DOI: 10.14464/gammas.v2i1.420 GAMMAS 2020, Vol. 2, No. 1, p. 1–13

International Association
of Applied Mathematics and Mechanics
– Archive for Students –

Low-Rank Alternating Direction Implicit Iteration in pyMOR

Linus Balicki[a,*]

[a] Otto von Guericke University Magdeburg, Germany

received 13.06.2019, accepted 29.12.2019, published 16.02.2020

* corresponding author: linus.balicki@ovgu.de
supervisors: Jens Saak and Petar Mlinarić, Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg

Abstract: *The low-rank alternating direction implicit (LR-ADI) iteration is an effective method for solving large-scale Lyapunov equations. In the software library `pyMOR`, solutions to Lyapunov equations play an important role when reducing a model using the balanced truncation method. In this article, we introduce the LR-ADI iteration as well as `pyMOR`, while focusing on its features which are relevant for integrating the iteration into the library. We present the results of numerical experiments, which indicate that the iteration's pure `pyMOR` implementation outperforms external libraries when dealing with large problem dimensions.*

Keywords: model order reduction, matrix equations, balanced truncation, alternating direction implicit iteration, pyMOR

1 Introduction

The basis for numerous practical applications and research areas, such as systems and control theory, consists of modeling large-scale technical and dynamical systems. Due to the growing complexity of modern applications, the occurrence of systems of differential equations that are too large for numerical computations or simulations is not uncommon. To overcome this issue, a variety of model order reduction methods have been developed. The main goal of some of these methods is to approximate a high-dimensional system with a much smaller one, which can be used effectively in computations but also estimates the input-output behavior of the original system well. In the process of modeling, linear time-invariant (LTI) systems of the form

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{1}$$

with $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, and $D \in \mathbb{R}^{p\times m}$ play an important role. Furthermore, $u(t) \in \mathbb{R}^m$ describes the input, $y(t) \in \mathbb{R}^p$ the output and $x(t) \in \mathbb{R}^n$ the state of the underlying system. An established method which is used to reduce this type of system is *balanced truncation*, which requires solving *Lyapunov equations* of the type

$$\begin{aligned} AX + XA^{\mathrm{T}} + BB^{\mathrm{T}} &= 0, \\ A^{\mathrm{T}}X + XA + C^{\mathrm{T}}C &= 0, \end{aligned}$$

as a central component. This type of equation appears in various applications [2, 25, 26], where the system matrix A is either small and dense or large and sparse. When dealing with small Lyapunov equations, it is appropriate to use direct solvers like the Bartels-Stewart method [6] and Hammarling's algorithm [15], which are based on the Schur decomposition of the system matrix

A. For large Lyapunov equations, it is usually too costly to compute the Schur decomposition. Additionally, it would not be possible to store the resulting dense solution matrix $X \in \mathbb{R}^{n\times n}$. In order to avoid these issues, it is wise to use iterative solvers, such as the LR-ADI iteration, which compute low-rank factors $Z \in \mathbb{R}^{n\times \ell}$ with $\ell \ll n$ such that the solution is approximated via $X \approx ZZ^{\mathrm{T}}$.

There exist a variety of software packages, such as SLICOT [20] and M.E.S.S. [8], which provide solvers for problems that occur in model order reduction. SLICOT focuses on Fortran 77 implementations which are relevant for applications in systems and control theory whereas M.E.S.S. has originally been developed to solve sparse matrix equations in MATLAB. A model order reduction framework written in the increasingly popular programming language Python is represented by the library pyMOR [5, 18, 23].

This paper revisits the LR-ADI iteration for solving Lyapunov equations and discusses aspects of implementing the corresponding algorithm in pyMOR respecting the library's design pattern, which enforces the abstraction of high-dimensional operations. The final implementation allows the library to work without depending on external solvers for large and sparse Lyapunov equations. Additionally, we discuss pyMOR's design paradigm, which allows developers to easily integrate external solvers for partial differential equations and extend operations, such that these can be used in parallel distributed systems. Finally, we compare the run time of external libraries and the implementation that is solely based on abstract operations available in the pyMOR environment with numerical experiments. For the comparison, we use C-M.E.S.S. and Py-M.E.S.S., which represent implementations of the M.E.S.S. library in C and a wrapper in Python, respectively.

In Section 2, we introduce balanced truncation, which serves as a motivation for solving Lyapunov equations. The LR-ADI iteration is discussed in Section 3, where we derive real low-rank formulations for the solution and the residual of Lyapunov equations. Additionally, we present appropriate choices for shift parameters and consider generalized Lyapunov equations. In Section 4, we introduce pyMOR and discuss practical aspects of implementing the LR-ADI iteration. Numerical experiments, which compare the run time of different implementations available in pyMOR, are presented in Section 5. Finally, we provide a summary and outlook in Section 6.

Throughout this paper we use the following notation: the symbol $\mathbb{R}_-$ denotes the strictly negative real numbers, whereas $\mathbb{C}_-$ represents the open left-half plane. The symbols $\Lambda(A)$ and $\Lambda(A,E)$ denote the spectra of the matrix A and the pair of matrices (A,E), respectively. Furthermore, $\operatorname{diag}(x_1,\dots,x_n)$ is the $n\times n$ diagonal matrix which has the values $x_1,\dots,x_n$ on its diagonal. For a complex matrix $A \in \mathbb{C}^{n\times m}$ we define $A^{\mathrm{H}} := \overline{A}^{\mathrm{T}} \in \mathbb{C}^{m\times n}$ as the complex conjugated transposed matrix.

2 Balanced Truncation

The primary motivation for solving Lyapunov equations in the context of model order reduction is their relevance for the balanced truncation (BT) method. In this section, we shortly discuss BT based on [2]. Note that we focus on continuous-time LTI systems, which are asymptotically stable, hence satisfy $\Lambda(A) \subset \mathbb{C}_-$. This ensures that the resulting Lyapunov equations have a unique positive semidefinite solution. Additionally, we solely consider homogeneous initial conditions for systems of differential equations that occur within this article.

Generally speaking, BT determines weakly observable and controllable states of a system and then eliminates those appropriately. In order to distinguish between weakly and strongly observable and controllable states, some measure for these properties needs to be provided. The *reachability Gramian*

$$\mathcal{P} = \int_0^\infty e^{At} B B^{\mathrm{T}} e^{A^{\mathrm{T}} t}\,\mathrm{d}t, \tag{2}$$

and *observability Gramian*

$$\mathcal{Q} = \int_0^\infty e^{A^{\mathrm{T}} t} C^{\mathrm{T}} C e^{At}\,\mathrm{d}t, \tag{3}$$

serve exactly this purpose: Small eigenvalues of $\mathcal{P}$ and $\mathcal{Q}$ can be associated with weakly controllable and weakly observable states, respectively. In general, there is no guarantee that weakly controllable states coincide with those that are weakly observable. For this reason, the first step in BT consists of determining a matrix $T \in \mathbb{R}^{n\times n}$ such that

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

where $\Sigma_1 \in \mathbb{R}^{r\times r}$ and $\Sigma_2 \in \mathbb{R}^{(n-r)\times(n-r)}$. Using this transformation, the original system's realization, defined by the tuple

$$(A,B,C,D) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m} \times \mathbb{R}^{p\times n} \times \mathbb{R}^{p\times m},$$

can be brought into a balanced realization

$$\left(TAT^{-1}, TB, CT^{-1}, D\right) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m} \times \mathbb{R}^{p\times n} \times \mathbb{R}^{p\times m}$$

which describes the original dynamical system, but has the property that weakly controllable states are simultaneously weakly observable.

