Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Matematica

On the Numerical Solution of Large-scale Differential Riccati Equations

Tesi di Laurea in Analisi Numerica

Relatore:

Chiar.mo Dott. Davide Palitta

Presentata da:

Eugenio Mancinelli

Correlatore:

Chiar.mo Dott. Jens Saak

Anno Accademico 2025-2026

Alla mia famiglia,
agli amici,
e a Bologna.

Abstract

This thesis investigates efficient numerical methods for solving large-scale Differential Riccati Equations (DREs), which commonly arise in optimal control and filtering problems. The DRE is discretized in time using Backward Differentiation Formula (BDF) schemes, leading to a sequence of Algebraic Riccati Equations (AREs) solved via iterative techniques such as the Newton–Kleinman method, Newton's method with exact line search, and low-rank ADI-based algorithms. A modified ADI scheme is then proposed, allowing for nonzero initial iterates and enabling the reuse of previous solutions to accelerate convergence. The performance of the low-rank methods is validated on the rail profile cooling problem, modeled within the Linear Quadratic Regulator (LQR) framework. Numerical experiments confirm the expected convergence order $\mathcal{O}(h^p)$ and demonstrate that the proposed low-rank approaches substantially improve computational efficiency and scalability for high-dimensional problems.

Index

List of Figures

List of Tables

List of Algorithms

Introduction

This thesis aims to provide a comprehensive treatment of the fundamental theory behind Differential Riccati Equations, along with a detailed description of advanced algorithms for their solution.

Differential Riccati Equations (DREs) form a class of matrix equations that arise frequently in optimal control and filtering problems, particularly in large-scale systems governed by complex dynamics. The increasing demand from a wide range of applications has, over the years, stimulated significant research into the design and analysis of efficient numerical algorithms for solving these equations.

Their numerical solution involves a sequence of techniques that gradually transform the original problem into a more tractable form. We begin by discretizing the DRE in time using either a Backward Differentiation Formula (BDF) or a Rosenbrock method. This leads to the need to solve an Algebraic Riccati Equation (ARE) at each time step. Since the resulting ARE is generally nonlinear, we employ iterative linearization techniques such as the Newton-Kleinman scheme or Newton's method with exact line search.

At each Newton's iteration, we must solve a Sylvester or, more commonly, a Lyapunov equation. For this task, we adopt the Alternating Direction Implicit (ADI) method, which is well-suited to large-scale problems due to its favorable memory and computational properties.

While direct methods, such as the Bartels-Stewart algorithm or invariant subspace methods, can also be used to solve the linear step, they rely on the Schur decomposition of the coefficient matrices, which has a computational cost of $\mathcal{O}(n^3)$, n being the problem dimension, making them impractical for high-dimensional systems.

To ensure computational and memory efficiency, it is essential to exploit the low-rank

structure often present in the problem's data. However, this is not straightforward due to the interaction of multiple parameters across the different algorithms.

The situation becomes even more challenging in the non-autonomous case, where the coefficient matrices of the differential equation depend explicitly on time. In this setting, each time step requires the evaluation of time-varying matrices, and greater care must be taken in selecting low-rank approximations, as the rank of the solution may significantly change over time.

The aforementioned solvers can be extended to this setting, but the optimal way to do that, in order to speed up the convergence, remains an open question. This thesis focuses on improving solvers for large-scale DREs by analyzing the key parameters that influence convergence and by implementing a modified ADI scheme that allows for non-zero initial iterates, specifically, reusing the solution from the previous time step. This approach has already shown to be beneficial for AREs and it brings promising computational advantages in the time-dependent case as well.

This thesis is structured into five chapters as follows. The first chapter provides an overview of AREs and DREs, including useful definitions and theoretical results. The second chapter presents in detail various numerical schemes for solving AREs. It begins with a description of the ADI method, discussing its convergence behavior and the techniques used for selecting shift parameters. Next, it presents and compares two iterative methods commonly employed to solve quadratic matrix equations: the Newton–Kleinman method and Newton's method with exact line search. The chapter then introduces a more recent approach for solving low-rank AREs: the RADI method. Finally, it discusses how the previously described algorithms can be adapted to handle time and memory constraints efficiently and introduces the CF-ADI and LDL^{T}-ADI algorithms. The third chapter presents a modified ADI scheme that allows for nonzero initial iterates, with the aim of accelerating convergence. It also provides the MATLAB implementation of this scheme and reports two new results concerning the numerical solutions computed by using the method. The fourth chapter covers the time-discretization techniques used to transform a DRE into a sequence of AREs, focusing on BDF schemes and Rosenbrock methods. It concludes with the presentation of two classical, yet important, applications in control theory: the Linear Quadratic Regulator (LQR) problem

