\frac{r-s}{r+s}\right|$
2: **if** Im(p) $= 0$ **then** $\mathcal{S} = \{p\}$ **else** $\mathcal{S} = \{p, \overline{p}\}$
3: **while** $|\mathcal{S}| < l$ **do**
4: Finde $p \in R$ mit $\left|\prod_{s\in\mathcal{S}} \frac{s-p}{s+p}\right| = \max_{r\in R} \left|\prod_{s\in\mathcal{S}} \frac{r-s}{r+s}\right|$
5: **if** Im(p) $= 0$ **then** $\mathcal{S} = \mathcal{S} \cup \{p\}$ **else** $\mathcal{S} = \mathcal{S} \cup \{p, \overline{p}\}$

Sofern die LR ADI Iteration nach dem Verwenden aller so generierten Shifts nicht

konvergiert ist, können die selben Shiftparameter erneut benutzt werden.

3.4.2 Im Iterationsverlauf berechnete Shifts

Die bisher vorgestellten Verfahren für die Ermittlung von Shiftparametern finden im Regelfall vor der eigentlichen Iteration statt. Es wird nun eine Variante aus [18, 6, 19] betrachtet in welcher die Shifts im Verlauf des Algorithmus über *Galerkin Projektionen* der Matrizen A und E berechnet werden. Dabei wird zunächst die Menge der ersten Shiftparameter $\{\alpha_1, \ldots, \alpha_{k_1}\} = \Lambda(\tilde{B}^T A\tilde{B}, \tilde{B}^T E\tilde{B}) \cap \mathbb{C}_-$ initialisiert, wobei die Spalten von $\tilde{B}$ eine orthonormale Basis für span$\{B\}$ darstellen. Da für die Matrix $B \in \mathbb{R}^{n\times m}$ aus der Gleichung (2.2) gilt $m \ll n$, handelt es sich bei den Projektionen[1] $\tilde{B}^T A\tilde{B}$ und $\tilde{B}^T E\tilde{B}$ um kleine quadratische Matrizen, deren Eigenwerte sich mit geringem Aufwand bestimmen lassen. Auch die Orthonormalisierung von B ist wegen $m \ll n$ in einem angemessenen zeitlichen Rahmen durchführbar. Der Schnitt des Spektrums $\Lambda(\tilde{B}^T A\tilde{B}, \tilde{B}^T E\tilde{B})$ mit $\mathbb{C}_-$ garantiert dabei, dass alle Shiftparameter zulässig sind. Falls es zu dem unwahrscheinlichen Fall kommen sollte, dass $\Lambda(\tilde{B}^T A\tilde{B}, \tilde{B}^T E\tilde{B}) \cap \mathbb{C}_- = \emptyset$, kann anstelle von B solange eine zufällige Matrix Q für die Projektion benutzt werden, bis $\Lambda(Q^T AQ, Q^T EQ) \cap \mathbb{C}_-$ ausreichend viele Shifts liefert.
Nach k_1 Schritten der Iteration werden neue Shiftparameter benötigt. Hierbei stehen verschiedene Möglichkeiten zur Auswahl:

1. Nutze $\{\alpha_{k_1+1}, \ldots, \alpha_{k_2}\} = \Lambda(\tilde{W}_{k_1}^T A\tilde{W}_{k_1}, \tilde{W}_{k_1}^T E\tilde{W}_{k_1}) \cap \mathbb{C}_-$, mit den Spalten von $\tilde{W}_{k_1}$ als orthonormale Basis für span$\{W_{k_1}\}$.

2. Nutze $\{\alpha_{k_1+1}, \ldots, \alpha_{k_2}\} = \Lambda(\tilde{V}_{k_1}^T A\tilde{V}_{k_1}, \tilde{V}_{k_1}^T E\tilde{V}_{k_1}) \cap \mathbb{C}_-$, mit den Spalten von $\tilde{V}_{k_1}$ als orthonormale Basis für span$\{V_{k_1}\}$.

3. Nutze $\{\alpha_{k_1+1}, \ldots, \alpha_{k_2}\} = \Lambda(\tilde{V}_{k_1}(u)^T A\tilde{V}_{k_1}(u), \tilde{V}_{k_1}(u)^T E\tilde{V}_{k_1}(u)) \cap \mathbb{C}_-$, mit den Spalten von $\tilde{V}_{k_1}(u)$ als orthonormale Basis für span$\{V_{k_1}(u)\}$ und $V_{k_1}(u) := [V_1, \ldots, V_{k_1}]$.

Falls hierbei keine Eigenwerte in $\mathbb{C}_-$ liegen, können die vorherigen Shiftparameter nochmal verwendet werden. Außerdem hat es sich als sinnvoll herausgestellt, dass sofern V_{k_1} eine komplexe Matrix ist, die Spalten von $\tilde{V}_{k_1}$ als orthonormale Basis für span$\{\mathrm{Re}(V_{k_1}), \mathrm{Im}(V_{k_1})\}$ gewählt werden. Bei der dritten Möglichkeit setzt sich $V_{k_1}(u)$ aus den letzten u in der Iteration berechneten Matrizen V_i aus den Gleichungen (3.7) zusammen. Dementsprechend können bei dem Bilden des Unterraums span$\{V_{k_1}(u)\}$ die letzten uk_1 Spalten der Lösung Z_{k_1} benutzt werden. Dabei sollte beachtet werden, dass falls $u \geq k_1$ entsprechend auch $V_{k_1}(u) = [V_1, \ldots, V_{k_1}]$ gilt. In dem Fall, dass $\alpha_{k_1-u} = \overline{\alpha_{k_1-u+1}}$ ist es jedoch sinnvoll die letzten $(u+1)k_1$ Spalten von Z_{k_1} zu betrachten.

Neben den hier vorgestellten Verfahren existiert noch eine Vielzahl weiterer Möglichkeiten, um geeignete Shiftparameter zu generieren. Es stellt sich somit die Frage, welche der Varianten zu bevorzugen ist. Numerische Experimente in [6, 18] haben dabei gezeigt, dass die effektivste Wahl der Shiftparameter stets von dem gegebenen Problem abhängt. Eine konsistent performante Methode war dabei die Shifts zu benutzen,

[1] Projektionen im Sinne, dass die Koeffizientenmatrizen A und E auf den durch *span*$\{\hat{B}\}$ definierten Unterraum projiziert werden

welche sich über die Galerkin Projektionen der V_i bestimmen lassen. Darüber hinaus funktionieren die im Iterationsverlauf berechneten Shifts voll-automatisch, verlangen also nicht, dass der Nutzer wie beispielsweise bei den heuristischen Shifts, eine Vielzahl von relevanten Parametern (k_-, k_+, l) angeben muss. Entsprechend wurde auch diese Methode für die Implementierung der LR ADI Iteration in `pyMOR` gewählt.