In order to determine the transformation matrix T, it is crucial to compute the Gramians from equations (2) and (3). It can be shown [17] that the Gramians are solutions to the dual Lyapunov equations

$$\begin{aligned} A\mathscr{P} + \mathscr{P}A^{\mathrm{T}} + BB^{\mathrm{T}} &= 0, \\ A^{\mathrm{T}}\mathscr{Q} + \mathscr{Q}A + C^{\mathrm{T}}C &= 0. \end{aligned}$$

After computing the Gramians, Cholesky-like factorizations of the form $\mathscr{P} = Z_{\mathscr{P}}Z_{\mathscr{P}}^{\mathrm{T}}$ and $\mathscr{Q} = Z_{\mathscr{Q}}Z_{\mathscr{Q}}^{\mathrm{T}}$ allow the formulation of the singular value decomposition

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

where $\Sigma_1 \in \mathbb{R}^{r\times r}$ and the matrices L_1 and R_1 have r columns, respectively. Finally, we can define the transformation matrix via $T = \Sigma^{-\frac{1}{2}}R^{\mathrm{T}}Z_{\mathscr{Q}}^{\mathrm{T}}$, where it holds that $T^{-1} = Z_{\mathscr{P}}L\Sigma^{-\frac{1}{2}}$ and $\Sigma^{-\frac{1}{2}} = \operatorname{diag}(1/\sqrt{\sigma_1},\ldots,1/\sqrt{\sigma_n})$. As a next step, we want to truncate the system such that states that can be associated with the singular values $\sigma_{r+1},\ldots,\sigma_n$ will be eliminated. Let therefore

$$T_L := \Sigma_1^{-\frac{1}{2}}R_1^{\mathrm{T}}Z_{\mathscr{Q}}^{\mathrm{T}} \quad \text{and} \quad T_R := Z_{\mathscr{P}}L_1\Sigma_1^{-\frac{1}{2}},$$

which define $\widetilde{A} := T_LAT_R$, $\widetilde{B} := T_LB$, $\widetilde{C} := CT_R$ and thus the reduced-order model

$$\begin{aligned} \dot{\widetilde{x}}(t) &= \widetilde{A}\widetilde{x}(t) + \widetilde{B}u(t), \\ \widetilde{y}(t) &= \widetilde{C}\widetilde{x}(t) + Du(t), \end{aligned}$$

with $\widetilde{A} \in \mathbb{R}^{r\times r}$, $\widetilde{B} \in \mathbb{R}^{r\times m}$, $\widetilde{C} \in \mathbb{R}^{p\times r}$, $\widetilde{x}(t) \in \mathbb{R}^{r}$, $\widetilde{y}(t) \in \mathbb{R}^{p}$.

3 Low-Rank ADI Iteration for Lyapunov Equations

In practical applications, the coefficient matrix A of a Lyapunov equation is often sparse and the matrix BB^{T}, which we refer to as the *right-hand side*, has a low rank (i.e., $B \in \mathbb{R}^{n\times m}$ with $m \ll n$). In this case, it can be shown [3] that the solution of the underlying Lyapunov equation often has a low numerical rank as well. This serves as a motivation to approximate it with low-rank factors $Z \in \mathbb{R}^{n\times \ell}$ such that $\ell \ll n$ and $X \approx ZZ^{\mathrm{T}}$. An iterative approach for solving Lyapunov equations, which takes advantage of these properties, is the LR-ADI iteration. In this section, we introduce the iteration for standard Lyapunov equations

$$AX + XA^{\mathrm{T}} + BB^{\mathrm{T}} = 0, \tag{4}$$

and later extend the resulting algorithm such that it can be applied to generalized Lyapunov equations

$$AXE^{\mathrm{T}} + EXA^{\mathrm{T}} + BB^{\mathrm{T}} = 0, \tag{5}$$

where $E \in \mathbb{R}^{n\times n}$ is an invertible matrix.

3.1 ADI Iteration

The basis for deriving the LR-ADI iteration usually consists of formulating the two-step iteration [17, 22]

$$\begin{aligned} (A+\alpha_iI)X_{i-\frac{1}{2}} &= -BB^{\mathrm{T}} - X_{i-1}\left(A^{\mathrm{T}} - \alpha_iI\right), \\ (A+\alpha_iI)X_i^{\mathrm{T}} &= -BB^{\mathrm{T}} - X_{i-\frac{1}{2}}^{\mathrm{T}}\left(A^{\mathrm{T}} - \alpha_iI\right) \end{aligned}$$

with complex shift parameters $\alpha_i \in \mathbb{C}_-$ and an initial guess $X_0 = 0 \in \mathbb{R}^{n\times n}$. We discuss an appropriate choice of shift parameters in Section 3.6. The above expression is equivalent to the single iteration step

$$\begin{aligned} X_i = {} & (A+\alpha_iI)^{-1}\left(A-\overline{\alpha_i}I\right)X_{i-1}\left(A-\overline{\alpha_i}I\right)^{\mathrm{H}}(A+\alpha_iI)^{-\mathrm{H}} \\ & -2\operatorname{Re}(\alpha_i)(A+\alpha_iI)^{-1}BB^{\mathrm{T}}(A+\alpha_iI)^{-\mathrm{H}}. \end{aligned} \tag{6}$$

This formulation neither takes advantage of the right-hand side's low rank nor does it compute the desired low-rank solution factors. In the following paragraphs, we derive a formulation of the ADI iteration, which tackles these issues, and summarize the results in a first algorithm.

3.2 Low-Rank Solution Factors

By rearranging (6) and considering $X_0 = Z_0Z_0^{\mathrm{H}}$ with initial value $Z_0 = [\,] \in \mathbb{C}^{n\times 0}$, we obtain the expression

$$\begin{aligned} Z_iZ_i^{\mathrm{H}} = {} & \left((A+\alpha_iI)^{-1}\left(A-\overline{\alpha_i}I\right)Z_{i-1}\right) \\ & \times\left((A+\alpha_iI)^{-1}\left(A-\overline{\alpha_i}I\right)Z_{i-1}\right)^{\mathrm{H}} \\ & + \left(\sqrt{-2\operatorname{Re}(\alpha_i)}(A+\alpha_iI)^{-1}B\right) \\ & \times\left(\sqrt{-2\operatorname{Re}(\alpha_i)}(A+\alpha_iI)^{-1}B\right)^{\mathrm{H}}. \end{aligned}$$

A single low-rank factor is then given by

$$\begin{aligned} Z_i = \Big[& \sqrt{-2\operatorname{Re}(\alpha_i)}(A+\alpha_iI)^{-1}B, \\ & (A+\alpha_iI)^{-1}\left(A-\overline{\alpha_i}I\right)Z_{i-1}\Big]. \end{aligned}$$

Repeatedly replacing Z_j for $j = i-1,\ldots,0$ on the right-hand side of the expression above yields

$$\begin{aligned} Z_i = \Big[& \sqrt{-2\operatorname{Re}(\alpha_i)}(A+\alpha_iI)^{-1}B,\ \ldots, \\ & \sqrt{-2\operatorname{Re}(\alpha_1)}T_i\cdots T_2(A+\alpha_1I)^{-1}B\Big], \end{aligned} \tag{7}$$

Algorithm 1 LR-ADI Iteration Version 1

```
Input: $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $\{\alpha_1,\dots,\alpha_s\} \subset \mathbb{C}_-$
Output: $Z \in \mathbb{C}^{n\times sm}$
1: solve $(A+\alpha_1 I)V_1 = B$ for $V_1$
2: $Z_1 = \sqrt{-2\operatorname{Re}(\alpha_1)}V_1$
3: for $i = 2,\dots,s$ do
4:     solve $(A+\alpha_i I)\hat{V} = V_{i-1}$ for $\hat{V}$
5:     $V_i = V_{i-1} - \left(\alpha_i + \overline{\alpha_{i-1}}\right)\hat{V}$
6:     $Z_i = \left[Z_{i-1},\ \sqrt{-2\operatorname{Re}(\alpha_i)}V_i\right]$
7: end for
8: $Z = Z_s$
```

where $T_j := (A+\alpha_j I)^{-1}(A-\overline{\alpha_j}I)$. Considering the fact that the matrices $(A\pm\alpha I)^{\pm1}$ and $(A\pm\beta I)^{\pm1}$ commute for arbitrary $\alpha,\beta\in\mathbb{C}$ as long as the inverse exists [17], allows us to rearrange the factors occurring in the product $T_i\cdots T_2$. Reversing the order of the shifts yields a more beneficial expression for (7) with

$$\begin{aligned} Z_1 &= \sqrt{-2\operatorname{Re}(\alpha_1)}V_1,\\ Z_i &= \left[Z_{i-1},\ \sqrt{-2\operatorname{Re}(\alpha_i)}V_i\right], \end{aligned} \tag{8}$$

where

$$\begin{aligned} V_1 &= (A+\alpha_1 I)^{-1}B,\\ V_i &= \left(A-\overline{\alpha_{i-1}}I\right)(A+\alpha_i I)^{-1}V_{i-1} &\text{(9a)}\\ &= V_{i-1} - \left(\alpha_i+\overline{\alpha_{i-1}}\right)(A+\alpha_i I)^{-1}V_{i-1}. &\text{(9b)} \end{aligned}$$