and the Tracking problem. Finally, the fifth chapter describes the rail profile cooling problem and presents its mathematical model. In control theory, this problem is solved via the LQR approach, which gives rise to a differential Riccati equation. For different problem sizes, we discretize the corresponding DRE using a BDF scheme and apply some of the low-rank methods introduced in Chapter 3 to solve the resulting algebraic Riccati equations. We first investigate the error behavior of the solution, demonstrating that it follows the expected order, i.e. $\mathcal{O}(h^p)$ where h is the time step size of the discretization and p is the BDF order. We then compare the different ARE solvers in terms of runtime, average number of iterations, and average rank of the solution, focusing on large-dimensional problems.

Notation

We present here the notation that will be used throughout this work.

- flops : floating point operations;
- $\mathbb{N} := \{1, 2, \ldots\}$: set of natural numbers;
- $\mathbb{R}$: set of real numbers;
- $\mathbb{R}_{\geq 0} := \{x \in \mathbb{R} : x \geq 0\}$, $\mathbb{R}_{<} := \{x \in \mathbb{R} : x < 0\}$;
- $\mathbb{C}$: set of complex numbers;
- If $z \in \mathbb{C}$, we denote the real part of z by $\Re(z)$, and its imaginary part by $\Im(z)$;
- $\mathbb{C}_{<} := \{z \in \mathbb{C} : \Re(z) < 0\}$;
- Let $\mathbb{M} = \mathbb{R}$ or $\mathbb{M} = \mathbb{C}$ and $m, n \in \mathbb{N}$. $\mathbb{M}^{m \times n}$ is the set of $m \times n$ matrices with entries in $\mathbb{M}$;
- I_n is the *identity matrix* of size $n \times n$;
- Given a matrix A, we denote with A^{T} its transpose and with $\overline{A}$ the matrix obtained by applying the complex conjugate to its entries. The *conjugate transpose* of A is defined as $A^{\mathsf{H}} := \left(\overline{A}\right)^{\mathsf{T}}$. We apply the same definitions to vectors;
- Given $A \in \mathbb{R}^{n \times n}$, we say that it is *symmetric* if $A = A^{\mathsf{T}}$. If A is complex and it holds $A = A^{\mathsf{H}}$, then we say that A is *Hermitian*;
- Given $A \in \mathbb{R}^{n \times n}$, we say that it is *orthogonal* if $AA^{\mathsf{T}} = A^{\mathsf{T}}A = I_n$. If $A \in \mathbb{C}^{n \times n}$, we say that it is *unitary* if $AA^{\mathsf{H}} = A^{\mathsf{H}}A = I_n$;

- Given a matrix A, its *spectrum* $\mathrm{spec}(A)$ is the set of all the eigenvalues of A. We define the *spectral radius* of A as the quantity $\rho(A) := \max\{|\lambda| : \lambda \in \mathrm{spec}(A)\}$;
- Given an $n \times n$ square matrix $A = (a_{i,j})_{i,j=1,\dots,n}$, we define $\mathrm{trace}(A) := \sum_{i=1}^{n} a_{i,i}$;
- Let $X : \mathbb{R} \to \mathbb{C}^{n\times m},\ t \mapsto X(t)$, be a differentiable function, where $n, m \in \mathbb{N}$. We denote its derivative by $\dot{X} := \frac{\mathrm{d}X(t)}{\mathrm{d}t}$.

Preliminaries

Matrix norms

Definition 0.0.1. Let $\mathbb{M}$ be the set of either real or complex numbers and $m, n \in \mathbb{N}$. A *matrix norm* is a function $||\cdot|| : \mathbb{M}^{m\times n} \to \mathbb{R}_{\geq 0}$ that satisfies the following properties: For all scalars $\alpha \in \mathbb{M}$ and matrices $A, B \in \mathbb{M}^{m\times n}$,

- $||A|| \geq 0$;
- $||A|| = 0 \iff A = 0_{m,n}$;
- $||\alpha A|| = |\alpha|\ ||A||$;
- $||A + B|| \leq ||A|| + ||B||$;
- $||AB|| \leq ||A||\ ||B||$;

where $0_{m,n}$ is the $m \times n$ zero matrix.