4 Software und Implementierung

Basierend auf den mathematischen Grundlagen der letzten Kapitel kann eine vollständige Version der LR ADI Iteration implementiert werden. Im Kontrast zu den bisher diskutierten ausschließlich theoretischen Aspekten des Algorithmus, widmet sich dieses Kapitel der praktischen Umsetzung innerhalb des durch die `pyMOR`-Bibliothek vorgegebenen Rahmens.

4.1 pyMOR

Wie bereits erwähnt, ist ein Ziel dieser Arbeit die LR ADI Iteration für die Softwarebibliothek `pyMOR` zu implementieren. Der ursprüngliche Fokus dieser Bibliothek war dabei die Anwendung reduzierter Basismethoden auf parametrisierte partielle Differentialgleichungen, wobei dieser derzeit vor allem um systemtheoretische Modellreduktionsmethoden erweitert wird. Bei ersteren Reduktionsverfahren lässt sich feststellen, dass alle hochdimensionalen Operationen auf den aus ihnen resultierenden Komponenten durch einige wenige Operationen auf Vektoren und Operatoren realisierbar sind. Daraus hat sich auch das Designparadigma von `pyMOR` ergeben, welches sich aus zwei Aspekten zusammensetzt. Zum Einen sollen für alle in den Modellreduktionsmethoden auftretenden mathematischen Objekte (zum Beispiel Operatoren, Vektoren, Diskretisierungen) plattformunabhängige Schnittstellen bereitgestellt werden. Zum Anderen soll bei allen implementierten Algorithmen innerhalb `pyMOR`s möglichst ausschließlich auf diese abstrakten Schnittstellen zurückgegriffen werden.

Diese Vorgehensweise bietet im Wesentlichen zwei maßgebliche Vorteile. Ersterer ist dabei, dass eine flexible und einfache Einbettung externer Software zum Lösen partieller Differentialgleichungen ermöglicht wird. Für die Integration müssen nur einige wenige Schnittstellen ergänzt werden, wobei anschließend alle Modellreduktionsmethoden ohne weiteren Aufwand zur Verfügung stehen. Bisher werden beispielsweise die externen Programmbibliotheken `FEniCS`[20], `DEAL.II` [21], `DUNE` [22] und `NGSolve`[23] unterstützt. Dem vorgestellten Designparadigma entsprechend existieren für die aus diesen Programmen stammenden Objekte abstrakte Schnittstellen innerhalb von `pyMOR`, wie beispielsweise das `OperatorInterface` mit den Methoden `apply` zum Berechnen von Matrixvektorprodukten oder `apply_inverse` welches genutzt werden kann um lineare Gleichungssysteme zu lösen. Damit ergibt sich auch der zweite Vorteil, welcher sich auf die Parallelität der Berechnungen innerhalb `pyMOR`s bezieht. Beispielsweise ist die `apply_inverse`-Methode so konzipiert, dass das lineare Gleichungssystem im Regelfall nicht über eine `pyMOR`-Implementierung sondern über eine vom Ursprung des Operators abhängige externe Implementierung gelöst wird. In diesem Sinne wird die Verantwortung für eine parallele und effiziente Berechnung auf die externe Bibliothek übertragen. Darüber hinaus existieren auch Schnittstellen für Operatoren innerhalb `pyMOR`s, welche speziell auf das verteilt parallele Rechnen ausgelegt sind. Dies bedeutet, dass aufwendige Berechnungen potentiell auf mehrere Prozessoren verteilt werden können.

4.2 Aspekte einer Implementierung in pyMOR

Entsprechend des im letzten Unterkapitel vorgestellten Designparadigmas, ist ein zentraler Bestandteil jeder Implementierung innerhalb `pyMOR`s die Nutzung und Bewahrung des hohen Abstraktionsgrads. In diesem Sinne werden auch alle potentiell hochdimensionalen Operationen innerhalb der Implementierung der LR ADI Iteration auf den Schnittstellen `OperatorInterface` und `VectorArrayInterface` ausgeführt, wobei Erstere vor allem als Ausdruck für die Matrizen A, E und B fungiert und Letztere als Darstellung der V_i, W_i und Z_i aus Algorithmus 3 dient. Eine weitere in diesem Sinne entwickelte Implementierung in `pyMOR` stellt das *Gram-Schmidt-Orthonormalisierungsverfahren* dar. Auf dieses wird beispielsweise bei der Ermittlung einer orthonormalen Basis zurückgegriffen, welche bei der Generierung der Shiftparamter benötigt wird.

Neben den Operationen auf den abstrakten Schnittstellen, existieren auch einige Verfahren für welche keine `pyMOR`-Implementierung zur Verfügung steht. Beispiele dafür sind das (verallgemeinerte) Eigenwertproblem, welches einen wichtigen Bestandteil bei der Bestimmung von Shiftparametern darstellt und das Berechnen der Spektralnorm einer Matrix, welche eine Rolle beim Ermitteln des Abbruchkriteriums der Iteration spielt. Wie sich leicht feststellen lässt, sind die Dimensionen der auftretenden Probleme relativ niedrig, da sowohl das Residuum $W_i^T W_i$ als auch die Projektionen $V_i^T A V_i$ und $V_i^T E V_i$ aus Abschnitt 3.4 kleine Matrizen darstellen. Dementsprechend ist auch ein bezüglich des `pyMOR` Designparadigmas konsistentes Vorgehen die abstrakte Ummantelung dieser niedrigdimensionalen Objekte abzulegen und andere Methoden welche in `Python` zur Verfügung stehen auf diese anzuwenden. Insbesondere wird hierbei auf die beiden populären `Python`-Bibliotheken `NumPy` [24] und `SciPy` [25] zurückgegriffen, welche leistungsfähige Implementierungen für eine Vielzahl von numerischen Algorithmen bereitstellen. Für die beiden relevanten Probleme stehen beispielsweise `np.linalg.norm` zur Berechnung der Spektralnorm und `scipy.linalg.eigvals` für Eigenwertprobleme zur Verfügung.