This leads to our first formulation of the LR-ADI iteration, which requires the solution of a shifted linear system with multiple right-hand sides in each step of the iteration and is presented in Algorithm 1. Note that m columns are added to the solution factor in each iteration step since $V_i\in\mathbb{C}^{n\times m}$. A fundamental issue with this algorithm is the lack of information about the quality of the approximation ZZ^{H}, and in connection to that, it is unclear how many iteration steps s are necessary in order to obtain a sufficiently accurate approximation. A commonly used indicator for a high-quality approximation is a small *Lyapunov residual*

$$\mathscr{R}_i = AX_i + X_iA^{\mathrm{T}} + BB^{\mathrm{T}}.$$

Note that the Lyapunov equation is equivalent to a system of linear equations [2]. Due to the residual's relation to the error of the currently computed approximation [1], a stopping criterion for the LR-ADI iteration based on the residual is an obvious choice. Evaluating $\mathscr{R}_i$, as stated above, requires several large matrices to be multiplied. Since the computational effort for the multiplications is too large, we present a more favorable approach to evaluate the residual in the following subsection.

3.3 Low-Rank Residual Factors

Results presented in [10] have shown that not only the solution of a Lyapunov equation but also its residual has a low rank. In particular, it can be shown that the rank of the Lyapunov residual $\mathscr{R}_i$ coincides with the rank of the right-hand side BB^{T} if it holds $\{\alpha_1,\dots,\alpha_i\}\cap\Lambda(A)=\emptyset$. This serves as a motivation to analyze low-rank residual factors $W_i\in\mathbb{C}^{n\times m}$ such that $\mathscr{R}_i = W_iW_i^{\mathrm{H}}$.

In order to derive an expression for W_i, we consider the following formula which is a result of subtracting the solution X from equation (6)

$$\begin{aligned} X_i - X &= (A+\alpha_i I)^{-1}\Big(\left(A-\overline{\alpha_i}I\right)X_{i-1}\left(A-\overline{\alpha_i}I\right)^{\mathrm{H}}\\ &\qquad -2\operatorname{Re}(\alpha_i)BB^{\mathrm{T}}\Big)(A+\alpha_i I)^{-\mathrm{H}} - X\\ &= (A+\alpha_i I)^{-1}\Big(\left(A-\overline{\alpha_i}I\right)X_{i-1}\left(A-\overline{\alpha_i}I\right)^{\mathrm{H}}\\ &\qquad +\left(\alpha_i+\overline{\alpha_i}\right)\left(AX+XA^{\mathrm{T}}\right)\\ &\qquad -(A+\alpha_i I)X(A+\alpha_i I)^{\mathrm{H}}\Big)(A+\alpha_i I)^{-\mathrm{H}}\\ &= (A+\alpha_i I)^{-1}\left(A-\overline{\alpha_i}I\right)(X_{i-1}-X)\\ &\qquad \times\left(A-\overline{\alpha_i}I\right)^{\mathrm{H}}(A+\alpha_i I)^{-\mathrm{H}}. \end{aligned}$$

Repeatedly plugging this into the expression

$$\mathscr{R}_i = A(X_i - X) + (X_i - X)A^{\mathrm{T}} = W_iW_i^{\mathrm{H}}$$

leads to

$$\begin{aligned} W_iW_i^{\mathrm{H}} &= A\left(\prod_{k=1}^{i}T_k\right)(-X)\left(\prod_{k=1}^{i}T_k^{\mathrm{H}}\right)\\ &\quad + \left(\prod_{k=1}^{i}T_k\right)(-X)\left(\prod_{k=1}^{i}T_k^{\mathrm{H}}\right)A^{\mathrm{T}}\\ &= \left(\prod_{k=1}^{i}T_k\right)BB^{\mathrm{T}}\left(\prod_{k=1}^{i}T_k^{\mathrm{H}}\right), \end{aligned}$$

with $T_j = (A+\alpha_j I)^{-1}(A-\overline{\alpha_j}I)$ as previously defined. Clearly, a single low-rank residual factor can then be written as

$$W_i = \left(\prod_{k=1}^{i}T_k\right)B.$$

Reconsidering the V_i defined in (9a) allows us to derive the following expression by repeatedly replacing the V_{i-1} and afterwards rearranging the remaining factors appropriately

$$\begin{aligned} V_i &= \left(A-\overline{\alpha_{i-1}}I\right)(A+\alpha_i I)^{-1}V_{i-1}\\ &= (A+\alpha_i I)^{-1}\left(\prod_{k=1}^{i-1}T_k\right)B\\ &= (A+\alpha_i I)^{-1}W_{i-1} &\text{(10a)}\\ &= \left(A-\overline{\alpha_i}I\right)^{-1}W_i. &\text{(10b)} \end{aligned}$$

Algorithm 2 LR-ADI Iteration Version 2

```
Input: $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $\{\alpha_1,\dots,\alpha_s\} \subset \mathbb{C}_-$, $0 < \epsilon \ll 1$
Output: $Z \in \mathbb{C}^{n\times sm}$
1: $Z_0 = [\,]$, $W_0 = B$, $i = 1$
2: while $\left\| W_{i-1}^{\mathrm{H}} W_{i-1} \right\| > \epsilon \|B^{\mathrm{T}} B\|$ do
3:     solve $(A + \alpha_i I)V_i = W_{i-1}$ for $V_i$
4:     $W_i = W_{i-1} - 2\operatorname{Re}(\alpha_i)V_i$
5:     $Z_i = \left[Z_{i-1},\ \sqrt{-2\operatorname{Re}(\alpha_i)}V_i\right]$
6:     $i = i + 1$
7: end while
8: $Z = Z_s$
```

Finally, we obtain the formulation

$$\begin{aligned} W_i &= \left(A - \overline{\alpha_i} I\right) V_i = (A + \alpha_i I - 2\operatorname{Re}(\alpha_i) I) V_i \\ &= W_{i-1} - 2\operatorname{Re}(\alpha_i) V_i, \end{aligned} \tag{11}$$

for the residual factors.

The major benefit of expressing the residual as a product of two low-rank factors is that for several matrix norms it holds [10]

$$\|\mathscr{R}_i\| = \left\| W_i W_i^{\mathrm{H}} \right\| = \left\| W_i^{\mathrm{H}} W_i \right\|.$$

Since $W_i^{\mathrm{H}} W_i \in \mathbb{R}^{m\times m}$ is a small matrix, eigenvalues can be cheaply computed, which allows fast evaluation of the matrix norm, for instance when using $\|\cdot\|_2$.

Thus, a stopping criterion for the LR-ADI iteration is provided by the relative residual

$$\frac{\|\mathscr{R}_i\|}{\|\mathscr{R}_0\|} = \frac{\left\| W_i^{\mathrm{H}} W_i \right\|}{\left\| B^{\mathrm{T}} B \right\|} \le \epsilon, \tag{12}$$

where $0 < \epsilon \ll 1$. Integrating this expression into the iteration leads to an improved version of Algorithm 1 summarized in Algorithm 2. Note that for a clearer presentation we consistently assume that the presented algorithms require exactly s iteration steps until the convergence criterion is fulfilled.

3.4 Real Low-Rank Factors

So far, we have considered the matrices W_i and Z_i to be complex, since the shift parameters $\{\alpha_1,\dots,\alpha_s\}$ are complex as well. In [9], real formulations for these low-rank factors have been derived, based on the assumption that complex shifts only appear in pairs of complex conjugates $\{\alpha_i, \alpha_{i+1} = \overline{\alpha_i}\}$.

