SIAM J. MATRIX ANAL. APPL.
Vol. 31, No. 5, pp. 2553–2579

A RIEMANNIAN OPTIMIZATION APPROACH FOR COMPUTING LOW-RANK SOLUTIONS OF LYAPUNOV EQUATIONS*

BART VANDEREYCKEN† AND STEFAN VANDEWALLE†

Abstract. We propose a new framework based on optimization on manifolds to approximate the solution of a Lyapunov matrix equation by a low-rank matrix. The method minimizes the error on the Riemannian manifold of symmetric positive semidefinite matrices of fixed rank. We detail how objects from differential geometry, like the Riemannian gradient and Hessian, can be efficiently computed for this manifold. As a minimization algorithm we use the Riemannian trust-region method of [P.-A. Absil, C. Baker, and K. Gallivan, *Found. Comput. Math.*, 7 (2007), pp. 303–330] based on a second-order model of the objective function on the manifold. Together with an efficient preconditioner, this method can find low-rank solutions with very little memory. We illustrate our results with numerical examples.

Key words. Lyapunov matrix equations, low-rank approximations, numerical optimization on manifolds, Riemannian manifold, large-scale equations

AMS subject classifications. 65F10, 65N22, 93B40, 65K05, 90C26, 58C05

DOI. 10.1137/090764566

1. Introduction.

The subject of this paper is the problem of approximating solutions of large-scale matrix equations by iterative methods. We will focus on the generalized Lyapunov equation

$$AXM + MXA = C \tag{1.1}$$

with given symmetric matrices $A, M, C \in \mathbb{R}^{n\times n}$. Furthermore, we assume that A and M are positive definite. Equation (1.1) is a linear matrix equation in $X \in \mathbb{R}^{n\times n}$ and, by our assumptions, its solution X is symmetric and unique; see [14] or [36].

Lyapunov equations are of significant importance in control theory [8], model reduction [33], and stochastic analysis of dynamical systems [41]; see [4] for a general overview. A recurring pattern is that the solution X of the Lyapunov equation (1.1) can be associated with a Gramian of the linear time-invariant system

$$M\dot{x}(t) = Ax(t) + Bu, \quad x(0) = x_0$$

with state x and input u. These Gramians capture useful information about the energy of a system such as the controllability, observability, and covariance matrices. In general, this matrix X is dense even if the system is modeled by sparse matrices.

Large-scale matrix equations arise naturally when the system is modeled by a system of partial differential equations (PDEs). Semidiscretization in space by, e.g., the finite element method (FEM) results in a Lyapunov equation with a sparse system matrix A and a sparse mass matrix M. The dimension n of these matrices is usually

*Received by the editors July 10, 2009; accepted for publication (in revised form) by M. E. Hochstenbach June 18, 2010; published electronically September 9, 2010. The first author is a Research Assistant of the Research Foundation–Flanders (FWO). This work was partially supported by the Research Council K.U. Leuven, CoE EF/05/006 Optimization in Engineering (OPTEC) and presents results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office. The scientific responsibility rests with its authors.

http://www.siam.org/journals/simax/31-5/76456.html

†Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium (bart.vandereycken@cs.kuleuven.be, stefan.vandewalle@cs.kuleuven.be).

very large. A major challenge when solving matrix equations for such a large-scale problem is that the dense matrix X has n^2 entries. Storing this matrix when $n \gg 1000$ will be problematic, let alone solving the Lyapunov equation itself.

1.1. Low-rank approximations for large-scale problems. Direct methods for solving a Lyapunov equation, such as the Bartels–Stewart algorithm [7], compute the $n \times n$ matrix X in $O(n^3)$ floating point operations. This makes them suitable only for small-scale problems up to $n \simeq 1000$. Even optimal solvers, like multigrid, will be out of reach, since their $O(n^2)$ complexity is still too large in practice. The origin of this problem lies in the fact that these matrix equations are defined on the tensor product space $\mathbb{R}^n \times \mathbb{R}^n$, whereas the physical system, modeled by A and M, is defined on $\mathbb{R}^n$. This is termed the curse of dimensionality and it requires special consideration regarding discretizations and solvers.

A popular algebraic technique for alleviating this is approximating the solution by a low-rank matrix of rank $k \ll n$. In this way the number of unknowns is reduced to $O(nk)$. If one can compute this low-rank approximation in $O(nk^c)$ flops with c small, then solving such a large-scale matrix becomes feasible. There already exist a significant number of low-rank Lyapunov solvers using this principle. Some of these solvers are based on either the ADI or Smith method (see [37], [31], and [20]); on Krylov subspace techniques (see [39], [24], [25], and [42]); or on low-rank arithmetic (see [19]). Our method is based on Riemannian optimization.

Clearly, low-rank approximations are not always suitable. Although it is reasonable to expect that the quality of the approximation will improve with growing rank, this rank can be very large, maybe too large to be of any practical use. Indeed, consider the equation $AX + XA = 2A$ with solution $X = I$, the identity matrix. Any low-rank approximation will unavoidably be very poor. However, there are a significant number of applications where the solution does exhibit the so-called *low-rank property*: the eigenvalues of the matrix X have an exponential decay and the accuracy of the best low-rank approximation increases rapidly with growing rank. This has been studied in [38], [43], [5], and [18] for the case of $M = I$ and a low-rank matrix C in (1.1). There, bounds have been proposed that depend on the spectrum of A and, to some extent, explain the low-rank phenomenon. If we consider, e.g., a matrix A with condition number κ, these bounds suggest that we can approximate X with a relative accuracy of ϵ using a rank $k = O(\log(\kappa)\log(1/\epsilon))$; see [18, Remark 1].

The low-rank solvers cited above (except for [19]) are all based on a clever reformulation of a well-known iterative method, e.g., the ADI method, to a low-rank setting. In each step i, the algorithm can be reformulated exactly to work on a factor Y_i of the iterate $X_i = Y_i Y_i^T$ and, with each iteration, the rank of the approximation X_i will grow. If convergence is fast, the approximation will have low rank. In this case, these solvers can be very efficient, since they are relatively cheap per iteration. If convergence is slow, however, there is not much that can be done, except to keep on iterating and possibly truncating iterates along the way; see [20]. A major problem is that there seems to be little room to incorporate preconditioning compatible with the structure of a Lyapunov equation, although the method in [42] can be viewed as preconditioning with A^{-1}. Another problem is that these algorithms need to form the product of A with a square root of C. For general matrices, this square root can be costly to compute. Therefore, the algorithms are usually only applied to the case where $C = bb^T$ as this gives a trivial square root b.

The solver [19] is based on a so-called low-rank arithmetic: if the addition of two matrices of rank k is followed by a projection onto the set of rank k matrices, one can

perform a standard solver for linear systems, like multigrid, efficiently with low-rank matrices. In each iteration, the rank of the iterate will temporarily grow, only to be truncated back to low rank. This way, solver [19] circumvents the problem of slow convergence by doing a geometric multigrid iteration with low-rank matrices. It has the benefit that it can combine an optimal and fast solver with a low-rank solution, provided that this low-rank arithmetic does not destroy convergence. However, this can only be seen a posteriori and the algorithm seems rather sensitive to the rank chosen at each level of the multigrid algorithm. Furthermore, geometric multigrid is usually used with better smoothers and acting as a preconditioner, or exchanged in favor of algebraic multigrid. It is far from clear how to perform this in low-rank arithmetic.

1.2. Contributions and outline. The major contribution of this paper is the introduction of a new geometric framework for computing low-rank approximations to solutions of matrix equations. The method is based on optimizing an objective function on the Riemannian manifold of symmetric positive semidefinite matrices of fixed rank. We focus on the stable generalized Lyapunov equation and show that this Riemannian optimization approach can lead to an efficient and scalable solver which is competitive with the state-of-the-art low-rank solvers. The results on the geometric properties of the manifold are kept general in the expectation that the geometric framework can be applied also to other matrix equations, e.g., Riccati, and to similar matrix manifolds, e.g., nonsymmetric fixed rank.

We begin in section 3 by discussing the objective function in our approach, and by showing that it represents the energy norm of the error made by the low-rank approximation. After a brief introduction to Riemannian optimization in section 4, we derive some properties of the manifold essential for the Riemannian trust-region (RTR) method in section 5, which include the tangent space, the Riemannian gradient and Hessian, and the retraction. Special attention is given to the efficient implementation and application of these geometric objects.

In section 6 we will show how the previous results can be applied to the Lyapunov equation. First, a second-order model is derived, after which we discuss the efficient implementation of the RTR method. Due to the large-scale nature of the problem, we devote section 7 to the problem of finding a suitable preconditioner. Again, most attention is given to the efficient computation of this preconditioner. Next, in section 8 we present some numerical results and compare our approach with other low-rank Lyapunov solvers. We conclude in section 9 and give some possibilities for future work.

1.3. Notational conventions. The space of $n \times k$ real matrices is denoted by $\mathbb{R}^{n\times k}$. The symbol $\mathbb{R}_*^{n\times k}$ is used to denote the restriction to full-rank matrices. With $Y_\perp \in \mathbb{R}^{n\times n-k}$ we mean the normalized orthogonal complement of $Y \in \mathbb{R}^{n\times k}$, $k < n$, such that $Y_\perp^T Y = 0$ and $Y_\perp^T Y_\perp = I_k$. With $A \succ 0$ we denote that A is symmetric positive definite (s.p.d.) and, likewise, $A \succeq 0$ means that A is symmetric positive semidefinite (s.p.s.d.). The set of eigenvalues of a matrix A is denoted by $\lambda(A)$.

The directional derivative of a function f at x in the direction of ξ is denoted by $\mathrm{D}\, f(x)[\xi]$. Derivatives w.r.t. the parametrization parameter t of a curve $\gamma(t)$ are denoted by dots, so $\dot\gamma(t) := \mathrm{d}\gamma/\mathrm{d}t$ and $\ddot\gamma(t) := \mathrm{d}^2\gamma/\mathrm{d}t^2$. We do not assume anything specific about the parametrization except for sufficient differentiability.

We use the following spaces for $n \times n$ matrices: the orthogonal group O_n, the symmetric matrices $\mathrm{S}_n^{\mathrm{sym}}$, and the skew-symmetric matrices $\mathrm{S}_n^{\mathrm{skew}}$. In order to differentiate between abstract elements and concrete matrices, we use the following notation:

x, y, z are abstract elements of the manifold $\mathcal{M}$, while capital letterss X, Y, Z are matrices, as stored on a computer. The statement $x = YY^T$ means that the low-rank s.p.s.d. matrix x is implemented as a factorization with matrix Y. The inner product at an element x of a Riemannian manifold is denoted by $\langle \xi, \nu \rangle_x$, with ξ and ν tangent vectors of x. With $\mathrm{tr}(\cdot)$ we mean the trace of a matrix.

2. Basic principles of the new method. The method we propose finds a low-rank approximation of X in (1.1) by minimizing the objective function

$$f : \mathcal{M} \to \mathbb{R},\ X \mapsto \mathrm{tr}(XAXM) - \mathrm{tr}(XC) \tag{2.1}$$

on the manifold $\mathcal{M}$ of s.p.s.d. matrices of rank k,

$$\mathcal{M} = \{X : X \in \mathrm{S}_n^{\mathrm{sym}},\ X \succeq 0,\ \mathrm{rank}(X) = k\}. \tag{2.2}$$

This results in the optimization problem

$$\min_x f(x) \qquad \text{subject to } x \in \mathcal{M}. \tag{2.3}$$

We solve (2.3) by a robust second-order method in the framework of Riemannian optimization, namely the RTR method from [1]. This optimization algorithm exploits that $\mathcal{M}$ is a smooth manifold. We will show in the next section that f is related to the energy norm of the error made by the low-rank approximation. As a consequence, the minimizer of (2.3) will be locally the best low-rank solution in this norm.

Instead of minimizing f one could also minimize the residual of (1.1) on $\mathcal{M}$. This way one can solve nonsymmetric Lyapunov equations and nonlinear matrix equations. The reason for choosing f, however, is that the energy norm is in some sense more natural when solving positive definite systems like (1.1): we get concise expressions for the gradient and the Hessian, choosing a preconditioner for the iterative solver will turn out to be straightforward, and the energy norm is a better estimator for the real error than the residual. Minimizing the energy norm of the error over some (sub)space is also very common in the context of Krylov solver for s.p.d. systems; see, e.g., in a more general setting [28]. On the other hand, minimizing the residual is more intricate and is beyond the scope of this paper, which is to introduce the framework of solving matrix equations by optimization on manifolds.