Abgesehen von den bisher diskutieren Aspekten welche vor allem die Software und das Design betrafen, existieren noch weitere für die praktische Implementierung relevante Gesichtspunkte. Beispielsweise ist es gegebenenfalls notwendig die in Gleichung (1.2) spezifizierte duale verallgemeinerte Lyapunovgleichung

$$A^T X E + E^T X A + C^T C = 0$$

zu lösen. Im Wesentlichen muss dafür Algorithmus 3 nur leicht modifiziert werden, sodass beim Lösen der linearen Gleichungssysteme die Koeffizientenmatrix, sowie bei der Multiplikation zweier Matrizen der linke Faktor transponiert wird. Entsprechende Methoden werden auch von `pyMOR` mit `apply_inverse_transpose` und `apply_transpose` zur Verfügung gestellt. Des Weiteren sollte auch die Standard Lyapunovgleichung (2.1) effizient gelöst werden können. Dafür kann beispielsweise der `IdentityOperator` welcher als Einheitsmatrix fungiert benutzt werden, indem in Algorithmus 3 die Matrix E durch ebendiesen ersetzt wird. Sollte der `IdentityOperator` auf ein Element angewendet werden, wie beispielsweise in den Zeilen 5 und 10 von Algorithmus 3, so wird ohne jegliche Berechnungen auszuführen ebendieses zurückgegeben.

4.3 C-M.E.S.S. und Py-M.E.S.S.

M.E.S.S. [2] ist der Nachfolger der Lyapack-Toolbox für MATLAB, welche zum Lösen großer dünnbesetzter Matrixgleichungen entwickelt wurde. Eine in der Programmiersprache C implementierte Version der M.E.S.S.-Bibliothek stellt dabei C-M.E.S.S. dar, wobei diese insbesondere auch eine Implementierung der LR ADI Iteration für Lyapunovgleichungen bereitstellt. Die Programmiersprache Python hat nun die nützliche Eigenschaft, dass einfach zu nutzende Schnittstellen für C-Programmcode existieren. Py-M.E.S.S. nutzt genau diese Schnittstellen aus und stellt ein Bindeglied zwischen der C-M.E.S.S.-Bibliothek und Python dar. Dementsprechend besteht neben einer direkten Implementierung des LR ADI Algorithmus in pyMOR auch die Möglichkeit die Lyapunovgleichung mit der über die durch Py-M.E.S.S. erreichbare Implementierung innerhalb der C-M.E.S.S.-Bibliothek zu lösen. Hierfür ist bereits eine Schnittstelle in pyMOR implementiert worden, welche die Matrizen der betrachteten Lyapunovgleichung, aber auch Referenzen auf Funktionen für das Berechnen von Matrixvektorprodukten oder das Lösen von linearen Gleichungssystemen an py-M.E.S.S. übergibt. Auf diese Weise kann der in der C-M.E.S.S. Bibliothek spezifizierte Algorithmus innerhalb pyMORs verwendet werden, wobei weiterhin die Designvorgaben pyMORs eingehalten werden. Das Transferieren der Daten von pyMOR nach C erfordert dabei beispielsweise das Kopieren von Matrizen und Vektoren in jedem Schritt der Iteration, welches bei großen Dimensionen potentiell mit erhöhtem zeitlichen Aufwand verbunden ist. Das nächste Kapitel soll dementsprechend unter Betrachtung diverser numerischer Tests Klarheit darüber verschaffen, welche der Umsetzungen in pyMOR eine bessere Performance erzielen kann.

5 Numerische Vergleiche

Wie bereits angedeutet hängen sowohl `pyMOR` als auch `C-M.E.S.S.` von externen Softwarekomponenten ab. Beispiele dafür sind die `BLAS` und `LAPACK` Bibliotheken, welche optimierte Routinen für das Lösen von Problemen aus der linearen Algebra zur Verfügung stellen. Um die Reproduzierbarkeit und Qualität der im Folgenden durchgeführten numerischen Experimente sicherzustellen, muss gewährleistet werden, dass bei allen Berechnungen die Versionen der benutzten Softwarepakete übereinstimmen. Für die in diesem Kapitel durchgeführten Versuche wird stets die in Tabelle 1 dargestellte Konstellation verwendet.

Software	Version
Python	3.6
pyMOR	0.5
C-/Py-M.E.S.S.	1.0.0
BLAS	OpenBLAS 0.2.18
LAPACK	3.6
SuiteSparse	4.4.1
NumPy	1.15.2
SciPy	1.1.0
Betriebssystem	Ubuntu 16.04

Tabelle 1: Software und Versionen

Neben der Software spielen auch die verwendeten Hardwarekomponenten eine zentrale Rolle bei den letztendlichen Resultaten der Tests. Entsprechend wird hierfür die in Tabelle 2 spezifizierte Konstellation von Hardwarekomponenten verwendet, welche für alle Berechnungen eingehalten wird.

Komponente	Typ
Systemmodell	HP 250 G6 Notebook PC
Prozessor	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, 2701 MHz, 2 Kerne, 4 logische Prozessoren
Arbeitsspeicher	8.00 GB

Tabelle 2: Hardwarekomponenten

Des Weiteren wurden die im Folgenden betrachteten Experimente jeweils fünf mal ausgeführt, wobei stets das Ergebnis mit der kürzesten Laufzeit in die Auswertung einbezogen wurde. Dabei stoppt der Algorithmus sobald das in Abschnitt 3.3.2 spezifizierte relative Lyapunovresiduum die Grenze $\epsilon = 10^{-10}$ unterschreitet. Darüber hinaus sollte erwähnt werden, dass sowohl die `pyMOR` als auch die `C-M.E.S.S.` Implementierung das in Abschnitt 3.4 vorgestellte Verfahren zur Generierung von Shiftparametern über Projektionen der Matrix V_i nutzen und somit auch bei allen betrachteten Experimenten die Algorithmen nach der selben Anzahl von Iterationen konvergieren. Außerdem wird

im Folgenden die `py-M.E.S.S.` Implementierung nicht direkt verwendet, sondern wie bereits in Abschnitt 4.3 erwähnt, die von `pyMOR` zur Verfügung gestellte Schnittstelle zu `py-M.E.S.S.`.

5.1 Standard Lyapunovgleichungen

In diesem Abschnitt wird erneut das in Abschnitt 2.3 eingeführte Beispiel betrachtet, welches die Wärmeausbreitung auf einem 1-dimensionalen Segment modelliert. Dementsprechend ergeben sich die im Folgenden dargestellten Resultate aus dem Lösungsprozess der Standard Lyapunovgleichung (2.1). Dabei befinden sich die Messwerte zwischen $n = 2\,000$ und $n = 300\,000$, wobei die Matrizen durch die in der Datei `heat.py` definierte Funktion gebildet werden.