In the following, we assume that the LR-ADI iteration is in the k-th step and that the previously computed residual factor W_{k-1} and solution factor Z_{k-1} are real-valued. Letting complex shift parameters appear as pairs of complex conjugates exclusively, we now derive real formulations for W_k and Z_k or W_{k+1} and Z_{k+1}, respectively, by analyzing every relevant case that can occur within the iteration:

1. Case $\alpha_k \in \mathbb{R}_-$: Due to (10a) and (10b) it holds

$$W_k = (A - \overline{\alpha_k} I)(A + \alpha_k I)^{-1} W_{k-1}.$$

Since W_{k-1} is a real matrix and $\alpha_k \in \mathbb{R}_-$ we obtain $W_k \in \mathbb{R}^{n\times m}$. Hence, equation (10b) also provides a real-valued representation for V_k. Finally, from (8) we immediately obtain that the solution factor Z_k is real-valued as well.

2. Case $\alpha_k \in \mathbb{C}_- \setminus \mathbb{R}$, $\alpha_{k+1} = \overline{\alpha_k}$: Splitting equation (10a) into its real and imaginary part yields

$$\begin{aligned} W_{k-1} &= (A + \alpha_k I) V_k \\ &= (A + \operatorname{Re}(\alpha_k) I + \imath \operatorname{Im}(\alpha_k) I) \\ &\quad \times (\operatorname{Re}(V_k) + \imath \operatorname{Im}(V_k)) \\ &= A\operatorname{Re}(V_k) + \operatorname{Re}(\alpha_k)\operatorname{Re}(V_k) - \operatorname{Im}(\alpha_k)\operatorname{Im}(V_k) \\ &\quad + \imath(\operatorname{Im}(\alpha_k)\operatorname{Re}(V_k) + A\operatorname{Im}(V_k) \\ &\qquad + \operatorname{Re}(\alpha_k)\operatorname{Im}(V_k)). \end{aligned}$$

According to our assumption we deduce

$$\begin{aligned} 0 &= \operatorname{Im}(W_{k-1}) \\ &= \operatorname{Im}(\alpha_k)\operatorname{Re}(V_k) + A\operatorname{Im}(V_k) + \operatorname{Re}(\alpha_k)\operatorname{Im}(V_k) \\ &= \left(A + \overline{\alpha_k} I\right)\operatorname{Im}(V_k) + \operatorname{Im}(\alpha_k) V_k, \end{aligned}$$

which is equivalent to

$$0 = \frac{1}{\operatorname{Im}(\alpha_k)} \operatorname{Im}(V_k) + \left(A + \overline{\alpha_k} I\right)^{-1} V_k$$

in the currently discussed case. Using $\alpha_{k+1} = \overline{\alpha_k}$ and (9b), we obtain

$$\begin{aligned} V_{k+1} &= V_k - 2\overline{\alpha_k}\left(A + \overline{\alpha_k} I\right)^{-1} V_k \\ &= V_k + 2\frac{\operatorname{Re}(\alpha_k) - \imath \operatorname{Im}(\alpha_k)}{\operatorname{Im}(\alpha_k)} \operatorname{Im}(V_k) \\ &= \overline{V_k} + 2\frac{\operatorname{Re}(\alpha_k)}{\operatorname{Im}(\alpha_k)} \operatorname{Im}(V_k). \end{aligned} \tag{13}$$

Finally, equation (11) yields

$$\begin{aligned} W_{k+1} &= W_k - 2\operatorname{Re}(\alpha_k) V_{k+1} \\ &= W_{k-1} - 2\operatorname{Re}(\alpha_k) V_k - 2\operatorname{Re}(\alpha_k) V_{k+1} \\ &= W_{k-1} \\ &\quad - 2\operatorname{Re}(\alpha_k)\left(V_k + \overline{V_k} + \frac{2\operatorname{Re}(\alpha_k)}{\operatorname{Im}(\alpha_k)} \operatorname{Im}(V_k)\right) \\ &= W_{k-1} \\ &\quad - 4\operatorname{Re}(\alpha_k)\left(\operatorname{Re}(V_k) + \frac{\operatorname{Re}(\alpha_k)}{\operatorname{Im}(\alpha_k)} \operatorname{Im}(V_k)\right), \end{aligned} \tag{14}$$

where clearly $W_{k+1} \in \mathbb{R}^{n\times m}$.
Based on the derived real-valued expression for the residual factor, we can show that Z_{k+1} has a real representation as well. Let us therefore consider equation (8), which clearly yields

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

By observing

$$V_k\overline{V_k}^{\mathrm{T}} + \overline{V_k}V_k^{\mathrm{T}} = 2\operatorname{Re}(V_k)\operatorname{Re}(V_k)^{\mathrm{T}} + 2\operatorname{Im}(V_k)\operatorname{Im}(V_k)^{\mathrm{T}},$$

introducing the notation $\delta_k := \frac{\operatorname{Re}(\alpha_k)}{\operatorname{Im}(\alpha_k)}$, and considering (13) we obtain

$$\begin{aligned} X_{k+1} &= Z_{k+1}Z_{k+1}^{\mathrm{H}} \\ &= Z_{k-1}Z_{k-1}^{\mathrm{T}} - 2\operatorname{Re}(\alpha_k)\left(V_k\overline{V}_k^{\mathrm{T}} + V_{k+1}\overline{V}_{k+1}^{\mathrm{T}}\right) \\ &= Z_{k-1}Z_{k-1}^{\mathrm{T}} \\ &\quad - 4\operatorname{Re}(\alpha_k)\Big(\operatorname{Re}(V_k)\operatorname{Re}(V_k)^{\mathrm{T}} + \operatorname{Im}(V_k)\operatorname{Im}(V_k)^{\mathrm{T}} \\ &\qquad + \delta_k\operatorname{Im}(V_k)V_k^{\mathrm{T}} + \delta_k\overline{V_k}\operatorname{Im}(V_k)^{\mathrm{T}} \\ &\qquad + 2\delta_k^2\operatorname{Im}(V_k)\operatorname{Im}(V_k)^{\mathrm{T}}\Big) \\ &= Z_{k-1}Z_{k-1}^{\mathrm{T}} \\ &\quad - 4\operatorname{Re}(\alpha_k)\Big((\operatorname{Re}(V_k) + \delta_k\operatorname{Im}(V_k)) \\ &\qquad \times (\operatorname{Re}(V_k) + \delta_k\operatorname{Im}(V_k))^{\mathrm{T}} \\ &\qquad + \left(\delta_k^2 + 1\right)\operatorname{Im}(V_k)\operatorname{Im}(V_k)^{\mathrm{T}}\Big). \end{aligned}$$

We can now extract a single real solution factor for a double iteration step, which is given by

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

Using the fact that $W_0 = B \in \mathbb{R}^{n\times m}$ and considering $Z_0 = [\,] \in \mathbb{R}^{n\times 0}$, we can use induction to show that we always obtain real-valued iterates W_i and Z_i when shift parameters appear in pairs of complex conjugates. In conclusion, the presented cases yield that real shift parameters always result in real residual and solution factors. Additionally, when a pair of complex shift parameters $\{\alpha_k, \alpha_{k+1} = \overline{\alpha}_k\}$ occurs, a double step can be performed in the iteration, in order to obtain real-valued low-rank factors.

These results allow us to formulate the final version of the LR-ADI iteration for standard Lyapunov equations, which we summarize in Algorithm 3. Using real low-rank factors has two significant benefits: For one, real values require about half as much storage as complex values and secondly, whenever a pair of complex shift parameters occurs, only one instead of two shifted linear systems has to be solved within the iteration in order to perform two steps.