3. The objective function. Matrix equation (1.1) can be written as a standard linear equation by means of vectorization. Let $\mathrm{vec}(\cdot)$ denote the operator that makes a vector from a matrix by columnwise stacking, and let $\otimes$ denote the Kronecker product. Then we have that (1.1) is equivalent to

$$\mathcal{L}\,\mathrm{vec}(X) = \mathrm{vec}(C) \quad \text{with } \mathcal{L} = A \otimes M + M \otimes A. \tag{3.1}$$

This is easy to see once one observes that $\mathrm{vec}(ABC) = (C^T \otimes A)\,\mathrm{vec}(B)$ for square matrices A, B, C of equal size. Note that the $\mathrm{vec}(\cdot)$ operator defines an isomorphism between the Euclidean spaces $\mathbb{R}^{n\times n}$ and $\mathbb{R}^{n^2}$ where the inner products relate to each other by

$$\langle X, Y \rangle_{\mathbb{R}^{n\times n}} = \langle \mathrm{vec}(X), \mathrm{vec}(Y) \rangle_{\mathbb{R}^{n^2}} \iff \mathrm{tr}(X^T Y) = \mathrm{vec}(X)^T\,\mathrm{vec}(Y).$$

See [29] and [23] for more on these properties. Further on, we will need the $\mathcal{L}$-norm

$$\|\cdot\|_{\mathcal{L}} = \sqrt{\langle \cdot, \cdot \rangle_{\mathcal{L}}} \quad \text{with } \langle u, v \rangle_{\mathcal{L}} = \langle u, \mathcal{L}v \rangle.$$

In order for this norm to make sense, $\mathcal{L}$, as defined in (3.1), should be symmetric and positive definite. We show that this is always the case by our assumptions on A and M, as detailed in the introduction.

PROPOSITION 3.1. *Suppose $A, M \succ 0$, then $\mathcal{L} \succ 0$.*

Proof. From the symmetry of A and M, we have that $\mathcal{L} = \mathcal{L}^T$. To prove that $\mathcal{L}$ is positive definite, we show that $\lambda(A \otimes M + M \otimes A) > 0$. From [36], we know that the eigenvalues of $A \otimes M + M \otimes A$ consist of the set of all numbers $\mu_{i,j} := \mu_i + \mu_j$, where μ_i, μ_j are the eigenvalues of the symmetric/positive-definite pencil $A - \lambda M$. Since $A \succ 0$, the eigenvalues $\mu_{i,j}$ of this pencil are strictly positive (see [45, Thm. 3.4.2]), and so $\mu_i + \mu_j > 0$. ∎

Now we can work out the $\mathcal{L}$-norm of the error $E = X - X_*$ of $X \in \mathcal{M}$, with X_* the true solution of (1.1). Since X is symmetric, E is symmetric too, and using the relations from above, we get

$$\begin{aligned}\|\mathrm{vec}(E)\|_{\mathcal{L}}^2 &= \mathrm{vec}(E)^T (A \otimes M + M \otimes A)\,\mathrm{vec}(E),\\ &= \mathrm{vec}(E)^T\,\mathrm{vec}(MEA) + \mathrm{vec}(E)^T\,\mathrm{vec}(AEM),\\ &= 2\,\mathrm{tr}(EMEA),\end{aligned}$$

where we used the identity $\mathrm{tr}(FG) = \mathrm{tr}(GF)$. Inserting $E = X - X_*$, we continue

$$\begin{aligned}\|\mathrm{vec}(E)\|_{\mathcal{L}}^2 &= 2\,\mathrm{tr}[(X - X_*)M(X - X_*)A],\\ &= 2\,\mathrm{tr}(XMXA) - 2\,\mathrm{tr}(XMX_*A + XAX_*M) + 2\,\mathrm{tr}(X_*MX_*A),\\ &= 2\,\mathrm{tr}(XMXA) - 2\,\mathrm{tr}(XC) + 2\,\mathrm{tr}(X_*MX_*A),\\ &= 2f(X) + 2\,\mathrm{tr}(X_*MX_*A).\end{aligned}$$

Since $\mathrm{tr}(X_*MX_*A)$ is a constant, minimizing $f(X)$, as defined in (2.1), amounts to minimizing the $\mathcal{L}$-norm of the error of X.

The optimization problem (2.3) is formulated on the set of s.p.s.d. matrices of rank k. One may wonder whether this is a restriction in comparison to optimizing over the set of s.p.s.d. matrices with rank less than or equal to k, i.e.,

$$\min_x f(x) \quad \text{s.t. } x \in \{X : X \in \mathrm{S}_n^{\mathrm{sym}},\ X \succeq 0,\ \mathrm{rank}(X) \leq k\}. \tag{3.2}$$

Intuitively, we can expect that at least one of the minimizers of problem (3.2) will not be of rank lower than k if the rank of the exact solution X_* is at least k. This can be proved rigorously as follows.

PROPOSITION 3.2. *Let $A, M \succ 0$ and $\mathrm{rank}(X_*) \geq k$, with X_* the exact solution of (1.1), then every minimizer of (3.2) has rank k.*

Proof. The proof mimics the proof of a similar result in [22, Prop. 2.4], but is based using the $\mathcal{L}$-norm instead of the Euclidean norm. Let $g(X) := \frac{1}{2}\|\mathrm{vec}\,(X - X_*)\|_{\mathcal{L}}^2 = \mathrm{tr}[(X - X_*)M(X - X_*)A]$. Since $g(X) = f(X) + c$, $c \in \mathbb{R}$, replacing f with g in (3.2) does not change the minimizers. Suppose $\hat{X}$ is such a minimizer and $\mathrm{rank}(\hat{X}) = l < k$. Then the rank one perturbation $\hat{X} + \epsilon bb^T$, with $\epsilon \geq 0$ and $b \in \mathbb{R}^{n\times 1}$ satisfying $\mathrm{tr}(bb^TMbb^TA) = 1$, will not have a function value lower than $g(\hat{X})$. This gives

$$g(\hat{X} + \epsilon bb^T) = g(\hat{X}) - 2\epsilon\,\mathrm{tr}[(X_* - \hat{X})Abb^TM] + \epsilon^2 \geq g(\hat{X}).$$

So for all ϵ and b we have that $2\,\mathrm{tr}[(X_* - \hat{X})Abb^TM] \leq \epsilon$ and this means that $\mathrm{tr}[(X_* - \hat{X})Abb^TM] = 0$. Since $\mathcal{L} \succ 0$ and b arbitrary, we can conclude $X_* = \hat{X}$. This contradicts the assumption that $\mathrm{rank}(\hat{X}) = l < k$. ∎

Hence it suffices to restrict the optimization to $\mathcal{M}$.

4. Riemannian optimization. Having defined the objective function, we need to decide on the numerical method to solve problem (2.3). It may be tempting to use standard techniques for constrained optimization to solve (2.3), but the presence of the low-rank constraint makes this very difficult. We address this problem by exploiting that $\mathcal{M}$ is a *smooth manifold*, which enables us to use the framework of numerical methods on manifolds and, in particular, *Riemannian optimization*; see, e.g., [21], [15], [27], and [3]. Riemannian optimizers abandon the flat, Euclidean space $\mathbb{R}^{n\times n}$ and formulate the problem on a curved, Riemannian manifold instead. In exchange, we can eliminate the low-rank constraint and get an unconstrained optimization problem that, by construction, will only use feasible points. Although optimizing on a smooth manifold is more complicated than optimizing on $\mathbb{R}^{n\times n}$, there are general techniques available; see [3] for an overview in case of matrix manifolds.

In this paper, we will use the RTR method of [1], which is a matrix-free and robust second-order method suitable for large-scale optimization. Simply put, the method is a generalization of the classic unconstrained trust-region (TR) method to Riemannian manifolds. Each iteration consists of two phases: first, approximating the solution of the so-called trust-region subproblem, followed by the computation of a new iterate. The algorithm is summarized below:

1: **for** $i = 1, 2, \ldots$ **do**
2: Approximately minimize the trust-region subproblem

$$\min_{\xi\in T_x\mathcal{M}, \langle\xi,\xi\rangle_x \le \Delta^2} m_x(\xi) := f(x) + \langle \operatorname{grad} f(x), \xi\rangle_x + \frac{1}{2}\langle \operatorname{Hess} f(x)[\xi], \xi\rangle_x. \tag{4.1}$$

3: Construct the new iterate $x \leftarrow R_x(\xi)$ and update the trust-region radius Δ depending on the quality of the new iterate x.
4: **end for**

The algorithm computes a series of approximations $x \in \mathcal{M}$ by using a series of second-order models $m_x : T_x\mathcal{M} \to \mathbb{R}$ associated with every x. These models are each defined on the tangent space at x, denoted by $T_x\mathcal{M}$, are based on the Riemannian gradient and Hessian and can be evaluated by means of the Riemannian metric, denoted by $\langle\cdot,\cdot\rangle_x$. The optimized tangent vector ξ that solves (4.1) is then used to get a new iterate. To do this we need a mapping $R_x : T_x\mathcal{M} \to \mathcal{M}$, called the retraction, that maps tangent vectors $\xi \in T_x\mathcal{M}$ to the manifold.

Remark 4.1. A different approach is to parametrize $x = YY^T$ with $Y \in \mathbb{R}^{n\times k}_*$ and minimize $f(YY^T)$ over $\mathbb{R}^{n\times k}_*$. This has been used successfully in the context of low-rank SDP solvers [13]. While this effectively lowers the dimension of the search space, it suffers from nonlocal minimizers which can cause problems for second-order methods like Newton's method. Indeed, every $Z = YQ$, with $Q \in \mathrm{O}_k$, is also a minimizer. A possible solution to obtain isolated minimizers (see [2] and [26]) is to optimize over the quotient space $\mathbb{R}^{n\times k}_*/\mathrm{O}_k$. This approach is in fact closely related to optimizing on $\mathcal{M}$ since these two manifolds are diffeomorphic.

5. The manifold. This section is devoted to the Riemannian geometry of $\mathcal{M}$. Manifold $\mathcal{M}$ has already been studied in [21] and [22], where the fact that $\mathcal{M}$ is a smooth manifold is proved.

THEOREM 5.1 (see [21, Chap. 5] and [22, Prop. 2.1]). *The set $\mathcal{M}$ is a smooth embedded submanifold of $\mathbb{R}^{n\times n}$ with dimension $nk - \frac{1}{2}k(k-1)$. The tangent space of*

$\mathcal{M}$ at an element x is

$$T_x\mathcal{M} = \{\Delta x + x\Delta^T : \Delta \in \mathbb{R}^{n\times n}\}. \tag{5.1}$$

In order to be able to use this manifold in our Riemannian optimization algorithm, we shall require some additional insights into its structure. First, for computational reasons, the description of the tangent space by (5.1) is not suitable since $\Delta \in \mathbb{R}^{n\times n}$. In the next subsection, we will, therefore, derive a more efficient representation. Next, after having specified the metric, we will show that the Riemannian gradient and Hessian can be computed by means of orthogonal projections onto the tangent space. Finally, we will construct the retraction mapping $R_x : T_x\mathcal{M} \to \mathcal{M}$ and outline some of its properties. In addition, we will construct a second-order expansion of this retraction that can be used to compute the Riemannian Hessian of (2.1) analytically.

5.1. The tangent space. Before deriving the tangent space, we need an efficient representation of the elements $x \in \mathcal{M}$. Since every s.p.s.d. matrix of rank k can be factorized as the product of an $n \times k$ matrix with its transpose, we make the obvious choice of $x = YY^T$ with $Y \in \mathbb{R}^{n\times k}_*$. Although this representation is not unique, this is not a problem since uniqueness of Y will never be required in our algorithm.

The dimension of the tangent space is, by definition, the dimension of the manifold. However, expression (5.1) for $T_x\mathcal{M}$ is clearly an overparametrization. A minimal parametrization is given by the following proposition.

PROPOSITION 5.2. *The tangent space of $\mathcal{M}$ at $x = YY^T$ is given by*

$$T_x\mathcal{M} = \left\{ \begin{bmatrix} Y & Y_\perp \end{bmatrix} \begin{bmatrix} S & N^T \\ N & 0 \end{bmatrix} \begin{bmatrix} Y^T \\ Y_\perp^T \end{bmatrix} : S \in \mathrm{S}_k^{sym},\ N \in \mathbb{R}^{n-k\times k} \right\}. \tag{5.2}$$

Proof. The right-hand side of (5.2) has the correct number of degrees of freedom, it is a linear space, and, by taking $\Delta = (YS/2 + Y_\perp N)(Y^TY)^{-1}Y^T$, it is included in $T_x\mathcal{M}$ of (5.1). ∎

This representation shows that we can parametrize every tangent vector with the matrices S and N and this representation uses the minimal number of parameters. Note that an algorithm will not require explicit computation and storage of $Y_\perp$, which would be unacceptably expensive. It will require only products of the form $Y_p := Y_\perp N \in \mathbb{R}^{n\times k}$; see also section 6.3.

5.2. The Riemannian metric and gradient. In order to define the gradient of an objective function, we need to decide on a metric. The aim is to smoothly assign at each point x an inner product $\langle \xi, \psi \rangle_x$ for all tangent vectors $\xi, \psi \in T_x\mathcal{M}$. This will turn $\mathcal{M}$ into a Riemannian manifold. Because $\mathcal{M}$ is an embedded submanifold of $\mathbb{R}^{n\times n}$ and $T_x\mathcal{M} \subset \mathbb{R}^{n\times n}$, the choice of the classic Euclidean inner product

$$\langle \xi, \psi \rangle_x := \mathrm{tr}(\xi^T\psi) \tag{5.3}$$

as the *Riemannian metric* is obvious. Since this metric does not depend on x, we will drop the subscript $_x$ in the notations. The norm induced from (5.3) is the Frobenius norm and will be denoted by $\|\xi\| := \sqrt{\langle \xi, \xi \rangle}$. Furthermore, orthogonality will always mean orthogonal w.r.t. (5.3).