Definition 0.0.2. Given a vector norm $||\cdot||$, i.e. a norm whose argument is a vector, the corresponding matrix norm on $\mathbb{C}^{m\times n}$ is defined by

$$||A|| = \max_{x\in\mathbb{C}^n, ||x||=1} ||Ax||.$$

A few commonly used norms in $\mathbb{C}^n$ are:

- $||x||_1 = \sum_{i=1}^{n} |x_i|$,
- $||x||_2 = \left(\sum_{i=1}^{n} |x_i|^2\right)^{\frac{1}{2}}$,

- $||x||_\infty = \max\limits_{i=1,\dots,n} |x_i|$.

The corresponding matrix norms on $\mathbb{C}^{m\times n}$ are given by

- $||A||_1 = \max\limits_{j=\dots,n} \sum\limits_{i=1}^{m} |a_{i,j}|$,
- $||A||_2 = \rho(A^{\mathsf{H}}A)^{1/2}$,
- $||A||_\infty = \max\limits_{i=1,\dots,m} \sum\limits_{j=1}^{n} |a_{i,j}|$.

The above norms are called the *1-norm*, *2-norm*, and ∞*-norm*, respectively. Another matrix norm that is used frequently, especially for numerical experiments, is the Frobenius norm $||\cdot||_F$.

Definition 0.0.3. Let $A \in \mathbb{C}^{m\times n}$. We define the *Frobenius norm* of A as

$$||A||_F := \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{i,j}|^2\right)^{\frac{1}{2}} = (\mathrm{trace}(A^{\mathsf{H}}A))^{\frac{1}{2}}.$$

Definite and semi-definite matrices

Definition 0.0.4. We say that a matrix $A \in \mathbb{C}^{n\times n}$ is *positive definite* if $\Re(x^{\mathsf{H}}Ax) > 0$ for any $x \in \mathbb{C}^n \setminus \{0\}$ with $||x||_2 = 1$. In this case, we write $A \succ 0$.

Definition 0.0.5. We say that a matrix $A \in \mathbb{C}^{n\times n}$ is *positive semi-definite* if $\Re(x^{\mathsf{H}}Ax) \geq 0$ for any $x \in \mathbb{C}^n$ with $||x||_2 = 1$. In this case, we write $A \succeq 0$.

Given two matrices A and B, the notation $B \succeq A$ means that $B - A \succeq 0$. Let us now recall a few important characterizations.

Theorem 0.0.6. Let $A \in \mathbb{C}^{n\times n}$. A is positive semi-definite if and only if $A = B^{\mathsf{H}}B$, for some B. If A is real, B can be real as well and the decomposition can be written as $A = B^{\mathsf{T}}B$.
Moreover, A is positive definite if and only if such a decomposition exists with B invertible.
More generally, A is positive semi-definite with rank k if and only if a decomposition exists with a $k \times n$ matrix B of full rank, i.e. of rank k.

Let us consider the negative counterpart of the previous definitions.

Definition 0.0.7. We say that a matrix $A \in \mathbb{C}^{n\times n}$ is *negative definite* if $-A$ is positive definite. In this case, we write $A \prec 0$.

Definition 0.0.8. We say that a matrix $A \in \mathbb{C}^{n\times n}$ is *negative semi-definite* if $-A$ is positive semi-definite. In this case, we write $A \preceq 0$.

Remark 0.0.9. A is negative (semi)-definite if and only if $-A$ is positive (semi)-definite.

Definition 0.0.10. We say that a matrix is *indefinite* if it is neither positive semi-definite nor negative semi-definite.

The following result characterizes the definiteness of a matrix based on its spectrum.

Theorem 0.0.11. Let $A \in \mathbb{R}^{n\times n}$ be symmetric. Then all its eigenvalues are real and their sign characterize the definiteness of A as follows:

- A is positive definite if and only if all its eigenvalues are positive;
- A is positive semi-definite if and only if all of its eigenvalues are non-negative;
- A is negative definite if and only if all of its eigenvalues are negative;
- A is negative semi-definite if and only if all of its eigenvalues are non-positive;
- A is indefinite if and only if it has both positive and negative eigenvalues.