Algorithm 3 LR-ADI Iteration Version 3

```
Input: $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $\{\alpha_1,\dots,\alpha_s\} \subset \mathbb{C}_-$, $0 < \epsilon \ll 1$
Output: $Z \in \mathbb{R}^{n\times sm}$
1:  $Z_0 = [\,]$, $W_0 = B$, $i = 1$
2:  while $\|W_{i-1}^{\mathrm{T}}W_{i-1}\| > \epsilon\|B^{\mathrm{T}}B\|$ do
3:      solve $(A + \alpha_i I)V_i = W_{i-1}$ for $V_i$
4:      if $\operatorname{Im}(\alpha_i) = 0$ then
5:          $W_i = W_{i-1} - 2\operatorname{Re}(\alpha_i)V_i$
6:          $Z_i = [Z_{i-1},\ \sqrt{-2\operatorname{Re}(\alpha_i)}V_i]$
7:          $i = i + 1$
8:      else
9:          $\delta_i = \frac{\operatorname{Re}(\alpha_i)}{\operatorname{Im}(\alpha_i)}$, $\beta_i = 2\sqrt{-\operatorname{Re}(\alpha_i)}$
10:         $W_{i+1} = W_{i-1} + \beta_i^2(\operatorname{Re}(V_i) + \delta_i\operatorname{Im}(V_i))$
11:         $Z_{i+1} = \Big[Z_{i-1},\ \beta_i(\operatorname{Re}(V_i) + \delta_i\operatorname{Im}(V_i)),\ \beta_i\sqrt{\delta_i^2+1}\operatorname{Im}(V_i)\Big]$
12:         $i = i + 2$
13:     end if
14: end while
15: $Z = Z_s$
```

3.5 Low-Rank ADI Iteration for Generalized Lyapunov Equations

As already pointed out, we want to extend Algorithm 3 such that it can be applied to generalized Lyapunov equations

$$AXE^{\mathrm{T}} + EXA^{\mathrm{T}} + BB^{\mathrm{T}} = 0, \tag{15}$$

where $E \in \mathbb{R}^{n\times n}$ is an invertible matrix. This type of equation plays an important role when applying the BT method to generalized LTI systems of the type

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t). \end{aligned} \tag{16}$$

We formulate equation (15) as a standard Lyapunov equation

$$\widetilde{A}X + X\widetilde{A}^{\mathrm{T}} + \widetilde{B}\widetilde{B}^{\mathrm{T}} = 0,$$

with $\widetilde{A} := E^{-1}A$ and $\widetilde{B} := E^{-1}B$. Let $\widetilde{W_i} := EW_i$. We observe that applying Algorithm 3 to the transformed generalized Lyapunov equation yields the following linear system in Line 3 of the iteration

$$\left(\widetilde{A} + \alpha_i I\right)V_i = W_{i-1} \iff (A + \alpha_i E)V_i = \widetilde{W}_{i-1}.$$

Algorithm 4 LR-ADI Iteration for generalized Lyapunov equations

Input: $A, E \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $\{\alpha_1, \dots, \alpha_s\} \subset \mathbb{C}_-$, $0 < \epsilon \ll 1$

Output: $Z \in \mathbb{R}^{n\times sm}$

```
1:  $Z_0 = [\,]$, $W_0 = B$, $i = 1$
2:  while $\|W_{i-1}^{\mathrm{T}} W_{i-1}\| > \varepsilon \|B^{\mathrm{T}} B\|$ 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:         $W_{i+1} = W_{i-1} + \beta_i^2 E(\mathrm{Re}(V_i) + \delta_i\,\mathrm{Im}(V_i))$
11:         $Z_{i+1} = \Big[Z_{i-1},\ \beta_i(\mathrm{Re}(V_i) + \delta_i\,\mathrm{Im}(V_i)),$
                $\beta_i\sqrt{\delta_i^2 + 1}\,\mathrm{Im}(V_i)\Big]$
12:         $i = i + 2$
13:     end if
14: end while
15: $Z = Z_s$
```

Adapting Line 5 and Line 10 of the algorithm accordingly, results in the expressions

$$W_i = W_{i-1} - 2\,\mathrm{Re}(\alpha_i)V_i \iff \widetilde{W}_i = \widetilde{W}_{i-1} - 2\,\mathrm{Re}(\alpha_i)EV_i$$

and

$$\begin{aligned} W_{i+1} &= W_{i-1} - \beta_i^2(\mathrm{Re}(V_i) + \delta_i\,\mathrm{Im}(V_i)) \\ &\iff \widetilde{W}_{i+1} = \widetilde{W}_{i-1} - \beta_i^2 E(\mathrm{Re}(V_i) + \delta_i\,\mathrm{Im}(V_i)) \end{aligned}$$

where $\widetilde{W}_0 = EW_0 = E\widetilde{B} = B$. Using the derived expressions, we can formulate Algorithm 4, which is a slightly altered version of Algorithm 3 and can be applied to generalized Lyapunov equations.

3.6 Shift Parameters

A fundamental component of the iteration, which we did not discuss yet, is the choice of the shift parameters $\{\alpha_1, \dots, \alpha_s\}$. We shortly discuss approaches for computing such parameters, while focusing on generalized Lyapunov equations.

It can be shown [17, 24] that a superlinear convergence of the LR-ADI iteration can be achieved by using shifts with a negative real part. Furthermore, shift parameters that lead to a fast convergence can be obtained by solving the ADI shift parameter problem [27]

$$\{\alpha_1, \dots, \alpha_s\} = \underset{\{\mu_1,\dots,\mu_s\}\subset\mathbb{C}_-}{\arg\min} \left(\max_{\lambda\in\Lambda(A,E)} \left| \prod_{k=1}^{s} \frac{\lambda - \overline{\mu_k}}{\lambda + \mu_k} \right|^2 \right). \quad (17)$$

Since this min-max problem requires knowledge about the spectrum $\Lambda(A,E)$, the computational effort for computing exact solutions of equation (17) is too large in most cases. Nonetheless, several approaches that provide approximate solutions to the ADI shift parameter problem have been established. These are usually based on computing a certain amount of Ritz values of $E^{-1}A$ and $A^{-1}E$, which can then be used to estimate a domain $\Omega \subset \mathbb{C}_-$ such that $\Lambda(A,E) \subset \Omega$. This allows for an approximation of equation (17) [17, 24]. Alternatively, the Ritz values are used in heuristic approaches (e.g., in [22]) where (17) is evaluated by using several combinations of possible shift parameters.

A major drawback of these approaches is that several relevant parameters have to be specified. For instance, the amount of Ritz values that need to be computed in order to obtain high-quality shifts strongly depends on the underlying problem. A more user-friendly approach was introduced in [11], whereas further variants of the method have been discussed in [17]. The main idea of the approach is to use eigenvalues generated by low-dimensional subspaces as shift parameters, instead of computing Ritz values of large matrices such as $E^{-1}A$. Despite the lack of a deep theoretical understanding, these types of shift parameters have performed well in a variety of numerical experiments. Accordingly, we chose to implement variants of this approach for the `pyMOR` project and take a closer look at the approach in the following paragraphs. Note that we focus solely on variants that use the iterate V_i, which allows for a clearer structure in the subsequent lines as well as in the implementation.

An initial set of shift parameters is provided by

$$\{\alpha_1, \dots, \alpha_{k_1}\} = \Lambda(\hat{B}^{\mathrm{T}} A\hat{B}, \hat{B}^{\mathrm{T}} E\hat{B}) \cap \mathbb{C}_-,$$

where the columns of $\hat{B}$ form an orthonormal basis for $\mathrm{span}\{B\}$. Since we consider $B \in \mathbb{R}^{n\times m}$ with $m \ll n$, the orthonormal basis can be computed in an appropriate time frame. Additionally, it holds that $\hat{B}^{\mathrm{T}} A\hat{B}$ and $\hat{B}^{\mathrm{T}} E\hat{B}$ are small square matrices, which allows for solving the underlying generalized eigenvalue problem in a negligibly small period of time as well. If none of the computed eigenvalues lie in $\mathbb{C}_-$, we use a randomly generated matrix Q and initialize the shifts with

$$\{\alpha_1, \dots, \alpha_{k_1}\} = \Lambda(Q^{\mathrm{T}} AQ, Q^{\mathrm{T}} EQ) \cap \mathbb{C}_-.$$

Note that the authors in [11] and [17] suggest to reflect unstable shift parameters across the imaginary axis instead of computing new shifts using the matrix Q. However, the implementation of the LR-ADI iteration in `Py-M.E.S.S.` is based on the variant which we stated first. In order to present a meaningful comparison of the pyMOR implementation and the Py-M.E.S.S. variant in Section 5, we decided to use the approach based on the projection with the randomly generated matrix Q in pyMOR. After k_1 steps of the iteration, new shifts can be computed by one of the following options using the iterates V_i:

1. Use $\{\alpha_{k_1+1},\dots,\alpha_{k_2}\} = \Lambda(\widetilde{V}_{k_1}^{\mathrm{T}} A \widetilde{V}_{k_1}, \widetilde{V}_{k_1}^{\mathrm{T}} E \widetilde{V}_{k_1}) \cap \mathbb{C}_-$, with the columns of $\widetilde{V}_{k_1}$ as an orthonormal basis for $\operatorname{span}\{\operatorname{Re}(V_{k_1}), \operatorname{Im}(V_{k_1})\}$.
2. Use $\{\alpha_{k_1+1},\dots,\alpha_{k_2}\} = \Lambda(\widetilde{V}_{k_1}(u)^{\mathrm{T}} A \widetilde{V}_{k_1}(u), \widetilde{V}_{k_1}(u)^{\mathrm{T}} E \widetilde{V}_{k_1}(u)) \cap \mathbb{C}_-$, with the columns of $\widetilde{V}_{k_1}(u)$ as an orthonormal basis for $\operatorname{span}\{V_{k_1}(u)\}$ and
$$V_{k_1}(u) := [V_{k_1-u},\dots,V_{k_1}].$$

If none of these variants yield new shifts, the previous set of shift parameters can be reused. In the case $u \geq k_1$, we use $V_{k_1}(u) := [V_1,\dots,V_{k_1}]$. Note that in the second approach it is possible to use the last um columns added to the solution factor instead of using stored values of the iterates V_i. In the case that $\alpha_{k_1-u-1} = \overline{\alpha_{k_1-u}}$, it is wise to consider the last $u(m+1)$ columns instead.

4 Software and Implementation

Using the rather theoretical results summarized in the previous sections, we discuss the implementation of the LR-ADI iteration in the following paragraphs. In the course of this, we introduce the pyMOR framework and briefly consider practical aspects of the implementation.

4.1 pyMOR

The software library pyMOR provides a framework for model order reduction applications in the programming language Python. Amongst others, reduced basis methods and system-theoretic model order reduction approaches such as balanced truncation have been implemented. One of the central design goals of pyMOR is the realization of model order reduction algorithms such that external libraries for solving partial differential equations (PDE) can be easily integrated into the library. In connection to that, the implementations should be formulated in a generic way such that the algorithms do not have to be reimplemented for each individual PDE solver.

In order to achieve these goals, all implementations in pyMOR follow a strict design paradigm, which enforces the usage of abstract interfaces for any high-dimensional objects (e.g., operators, vectors, discretizations, ...) that occur in the process of model order reduction. In the currently discussed context, interfaces are realized via abstract base classes (ABCs). For instance, the ABCs VectorArrayInterface and OperatorInterface indicate what kind of methods have to be provided by Operator and VectorArray objects in pyMOR. The exact behavior of these objects depends on their origin, which is usually the corresponding PDE library. For the PDE solvers FEniCS [14], deal.II [12], DUNE [13] and NGSolve [19] support has already been added to pyMOR. Accordingly, there exist implementations, so-called *specializations* of the underlying ABCs, that specify the behavior of Operator and VectorArray objects for each of these solvers. Most importantly, algorithms that are implemented using the previously mentioned interface classes can afterward be executed with any suitable specialization of the underlying ABC. Thus, model order reduction algorithms do not have to be implemented for every single external PDE solver, but only once using the interface classes for operators and vectors. In the following, we refer to these implementations as *abstract implementations.*

Furthermore, support for new PDE solvers can be easily added by specifying a few specializations for operators and vectors. Another benefit of the presented approach evolves around the parallelism of pyMOR's implementations. Since the specializations for the high-dimensional objects from the PDE libraries are based on interfaces provided by just these, the responsibility for a parallel implementation is transferred to the external library. Additionally, there exist operators in pyMOR, which easily enable parallel distributed computations with pyMOR.

4.2 Low-Rank ADI Iteration in pyMOR

In order to respect the pyMOR design paradigm introduced in the previous subsection, implementations for the library have to preserve the high level of abstraction. For the LR-ADI iteration, two abstract pyMOR components play an important role. One of them is the OperatorInterface with which the matrices A and E from Algorithm 4 can be implemented. In order to deal with the right-hand side B, the iterates V_i and W_i, as well as the solution factor Z_i, the VectorArrayInterface can be used.

An algorithm that has already been integrated into the pyMOR library is the *Gram-Schmidt Orthonormalization Method*. The algorithm can be applied to specializations of the VectorArrayInterface and plays an important role when generating shift parameters, as suggested in Section 3.6. Another question which arises in the process of implementing the LR-ADI iteration is, how to deal with standard and generalized Lyapunov equations, respectively. Using the IdentityOperator provided by pyMOR, standard Lyapunov equations can be solved with Algorithm 4 by simply replacing the matrix E with the identity. When applying the IdentityOperator to an object, it will be returned without further computations.

Besides all of the high-dimensional operations, there exist a few properties within the algorithm which can not be computed using abstract implementations available in pyMOR. Examples for these properties are the eigenvalues, which are crucial in the process of generating shift parameters. Additionally, the spectral norm needs to be computed in order to evaluate the stopping criterion of the iteration. Since these computations use small matrices (e.g., $\hat{B}^{\mathrm{T}} A \hat{B}, \hat{B}^{\mathrm{T}} E \hat{B}, W_i^{\mathrm{T}} W_i \in \mathbb{R}^{m \times m}$, $m \ll n$), non-abstract implementations can be used without violating the design standards. For the mentioned problems, the popular libraries NumPy and SciPy with the suitable functions numpy.linalg.norm and scipy.linalg.eigvals can be used.

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

M.E.S.S. [8] is the successor of the Lyapack toolbox for MATLAB, which has been developed to solve large, sparse matrix equations. A version of the M.E.S.S. library, written in the programming language C, is represented by C-M.E.S.S., which in particular provides an implementation of the LR-ADI iteration. A beneficial property of Python is that C code can be easily executed from Python scripts. Py-M.E.S.S. takes advantage of this property and represents a link between Python and the C-M.E.S.S. implementation. Accordingly, the LR-ADI iteration can either be realized using a pure pyMOR implementation or by accessing the C-M.E.S.S. library via Py-M.E.S.S.. The latter approach has already been integrated into the pyMOR library. In order to be consistent with the pyMOR design, C-M.E.S.S. is forced to use the functions for multiplying matrices and solving linear systems which are provided by the operators defined in the pyMOR library. This approach requires transferring matrices between Py-M.E.S.S. and pyMOR (i.e. VectorArrays) in each step of the iteration, which potentially becomes expensive for larger dimensions. Accordingly, we discuss a few numerical experiments which compare the run time of a pure pyMOR approach with the abstract Py-M.E.S.S. implementation.

5 Numerical Experiments

Since the performance of pyMOR and C-M.E.S.S. greatly depends on implementations of external software components like BLAS and LAPACK, we used the same versions of relevant software for all numerical tests. In order to ensure the reproducibility of the experiments, we summarized the versions in Table 1. Additionally, hardware components are just as relevant for the outcome of the experiments as the software versions. Accordingly, the used hardware is summarized in Table 2. The experiments introduced in the following subsections were executed five times, where we consistently considered the shortest run time for the comparisons. The iteration terminated once the relative Lyapunov residual specified in equation (12) was lower than 10^{-10}. Note that both approaches discussed in the subsequent paragraphs require the same amount of iteration steps and use shift parameters generated by projections using the iterate V_i as described in Section 3.6. Additionally, we focus on the equations introduced in (4) and (5) for the experiments and do not use Py-M.E.S.S. directly, but rather the adapted version, which uses pyMOR's bindings for Py-M.E.S.S.. Experiments discussed in [4] show that using Py-M.E.S.S. directly, results in drastically shorter run times compared to the approaches presented in the following subsections. However, accessing Py-M.E.S.S. directly does not preserve the high level of abstraction, since Py-M.E.S.S. can not deal with most objects created by external PDE solvers without further modifications.