Using the metric, we can define the normal space $N_x\mathcal{M}$ as the complementary subspace, orthogonal to $T_x\mathcal{M}$:

$$N_x\mathcal{M} := \left\{ \begin{bmatrix} Y & Y_\perp \end{bmatrix} \begin{bmatrix} E & -L^T \\ L & K \end{bmatrix} \begin{bmatrix} Y^T \\ Y_\perp^T \end{bmatrix} : E \in \mathrm{S}_k^{\mathrm{skew}}, L \in \mathbb{R}^{n-k\times k}, K \in \mathbb{R}^{n-k\times n-k} \right\}.$$

Additionally, we will denote the orthogonal projection along $N_x\mathcal{M}$ of a matrix $Z \in \mathbb{R}^{n\times n}$ onto the tangent space at x by $\mathrm{P}_x(Z)$.

The following orthogonal projectors will be convenient later on:

$$\mathrm{P}^s_x : \mathbb{R}^{n\times n} \to \{YSY^T : S \in \mathrm{S}^{\mathrm{sym}}_k\}, \tag{5.4}$$

$$\mathrm{P}^p_x : \mathbb{R}^{n\times n} \to \{Y_\perp NY^T + YN^TY_\perp^T : N \in \mathbb{R}^{n-k\times k}\}. \tag{5.5}$$

Observe that $\mathrm{range}(\mathrm{P}^s_x) \perp \mathrm{range}(\mathrm{P}^p_x)$ and that $\mathrm{P}_x = \mathrm{P}^s_x + \mathrm{P}^p_x$, which follows from $\mathrm{range}(\mathrm{P}^s_x) \cup \mathrm{range}(\mathrm{P}^p_x) = T_x\mathcal{M} = \mathrm{range}(\mathrm{P}_x)$. Considering Proposition 5.2, we see that every tangent vector $\xi \in T_x\mathcal{M}$ can be decomposed into two mutually orthogonal parts $\xi = \xi^s + \xi^p$ with $\xi^s := \mathrm{P}^s_x(\xi)$ and $\xi^p := \mathrm{P}^p_x(\xi)$. Using the standard orthogonal projectors onto the column span of Y and $Y^\perp$, namely $P_Y = Y(Y^TY)^{-1}Y^T$ and $P^\perp_Y = I - P_Y$, we can further specify the projectors (5.4)–(5.5) as

$$\mathrm{P}^s_x : Z \mapsto P_Y \frac{Z + Z^T}{2} P_Y, \tag{5.6}$$

$$\mathrm{P}^p_x : Z \mapsto P^\perp_Y \frac{Z + Z^T}{2} P_Y + P_Y \frac{Z + Z^T}{2} P^\perp_Y, \tag{5.7}$$

$$\mathrm{P}_x : Z \mapsto \mathrm{P}^s_x(Z) + \mathrm{P}^p_x(Z). \tag{5.8}$$

Since these projectors are linear operators, they can be represented as matrices. We can do this by vectorizing $\mathbb{R}^{n\times n}$ as in section 3, so that the projectors have $\mathbb{R}^{n^2}$ as domain. Applying the $\mathrm{vec}\,(\cdot)$ operator to the expressions (5.6)–(5.7), we obtain the $n^2 \times n^2$ matrices

$$P^s_x = \frac{1}{2}(P_Y \otimes P_Y)(I + \Pi), \tag{5.9}$$

$$P^p_x = \frac{1}{2}(P_Y \otimes P^\perp_Y + P^\perp_Y \otimes P_Y)(I + \Pi), \tag{5.10}$$

$$P_x = P^s_x + P^p_x. \tag{5.11}$$

Matrix Π is the symmetric permutation matrix, known in [47] as the *perfect shuffle* $S_{n,n}$, that satisfies $\mathrm{vec}(A^T) = \Pi\,\mathrm{vec}(A)$. To verify the symmetry of the projection matrices, we can use the property that Π allows one to switch Kronecker products as follows: $\Pi(A \otimes B)\Pi = B \otimes A$, for square A, B of equal size.

We will use the *Riemannian gradient* of an objective function f defined on $\mathcal{M}$ in our optimization algorithm. This gradient, denoted as $\mathrm{grad}\, f$, has the well-known interpretation of the direction of steepest ascent, but restricted to $\mathcal{M}$,

$$\frac{\mathrm{grad}\, f(x)}{\|\mathrm{grad}\, f(x)\|} = \operatorname*{arg\,max}_{\xi \in T_x\mathcal{M},\, \|\xi\|=1} \mathrm{D}\, f(x)[\xi].$$

In addition, the gradient can be identified from the relation

$$\langle \mathrm{grad}\, f(x), \xi\rangle = \mathrm{D}\, f(x)[\xi] \qquad \text{for all } \xi \in T_x\mathcal{M}. \tag{5.12}$$

With the aid of the following theorem, the Riemannian gradient can be computed also by the orthogonal projection P_x from the gradient in the embedding space.

THEOREM 5.3 (see [3, Chap. 3.6]). *Suppose a function $f : \mathbb{R}^{n\times n} \to \mathbb{R}$ has matrix G_x as Euclidean gradient in point $x \in \mathcal{M}$, then the Riemannian gradient of $f : \mathcal{M} \to \mathbb{R}$ is given by* $\mathrm{grad}\, f(x) = \mathrm{P}_x(G_x)$.

5.3. The Riemannian Hessian. The RTR method uses a second-order model that is based on the Hessian. In case of a Riemannian manifold, the *Riemannian Hessian* of a function f at $x \in \mathcal{M}$ is the unique symmetric and linear operator $\operatorname{Hess} f(x) : T_x\mathcal{M} \to T_x\mathcal{M}$ that satisfies [15]

$$\langle \operatorname{Hess} f(x)[\xi], \xi \rangle = \left. \frac{\mathrm{d}^2}{\mathrm{d}\, t^2} \right|_{t=0} f(\gamma_\xi(t)), \tag{5.13}$$

where $\gamma_\xi(t)$ is a geodesic with $\gamma_\xi(0) = x$ and $\dot{\gamma}_\xi(0) = \xi$. In differential geometry, this Hessian has a rigorous meaning by aid of the Levi–Civita connection. We will not explore this further, except from noting that it is a standard result that this connection is unique and always defined for smooth f [11, Thm. 3.3].

5.4. The retraction. As explained in section 4, we need a mapping, called the retraction R_x, to map updates in the tangent space onto the manifold. There is considerable freedom in this, but a well-chosen retraction is crucial for the performance of a Riemannian optimization algorithm. A generic choice for a retraction $R_x(\xi)$ is the so-called exponential mapping [11, Def. VII.6.3]. It is defined as the image of the geodesic $\gamma_\xi(t)$ at $t = 1$ and it always exists in a neighborhood $B_x \subset T_x\mathcal{M}$. However, the use of the exponential mapping is computationally expensive and not really necessary. Although the performance of a practical algorithm is very dependent on the efficient implementation of a specific retraction, in theory almost any smooth mapping that maps tangent vectors rigidly onto the manifold suffices and will not hamper convergence. These retractions have to fulfill some properties.

DEFINITION 5.4 (see [3, Def. 4.1.1]). *A* first-order retraction *on $\mathcal{M}$ is a mapping R, smooth around zero, from the tangent bundle $T\mathcal{M}$ onto $\mathcal{M}$ with the following properties. Let R_x be the restriction of R to $T_x\mathcal{M}$, then*

1. *$R_x(0) = x$,*
2. *Local rigidity: For every tangent vector $\xi \in T_x\mathcal{M}$, the curve $\gamma_\xi : t \mapsto R_x(t\xi)$ satisfies $\dot{\gamma}_\xi(0) = \xi$.*

As the presence of "first order" suggests, these retractions approximate the exponential mapping up to first order. The next step is a *second-order* retraction where the second-order derivative must be interpreted in the sense of section 5.3.

DEFINITION 5.5 (see [3, Prop. 5.5.5]). *A* second-order retraction *on $\mathcal{M}$ is a first-order retraction which satisfies in addition the zero initial acceleration condition:*

$$\mathrm{P}_x \left(\left. \frac{\mathrm{d}^2}{\mathrm{d}\, t^2} \right|_{t=0} R_x(t\xi) \right) = 0 \qquad \textit{for all } \xi \in T_x\mathcal{M}.$$

As far as convergence of Riemannian optimization methods goes, first-order accuracy is sufficient [3, Chap. 4], but second-order retractions enjoy a very nice property: the Riemannian Hessian of a cost function f coincides with the Euclidean Hessian of the lifted cost function $\widehat{f}_x := f \circ R_x$.

THEOREM 5.6 (see [3, Prop. 5.5.5]). *Let R_x be a second-order retraction on $\mathcal{M}$, then*

$$\operatorname{Hess} f(x) = \operatorname{Hess}(f \circ R_x)(0) \qquad \textit{for all } x \in \mathcal{M}.$$

A popular retraction is simply the projection onto $\mathcal{M}$,

$$R_x^{\mathrm{P}}(\xi) := \mathrm{P}_{\mathcal{M}}(x + \xi), \tag{5.14}$$

where the operator $\mathrm{P}_{\mathcal{M}}$ selects the nearest element to $\mathcal{M}$ in the Frobenius norm, i.e.,

$$\mathrm{P}_{\mathcal{M}} : \mathbb{R}^{n\times n} \to \mathcal{M}, \quad X \mapsto \operatorname*{arg\,min}_{z\in\mathcal{M}} \|X - z\|.$$

It is used both in the general context of retraction-based Riemannian optimization (see [3] and the references therein) and in the specific context of low-rank solvers for matrix equations (see [20] and [19]). It will come as no surprise that this choice owes greatly to the fact that we minimize the error after retraction and stay as close to the manifold as possible. However, there is a caveat: since we project onto a nonconvex set, the projection is not always well defined, except in a (possibly very small) neighborhood of x. The following theorem gives a characterization of this projection and shows that the retraction is not defined if the matrix to project does not have enough positive eigenvalues.

THEOREM 5.7 (see [21, Cor. 2.3]). *Let $A \in \mathrm{S}_n^{sym}$ have n_+ positive and n_- negative eigenvalues. Let its eigenvalue decomposition be $A = V\operatorname{diag}(\lambda_1,\dots,\lambda_n)V^T$ with $\lambda_1 \geq \cdots \geq \lambda_{n_+} > 0 > \lambda_{n-n_-+1} \geq \cdots \geq \lambda_n$. The best s.p.s.d. approximation of rank k in the Frobenius norm exists if and only if $n_+ \geq k$. One such minimizer is given by*

$$\mathrm{P}_{\mathcal{M}}(A) = V\operatorname{diag}(\lambda_1,\dots,\lambda_k,0,\dots,0)V^T.$$

We shall use this theorem to compute the retraction provided $n_+ \geq k$. Since the domain of $R_x^{\mathrm{P}}(\xi)$ is restricted to the tangent space, we can further analyze when the retraction is well defined and that it is of second order.

PROPOSITION 5.8. *Suppose $n \geq 2k$. Retraction $R_x^{\mathrm{P}}(\xi)$ defined by* (5.14) *exists if*

(i) *the rank of $\mathrm{P}_x^p(\xi)$ is $2k$, or*

(ii) *$\mathrm{P}_x^s(x+\xi)$ has k strictly positive eigenvalues.*

Condition (ii) *can always be satisfied for ξ small enough.*

Proof. Elaboration of $x+\xi$ with $x = YY^T$ and $\xi \in T_x\mathcal{M}$ as in Proposition 5.2, gives

$$x+\xi = \begin{bmatrix} Y & Y_\perp \end{bmatrix} T \begin{bmatrix} Y^T \\ Y_\perp^T \end{bmatrix} \quad \text{with } T = \begin{bmatrix} I_k + S & N^T \\ N & 0 \end{bmatrix},\ S \in \mathrm{S}_k^{\mathrm{sym}},\ N \in \mathbb{R}^{n-k\times k}.$$

Theorem 5.7 guarantees that the retraction exists if the $n\times n$ matrix T has at least k positive eigenvalues. By [34, Thm. 16.6], we have $\operatorname{inertia}(T) = \operatorname{inertia}(Z^T(I_k+S)Z) + (p,p,0)$ with $p = \operatorname{rank}(N)$ and Z a basis for the null space of N. Since $n \geq 2k$, p equals the column rank of N and Z will have $k-p \leq k$ columns. Suppose condition (i) is true, then N will be full rank, so $p = k$, and T will have at least k strictly positive eigenvalues. If condition (ii) is satisfied, then all k eigenvalues of $I_k + S$ are strictly positive. Now all $k-p$ eigenvalues of $Z^T(I_k+S)Z$ are strictly positive as well and T will have $k-p+p = k$ strictly positive eigenvalues. The fact that condition (ii) can always be satisfied for ξ small enough follows from the observation that $I_k + S \succ 0$ for S small enough. ∎

PROPOSITION 5.9. *Retraction R_x^{P} defined by* (5.14) *is a second-order retraction.*

Proof. The property $R_x^{\mathrm{P}}(0) = x$ is trivially satisfied.

First order. Smoothness around zero and local rigidity follow from Lemma 2.1 in [30] for projections on general submanifolds, which states that $\operatorname{grad}\mathrm{P}_{\mathcal{M}}(x) = \mathrm{P}_x$. The existence in a neighborhood of each x follows from Proposition 5.8 for ξ small enough.