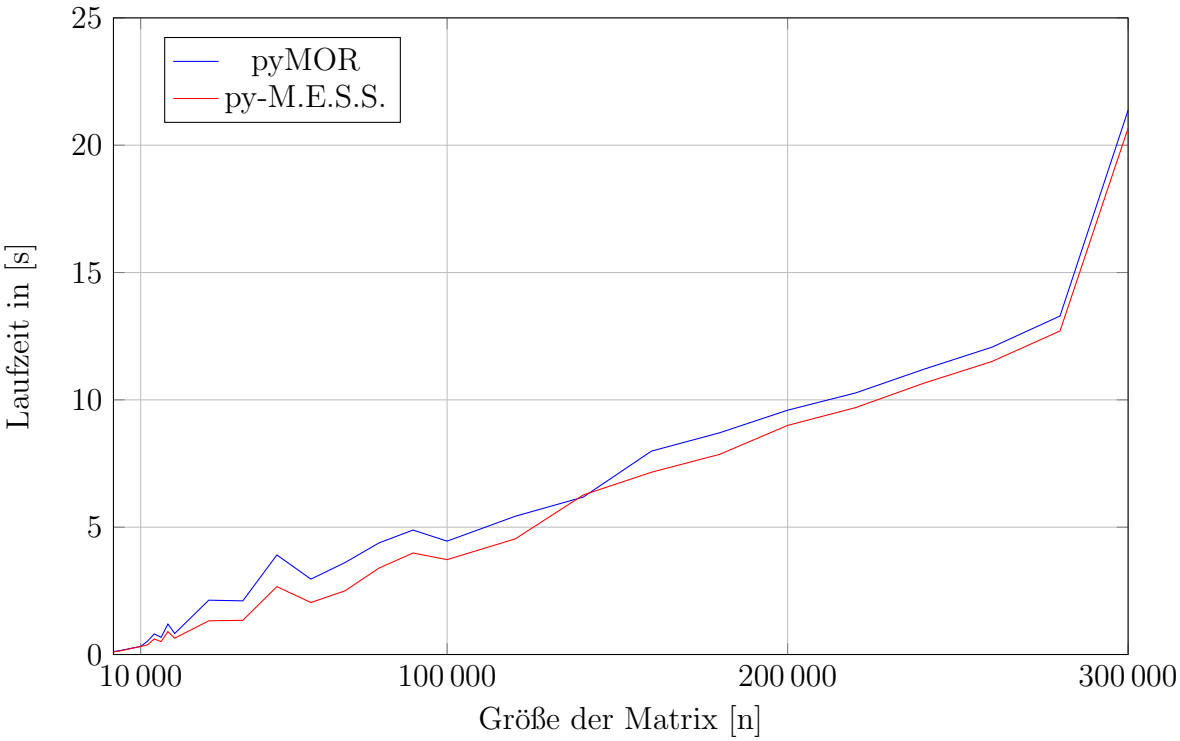

Abbildung 1: Laufzeiten bei Standard Lyapunovgleichungen

Wie sich in Abbildung 1 erkennen lässt, verhalten sich die Laufzeiten der beiden Implementierungen im Wesentlichen ähnlich. Dabei ist jedoch bis auf das Testergebnis bei $n = 140\,000$ die `py-M.E.S.S.` Variante stets überlegen, wobei sich die Laufzeiten bei $n = 50\,000$ um den maximalen Wert von 1.24 Sekunden unterscheiden. In dem einzelnen Punkt bei welchem `pyMOR` die bessere Laufzeit erzielen konnte, beträgt die Laufzeitdifferenz lediglich 0.08 Sekunden.

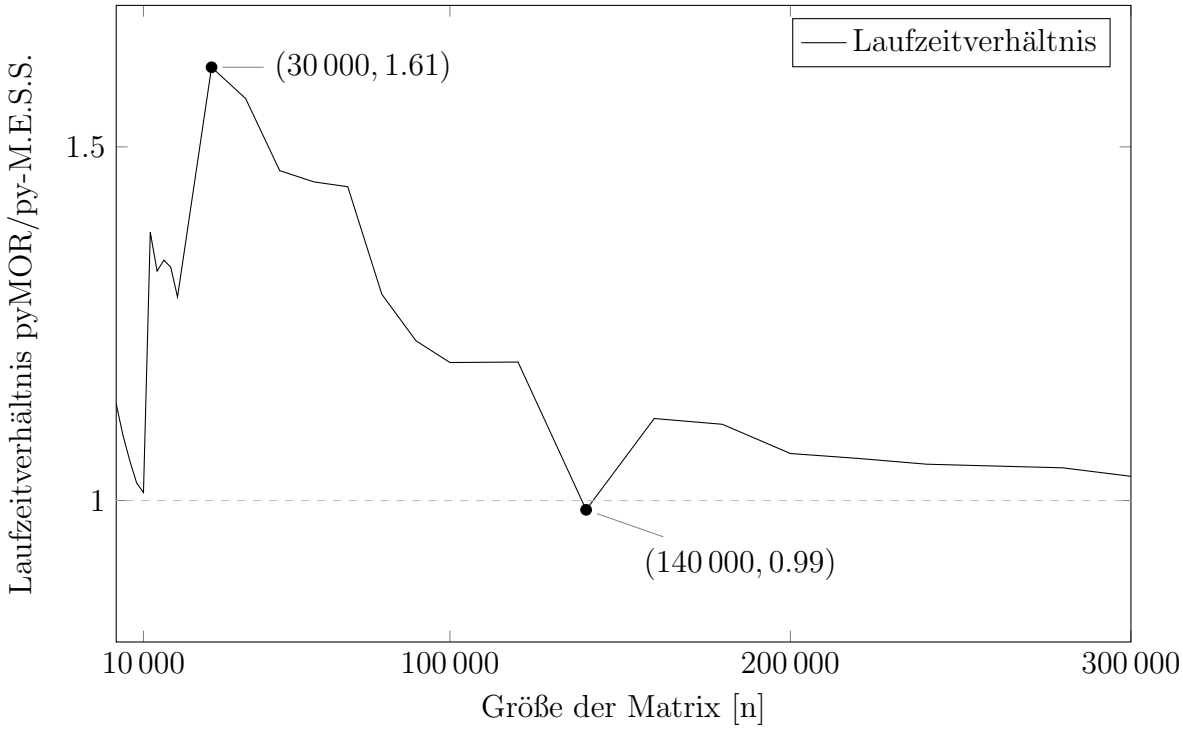

Abbildung 2: Laufzeitverhältins von pyMOR zu py-M.E.S.S. bei Standard Lyapunovgleichungen

Abgesehen von einigen kleinen Schwankungen steigt das Laufzeitverhältnis zwischen $n = 2\,000$ und $n = 30\,000$ im Wesentlichen an, wobei die `py-M.E.S.S.` Implementierung maximal 61% schneller ist als die `pyMOR` Variante. Der verhältnismäßig geringste Unterschied bezüglich der Laufzeit wird bei $n = 140\,000$ angenommen, wobei in diesem Fall `py-M.E.S.S.` 1% langsamer ist als `pyMOR`. Wird von diesem Punkt abgesehen, sinkt das Laufzeitverhältnis zwischen $n = 30\,000$ und $n = 300\,000$ konstant.