Table 1 – Software and versions

Software	Version
Python	3.6
pyMOR	Forked repository [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
Operating system	Ubuntu 16.04

Table 2 – Hardware

Component	Type
Model	HP 250 G6 Notebook PC
Processor	Intel® Core™ i5-7200U CPU @ 2.50GHz, 2 701 MHz, 2 Core, 4 logical Processors, hyper-threading activated
RAM	8.00 GB
Cache	3 072 KB

5.1 Standard Lyapunov Equations

For the first experiment, we consider the heat equation on a one-dimensional segment [5]

$$\begin{aligned} \frac{\partial}{\partial t} T(z,t) &= \frac{\partial^2}{\partial z^2} T(z,t), \quad t > 0,\ z \in (0,1), \\ \hat{y}(t) &= T(1,t), \end{aligned}$$

with the output $\hat{y}(t)$, input $u(t)$ and the following boundary conditions

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

Applying the finite differences method using central differences yields the LTI system

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

defined by matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times 1}$ and $C \in \mathbb{R}^{1\times n}$, which we use to test the implementations for the LR-ADI iteration. In the following, the values which define the size of the system's matrices lie between $n = 2\,000$ and $n = 300\,000$.

In Figure 1(a), we see that the run times for both approaches behave very similarly. Overall, the interface to the Py-M.E.S.S. implementation is superior in this experiment. The largest time difference occurs at $n = 50\,000$ where Py-M.E.S.S. is 1.24 seconds faster than the pure pyMOR approach. Only for $n = 140\,000$, we can observe a better performance of the new implementation with an insignificant benefit of 0.08 seconds. Looking at the relative run time presented in Figure 1(b), we observe that Py-M.E.S.S. is at most 61% faster than the pyMOR implementation. After this peak at $n = 30\,000$, the relative run time continuously decreases.

The implementation based on Py-M.E.S.S. requires copying the iterate V_i and the residual factor W_i in each step of the iteration. Additionally, the solution factor Z_i is copied once after the algorithm terminates. Since the matrix B, and thus V_i and W_i, consist of a single column in this experiment, the amount of data that needs to be copied is relatively low. This serves as an explanation for the good performance of the Py-M.E.S.S. approach. Furthermore, an observation worth noting is that the relative run time clearly approaches 1 for large problem dimensions n. We can explain this behavior by considering the cache size mentioned in Table 2: For large n, either implementation's performance is limited by the amount of available memory, and thus, similar run times are achieved by both approaches when considering large problem dimensions. Another fact that explains this observation is that the pure pyMOR approach does not require copies, in contrast to the Py-M.E.S.S. variant. This drawback has a large impact on the performance, especially when considering high-dimensional problems. Nonetheless, the question remains as to why the implementation based on Py-M.E.S.S. performs better in the presented experiment. Regarding the answer to this question, the most relevant aspect evolves around the parallelism in the presented implementations: The Py-M.E.S.S. variant uses a single thread throughout the entire computation, whereas the pyMOR implementation operates with multiple threads. Especially for small problem dimensions, the initialization of multiple threads creates a relatively large overhead, which outweighs the performance gained through the parallelization in this experiment. For larger problems, the iteration does not significantly benefit from parallelization either, due to the previously mentioned fact that the performance is limited by the available memory. Finally, these connections serve as an explanation for the resulting run times.

5.2 Symmetric Generalized Lyapunov Equations

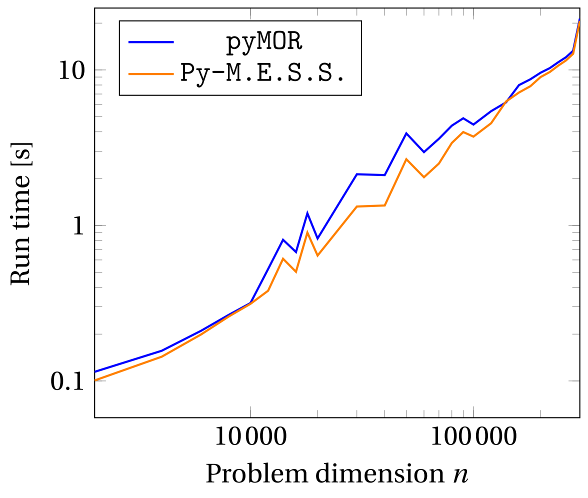
The numerical experiments in this subsection are based on the *Steel Profile* benchmark from [7, 21]. In this benchmark, an LTI system of the type

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

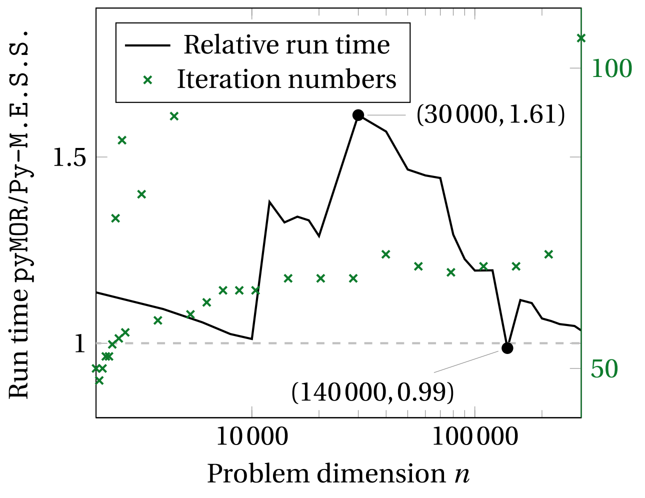
occurs in the process of modeling optimal cooling of steel profiles. For the system matrices it holds $E = E^{\mathrm{T}}$, $A = A^{\mathrm{T}} \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times 7}$. Note that this benchmark is only available with a few different values for n. Accordingly, we present all of the resulting run times in Table 3.

Looking at the results, we observe that the approach using Py-M.E.S.S. only performs better for the matrix size $n = 1\,357$. In all of the other cases, a shorter run time is achieved by the pure pyMOR implementation, where the difference does not exceed 4%.

The iterates V_i and W_i have significantly more columns, and thus, it requires more time to copy them. This provides an explanation for the superiority of the pyMOR approach compared to the Py-M.E.S.S. variant. Note that in each step of the iteration, a fixed number of columns is added to the solution factor Z_i, which also needs to be copied once the algorithm terminates.



(a) Run time



(b) Relative run time and iteration numbers

Figure 1 – Results for standard Lyapunov equations

Table 3 – Run times for symmetric generalized Lyapunov equations

n	pyMOR run time [s]	py-M.E.S.S. run time [s]	Relative run time	Iteration steps
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

5.3 Unsymmetric Generalized Lyapunov Equations

In this subsection, an experiment based on the oscillator model with $\tilde{n} \in \mathbb{N}$ masses and three dampers introduced in [26] is discussed. In this model, a second order LTI system of the type

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

with $M, D, K \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$, $F \in \mathbb{R}^{\tilde{n}\times\tilde{m}}$ and $C_1, C_2 \in \mathbb{C}^{\tilde{p}\times\tilde{n}}$ occurs which can be written as an equivalent generalized first-order LTI system of the order $n = 2\tilde{n}$ as described in equation (16). In our case, the matrices defining the Lyapunov equation are given by

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

and

$$B = \begin{bmatrix} 0 \\ F \end{bmatrix} \in \mathbb{R}^{2\tilde{n}\times\tilde{m}}.$$