Second order. Slight variation of [32, Thm. 4.9]. We claim that $R_x^{\mathrm{P}}(t\xi) - \gamma_\xi(t) = O(t^3)$ with $\gamma_\xi(t)$ the geodesic with foot x and direction ξ. Observing that $\gamma_\xi(t) \in \mathcal{M}$ for all t and expanding $\gamma_\xi(t)$ in series, we have the following error term:

$$\begin{aligned} R_x^{\mathrm{P}}(t\xi) - \gamma_x(t) &= R_x^{\mathrm{P}}(t\xi) - \mathrm{P}_{\mathcal{M}}\,\gamma_x(t) \\ &= R_x^{\mathrm{P}}(t\xi) - \mathrm{P}_{\mathcal{M}}\left[x + t\dot{\gamma}_x(0) + \frac{1}{2}t^2\ddot{\gamma}_x(0) + O(t^3)\right]. \end{aligned}$$

Since $\dot{\gamma}_\xi(0) = \xi$ and $\ddot{\gamma}_\xi(t)$ belongs to the normal space at $\gamma_\xi(t)$, the first-order terms cancel and the second-order term is projected out by $\mathrm{P}_{\mathcal{M}}$. Since all geodesics $\gamma_\xi(t)$ are locally the image of the exponential mapping in x [11, Thm. VII.6.9], the retraction R_x^{P} is of second order. ∎

Since R_x^{P} is a second-order retraction, we could derive the Riemannian Hessian of the objective function f by computing the Euclidean Hessian of $f \circ R_x^{\mathrm{P}}$; see Theorem 5.6. However, in order to derive an analytical expression of the Hessian, as we will do in section 6.1, it is convenient to have a second-order expansion of the retraction as well. Retraction R_x^{P} is not readily available in series, but we can construct an equivalent expansion by carefully inspecting $x + \xi$ if we split ξ into $\mathrm{P}_x^s(\xi) + \mathrm{P}_x^p(\xi)$.

PROPOSITION 5.10. *For any $x \in \mathcal{M}$, with $x^\dagger \in \mathcal{M}$ its pseudoinverse* [17]*, the mapping $R_x^{(2)} : T_x\mathcal{M} \to \mathcal{M}$, given by*

$$R_x^{(2)} : \xi \mapsto wx^\dagger w^T, \quad \text{with } w = x + \frac{1}{2}\xi^s + \xi^p - \frac{1}{8}\xi^s x^\dagger \xi^s - \frac{1}{2}\xi^p x^\dagger \xi^s, \tag{5.15}$$

where $\xi^s = \mathrm{P}_x^s(\xi)$ and $\xi^p = \mathrm{P}_x^p(\xi)$, is a second-order retraction on $\mathcal{M}$.

Proof. First, we show that $R_x^{(2)}$ is a *retraction*, i.e., a mapping $B_x \to \mathcal{M}$ in a neighborhood $B_x \subset T_x\mathcal{M}$. Choose $x = YY^T$, then $x^\dagger = Y(Y^TY)^{-2}Y^T$. This allows us to write the components of the tangent vector $\xi = \xi^s + \xi^p$ as $\xi^s = YSY^T$ and $\xi^p = Y_\perp NY^T + YN^TY_\perp^T$. Elaborating the term w in (5.15) and using the relations $\xi^p x^\dagger = Y_\perp NY^T x^\dagger$, $\xi^s x^\dagger \xi^s = YS^2Y^T$, and $\xi^p x^\dagger \xi^s = Y_\perp NSY^T$, we arrive at

$$w = \Big(Y + \frac{1}{2}YS + Y_\perp N - \frac{1}{8}YS^2 - \frac{1}{2}Y_\perp NS\Big)Y^T + YN^TY_\perp^T = ZY^T + YN^TY_\perp^T,$$

where we introduced the matrix $Z \in \mathbb{R}^{n\times k}$. Since $Y^Tx^\dagger Y = I_k$, we see that $R_x^{(2)}$ can be written as $R_x^{(2)} : \xi \mapsto wx^\dagger w^T = ZZ^T$. Thus, the image of $R_x^{(2)}$ consists of s.p.s.d. matrices of rank not larger than k. Furthermore, there will always be a neighborhood of $T_x\mathcal{M}$ that results in matrices Z of full rank k (take S small enough). Next, we prove that $R_x^{(2)}$ is of *second order*. Expanding $wx^\dagger w$ fully up to second order terms in ξ^s and ξ^p and using the relations $xx^\dagger\xi^s = \xi^s xx^\dagger = \xi^s$ and $xx^\dagger\xi^p + \xi^p xx^\dagger = \xi^p$, we see that many of the second-order terms cancel. Finally, we obtain

$$R_x^{(2)}(\xi) = x + \xi^s + \xi^p + \xi^p x^\dagger \xi^p + O(\|\xi\|^3). \tag{5.16}$$

From this, $R_x^{(2)}(0) = x$ and local rigidity (first order) are obvious. Since $\xi^p x^\dagger \xi^p = Y_\perp N^2 Y_\perp^T \in N_x\mathcal{M}$, zero acceleration (second order) is proved. ∎

6. The Riemannian trust-region method. We will use the TR algorithm adapted for Riemannian manifolds from [1], as listed in Algorithm 1. Except for the model definition, which we will explain in section 6.1, only the step calculation by aid of the retraction is different from a classic TR method. Due to the large-scale nature of the TR subproblems, we minimize them with a truncated conjugate gradient (tCG) method [46], [44] preconditioned by the projected Lyapunov equation of section 7.

Algorithm 1. Riemannian trust-region (RTR) [1] with TR strategy from [34]

Require: $\bar{\Delta} > 0, \Delta_1 \in (0, \bar{\Delta})$

1: **for** $i = 1, 2, \ldots$ **do**

2: **Model definition**: define the second-order model

$$m_i : T_{x_i}\mathcal{M} \to \mathbb{R},\ \xi \mapsto f(x_i) + \langle \operatorname{grad} f(x_i), \xi\rangle + \frac{1}{2}\langle \operatorname{Hess} f(x_i)[\xi], \xi\rangle.$$

3: **Step calculation**: compute η_i by (approximately) solving

$$\eta_i = \arg\min m_i(\xi) \quad \text{s.t. } \|\xi\| \le \Delta_i. \tag{6.1}$$

4: **Acceptance of trial point**: compute $\rho_i = (\widehat{f}(0) - \widehat{f}_{x_i}(\eta_i))/(m_i(0) - m_i(\eta_i))$.

5: **if** $\rho_i \ge 0.05$ **then**

6: Accept step and set $x_{i+1} = R^{\mathrm{P}}_{x_i}(\eta_i)$.

7: **else**

8: Reject step and set $x_{i+1} = x_i$.

9: **end if**

10: **Trust-Region radius update**: set

$$\Delta_{i+1} = \begin{cases} \min(2\Delta_i, \bar{\Delta}) & \text{if } \rho_i \ge 0.75 \text{ and } \|\eta_i\| = \Delta_i, \\ 0.25\,\|\eta_i\| & \text{if } \rho_i \le 0.25, \\ \Delta_i & \text{otherwise.} \end{cases}$$

11: **end for**

6.1. The second-order model. The RTR method needs a second-order model of f around x. A convenient choice is to use a quadratic model m_x of the lifted objective function $\widehat{f}_x := f \circ R_x$; see [3, Chap. 7]. When R_x is a second-order retraction, this model can be based on the Riemannian gradient and Hessian, i.e.,

$$m_x : T_x\mathcal{M} \to \mathbb{R},\ \xi \mapsto f(x) + \langle \operatorname{grad} f(x), \xi\rangle + \frac{1}{2}\langle \operatorname{Hess} f(x)[\xi], \xi\rangle.$$

Making use of the properties (5.12) and (5.13), it is not difficult to show that $m_x(\xi)$ is indeed accurate up to second order, i.e., $|m_x(\xi) - f(R_x(\xi))| = O(\|\xi\|^3)$.

Thanks to Theorem 5.6, we can build this model by taking a classic Taylor expansion of $\widehat{f}_x := f_x \circ R^{(2)}_x$ with $R^{(2)}_x$ the second-order expansion (5.16). Let $\xi^p = \mathrm{P}^p_x(\xi)$, then

$$\begin{aligned} \widehat{f}_x(\xi) &= f(R^{(2)}_x(\xi)) \\ &= f(x + \xi + \xi^p x^\dagger \xi^p + O(\xi^3)) \\ &= \operatorname{tr}\left[(x + \xi + \xi^p x^\dagger \xi^p + O(\xi^3))A(x + \xi + \xi^p x^\dagger \xi^p + O(\xi^3))M\right] \\ &\quad - \operatorname{tr}\left[(x + \xi + \xi^p x^\dagger \xi^p + O(\xi^3))C\right] \\ &= f(x) + \operatorname{tr}(\xi R) + \operatorname{tr}(\xi A \xi M + \xi^p R \xi^p x^\dagger) + O(\|\xi\|^3), \end{aligned}$$

where R is the residual of x, i.e., $R := AxM + MxA - C$. In the truncated expression we can easily recognize the terms that contribute to the gradient and the Hessian, namely

$$\langle \xi, \operatorname{grad} f(x)\rangle = \operatorname{tr}[\xi R] \quad \text{and} \quad \langle \xi, \operatorname{Hess} f(x)[\xi]\rangle = 2\operatorname{tr}[\xi A \xi M + \xi^p R \xi^p x^\dagger].$$

Next, we need to manipulate these expressions to obtain the gradient as a tangent vector of $T_x\mathcal{M}$ and the Hessian as a linear and symmetric mapping of $T_x\mathcal{M} \to T_x\mathcal{M}$. This can be done by judiciously inserting some projectors, e.g., $\mathrm{P}_x(\xi) = \xi$.

The computation of the gradient is almost trivial since $R = R^T$,

$$\langle \xi, \operatorname{grad} f(x) \rangle = \operatorname{tr}(\xi R) = \langle \xi, R \rangle = \langle \mathrm{P}_x(\xi), R \rangle = \langle \xi, \mathrm{P}_x(R) \rangle,$$

and so we obtain

$$\operatorname{grad} f(x) = \mathrm{P}_x(AxM + MxA - C). \tag{6.2}$$

We recover the gradient as the orthogonal projection onto $T_x\mathcal{M}$ of the gradient in the full space, as it should be according to Theorem 5.3.

As for the Hessian, we need additionally $\xi^p = \mathrm{P}_x^p(\xi)$,

$$\begin{aligned}
\langle \xi, \operatorname{Hess} f(x)[\xi] \rangle &= 2 \operatorname{tr}(\xi A \xi M + \xi^p R \xi^p x^\dagger) \\
&= 2\langle \xi, A\xi M \rangle + 2\langle \xi^p, R\xi^p x^\dagger \rangle \\
&= 2\langle \xi, \mathrm{P}_x(A\xi M) \rangle + 2\langle \mathrm{P}_x^p(\xi), R\, \mathrm{P}_x^p(\xi) x^\dagger \rangle \\
&= \langle \xi, 2\,\mathrm{P}_x(A\xi M) + 2\,\mathrm{P}_x^p(R\,\mathrm{P}_x^p(\xi)x^\dagger) \rangle,
\end{aligned}$$

and so

$$\begin{aligned}
\operatorname{Hess} f(x)[\xi] &= 2\,\mathrm{P}_x(A\xi M) + 2\,\mathrm{P}_x^p(R\,\mathrm{P}_x^p(\xi)x^\dagger) \\
&= \mathrm{P}_x(A\xi M + M\xi A) + \mathrm{P}_x^p(R\,\mathrm{P}_x^p(\xi)x^\dagger + x^\dagger\,\mathrm{P}_x^p(\xi)R).
\end{aligned} \tag{6.3}$$

From looking at this expression, it may not be clear whether all properties of a Hessian are satisfied. Luckily, we can get a much more familiar representation of the Hessian after vectorization. Recalling the derivation of the matrices (5.9)–(5.11), we apply the $\operatorname{vec}(\cdot)$ operator to (6.3),

$$\begin{aligned}
\operatorname{vec}(\operatorname{Hess} f(x)[\xi]) &= \operatorname{vec}\left(\mathrm{P}_x(A\xi M + M\xi A) + \mathrm{P}_x^p(R\,\mathrm{P}_x^p(\xi)x^\dagger + x^\dagger\,\mathrm{P}_x^p(\xi)R)\right) \\
&= P_x \operatorname{vec}(A\xi M + M\xi A) + P_x^p \operatorname{vec}\left(R\,\mathrm{P}_x^p(\xi)x^\dagger + x^\dagger\,\mathrm{P}_x^p(\xi)R\right) \\
&= P_x(A \otimes M + M \otimes A)\operatorname{vec}(\mathrm{P}_x\,\xi) + P_x^p(R \otimes x^\dagger + x^\dagger \otimes R)\operatorname{vec}(\mathrm{P}_x^p\,\xi) \\
&= \left[P_x(A \otimes M + M \otimes A)P_x + P_x^p(R \otimes x^\dagger + x^\dagger \otimes R)P_x^p\right]\operatorname{vec}(\xi).
\end{aligned}$$

Finally, the Hessian is given by the matrix

$$\mathcal{H}_x := P_x(A \otimes M + M \otimes A)P_x + P_x^p(x^\dagger \otimes R + R \otimes x^\dagger)P_x^p. \tag{6.4}$$

This matrix is clearly a linear and symmetric operator, and due to the presence of P_x and P_x^p, it has $T_x\mathcal{M}$ as its domain and range.