Dass `py-M.E.S.S.` bei diesem Experiment klar überlegen ist, lässt sich zum Beispiel dadurch erklären, dass die betrachteten Lyapunovgleichungen keine Matrix E beinhalten. Somit ist insgesamt der Aufwand des Kopierens der Matrizen relativ gering und `py-M.E.S.S.` kann durch die vorhandenen Optimierungen eine bessere Laufzeit erzielen. Des Weiteren ist auch das fast konstante Abnehmen des Laufzeitverhältnisses zwischen $n = 30\,000$ und $n = 300\,000$ über den Kopieraufwand erklärbar. Offensichtlich bedeutet eine größere Dimension der Matrizen auch, dass mehr Daten kopiert werden müssen, wodurch `pyMOR` bei Koeffizientenmatrizen höherer Dimension ein wenig besser dasteht als bei niedrigen, wobei selbst bei $n = 300\,000$ die `py-M.E.S.S.` Implementierung zu bevorzugen wäre.

5.2 Symmetrische verallgemeinerte Lyapunovgleichungen

Das in diesem Unterkapitel diskutierte Experiment bezieht sich auf die *Steel Profile* Benchmark aus [26, 27]. Dabei tritt bei der Modellierung von optimalen Kühlprozessen

von Stahlprofilen ein LTI System der Form

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned}$$

auf. Dementsprechend werden in diesem Beispiel verallgemeinerte Lyapunogleichungen (2.2) mit $E = E^T$, $A = A^T \in \mathbb{R}^{n\times n}$ und $B \in \mathbb{R}^{n\times 7}$ betrachtet.

n	pyMOR Laufzeit	py-M.E.S.S. Laufzeit	Laufzeitverhältnis	Iterationen
1 357	0.28	0.27	1.04	49
5 177	1.26	1.29	0.98	56
20 209	8.29	8.63	0.96	70
79 841	78.07	79.87	0.98	77

Tabelle 3: Laufzeiten in Sekunden bei symmetrischen verallgemeinerten Lyapunovgleichungen

Im Gegensatz zum ersten Experiment lässt sich umgehend erkennen, dass die `py-M.E.S.S.` Implementierung ausschließlich bei der kleinsten Matrixgröße mit $n = 1\,357$ eine schnellere Laufzeit aufweist. Bei allen anderen Testgrößen ist `pyMOR` überlegen, wobei insgesamt maximal Schwankungen von 4% in der Laufzeit erkennbar sind.

Das erhaltene Ergebnis lässt sich unter anderem mit der Tatsache erklären, dass für das Lösen der verallgemeinerten Lyapunovgleichungen die Matrix E benötigt wird, wodurch sich der Kopieraufwand für die `py-M.E.S.S.` Implementierung fast verdoppelt. Insgesamt ist jedoch der Laufzeitunterschied nicht besonders groß und die `pyMOR` Variante ist hier nur geringfügig besser.

5.3 Unsymmetrische verallgemeinerte Lypaunovgleichungen

Für das in diesem Abschnitt betrachtete Experiment wird das in [28] vorgestellte Oszillator Modell mit $3n_0 + 1$ Massen und drei Dämpfern benutzt. Dabei wird ein LTI System zweiter Ordnung zur Modellierung verwendet, welches sich im Allgemeinen durch

$$\begin{aligned} M\ddot{q}(t) + D\dot{q}(t) + Kx(t) &= Fu(t) \\ y(t) &= C_1 q(t) + C_2 \dot{q}(t) \end{aligned}$$

beschreiben lässt. Hierbei sind M, D, $K \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$, $F \in \mathbb{R}^{\tilde{n}\times\tilde{m}}$, C_1, $C_2 \in \mathbb{C}^{\tilde{p}\times\tilde{n}}$ und $x(t)$, $u(t)$, $y(t)$, $q(t) \in \mathbb{R}^{\tilde{n}}$. Solche Systeme werden zum Lösen oft in ein äquivalentes LTI System erster Ordnung umformuliert. Insbesondere ergeben sich durch die für diesen Versuch gewählte Umformulierung die folgenden Koeffizientenmatrizen für die Lyapunovgleichungen:

$$E = \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \in \mathbb{R}^{2\tilde{n}\times 2\tilde{n}}, \quad A = \begin{bmatrix} 0 & I \\ -K & -D \end{bmatrix} \in \mathbb{R}^{2\tilde{n}\times 2\tilde{n}}, \quad B = \begin{bmatrix} 0 \\ F \end{bmatrix} \in \mathbb{R}^{2\tilde{n}\times\tilde{m}}.$$

Diese Matrizen werden nun durch die Routine in der Datei `chain.py` generiert, wobei aufgrund der speziellen Struktur des Systems $n_0 \in \mathbb{N}$ mit $n = 2\tilde{n} = 6n_0 + 2$ als Eingabeparameter gewählt wurde. Hierbei sollte erwähnt werden, dass anders als in [28] spezifiziert die Matrizen F und D als

$$F = \begin{bmatrix} \mathbb{1} & 0 & 0 \\ \mathbb{1} & \mathbb{1} & 0 \\ \mathbb{1} & \mathbb{1} & \mathbb{1} \\ 1 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{(3n_0+1)\times 3} \quad \text{und} \quad D = 0.02M + 0.5K \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$$

definiert sind, wobei $\mathbb{1} \in \mathbb{R}^{n_0}$. Die Messwerte befinden sich dabei im Bereich zwischen $n = 1\,502$ und $n = 300\,002$.