Hamiltonian matrices

As we will see later, Hamiltonian matrices play a crucial role in the context of algebraic Riccati equations .

Definition 0.0.12. Let $\mathcal{J}$ be the matrix $\begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix}$.
A $2n \times 2n$ complex matrix H is said to be *Hamiltonian* if $\mathcal{J}H$ is Hermitian, i.e. $\mathcal{J}H = H^{\mathsf{H}}\mathcal{J}^{\mathsf{H}}$.

Let's observe that $\mathcal{J}^{-1} = \mathcal{J}^{\mathsf{H}} = \mathcal{J}^{\mathsf{T}} = -\mathcal{J}$, and therefore H is Hamiltonian if and only if $\mathcal{J}^{\mathsf{T}} H \mathcal{J} = -H^{\mathsf{H}}$. By partitioning H into four $n \times n$ blocks and writing explicitly Definition 0.0.12, one can prove that H is Hamiltonian if and only if it has the form

$$H = \begin{bmatrix} A & G \\ Q & -A^{\mathsf{H}} \end{bmatrix},$$

where G and Q are Hermitian matrices of size n. It follows that $\mathcal{J}$ is Hamiltonian.

Remark 0.0.13. The property $\mathcal{J}^{\mathsf{T}} H \mathcal{J} = -H^{\mathsf{H}}$ implies that H is similar to $-H^{\mathsf{H}}$, so that its spectrum is symmetric with respect to the imaginary axis, i.e. the non-imaginary eigenvalues of H come in pairs $(\lambda, -\overline{\lambda})$, and therefore they can be ordered in such a way that

$$\Re(\lambda_1) \leq \ldots \leq \Re(\lambda_n) \leq 0 \leq \Re(\lambda_{n+1}) \leq \ldots \leq \Re(\lambda_{2n}).$$

Notice that if H has no eigenvalues that lie on the imaginary axis, then there are stable and antistable invariant subspaces corresponding to n eigenvalues with negative real part and positive real part, respectively.

Kronecker product and its properties

Definition 0.0.14. If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the *Kronecker product* $A \otimes B$ is the $pm \times qn$ block matrix

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

In the following, we list several properties that we will use later in this work.
Properties of the Kronecker product:

1. **Bilinearity and associativity**:

$$\begin{aligned} A \otimes (B + C) &= A \otimes B + A \otimes C, \\ (B + C) \otimes A &= B \otimes A + C \otimes A, \\ (kA) \otimes B = A \otimes (kB) &= k(A \otimes B), \\ (A \otimes B) \otimes C &= A \otimes (B \otimes C), \\ A \otimes 0 = 0 \otimes A &= 0, \end{aligned}$$

   where A, B, C are matrices, 0 is the zero matrix, and k is a scalar.

2. **Non-commutativity**:
   In general,

$$A \otimes B \neq B \otimes A.$$

3. **The mixed-product property**:
   Given A, B, C, D matrices of suitable sizes, then

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD).$$

4. **Vectorization property**:
   Given A, B, X matrices of suitable sizes, then

$$\operatorname{vec}(AXB) = (B^{\mathsf{T}} \otimes A) \operatorname{vec}(X),$$

   where $\operatorname{vec}(X)$ is the vectorization operator applied on X, formed by stacking its columns on top of one another.

5. **Spectrum**:
   Suppose that A and B are square matrices of size n and m respectively, and let $\operatorname{spec}(A)=\{\lambda_i : i = 1, \ldots, n\}$ and $\operatorname{spec}(B)=\{\mu_j : j = 1, \ldots, m\}$ be their spectra. Then the eigenvalues of $A \otimes B$ are $\lambda_i \mu_j$ for all $i = 1, \ldots, n,\ j = 1, \ldots, m$. Moreover, the eigenvalues of $I_m \otimes B + A \otimes I_n$ are given by $\lambda_i + \mu_j$ for all $i = 1, \ldots, n,\ j = 1, \ldots, m$.

Matrix Decompositions

In the following, we list a few useful factorizations that we will employ throughout the thesis.