If we compare this Hessian to the Hessian of the full space, $\mathcal{L} = A \otimes M + M \otimes A$, we see that besides the expected projector P_x there is a "correction term" due to curvature of the low-rank constraint. Furthermore, this term can make the Hessian indefinite and renders the optimization problem nonconvex (also in the Riemannian sense). This shows the need for a robust modification of Newton's method and motivates the TR approach.

The correction term in $\mathcal{H}_x$ is necessary to have a correct second-order model, as we illustrate in Figure 6.1. There we have plotted the maximum relative error of two models in function of the norm of the tangent vector for 1000 random vectors. The

first model, denoted in black, uses $\mathcal{H}_x$ as the Hessian and the second model, in white, uses $P_x \mathcal{L} P_x$. In addition to the second-order retraction $R_x^{(2)}$, shown with $\circ$, we have also used the projection-based retraction R_x^{P}, shown with $\diamond$. It is clearly visible that only the model with $\mathcal{H}_x$ as the Hessian gives rise to a second-order model. From the figure, we can also see that R_x^{P} does not deteriorate the second-order accuracy. Indeed, this is to be expected since this retraction is also of second order.

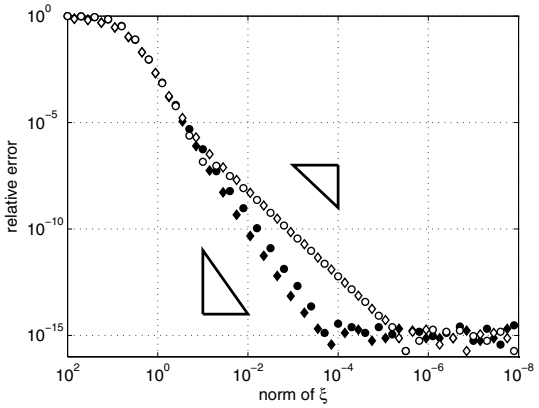

FIG. 6.1. *Relative error of the linear and quadratic models. The triangles indicate the second- and third-order convergence of the error.*

6.2. Final algorithm. In principle, the previous optimization formulated in Algorithm 1 suffices to find a low-rank approximation if the rank is known in advance. In practice, however, this rank is unknown since one is usually interested in an approximation that is better than a certain tolerance. We will take the relative residual in the Frobenius norm, i.e., $\|AXM + MXA - C\| / \|C\|$, as tolerance.

We, therefore, propose Algorithm 2 that computes a series of local optimizers to problem (2.3) with increasing rank k until the tolerance is satisfied. In order to ensure a monotonic decrease of the cost function, and thus the error, we can reuse the previous solution Y_i and append a zero column. Since the zero column does not increase the rank of $Y_{i+1} = \begin{bmatrix} Y_i & 0 \end{bmatrix}$, the Hessian would be singular due to $x^\dagger \notin \mathcal{M}$ in (6.3), and so we cannot use Y_{i+1} as the initial guess for Algorithm 1. Instead, we perform one steepest descent step to obtain a full-rank matrix Y_{i+1}. After that, we can find a minimizer with Algorithm 1. In our numerical experiments (see section 8), we found that δ is best kept small, say 2 to 6.

Due to numerical cancellation, the residual should not be computed based on the expression $\mathrm{tr}(R_i R_i)$ with $R_i = (AY_i)(Y_i^T M) + (MY_i)(Y_i^T A) - C$. Instead, we propose two ways of computing the residual. The first is similar to [37]. Suppose $x_i = Y_i Y_i^T$ and $C = cc^T$ with $c \in \mathbb{R}^{n \times l}$. This form of C is very common in control applications where $l \ll n$. After having computed the skinny QR factorization [17, Thm. 5.2.2] of $\begin{bmatrix} AY_i & MY_i & c \end{bmatrix} = Q_i T_i$, we can express the relative residual as

$$r_i = \|Ax_iM + Mx_iA - C\| / \|C\| = \left\| \begin{bmatrix} AY_i & MY_i & c \end{bmatrix} \begin{bmatrix} MY_i & AY_i & -c \end{bmatrix}^T \right\| / \left\| cc^T \right\| \tag{6.5}$$

$$= \left\| T_i \begin{bmatrix} 0 & I_k & 0 \\ I_k & 0 & 0 \\ 0 & 0 & -I_l \end{bmatrix} T_i^T \right\| / \left\| c^T c \right\|.$$

This approach requires that C is of low rank but the residual can also be computed without conditions on C. Since the matrix vector product of the residual with a given vector can be applied efficiently, it is possible to approximate r_i with a matrix-free eigensolver. Since the dominant eigenvalues of the symmetric matrix R_i are usually well separated, this estimation will converge quite fast.

The computational cost of r_i stays very moderate since the rank is small and we need only compute it once the minimizer is found. This is in contrast to [37] where the computation of the residual can be the most expensive step of the whole algorithm.

Algorithm 2. Final algorithm: RLyap

Require: initial guess $x_1 = Y_1 Y_1^T$ with $Y_1 \in \mathbb{R}_*^{n\times k_1}$, residual tolerance τ, rank increase δ.

1: **for** $i = 1, 2, \ldots$ **do**
2: **Find** x_i **as a minimizer of** (2.3): perform Algorithm 1 with $\mathcal{M}$ the manifold of rank k_i s.p.s.d. matrices.
3: **Compute the residual of** x_i: calculate r_i based on (6.5).
4: **if** $r_i \leq \tau$ **then**
5: **Solution found**: return x_i and quit.
6: **else**
7: **Increase rank**: $k_{i+1} = k_i + \delta$.
8: **Compute initial guess** $x_{i+1} = Y_{i+1} Y_{i+1}^T$: perform one step of steepest descent on $Y_{i+1} = \begin{bmatrix} Y_i & 0 \end{bmatrix}$.
9: **end if**
10: **end for**

6.3. Implementation aspects. We shall assume that the matrices A and M have a fast matrix vector product at the cost of $O(n)$. This is a very reasonable assumption when we are dealing with large-scale applications.

Factorization. Instead of using the factorization YY^T, we store x as VDV^T with $V \in \mathbb{R}_*^{n\times k}$ orthonormal and $D \in \mathbb{R}_*^{k\times k}$ diagonal, i.e., as a truncated eigenvalue decomposition. This is only slightly more costly but has the advantage that the projector P_Y can be applied as $P_V = VV^T$. The orthogonal projection onto $T_x\mathcal{M}$, given by (5.8), becomes

$$\mathrm{P}_x(Z) = VV^T \frac{Z + Z^T}{2} VV^T + (I - VV^T) \frac{Z + Z^T}{2} VV^T + VV^T \frac{Z + Z^T}{2} (I - VV^T).$$

Tangent vectors. We will also use this factorization for the tangent vectors. Suppose $x = YY^T = VDV^T$. According to Proposition 5.2, a tangent vector $\xi \in T_x\mathcal{M}$ in $x = YY^T$ is parametrized as

$$\xi = YUY^T + Y_\perp NY^T + YN^T Y_\perp^T, \quad U \in \mathrm{S}_k^{\mathrm{sym}},\ N \in \mathbb{R}^{n-k\times k}.$$

Since $Y = VR$ for some $R \in \mathbb{R}_*^{k\times k}$, the same ξ can be stored as

$$\xi = VSV^T + ZV^T + VZ^T, \quad S = RUR^T \in \mathrm{S}_k^{\mathrm{sym}}, \quad Z = Y_\perp NR^T \in \mathbb{R}^{n\times k}.$$

So we need only compute and store the matrices S and Z.

Objective function. Evaluation of $f(x)$ in $x = VDV^T$ can be done efficiently since

$$f(x) = \mathrm{tr}(xAxM) - \mathrm{tr}(xC) = \mathrm{tr}[(V^TAV)D(V^TMV)D] - \mathrm{tr}[(V^TCV)D].$$

The vectors AV, MV, and CV will be useful later on so we store them after each calculation of $f(x)$.

Gradient. The gradient at $x = VDV^T$ is given by the projection of the residual $R = AVDV^TM + MVDV^TA - C$ onto $T_x\mathcal{M}$; see (6.2). With the previous explained projectors, this becomes

$$\operatorname{grad} f(x) = VV^TRVV^T + (I - VV^T)RVV^T + VV^TR(I - VV^T).$$

After some manipulations and rearranging the terms for efficiency, we obtain that $\operatorname{grad} f(x)$ equals the tangent vector $VSV^T + ZV^T + VZ^T$ with

$$\begin{aligned} T &= (AV)D(V^TMV) + (MV)D(V^TAV) - (CV), \\ S &= V^TT, \\ Z &= T - VS. \end{aligned}$$

Hessian. The Hessian at x evaluated for $\xi = VS_\xi V^T + Z_\xi V^T + VZ_\xi^T$ is given by (6.3) where $x^\dagger = VD^{-1}V^T$. After similar but slightly more tedious manipulations as for the gradient, we get that $\operatorname{Hess} f(x)[\xi] = VSV^T + ZV^T + VZ^T$ with

$$\begin{aligned} T_1 &= (AV)S_\xi(V^TMV) + (MV)S_\xi(V^TAV) \\ &\quad + (AV)(Z_\xi^TMV) + (MV)(Z_\xi^TAV) + (AZ_\xi)(V^TMV) + (MZ_\xi)(V^TAV), \\ T_2 &= (AV)D(V^TMZ_\xi) + (MV)D(V^TAZ_\xi) - (CZ_\xi), \\ S &= V^TT_1, \\ Z &= T_1 - V(V^TT_1) + (T_2 - V(V^TT_2))D^{-1}. \end{aligned}$$

The dominating costs are the matrix vector products AZ_ξ, MZ_ξ, and CZ_ξ.

Retraction. The projection-based retraction $R_x^{\mathrm{P}}(\xi)$ can be computed with an eigenvalue decomposition of $x + \xi$; see Theorem 5.7. In general, computing this decomposition with a direct method implies an $O(n^3)$ cost. Luckily, we can exploit the fact that we need only retract from the tangent space. Suppose we need to retract $\xi = VSV^T + ZV^T + VZ^T$ in the point $x = VDV^T$, then $x + \xi$ can be written as

$$x + \xi = \begin{bmatrix} V & V_p \end{bmatrix} \begin{bmatrix} D + S & R^T \\ R & 0 \end{bmatrix} \begin{bmatrix} V^T \\ V_p^T \end{bmatrix}$$

with $Z = V_pR$ a skinny QR factorization. Observe that because $V^TZ = 0$, we have that $V^TV_p = 0$ and thus $\begin{bmatrix} V & V_p \end{bmatrix}$ is orthonormal. Since the Frobenius norm is unitary invariant, it suffices to compute the eigenvalue decomposition of a small $2k \times 2k$ matrix to project $x + \xi$:

$$R_x^{\mathrm{P}}(\xi) = P_{\mathcal{M}}(x + \xi) = \begin{bmatrix} V & V_p \end{bmatrix} P_{\widetilde{\mathcal{M}}} \left(\begin{bmatrix} D + S & R^T \\ R & 0 \end{bmatrix} \right) \begin{bmatrix} V^T \\ V_p^T \end{bmatrix}.$$

Here $\widetilde{\mathcal{M}}$ is the manifold of s.p.s.d. matrices of size $2k$ and rank k. This brings the dominating costs of this retraction to $O(k^2n)$ for the skinny QR and $O(k^3)$ for the eigenvalue decomposition.

7. Preconditioning. The computationally most expensive step of the RTR method is the solution of the TR subproblems (6.1). Since these problems are possibly very large, we solve them iteratively with the tCG algorithm [46], [44]. In each outer step of the RTR method, the second-order model is minimized with a classic matrix-free CG algorithm. This results in a number of inner iterations to solve (6.1)

up to a certain tolerance while still guaranteeing superlinear convergence of RTR. In addition, tCG employs two extra stopping criteria: the algorithm terminates if CG would use a search direction of negative or zero curvature, or if the new iterate would violate the TR bound. See [1, Alg. 2] for specific details.

Like classic CG, tCG lends itself excellently for preconditioning since a well-chosen preconditioner will have a good influence on the conditioning of each TR subproblem. The effect is that the number of inner iterations will be drastically lowered. It is, however, not directly obvious how we can define a matrix-free preconditioner that is symmetric and positive definite in all points x. In this section, we will derive such a preconditioner.

7.1. Projected Euclidean Hessian. From section 6.1 we know that the Riemannian Hessian of $f(x)$,

$$\mathcal{H}_x := P_x(A \otimes M + M \otimes A)P_x + P_x^p(x^\dagger \otimes R + R \otimes x^\dagger)P_x^p,$$

consists of two parts, namely a projection of the Euclidean Hessian $\mathcal{L} = A \otimes M + M \otimes A$ and a term involving the residual. For PDE-related Lyapunov equations, this projected Hessian $P_x\mathcal{L}P_x$ should make a good candidate for a preconditioner, since most of the bad conditioning of the TR subproblems can be attributed to $\mathcal{L}$, i.e., the PDE. This can be observed by solving problems of different size but with constant condition number for $\mathcal{L}$. In that case, the number of CG iterations required to solve the Newton equations is roughly independent of the size. Moreover, thanks to $\mathcal{L} \succ 0$, $P_x\mathcal{L}P_x$ is always symmetric and positive definite on $T_x\mathcal{M}$.