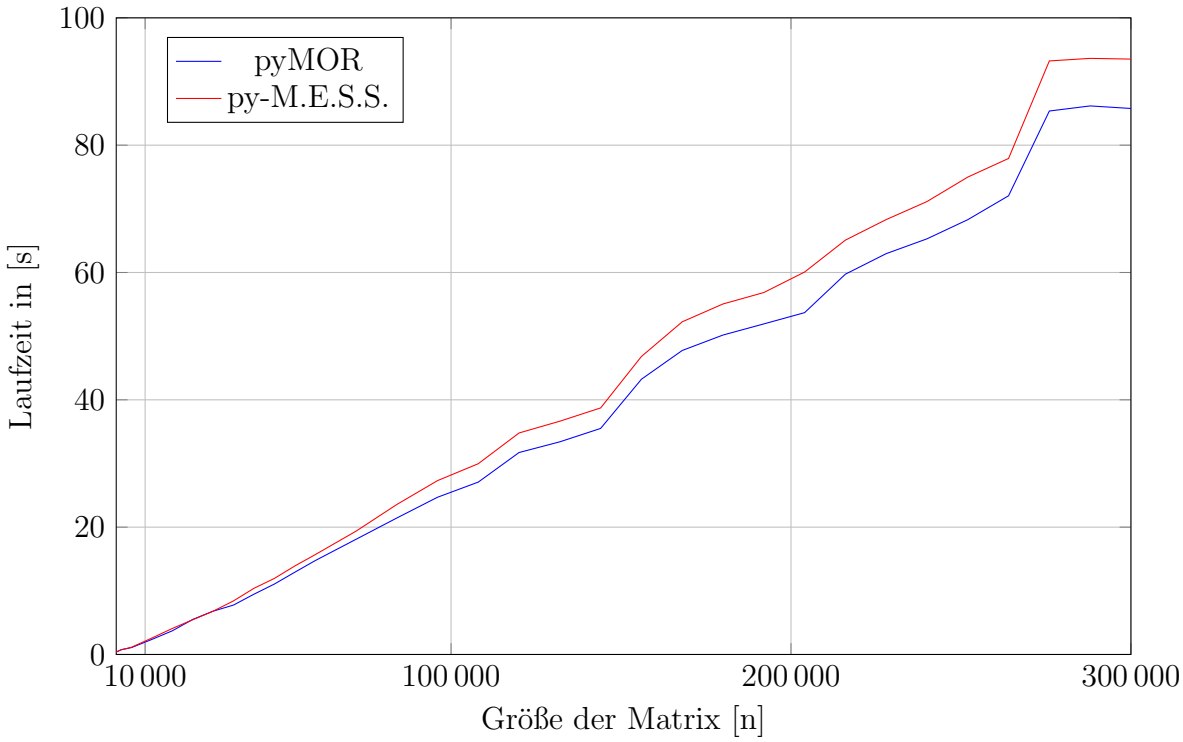

Abbildung 3: Laufzeiten bei unsymmetrischen verallgemeinerten Lyapunovgleichungen

Analog zu Abschnitt 5.1 verhalten sich die Laufzeiten der beiden Varianten ähnlich, wobei in diesem Experiment deutlich erkennbar die `pyMOR` Implementierung bei größer werdendem n besser abschneidet. Ausschließlich bei den Werten $n = 1\,502, 3\,002, 24\,002, 30\,002$ hat `py-M.E.S.S.` eine kürzere Laufzeit, wobei die Laufzeitdifferenz in diesen Fällen maximal 0.048 Sekunden beträgt. Im Gegensatz dazu ist bei $n = 276\,002$ die `pyMOR` Implementierung 7.87 Sekunden schneller.

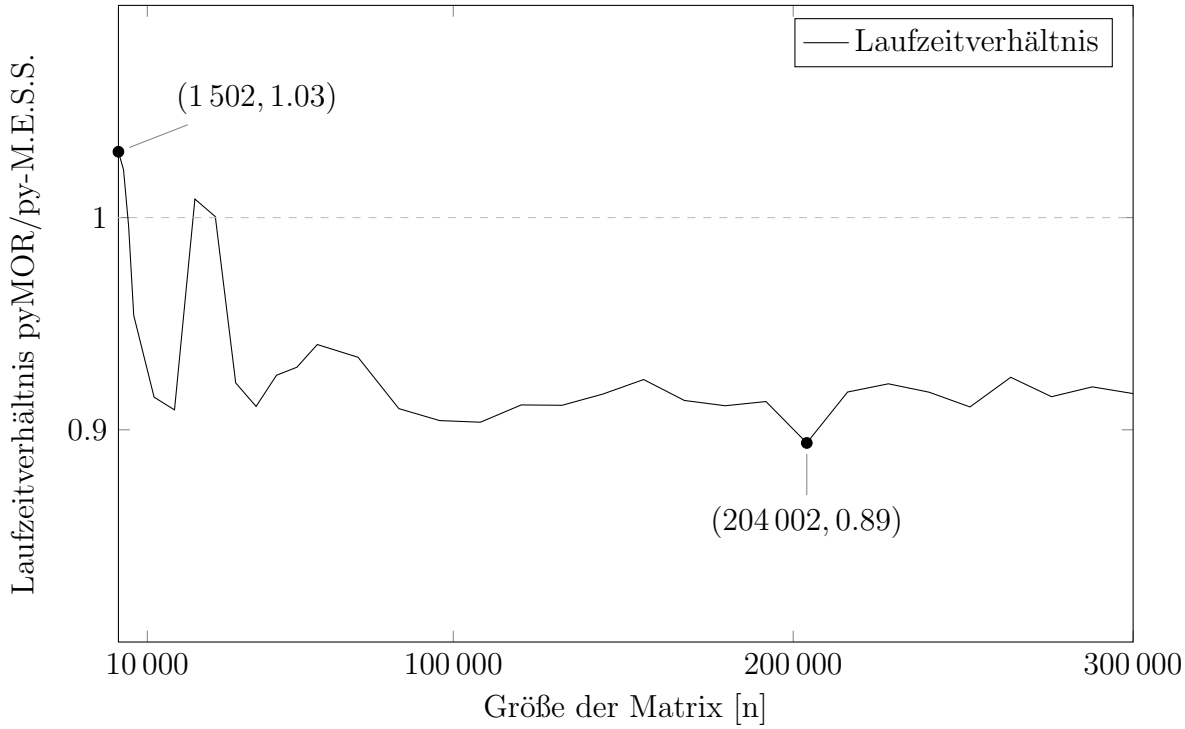

Abbildung 4: Laufzeitverhältins von pyMOR zu py-M.E.S.S. bei unsymmetrischen verallgemeinerten Lyapunovgleichungen

Des Weiteren ist bei $n = 1\,502$ `py-M.E.S.S.` ungefähr 3% schneller als die `pyMOR` Implementierung. Der maximale prozentuale Unterschied wird bei $n = 204\,002$ angenommen, wobei `pyMOR` hierbei eine 11% schnellere Laufzeit erzielen konnte. Außerdem schwankt das Laufzeitverhältnis nach $n = 30\,002$ nur leicht, wobei `py-M.E.S.S.` ab diesem Wert stets zwischen 6% und 11% langsamer ist als die `pyMOR` Variante.