Other than stated in [26], the matrices F and D are defined as

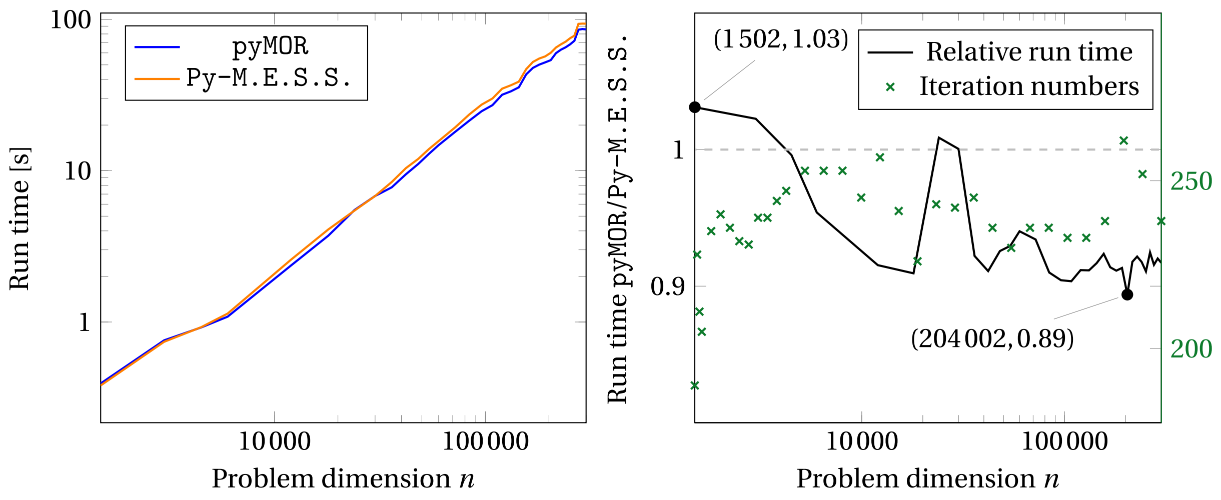
$$F = \begin{bmatrix} \mathbb{1} & 0 & 0 \\ \mathbb{1} & \mathbb{1} & 0 \\ \mathbb{1} & \mathbb{1} & \mathbb{1} \\ 1 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{\tilde{n}\times 3}$$

and

$$D = 0.02M + 0.5K \in \mathbb{R}^{\tilde{n}\times\tilde{n}},$$

in our experiments, which allows for a more convenient implementation of the model. Note that every entry in $\mathbb{1} \in \mathbb{R}^{\frac{\tilde{n}-1}{3}}$ has the value 1 and it holds $\frac{\tilde{n}-1}{3} \in \mathbb{N}$ due to the special structure of the system. Furthermore, we used values between $n = 1\,502$ and $n = 300\,002$ for the tests.

Analogous to Section 5.1, the run times for both variants behave similarly, with the significant difference that pyMOR performs better than the Py-M.E.S.S. approach in the experiment presented in this subsection. Especially for larger n, the pure pyMOR variant visibly (see Figure 2(a)) outperforms its alternative, where the largest difference is achieved at $n = 276\,002$ with 7.87 seconds. Only for the values $n = 1\,502, 3\,002, 24\,002, 30\,002$, we observe that the Py-M.E.S.S. implementation has a shorter run time with a maximal difference of 0.048 seconds.



(a) Run times (b) Relative run time and iteration numbers

Figure 2 – Results for generalized Lyapunov equations

In Figure 2(b) we observe that the approach based on Py-M.E.S.S. is approximately 3% faster than pyMOR for the value $n = 1502$. Between $n = 30002$ and $n = 300002$, there is only small fluctuations within the relative run time where pyMOR is consistently 6% to 11% faster than the Py-M.E.S.S. implementation.

Overall, the resulting run times can be explained once again by considering the amount of copies that the approach based on Py-M.E.S.S. needs to perform. For one, the matrices V_i and W_i have more columns in the currently discussed case compared to the experiment performed in Section 5.1. Additionally, significantly more iteration steps were necessary in this experiment in order to achieve convergence of the LR-ADI iteration. Since copies are performed in each step of the algorithm and the solution factor Z_i, which needs to be copied as well, becomes larger in every iteration step, the Py-M.E.S.S.-based implementation clearly has a disadvantage in this setting.

6 Conclusion

The LR-ADI iteration is an efficient approach to approximate the solution of Lyapunov equations, which benefits from low-rank formulations for the solution and residual. Additionally, the method benefits from real formulations for the low-rank factors via double iteration steps. Regarding the pyMOR library, an implementation based on Py-M.E.S.S. only performs well for Lyapunov equations where the matrix B has few columns, and the amount of iteration steps needed for convergence is rather low. Additionally, there are a few more options available in Py-M.E.S.S., which enable some fine-tuning of the LR-ADI iteration for certain problems. Most importantly, the pure pyMOR implementation of the LR-ADI iteration allows for solving large-scale Lyapunov equations in pyMOR without the need for external libraries, such as Py-M.E.S.S..

Another type of matrix equation which occurs in model order reduction and plays an important role in *linear-quadratic Gaussian balanced truncation* [16] is the algebraic Riccati equation

$$A^{\mathrm{T}}X + XA - XBB^{\mathrm{T}}X + C^{\mathrm{T}}C = 0.$$

Since pyMOR relies on Py-M.E.S.S. for solving these equations, an algorithm that approximates low-rank solutions for these, using the interface-based algorithms available in pyMOR, could be integrated in future development.

References

[1] G. Allaire and S. M. Kaber. *Numerical Linear Algebra*. Springer New York, 2008. ISBN 978-0-387-68918-0.

[2] A. C. Antoulas. *Approximation of Large-Scale Dynamical Systems*, volume 6 of *Adv. Des. Control*. SIAM Publications, Philadelphia, PA, 2005. ISBN 9780898715293.

[3] A. C. Antoulas, D. C. Sorensen, and Y. Zhou. On the decay rate of Hankel singular values and related issues. *Syst. Cont. Lett.*, 46(5):323–342, 2002.

[4] L. Balicki. *Eine abstrakte Implementierung der Low-Rank ADI Iteration für Lyapunovgleichungen in pyMOR*. Bachelor's Thesis, Otto-von-Guericke-Universität Magdeburg, 2019.

[5] L. Balicki. pyMOR forked repository. `https://github.com/lbalicki/pymor/tree/gammas`, March 2019.

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

[7] P. Benner and J. Saak. A semi-discretized heat transfer model for optimal cooling of steel profiles. In P. Benner, V. Mehrmann, and D. Sorensen, editors, *Dimension Reduction of Large-Scale*

Systems, volume 45 of *Lect. Notes Comput. Sci. Eng.*, pages 353–356. Springer-Verlag, Berlin/Heidelberg, Germany, 2005.

[8] 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`.

[9] P. Benner, P. Kürschner, and J. Saak. Efficient handling of complex shift parameters in the low-rank Cholesky factor ADI method. *Numer. Algorithms*, 62(2):225–251, 2013.

[10] P. Benner, P. Kürschner, and J. Saak. An improved numerical method for balanced truncation for symmetric second order systems. *Math. Comput. Model. Dyn. Sys.*, 19(6):593–615, 2013.

[11] 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.*, 43:142–162, 2014.

[12] deal.II authors. deal.II. `https://www.dealii.org`.

[13] DUNE developers. DUNE. `https://www.dune-project.org`.

[14] FEniCS developers. FEniCS. `https://fenicsproject.org`.

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

[16] E. Jonckheere and L. Silverman. A new set of invariants for linear systems–application to reduced order compensator design. *IEEE Transactions on Automatic Control*, 28(10):953–964, October 1983.

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

[18] R. Milk, S. Rave, and F. Schindler. pyMOR - generic algorithms and interfaces for model order reduction. *SIAM J. Sci. Comput.*, 38(5):S194–S216, 2016.

[19] NGSolve developers. NGSolve. `https://ngsolve.org`.

[20] Niconet e.V. SLICOT. `http://www.slicot.org`.

[21] Oberwolfach Benchmark Collection. Steel profile. hosted at MORwiki – Model Order Reduction Wiki, 2005.

[22] T. Penzl. A cyclic low rank Smith method for large sparse Lyapunov equations. *SIAM J. Sci. Comput.*, 21(4):1401–1418, 2000.

[23] pyMOR developers and contributors. pyMOR. `https://pymor.org/`.

[24] 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.

[25] J. M. Sanches and J. S. Marques. Image Denoising Using the Lyapunov Equation from Non-uniform Samples. In *Image Analysis and Recognition*, pages 351–358. Springer Berlin Heidelberg, 2006.

[26] 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.*, 31(1):18–39, 2009.

[27] E. L. Wachspress. *The ADI Model Problem*. Springer New York, 2013. ISBN 978-1-4614-5121-1.