In order to see how effective this preconditioner is, we have solved a series of Lyapunov equations resulting from a five-point discretized Laplace equation on a square with zero Dirichlet boundary conditions. The right-hand side is a random symmetric matrix of rank 3. The solutions were computed with Algorithm 1 and the iteration was stopped when the norm of the gradient was below 10^{-10}.

First, we check the dependence on the size n of the system. The condition number of $\mathcal{L}$ (and presumably $\mathcal{H}_x$) will grow $\sim n$ so we can expect more tCG iterations as the subproblems become larger. In Table 7.1 we see the number of outer RTR iterations and the total number of inner tCG iterations to solve for a rank $k = 15$ approximation. For this example, a rank 15 approximation has a relative residual (6.5) of about 10^{-5}, which is sufficient to assess the performance of the preconditioner. We have included the maximum number of tCG iterations as well since the last subproblems need to be solved up to high accuracy. For the unpreconditioned problem, this maximum number grows as $\sqrt{n}$, or, in other words, as the square root of the condition number of $\mathcal{L}$. This is in correspondence with the standard convergence analysis for CG. For the preconditioned problem on the other hand, this number is small for all sizes and, more importantly, it stays bounded. Furthermore, the preconditioner reduces the total number of outer and inner iterations drastically to a number almost independent of the size of the system.

Second, in Table 7.2 we investigate the dependence on k while the size is fixed to $n = 500^2$. For the unpreconditioned problem, the maximum number of inner tCG iterations is rather independent to the rank. This seems to support the hypothesis that most of the poor conditioning of $\mathcal{H}_x$ can be attributed to $\mathcal{L}$. For the preconditioner, on the other hand, we should expect some dependence on k since we did not take the second part of the Riemannian Hessian into account. Even though we observe in Table 7.2 an increase in the total number of inner iterations with growing rank, there

TABLE 7.1
Effect of preconditioning: dependence on n for problems with fixed rank $k = 15$.

Prec.	n	150^2	200^2	250^2	300^2	350^2	400^2	450^2	500^2
None	n_{outer}	46	44	49	44	43	44	56	48
	$\sum n_{\text{inner}}$	1913	2173	2984	3158	4076	4185	5375	5622
	$\max n_{\text{inner}}$	414	529	624	731	757	858	1004	1080
$P_x\mathcal{L}P_x$	n_{outer}	39	40	42	46	47	48	47	49
	$\sum n_{\text{inner}}$	83	83	91	94	96	101	88	93
	$\max n_{\text{inner}}$	14	13	15	13	13	13	12	10

TABLE 7.2
Effect of preconditioning: dependence on k for problems with fixed size $n = 500^2$.

Precond.	k	1	4	7	10	13	16	19
None	n_{outer}	20	34	35	40	50	51	69
	$\sum n_{\text{inner}}$	4921	4949	5295	4502	6039	5682	6211
	$\max n_{\text{inner}}$	1150	1066	1050	1064	1035	1078	1066
$P_x\mathcal{L}P_x$	n_{outer}	11	28	35	35	48	48	50
	$\sum n_{\text{inner}}$	26	65	73	70	98	92	100
	$\max n_{\text{inner}}$	5	8	8	11	10	11	11

is still a significant reduction thanks to preconditioning. Furthermore, the maximum number of inner iterations seems to stay constant with growing rank.

7.2. Applying the preconditioner. It is obvious from the tables that preconditioning with $P_x\mathcal{L}P_x$ greatly reduces the total number of inner iterations. However, the preconditioner will only be effective if it can be computed sufficiently fast, ideally at a cost of $O(n)$. We will show that this is possible for $M = I$ and when $(A+\lambda I)x = b$ with $\lambda > 0$ can be solved for x in $O(n)$.

Applying the preconditioner in $x = VDV^T$ means solving $\xi \in T_x\mathcal{M}$ such that

$$(P_x\mathcal{L}P_x)(\xi) = \eta, \quad \eta \in T_x\mathcal{M}. \tag{7.1}$$

First, we write (7.1) in matrix form using the projectors (5.6) and (5.7):

$$P_V(A\xi M + M\xi A)P_V + P_V^\perp(A\xi M + M\xi A)P_V + P_V(A\xi M + M\xi A)P_V^\perp = \eta,$$

which decomposes into

$$P_V(A\xi M + M\xi A)P_V = P_V\eta P_V \quad \text{and} \quad P_V^\perp(A\xi M + M\xi A)P_V = P_V^\perp\eta P_V. \tag{7.2}$$

From now on, let $M = I$. With the factorizations as explained in section 6.3, we can take the following matrix representations for the tangent vectors of $x = VDV^T$:

$$\xi = VS_\xi V^T + Z_\xi V^T + VZ_\xi^T \quad \text{and} \quad \eta = VS_\eta V^T + Z_\eta V^T + VZ_\eta^T.$$

System (7.2) can then be written as

$$V^TAVS_\xi + S_\xi V^TAV + V^TAZ_\xi + Z_\xi^TAV = S_\eta, \tag{7.3}$$

$$P_V^\perp(AVS_\xi + AZ_\xi + Z_\xi V^TAV) = Z_\xi \quad \text{s.t. } V^TZ_\xi = 0, \tag{7.4}$$

where $S_\xi \in \mathrm{S}_k^{\text{sym}}$ and $Z_\xi \in \mathbb{R}^{n\times k}$ are the unknown matrices. By taking the eigenvalue decomposition $V^TAV = Q\Lambda Q^T$, the previous system is equivalent to

$$\Lambda\widetilde{S}_\xi + \widetilde{S}_\xi\Lambda + \widetilde{V}^TA\widetilde{Z}_\xi + \widetilde{Z}_\xi^TA\widetilde{V} = \widetilde{S}_\eta, \tag{7.5}$$

$$P_{\widetilde{V}}^\perp(A\widetilde{V}\widetilde{S}_\xi + A\widetilde{Z}_\xi + \widetilde{Z}_\xi\Lambda) = \widetilde{Z}_\xi \quad \text{s.t. } \widetilde{V}^T\widetilde{Z}_\xi = 0, \tag{7.6}$$

where $\widetilde{S}_\xi = Q^T S_\xi Q \in \mathrm{S}_k^{\mathrm{sym}}$ and $\widetilde{Z}_\xi = Z_\xi Q \in \mathbb{R}^{n\times k}$ are transformed unknown matrices and $\widetilde{V} = VQ$.

We will now eliminate $\widetilde{Z}_\xi$ from (7.6) and substitute it into (7.5). Since $\Lambda = \mathrm{diag}(\lambda_i)$, we can solve in (7.6) for each column $\widetilde{Z}_\xi(:,i)$ independently (we use the notation $(:,i)$ to denote the ith column):

$$P_{\widetilde{V}}^{\perp}(A+\lambda_i I)\widetilde{Z}_\xi(:,i) = \widetilde{Z}_\eta(:,i) - P_{\widetilde{V}}^{\perp}A\widetilde{V}\widetilde{S}_\xi(:,i) \quad \text{s.t. } \widetilde{V}^T\widetilde{Z}_\xi(:,i) = 0. \tag{7.7}$$

By writing out the projector $P_{\widetilde{V}}^{\perp}$, it is straightforward to see that this system is equivalent to the following saddle-point problem:

$$\begin{bmatrix} A+\lambda_i I & \widetilde{V} \\ \widetilde{V}^T & 0 \end{bmatrix}\begin{bmatrix} \widetilde{Z}_\xi(:,i) \\ y \end{bmatrix} = \begin{bmatrix} \widetilde{Z}_\eta(:,i) - P_{\widetilde{V}}^{\perp}A\widetilde{V}\widetilde{S}_\xi(:,i) \\ 0 \end{bmatrix} \tag{7.8}$$

with $y \in \mathbb{R}^k$. This saddle-point problem can be efficiently solved by exploiting the sparsity of A but we will postpone the details to section 7.3. For now, we can formally write (7.7) as

$$\widetilde{Z}_\xi(:,i) = \mathcal{T}_i^{-1}(\widetilde{Z}_\eta(:,i)) - \mathcal{T}_i^{-1}(P_{\widetilde{V}}^{\perp}A\widetilde{V})\widetilde{S}_\xi(:,i), \tag{7.9}$$

where $\mathcal{T}_i^{-1}(B)$ indicates solving the ith saddle-point problem, corresponding to (7.8), with right-hand side B. Now, plugging (7.9) into (7.5), we obtain

$$\Lambda\widetilde{S}_\xi + \widetilde{S}_\xi\Lambda + \begin{bmatrix} v_1 - w_1 & \cdots & v_k - w_k \end{bmatrix} + \begin{bmatrix} v_1 - w_1 & \cdots & v_k - w_k \end{bmatrix}^T = \widetilde{S}_\eta$$

with $v_i = \widetilde{V}^T A\mathcal{T}_i^{-1}(\widetilde{Z}_\eta(:,i))$ and $w_i = \widetilde{V}^T A\mathcal{T}_i^{-1}(P_{\widetilde{V}}^{\perp}A\widetilde{V})\widetilde{S}_\xi(:,i)$. Let $K_i = \lambda_i I_k - \widetilde{V}^T A\mathcal{T}_i^{-1}(P_{\widetilde{V}}^{\perp}A\widetilde{V})$, then we can isolate $\widetilde{S}_\xi$ as

$$\begin{bmatrix} K_1\widetilde{S}_\xi(:,1) & \cdots & K_k\widetilde{S}_\xi(:,k) \end{bmatrix} + \begin{bmatrix} \widetilde{S}_\xi(1,:)K_1^T \\ \vdots \\ \widetilde{S}_\xi(k,:)K_k^T \end{bmatrix} = R, \tag{7.10}$$

with the known right-hand side matrix $R = \widetilde{S}_\eta - \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix} - \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix}^T$. Equation (7.10) is a linear equation in $\widetilde{S}_\xi$ with a special block structure. By vectorizing as in section 3, it is straightforward to see that the first part of (7.10) satisfies

$$\mathrm{vec}\begin{bmatrix} K_1\widetilde{S}_\xi(:,1) & \cdots & K_k\widetilde{S}_\xi(:,k) \end{bmatrix} = \begin{bmatrix} K_1 & & \\ & \ddots & \\ & & K_k \end{bmatrix}\begin{bmatrix} \widetilde{S}_\xi(:,1) \\ \vdots \\ \widetilde{S}_\xi(:,k) \end{bmatrix} = \mathcal{K}\,\mathrm{vec}\,(\widetilde{S}_\xi),$$

where $\mathcal{K}$ denotes the $k^2 \times k^2$ block-diagonal matrix $\mathrm{diag}(K_i)$. For the second part, we can use $\mathrm{vec}(X) = \Pi\,\mathrm{vec}(X^T)$ with Π the perfect-shuffle matrix to obtain

$$\mathrm{vec}\begin{bmatrix} \widetilde{S}_\xi(1,:)L_1^T \\ \vdots \\ \widetilde{S}_\xi(k,:)L_k^T \end{bmatrix} = \Pi\,\mathrm{vec}\begin{bmatrix} L_1\widetilde{S}_\xi^T(1,:) & \cdots & L_k\widetilde{S}_\xi^T(k,:) \end{bmatrix} = \Pi\mathcal{K}\,\mathrm{vec}(\widetilde{S}_\xi^T).$$

Finally, the whole equation (7.10) can be written as a linear system of size k^2:

$$\mathcal{K}\,\mathrm{vec}(\widetilde{S}_\xi) + \Pi\mathcal{K}\,\mathrm{vec}(\widetilde{S}_\xi^T) = \mathrm{vec}(R) \quad \Longleftrightarrow \quad (\mathcal{K} + \Pi\mathcal{K}\Pi)\,\mathrm{vec}(\widetilde{S}_\xi) = \mathrm{vec}(R). \tag{7.11}$$

After solving (7.11) for $\widetilde{S}_\xi$ we obtain $\widetilde{Z}_\xi$ from (7.9). Undoing the transformations by Q, we get $S_\xi = Q\widetilde{S}_\xi Q^T$ and $Z_\xi = \widetilde{Z}_\xi Q^T$, and thus ξ, such that (7.1) is satisfied.

In case $M \neq I$, we simply approximate M by I and use the previous techniques. Although this is a very crude approximation of M, the obtained preconditioner seems to work quite well in the numerical experiments. The reason is that for generalized Lyapunov equations M usually represents the Galerkin mass matrix of a FEM discretization. For quasi-uniform meshes with shape-regular elements this mass matrix is essentially a scaled identity matrix; see [16, Chap. 1.6]

7.3. Cost. There are two dominating costs for applying the preconditioner, namely solving the saddle-point problems (7.8) and solving the linear system (7.11).