QR decomposition

Theorem 0.0.15. Let $A \in \mathbb{C}^{m\times n}$ with $m \geq n$. Then there exist $Q \in \mathbb{C}^{m\times m}$ unitary and $R \in \mathbb{C}^{m\times n}$ upper triangular such that

$$A = QR = \begin{bmatrix} Q_1, Q_2 \end{bmatrix} \begin{bmatrix} R1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where Q_1 is $m \times n$, Q_2 is $m \times (m-n)$ and R_1 is $n \times n$ upper triangular matrix.

The factorization $A = QR$ is called *QR factorization* and the decomposition $A = Q_1 R_1$ is called *thin QR factorization*. In general, computing this decomposition requires $\mathcal{O}(mn^2)$ flops, which becomes $\mathcal{O}(n^3)$ flops when $m = n$.

Schur decomposition

Real Schur decomposition

Theorem 0.0.16. Let $A \in \mathbb{R}^{n\times n}$. Then A can be written as

$$A = QHQ^T,$$

where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $H \in \mathbb{R}^{n\times n}$ is either upper or lower quasi-triangular.

A *quasi-triangular* matrix is a block triangular matrix whose diagonal blocks are of size 1×1 or 2×2.

Complex Schur decomposition

Theorem 0.0.17. Let $A \in \mathbb{C}^{n\times n}$. Then A can be expressed as

$$A = QUQ^{\mathsf{H}},$$

for some unitary matrix $Q \in \mathbb{C}^{n\times n}$ and some upper triangular matrix $U \in \mathbb{C}^{n\times n}$.

Let us observe that in both cases, U and A are similar matrices and therefore share the same spectrum, which is given by the diagonal entries of U. Moreover, the computational complexity of both decompositions is $\mathcal{O}(n^3)$.

Singular Value Decomposition

Theorem 0.0.18. Let $A \in \mathbb{C}^{m\times n}$. Then A can be factorized as

$$A = U\Sigma V^{\mathsf{H}},$$

where $U \in \mathbb{C}^{m\times m}$ is a unitary matrix, $\Sigma \in \mathbb{R}^{m\times n}$ is a rectangular diagonal matrix with non-negative entries on the diagonal and $V \in \mathbb{C}^{n\times n}$ is a unitary matrix.

The factorization $A = U\Sigma V^{\mathsf{H}}$ is called *singular value decomposition* (SVD) of A.
If A is real, then U and V can be guaranteed to be real orthogonal matrices and, in that context, the SVD is denoted $U\Sigma V^{\mathsf{T}}$.
The diagonal entries $\sigma_i = \Sigma_{ii}$ of Σ are uniquely determined by A and are known as the *singular values* of A.
Without any assumption on A, the complexity of SVD is $\mathcal{O}(mn^2)$, thus $\mathcal{O}(n^3)$ if $m = n$.

Invariant subspaces

The concept of an invariant subspace for a matrix is crucial for the analysis and solution of algebraic Riccati equations.

Definition 0.0.19. Given an $n \times n$ matrix A and an m-dimensional subspace $\mathcal{V} \subseteq \mathbb{C}^n$, we denote by $A\mathcal{V}$ the subspace $\{y \in \mathbb{C}^n : \ y = Ax, \ x \in \mathcal{V}\}$. We say that $\mathcal{V}$ is an *invariant subspace* for A if $A\mathcal{V} \subseteq \mathcal{V}$, i.e. $Ax \in \mathcal{V}$ for any $x \in \mathcal{V}$.

Remark 0.0.20. If V is a $n \times m$ full rank matrix whose columns span the subspace $\mathcal{V}$, then $\mathcal{V}$ is invariant for A, if and only if there exists an $m \times m$ matrix Λ such that

$$AV = V\Lambda.$$

Observe that $\operatorname{spec}(\Lambda) \subseteq \operatorname{spec}(A)$.

Fréchet derivative

The concept of Fréchet derivative is used to extend the notion of classical derivative to functions defined on normed spaces.

Definition 0.0.21. Let V and W be normed vector spaces, and $U \subseteq V$ be an open subset of V. A function $\mathcal{F} : U \to W$ is called *Fréchet differentiable at* $x \in U$, if there exists a bounded linear operator $A : V \to W$ such that

$$\lim_{||h||_V \to 0} \frac{||\mathcal{F}(x+h) - \mathcal{F}(x) - Ah||_W}{||h||_V} = 0.$$

Equivalently, in Landau notation,

$$\mathcal{F}(x+h) = \mathcal{F}(x) + Ah + o(||h||_V).$$

If there exists such an operator A, it is unique, so we write $\mathcal{F}'_x = A$ and call it the *Fréchet derivative* of $\mathcal{F}$ at x.