Insgesamt lässt sich bei diesem Versuch erneut das Laufzeitverhalten wie bei den bisher betrachteten Experimenten über den mit dem Kopieren verbundenen Zeitaufwand erklären. Ein weiterer Aspekt welcher beachtet werden sollte, stellt die Anzahl der benötigten Iterationen für die Konvergenz der hier betrachteten Lyapunovgleichungen dar. Diese liegt in diesem Versuchsaufbau für jedes beliebige n deutlich über der in den Abschnitten 5.1 und 5.2 auftretenden Iterationsanzahl. Da bei jedem einzelnen Schleifendurchlauf der `py-M.E.S.S.` Implementierung Daten kopiert werden müssen, wirkt dies sich zusätzlich negativ auf die Laufzeit dieser Variante aus, wodurch `pyMOR` hier klar im Vorteil ist.

5.4 Direkter Vergleich zu py-M.E.S.S.

Wie bereits erwähnt bezogen sich die bisherigen Experimente lediglich auf die abstrakten, dem Designparadigma `pyMOR`s entsprechenden Umsetzungen der LR ADI Iteration. Wird anstatt der abstrakten `py-M.E.S.S.` Schnittstelle jedoch die direkte `py-M.E.S.S.` Implementierung verwendet, ergeben sich beispielsweise für die in Abschnitt 5.2 betrachteten Tests maßgeblich kürzere Laufzeiten.

n	pyMOR Laufzeit	Direkte py-M.E.S.S. Laufzeit	Laufzeitverhältnis
1 357	0.28	0.08	3.59
5 177	1.26	0.37	3.45
20 209	8.29	2.19	3.78
79 841	78.07	11.27	6.93

Tabelle 4: Laufzeitvergleich in Sekunden zu direkter py-M.E.S.S. Umsetzung

Offensichtlich ist hier die direkte Variante den abstrakten Umsetzungen deutlich überlegen. Jedoch würde das direkte Verwenden von `py-M.E.S.S.` innerhalb `pyMOR`s nicht den Designvorgaben entsprechen und entscheidende Konzepte, wie zum Beispiel das verteilt parallele Rechnen, würden sich nicht mehr sinnvoll umsetzen lassen.

6 Zusammenfassung

Es lässt sich festhalten, dass die Low-Rank ADI Iteration ein effizientes Verfahren zum approximativen Lösen von Lyapunovgleichungen darstellt, welches vor allem davon profitiert, dass in vielen Fällen sowohl die Lösung als auch das Residuum der Lyapunovgleichung durch Faktoren niedrigen Ranges darstellbar sind. Die Tatsache, dass diese Faktoren durch doppelte Iterationsschritte stets als reelle Matrizen angegeben werden können, trägt außerdem dazu bei, dass in der Praxis deutlich weniger Speicher für die Iteration benötigt wird. Des Weiteren ist die abstrakte Umsetzung des Verfahrens in `pyMOR` vor allem für verallgemeinerte Lyapunovgleichungen der Schnittstelle zur externen `C-M.E.S.S.` Bibliothek überlegen. Bei Standard Lyapunovgleichungen sollte jedoch für eine kurze Laufzeit trotzdem auf ebendiese zurückgegriffen werden. In jedem Fall können Lyapunovgleichungen in `pyMOR` nun weitestgehend unabhängig von externen Implementierungen gelöst werden.

In Zukunft könnte `pyMOR` auch um weitere Algorithmen zum Lösen von Matrixgleichungen wie beispielsweise der algebraischen Riccati-Gleichung

$$A^T X + XA - XBR^{-1}B^T X + Q = 0$$

erweitert werden. Gleichungen dieser Form treten im Kontext der Modellreduktion beispielsweise bei der *linear-quadratic Gaussian balanced truncation* auf. Außerdem stellt `C-M.E.S.S.` mehr Abbruchkriterien und Möglichkeiten zum Generieren von Shiftparametern für die LR ADI Iteration zur Verfügung als die direkte `pyMOR` Implementierung. Diese könnten im Sinne der Vollständigkeit und Flexibilität zukünftig auch in die `pyMOR` Implementierung integriert werden.

Anhang A

Testergebnisse bei Standard Lyapunovgleichungen

n	pyMOR Laufzeit	py-M.E.S.S. Laufzeit	Laufzeitverhältnis	Iterationen
2 000	0.11	0.10	1.14	50
4 000	0.16	0.14	1.09	48
6 000	0.21	0.20	1.06	50
8 000	0.27	0.26	1.02	52
10 000	0.32	0.31	1.01	52
12 000	0.52	0.38	1.38	54
14 000	0.81	0.61	1.32	75
16 000	0.67	0.50	1.34	55
18 000	1.20	0.90	1.33	88
20 000	0.82	0.64	1.29	56
30 000	2.13	1.32	1.61	79
40 000	2.11	1.34	1.57	58
50 000	3.91	2.67	1.47	92
60 000	2.96	2.04	1.45	59
70 000	3.61	2.50	1.44	61
80 000	4.38	3.39	1.29	63
90 000	4.89	3.99	1.23	63
100 000	4.45	3.73	1.20	63
120 000	5.43	4.54	1.20	65
140 000	6.18	6.27	0.99	65
160 000	7.99	7.16	1.12	65
180 000	8.71	7.86	1.11	69
200 000	9.59	9.00	1.07	67
220 000	10.27	9.70	1.06	66
240 000	11.20	10.65	1.05	67
260 000	12.07	11.51	1.05	67
280 000	13.29	12.71	1.05	69
300 000	21.38	20.67	1.03	105