Regarding (7.8), there is a vast amount of literature for solving large and sparse saddle-point problems of this kind; see [10] for an overview. The solution technique of the previous section requires solving $\mathcal{T}_i(X) = B$ for two different right-hand sides, or, equivalently, $B = [\widetilde{Z}_\eta(:,i) \quad P_{\widetilde{V}}^{\perp} A\widetilde{V}] \in \mathbb{R}^{n\times k+1}$; see (7.9). In our case, k is rather small, so we can solve $\mathcal{T}_i(X) = B$ by eliminating the (negative) Schur complement $S_i = \widetilde{V}^T(A+\lambda_i I)^{-1}\widetilde{V}$; see [10, sect. 5]. This gives

$$N = S_i^{-1}(\widetilde{V}^T(A+\lambda_i I)^{-1}B),$$
$$X = (A+\lambda_i I)^{-1}B - (A+\lambda_i I)^{-1}\widetilde{V}N.$$

For each $\mathcal{T}_i$, applying S_i^{-1} means an $O(k^3)$ cost for the Cholesky factorization of the dense matrix S_i and for the forward and back substitution to solve for $N \in \mathbb{R}^{k\times k+1}$. In addition, we need to apply $(A+\lambda_i I)^{-1}$ to B and $\widetilde{V}$. Assuming an optimal solver for the sparse s.p.d. matrix $A+\lambda_i I$, this implies a cost of $O(nk)$. In total, we get a cost of $O(nk^2) + O(k^4)$ to solve all k saddle-point problems. Usually $k \ll n$ and the $O(nk^2)$ cost dominates.

Since in every inner iteration of RTR the iterate x stays fixed, the $n \times k$ matrices $(A+\lambda_i I)^{-1}\widetilde{V}$ and $(A+\lambda_i I)^{-1}P_{\widetilde{V}}^{\perp}A\widetilde{V}$ remain the same and can be reused. If one is willing to cache these results for all k shifts, there is a significant speedup possible. The downside is that the memory requirements grow from $O(nk)$ to $O(nk^2)$.

Equation (7.11) is a linear and symmetric system of size k^2. Solving this vectorized system by a dense factorization results in an $O(k^6)$ cost which is prohibitively large, even for small k. However, in practice the equation can be solved much faster with an iterative method like CG. In all problems, we have observed convergence in only $O(\log k)$ steps. Together with (7.10) as a matrix-vector product with cost $O(k^3)$, this results in an empirical $O(k^3 \log k)$ cost for solving (7.11).

8. Numerical results. In this section we illustrate the performance of our Riemannian optimization approach with some numerical experiments. The computations were done with MATLAB R2009b on a 64-bit Intel Pentium Xeon 2.66 GHz with $\epsilon_{mach} \simeq 2 \cdot 10^{-16}$. The RTR algorithm was implemented using GenRTR [6], a generic MATLAB package for Riemannian trust-region.

8.1. Quality of the low-rank solutions. In Figure 8.1 we investigated the quality of the solutions from Algorithm 1 compared to the best rank k approximations of the exact solution. The generalized Lyapunov equation was based on the RAIL benchmark from [9] of size $n = 1357$. We simplified this benchmark to have a rank one matrix $C = B_1 B_1^T$ with B_1 the first column of the B matrix in [9]. Since RTR minimizes the error in the energy norm, we should expect that the best rank k approximations always have a better accuracy measured in the Frobenius norm. This

is verified in the left panel of Figure 8.1. In addition we see that the difference stays rather small and behaves uniformly in the rank. In other words, the RTR approximations are nearly as good as the best rank k approximations. Surprisingly, the errors of the residual of the RTR approximations are a little better than the best rank k approximations, as seen in the right panel of Figure 8.1.

Next, we performed the same comparison with the KPIK algorithm from [42]. Each step of KPIK appends a new column to the factor Y of the solution $x = YY^T$, and thus the rank will increase with every step. Although each step of the KPIK algorithm is cheap in comparison with the RTR method, in Figure 8.1 we can clearly see that these low-rank solutions are far from optimal.

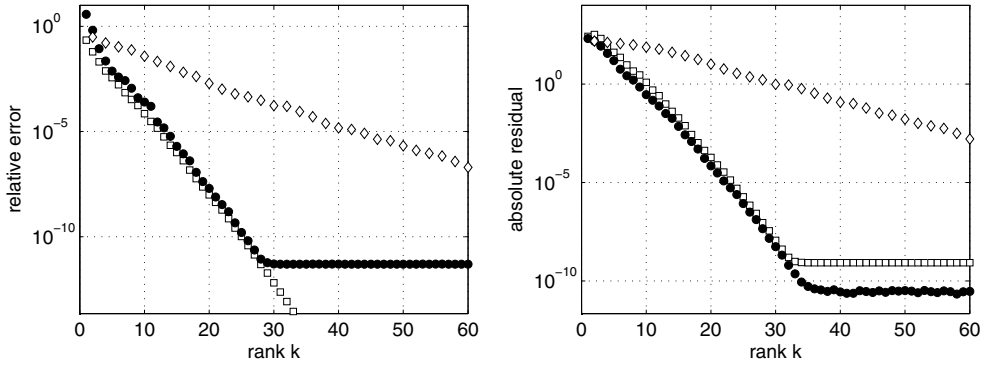

FIG. 8.1. *The relative error and the absolute residual for the simplified RAIL benchmark with* $n = 1357$. *The best rank k approximations* □ *are compared to approximations computed with RTR* • *and KPIK* ◊.

8.2. Comparison with existing low-rank solvers. In the following we will report on the performance of Algorithm 2, named RLyap, compared to two state-of-the-art low-rank Lyapunov solvers, namely the modified CFADI method and KPIK. For CFADI we used LyaPack 1.8 [40] and for KPIK the implementation of [42]. In addition, we will use an algebraic multigrid preconditioner [12] for solving iterative systems. All default options were kept.

The reported timings are wall times that include all necessary computations like setting up the preconditioner, computing sparse Cholesky factors, and determining shifts.

8.2.1. Accuracy of the linear systems. Most low-rank solvers, including RLyap with the preconditioner of section 7, need to solve (shifted) linear systems. For large-scale problems these systems will need to be solved iteratively by, e.g., a Krylov method preconditioned with AMG. Depending on the accuracy of the desired low-rank solution, KPIK and CFADI need to solve these systems quite accurately. An advantage of our method is that the AMG preconditioner can be used directly as approximate inverse for the shifted system in section 7.2. So instead of accurately solving shifted linear systems, we use AMG on the shifted system to precondition the actual TR subproblem. Since the AMG preconditioner is spectrally equivalent with the original shifted system, we get again a sound preconditioner for the TR subproblems.

We will investigate numerically how the accuracy of this inverse influences RLyap and CFADI. For RLyap we take a fixed number of V-cyles in the preconditioner. For

CFADI, each linear system is solved by a fixed number of CG steps, preconditioned by AMG. We can see in Table 8.1 that RLyap converges with every choice of number of V-cycles, while CFADI stagnates if the linear system is solved too crudely. Apparently only one V-cycle in RLyap's preconditioner gives the fastest wall time. One can argue that even one V-cycle is still too costly for RLyap's preconditioner. We did not pursue this further, but a more careful convergence analysis of the inner-outer tolerances could give a significant speedup.

TABLE 8.1
Effect of the accuracy when solving linear systems for different V-cycles of AMG. The corresponding average reduction of the residual is indicated by avg. tol.

	V-cycles	1	2	3	4	5	6	7
	avg. tol.	3e−1	5e−2	2e−3	7e−5	3e−6	1e−7	5e−9
CFADI	time (s.)	26	32	37	54	63	65	68
	rel. res.	7e−1	9e−2	5e−3	3e−4	2e−5	8e−7	5e−7
RLyap	time (s.)	42	52	70	85	114	134	155
	rel. res.	2e−7	2e−7	2e−7	2e−7	2e−7	2e−7	2e−7

8.2.2. Without mass matrix. The Lyapunov equation (1.1) with A the two-dimensional Poisson problem on the square, $M = I$, and C a rank one right-hand side is a much-used benchmark. The results of the performance of RLyap, CFADI, and KPIK for a relative residual of 10^{-6} is listed in Table 8.2. Solving $(A + \lambda I)x = b$ was done with a sparse direct solver (CHOLMOD with AMD reordering) or with an iterative solver (CG preconditioned with AMG). In the case of KPIK and CFADI, the inner tolerance for the iterative solver was 10^{-10} (a lower tolerance resulted in stagnation for the biggest problem) whereas for RLyap only one AMG V-cycle sufficed; see also section 8.2.1.

It is clear from the table that RLyap is significantly slower with a direct solver than with an iterative solver, while the situation is reversed for CFADI and KPIK. This seems to indicate that the preconditioner in RLyap is too crude to warrant solving it very accurately, i.e., by a direct solver. We remark, however, that it is possible to get a significant speedup for the sparse direct solver since the symbolic factorization has to be done only once. For this problem, this phase actually accounts for almost half the time of the total solve.

Since our MATLAB implementation of RLyap is not competitive with a sparse direct solver, we will only compare the iterative approach. We can observe that RLyap performs quite well for this problem: it is only slightly slower than the fastest method, KPIK, and it is several times faster than CFADI. Furthermore, the difference with KPIK becomes smaller for bigger problems: while the smallest problem is 62% slower, the largest is only 25% slower. If we compare the ranks of the solutions, we observe that RLyap always delivers the smallest rank. The rank of KPIK is significantly higher and grows with problem size.

The previous problem can be regarded as relatively easy since the grid is very isotropic. We, therefore, constructed a problem with a more irregular triangular mesh by discretizing the three-dimensional Poisson equation on the cube with piecewise linear finite elements. The right-hand side is $C = bb^T$ with b the FEM discretization of the unit function. This is a problem were CHOLMOD cannot be used so this shows the necessity of the iterative approach. The results of the comparison is visible in Table 8.3 and are almost similar to the previous 2D problem. Again KPIK is the fastest method, but now CFADI performs significantly better than RLyap for the

TABLE 8.2
Performance for the finite difference discretized 2D Poisson problem on the square. Tolerance on the relative residual was 10^{-6}.

		CHOLMOD with AMD			PCG with AMG		
		RLyap	CFADI	KPIK	RLyap	CFADI	KPIK
$n = 500^2$	time (s.)	101	55	13	40	70	24
	rank X	12	20	36	12	19	36
$n = 1000^2$	time (s.)	513	104	65	175	310	118
	rank X	12	20	38	12	18	38
$n = 1500^2$	time (s.)	1495	267	189	443	811	354
	rank X	12	20	19	12	19	44

bigger problem. The reason that RLyap is slower for the bigger problem is that the quality of the AMG deteriorates drastically for the bigger problem.

TABLE 8.3
Performance for the finite element discretized 3D Poisson problem on the cube. Tolerance on the relative residual was 10^{-6}.

		PCG with AMG		
		RLyap	CFADI	KPIK
$n = 132745$	time (s.)	81	115	58
	rank X	14	14	34
$n = 306006$	time (s.)	275	328	168
	rank X	15	15	36
$n = 1068660$	time (s.)	1750	1430	882
	rank X	16	16	46

8.2.3. With mass matrix. We will now report how RLyap performs with a mass matrix. We took the RAIL benchmark [9] with the outer product of the first column of the B-matrix as the right-hand side. First, we solved a Lyapunov equation by neglecting M, i.e., we take only A of the benchmark. After that, we solved the actual system with M. The results are visible in Table 8.4. We can see that if $M = I$, all solvers behave as expected. If we use the actual system with $M = I$, the results are very different. For the biggest problem CFADI is twice as fast as KPIK, while RLyap performs disproportionately poorly. This can be explained by the approximation of $M = I$ in RLyap's preconditioner: since the condition number of M is about 400, this approximation is apparently too crude to give an efficient solver.

TABLE 8.4
Comparison for the simplified RAIL benchmark. Linear systems solved by PCG with AMG.

		Simplified $M = I$			Orig. M	$M = LL^T$	
		RLyap	CFADI	KPIK	RLyap	CFADI	KPIK
$n = 5177$	time (s.)	3.9	2.6	1.4	29	5.9	5.3
	rank X	22	21	54	26	28	80
$n = 20209$	time (s.)	13	12	7.8	103	39	49
	rank X	22	25	70	26	31	114
$n = 79841$	time (s.)	76	61	46	447	249	552
	rank X	29	28	96	30	34	170

8.2.4. Right-hand side matrix of high rank. An advantage of the proposed method is that it does not impose conditions on the form of the right-hand side matrix C. Since all matrices will eventually be projected onto the tangent space,

RLyap requires only the product of a vector with C. KPIK and CFADI, on the other hand, require that C is factored as $C = BB^T$ with $B \in \mathbb{R}^{n\times l}$. Furthermore, for computational efficiency it is important that l is small since systems of the form $(A + \lambda I)^{-1}B$ have to be solved in each step. If the numerical rank of the solution X is much smaller than l, these methods will not be efficient.

We will now show that RLyap can indeed be more efficient in solving systems when l is large. All of the existing benchmark examples for low-rank Lyapunov solvers, however, are formulated for relatively low-rank C; in fact, many use only a rank one matrix. We will, therefore, construct an example that has a matrix C with relatively high rank compared to rank of the approximation of X. Take A_n the $n \times n$ tridiagonal matrix of a discretized 1D Laplacian. Consider the Lyapunov equation

$$A_n X + X A_n = C \quad \text{with } C := A_n^{-1} \begin{bmatrix} 0_{n_2\times n_2} & 0_{n_2\times n_1} \\ 0_{n_1\times n_2} & I_{n_2\times n_2} \end{bmatrix} A_n^{-1} \tag{8.1}$$

and $n_2 = \lfloor n/10 \rfloor$ and $n_1 = n - n_2$.

We can solve (8.1) without modification with RLyap. The experimental results for different meshes are visible in Table 8.5. We used a direct solver for the preconditioner and a tolerance of 10^{-6} for the relative residual. This tolerance could not be satisfied for the biggest problem, so we relaxed in addition the tolerance to $5 \cdot 10^{-5}$. Now the method succeeds in finding a low-rank approximation for all problems.

Although matrix C is by construction available in factored form, its rank will grow as $n/10$. This is clearly unsuitable for CFADI or KPIK. Thanks to the pre- and postmultiplying by A^{-1}, matrix C will have decaying eigenvalues and a reasonably good low-rank approximation. One can compute such a rank k_C approximation with a matrix-free eigenvalue solver, e.g., `eigs` in MATLAB. The decay is, however, slow and only algebraic so the rank of the resulting approximation C_k can still be rather high. Furthermore, since C_k is an approximation of the true right-hand side C the solution of $AX + XA = C_k$ will again be an approximation of the true solution of (8.1). So it is important to take k_C sufficiently high but not too high. For the smallest problem, $k_C = 15$ turned out to be the smallest rank for which the tolerance on the residual can still be satisfied. We take in addition $k_C = 30$ to examine the effect of k_C.

With these low-rank matrices C_{15} and C_{30} at hand, we solved the same systems again with CFADI and RLyap. We did not compare with KPIK since this requires a block-Krylov implementation which is currently not available. In all cases, except the smallest problem, RLyap outperformed the modified CFADI method w.r.t. wall time, rank of the solution, and final accuracy.

RLyap with C_{15} was as expected faster than with C_{30} and both were faster than with C. The influence on the rank k_C was not big, however, and solving directly with the real C is much more user friendly. The CFADI method, on the other hand, is very sensitive to the rank of C_k, both in time and accuracy. Take, for example, the problem with size $n = 40000$. Here CFADI with C_{15} stagnates while RLyap succeeds in solving the problem. If the right-hand side is C_{30}, CFADI succeeds in solving the problem again, but the method became twice as slow. In addition, we see that RLyap gives more accurate final approximations than CFADI.

9. Conclusions and outlook. We proposed a new algorithm, RLyap, to solve for low-rank solutions of Lyapunov equations based on optimization on the manifold of fixed-rank matrices. The performance of RLyap seems to be between that of KPIK and ADI, although there are problems where the situation is reversed. The example

TABLE 8.5

Experimental results for problem (8.1) *computed with RLyap and CFADI for different meshes. The right-hand side matrices were the real matrix, C, and rank* 15 *and* 30 *approximations, C_{15} and C_{30}, respectively. Timings between parentheses indicate that convergence stagnated and tolerance τ on the residual could not be satisfied.*

	Solver	RLyap	CFADI	RLyap	CFADI	RLyap
	rhs	C	C_{15}	C_{15}	C_{30}	C_{30}
$n = 20000$	time (s.)	35.7	**15.4**	32.6	40.3	34.3
$\tau =$ 1e−6	rank X	25	35	27	49	25
	residual	9.27e−7	6.39e−7	7.03e−7	5.53e−7	9.27e−7
$n = 40000$	time (s.)	70.3	(38.7)	**48.9**	111.2	61.6
$\tau =$ 1e−6	rank X	23	35	25	49	27
	residual	9.87e−7	2.67e−6	9.30e−7	8.61e−7	9.86e−7
$n = 80000$	time (s.)	169.7	(103.1)	**116.8**	(232.1)	128.4
$\tau =$ 1e−6	rank X	25	35	25	50	25
	residual	9.89e−7	2.68e−6	9.81e−7	2.98e−6	9.90e−7
$n = 160000$	time (s.)	(560.6)	(183.4)	(400.1)	(404.9)	(516.3)
$\tau =$ 1e−6	rank X	27	36	31	50	30
	residual	1.74e−6	2.85e−5	1.98e−6	2.73e−5	1.56e−6
$n = 160000$	time (s.)	176.8	139.5	**104.7**	300.9	125.5
$\tau =$ 5e−5	rank X	12	33	12	48	12
	residual	1.44e−5	3.57e−5	3.35e−5	3.47e−5	1.44e−5

in section 8.2.4 shows that the solver can be significantly faster and yet be more user friendly when the rank of the solution is lower than that of the right-hand side matrix. Noting that CFADI and KPIK perform already quite well for symmetric problems, this leads us to think that the Riemannian approach is promising for nonsymmetric problems, where, e.g., the shift determination for CFADI is much more difficult.

Even though RLyap performs adequate, there is still room for improvement, especially w.r.t. the preconditioner. In the current implementation, most of the time is spent solving shifted linear systems with constantly changing shifts. Since these shifts do not always change significantly and the preconditioner does not need to be solved very accurately, there is great potential to lower the computational burden. A similar observation was made in [35] where the multiple shifts could be avoided by using a subspace technique and a single shift.

Acknowledgments. We thank Lars Grasedyck for helpful discussions on matrix equations and Sue Thorne for a prerelease version of the AMG solver in [12]. We gratefully acknowledge Pierre-Antoine Absil for introducing us to Riemannian optimization and his subsequent support. Finally, we thank both referees for carefully checking the paper and providing a number of helpful remarks.

REFERENCES

[1] P.-A. ABSIL, C. BAKER, AND K. A. GALLIVAN, *Trust-region methods on Riemannian manifolds*, Found. Comput. Math., 7 (2007), pp. 303–330.

[2] P.-A. ABSIL, M. ISHTEVA, L. DE LATHAUWER, AND S. VAN HUFFEL, *A geometric Newton method for Oja's vector field*, Neural Comput., 21 (2009), pp. 1415–1433.

[3] P.-A. ABSIL, R. MAHONY, AND R. SEPULCHRE, *Optimization Algorithms on Matrix Manifolds*, Princeton University Press, Princeton, NJ, 2008.

[4] A. C. ANTOULAS, *Approximation of Large-Scale Dynamical Systems*, Adv. Des. Control 6, SIAM, Philadelphia, 2005.

[5] A. C. ANTOULAS, D. C. SORENSEN, AND Y. ZHOU, *On the decay rate of Hankel singular values and related issues*, Systems Control Lett., 46 (2002), pp. 323–342.

[6] C. Baker, P.-A. Absil, and K. Gallivan, *GenRTR: A Riemannian Optimization Package*, available online at http://www.math.fsu.edu/∼cbaker/GenRTR.
[7] R. H. Bartels and G. W. Stewart, *Solution of the matrix equation $AX + XB = C$*, Comm. ACM, 15 (1972), pp. 820–826.
[8] P. Benner, *Control Theory*, Handbook of Linear Algebra, Chapman & Hall/CRC, Boca Raton, FL, 2006.
[9] P. Benner and J. Saak, *Efficient numerical solution of the LQR-problem for the heat equation*, Proc. Appl. Math. Mech., 4 (2004), pp. 648–649.
[10] M. Benzi, G. H. Golub, and J. Liesen, *Numerical solution of saddle point problems*, Acta Numer., 14 (2005), pp. 1–137.
[11] W. M. Boothby, *An Introduction to Differentiable Manifolds and Riemannian Geometry*, 2nd ed., Pure Appl. Math. 120, Academic Press, Orlando, 1986.
[12] J. Boyle, M. D. Mihajlovic, and J. A. Scott, *HSL_MI20: An efficient AMG preconditioner for finite element problems in 3d*, Internat. J. Numer. Methods Engrg., 82 (2010), pp. 64–98.
[13] S. Burer and R. D. C. Monteiro, *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization*, Math. Program., 95 (2003), pp. 329–357.
[14] K.-W. E. Chu, *The solution of the matrix equation $AXB - CXD = E$ and $(YA - DZ, YC - BZ) = (E, F)$*, Linear Algebra Appl., 93 (1987), pp. 93–105.
[15] A. Edelman, T. A. Arias, and S. T. Smith, *The geometry of algorithms with orthogonality constraints*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 303–353.
[16] H. Elman, D. Silvester, and A. Wathen, *Finite Element and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics*, Oxford University Press, New York, 2005.
[17] G. H. Golub and C. F. V. Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1989.
[18] L. Grasedyck, *Existence of a low rank or $\mathcal{H}$-matrix approximant to the solution of a Sylvester equation*, Numer. Linear Algebra Appl., 11 (2004), pp. 371–389.
[19] L. Grasedyck and W. Hackbusch, *A multigrid method to solve large scale Sylvester equations*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 870–894.
[20] S. Gugercin, D. C. Sorensen, and A. C. Antoulas, *A modified low-rank Smith method for large-scale Lyapunov equations*, Numer. Algorithms, 32 (2003), pp. 27–55.
[21] U. Helmke and J. B. Moore, *Optimization and Dynamical Systems*, Springer-Verlag, London, 1994.
[22] U. Helmke and M. A. Shayman, *Critical points of matrix least squares distance functions*, Linear Algebra Appl., 215 (1995), pp. 1–19.
[23] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1991.
[24] I. M. Jaimoukha and E. M. Kasenally, *Krylov subspace methods for solving large Lyapunov equations*, SIAM J. Numer. Anal., 31 (1994), pp. 227–251.
[25] K. Jbilou and A. J. Riquet, *Projection methods for large Lyapunov matrix equations*, Linear Algebra Appl., 415 (2006), pp. 344–358.
[26] M. Journée, F. Bach, P.-A. Absil, and R. Sepulchre, *Low-rank optimization for semidefinite convex problems*, SIAM J. Optim., 20 (2010), pp. 2327–2351.
[27] O. Koch and C. Lubich, *Dynamical low-rank approximation*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 434–454.
[28] D. Kressner and C. Tobler., *Krylov subspace methods for linear systems with tensor product structure*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 1688–1714.
[29] P. Lancaster and M. Tismenetsky, *The Theory of Matrices*, 2nd ed., Academic Press, Orlando, 1985.
[30] A. S. Lewis and J. Malick, *Alternating projections on manifolds*, Math. Oper. Res., 33 (2008), pp. 216–234.
[31] J.-R. Li and J. White, *Low-rank solution of Lyapunov equations*, SIAM Rev., 46 (2004), pp. 693–713.
[32] S. A. Miller and J. Malick, *Newton methods for nonsmooth convex minimization: Connections among $\mathcal{U}$-Lagrangian, Riemannian Newton and SQP methods*, Math. Program., 104 (2005), pp. 609–633.
[33] B. C. Moore, *Principal component analysis in linear systems: Controllability, observability, and model reduction*, IEEE Trans. Automat. Control, 26 (1981), pp. 17–32.
[34] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Ser. Oper. Res., Springer-Verlag, New York, 1999.

[35] R. Nong and D. C. Sorensen, *A Parameter Free ADI-like Method for the Numerical Solution of Large Scale Lyapunov Equations*, CAAM TR09-16, Computational and Applied Mathematics, Rice University, Houston, TX, 2009.
[36] T. Penzl, *Numerical solution of generalized Lyapunov equations*, Adv. Comput. Math., 8 (1998), pp. 33–48.
[37] T. Penzl, *A cyclic low-rank Smith method for large sparse Lyapunov equations*, SIAM J. Sci. Comput., 21 (2000), pp. 1401–1418.
[38] T. Penzl, *Eigenvalue decay bounds for solutions of Lyapunov equations: The symmetric case*, Systems Control Lett., 40 (2000), pp. 139–144.
[39] Y. Saad, *Numerical solution of large Lyapunov equations*, in Signal Processing, Scattering, Operator Theory, and Numerical Methods, M. A. Kaashoek, J. H. V. Schuppen, and A. C. M. Ran, eds., Birkhäuser Boston, Boston, MA, 1990, pp. 503–511.
[40] J. Saak, H. Mena, and P. Benner, *MESS (Matrix Equation Sparse Solver)*, available online at http://www-user.tu-chemnitz.de/∼saak/Software/mess.php.
[41] N. Scheerlinck, P. Verboven, J. D. Stigter, J. De Baerdemaeker, J. F. Van Impe, and B. M. Nicolai, *A variance propagation algorithm for stochastic heat and mass transfer problems in food processes*, Internat. J. Numer. Methods Engrg., 51 (2001), pp. 961–983.
[42] V. Simoncini, *A new iterative method for solving large-scale Lyapunov matrix equations*, SIAM J. Sci. Comput., 29 (2007), pp. 1268–1288.
[43] D. C. Sorensen and Y. Zhou, *Bounds on Eigenvalue Decay Rates and Sensitivity of Solutions to Lyapunov Equations*, CAAM TR02-07, Computational and Applied Mathematics, Rice University, Houston, TX, 2002.
[44] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM J. Numer. Anal., 20 (1983), pp. 626–637.
[45] G. W. Stewart, *Matrix Algorithms. Volume* II*: Eigensystems*, SIAM, Philadelphia, 2001.
[46] P. L. Toint, *Towards an efficient sparsity exploiting Newton method for minimization*, in Sparse Matrices and Their Uses, Academic Press, London, New York, 1981, pp. 57–88.
[47] C. F. Van Loan, *The ubiquitous Kronecker product*, J. Comput. Appl. Math., 123 (2000), pp. 85–100.