Tabelle 5: Laufzeiten in Sekunden bei Standard Lyapunovgleichungen

Anhang B

Testergebnisse bei unsymmetrischen verallgemeinerten Lyapunovgleichungen

n	pyMOR Laufzeit	py-M.E.S.S. Laufzeit	Laufzeitverhältnis	Iterationen
1 502	0.39	0.38	1.03	189
3 002	0.76	0.74	1.02	228
4 502	0.92	0.92	1.00	211
6 002	1.08	1.14	0.95	205
12 002	2.37	2.58	0.92	235
18 002	3.72	4.09	0.91	240
24 002	5.50	5.45	1.01	236
30 002	6.80	6.80	1.00	232
36 002	7.76	8.42	0.92	231
42 002	9.46	10.39	0.91	239
4 8002	11.05	11.94	0.93	239
54 002	12.91	13.89	0.93	244
60 002	14.73	15.67	0.94	247
72 002	18.08	19.35	0.93	253
84 002	21.42	23.54	0.91	253
96 002	24.69	27.30	0.90	253
108 002	27.07	29.96	0.90	245
120 002	31.72	34.79	0.91	257
132 002	33.40	36.64	0.91	241
144 002	35.51	38.73	0.92	226
156 002	43.24	46.81	0.92	243
168 002	47.75	52.25	0.91	242
180 002	50.18	55.06	0.91	245
192 002	51.92	56.85	0.91	236
204 002	53.69	60.07	0.89	230
216 002	59.73	65.08	0.92	236
228 002	62.96	68.31	0.92	236
240 002	65.29	71.14	0.92	233
252 002	68.30	74.99	0.91	233
264 002	72.05	77.91	0.92	238
276 002	85.36	93.23	0.92	262
288 002	86.16	93.63	0.92	252
300 002	85.76	93.52	0.92	238

Tabelle 6: Laufzeiten in Sekunden bei unsymmetrischen verallgemeinerten Lyapunovgleichungen

Literatur

[1] *SLICOT*. `http://www.slicot.org`.

[2] P. Benner, M. Köhler, and J. Saak, "M.E.S.S. – the matrix equations sparse solvers library." `https://www.mpi-magdeburg.mpg.de/projects/mess`.

[3] R. Milk, S. Rave, and F. Schindler, "pyMOR - model order reduction with python." `http://pymor.org/`.

[4] R. Milk, S. Rave, and F. Schindler, "pyMOR - generic algorithms and interfaces for model order reduction," *SIAM J. Sci. Comput.*, vol. 38, no. 5, pp. 194–216, 2016.

[5] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*, vol. 6 of *Adv. Des. Control*. Philadelphia, PA: SIAM Publications, 2005.

[6] P. Kürschner, *Efficient Low-Rank Solution of Large-Scale Matrix Equations*. Dissertation, Otto-von-Guericke-Universität, Magdeburg, Germany, Apr. 2016. Shaker Verlag, ISBN 978-3-8440-4385-3.

[7] T. Penzl, "Numerical solution of generalized Lyapunov equations," *Adv. Comp. Math.*, vol. 8, pp. 33–48, 1997.

[8] J. Saak, *Efficient Numerical Solution of Large Scale Algebraic Matrix Equations in PDE Control and Model Order Reduction*. Dissertation, Technische Universität Chemnitz, Chemnitz, Germany, July 2009.

[9] W. A. Strauss, *Partial Differential Equations: An Introduction*. Brown University: Whiley & Sons Ltd, 2008.

[10] R. H. Bartels and G. W. Stewart, "Solution of the matrix equation $AX+XB = C$: Algorithm 432," *Comm. ACM*, vol. 15, pp. 820–826, 1972.

[11] S. J. Hammarling, "Numerical solution of the stable, non-negative definite Lyapunov equation," *IMA J. Numer. Anal.*, vol. 2, pp. 303–323, 1982.

[12] T. Penzl, "A cyclic low rank Smith method for large sparse Lyapunov equations," *SIAM J. Sci. Comput.*, vol. 21, no. 4, pp. 1401–1418, 2000.

[13] A. C. Antoulas, D. C. Sorensen, and Y. Zhou, "On the decay rate of Hankel singular values and related issues," *Syst. Cont. Lett.*, vol. 46, no. 5, pp. 323–342, 2002.

[14] D. C. Sorensen and Y. Zhou, "Bounds on eigenvalue decay rates and sensitivity of solutions to Lyapunov equations," Tech. Rep. TR02-07, Dept. of Comp. Appl. Math., Rice University, Houston, TX, June 2002.

[15] P. Benner, P. Kürschner, and J. Saak, "An improved numerical method for balanced truncation for symmetric second order systems," *Math. Comput. Model. Dyn. Syst.*, vol. 19, no. 6, pp. 593–615, 2013.

[16] P. Benner, P. Kürschner, and J. Saak, "Efficient handling of complex shift parameters in the low-rank Cholesky factor ADI method," *Numer. Algorithms*, vol. 62, no. 2, pp. 225–251, 2013.

[17] E. L. Wachspress, *The ADI Model Problem*. Springer New York, 2013.

[18] P. Benner, P. Kürschner, and J. Saak, "Self-generating and efficient shift parameters in ADI methods for large Lyapunov and Sylvester equations," *Electron. Trans. Numer. Anal.*, vol. 43, pp. 142–162, 2014.

[19] M. Hund, "Adaptive Berechnung der ADI Shiftparameter in der Niedrig-Rang-Galerkin-ADI," bachelor, Otto-von-Guericke-Universität, Magdeburg, Germany, 2012.

[20] A. Logg, K.-A. Mardal, and G. Wells, eds., *Automated Solution of Differential Equations by the Finite Element Method*, vol. 84 of *Lect. Notes Comput. Sci. Eng.* Springer-Verlag, 1 ed., 2012.

[21] W. Bangerth, R. Hartmann, and G. Kanschat, "deal.II – a general purpose object oriented finite element library," *ACM Trans. Math. Softw.*, vol. 33, no. 4, pp. 24/1–24/27, 2007.

[22] "Dune." `https://www.dune-project.org/`.

[23] "Ngsolve." `https://ngsolve.org/`.

[24] "Numpy." `http://www.numpy.org/`.

[25] "Scipy." `https://www.scipy.org/`.

[26] J. G. Korvink and E. B. Rudnyi, "Oberwolfach benchmark collection," in *Dimension Reduction of Large-Scale Systems* (P. Benner, D. C. Sorensen, and V. Mehrmann, eds.), vol. 45 of *Lecture Notes in Computational Science and Engineering*, pp. 311–315, Springer Berlin Heidelberg, 2005.

[27] The MORwiki Community, "MORwiki - Model Order Reduction Wiki." `http://modelreduction.org`.

[28] N. Truhar and K. Veselić, "An efficient method for estimating the optimal dampers' viscosity for linear vibrating systems using Lyapunov equation," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 1, pp. 18–39, 2009.