For our purposes, U and V will be spaces of the form $\mathbb{C}^{n\times m}$. In these cases, the Fréchet derivative of $\mathcal{F} : \mathbb{C}^{n\times m} \to \mathbb{C}^{n\times m}$ in $X \in \mathbb{C}^{n\times m}$ is the function $\mathcal{F}'_X : \mathbb{C}^{n\times m} \to \mathbb{C}^{n\times m}$, $E \mapsto \mathcal{F}'_X[E]$.

Properties of the Fréchet derivative:

1. **Linearity**:
   Let $\mathcal{F}, \mathcal{G} : V \to W$ be differentiable at x and c a scalar. Then

$$(c\mathcal{F})'_x = c\mathcal{F}'_x,$$
$$(\mathcal{F}+\mathcal{G})'_x = \mathcal{F}'_x + \mathcal{G}'_x.$$

2. **Chain rule**:
   Let $\mathcal{F} : U \to Y$ be differentiable at $x \in U$ and $\mathcal{G} : Y \to W$ differentiable at $y = \mathcal{F}(x)$. Then the composition $\mathcal{G} \circ \mathcal{F} : U \to W$ is differentiable at x and it holds

$$(\mathcal{G} \circ \mathcal{F})'_x = \mathcal{G}'_y \circ \mathcal{F}'_x.$$

Notions from control theory

Throughout this work, we will focus on continuous-time systems, but the definitions and results that follow can be adapted to discrete-time systems as well.

Definition 0.0.22. We define the *stability region* as the open left half-plane $\mathbb{C}_< := \{z \in \mathbb{C} : \Re(z) < 0\}$ and we say that a matrix A is *stable* if all its eigenvalues lie in $\mathbb{C}_<$. We say that A is *antistable* if all its eigenvalues lie outside the closure of the stability region, i.e. in $\mathbb{C}_> := \{z \in \mathbb{C} : \Re(z) > 0\}$.

Let us now introduce the fundamental notions of controllability, observability, stabilizability, and detectability, for a dynamical system.

Definition 0.0.23. Let $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times m}, C \in \mathbb{C}^{m\times n}$, and $\Omega \subset \mathbb{C}$. The pair (A, B) is called *controllable* or *reachable at* $\lambda \in \mathbb{C}$ if $\operatorname{rank}([A - \lambda I_n, B]) = n$; *controllable in* Ω if it is controllable at any $\lambda \in \Omega$; and *controllable* if it is controllable at any $\lambda \in \mathbb{C}$.

Definition 0.0.24. Let $A \in \mathbb{C}^{n\times n}$ and $C \in \mathbb{C}^{m\times n}$. The pair (C, A) is called *observable* is $(A^{\mathsf{H}}, C^{\mathsf{H}})$ is controllable.

Definition 0.0.25. Let $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{n\times m}$. The pair (A, B) is called *stabilizable* if it is controllable outside the stability region, i.e. for all $\lambda \in \mathbb{C}_\geq := \{z \in \mathbb{C} : \Re(z) \geq 0\}$.

The following result illustrates the relationship between stability and stabilizability.

Theorem 0.0.26. The pair (A, B) is stabilizable if and only if there exists a matrix $K \in \mathbb{C}^{m\times n}$ such that $A - BK$ is stable. Moreover, if $m = n$, $B \succeq 0$, and (A, B) is stabilizable, then there exists $K \in \mathbb{C}^{n\times n}$ such that $K \succeq 0$ and $A - BK$ is stable.

Definition 0.0.27. Let $A \in \mathbb{C}^{n\times n}$ and $C \in \mathbb{C}^{m\times n}$. The pair (C, A) is called *detectable* if $(A^{\mathsf{H}}, C^{\mathsf{H}})$ is stabilizable.

Hamilton-Jacobi-Bellman equation

The Hamilton-Jacobi-Bellman (HJB) equation is a partial differential equation that describes how the value function of a system evolves over time and space. Before stating that result, we have to clarify the notions of cost function and value function for a system.

Definition 0.0.28. The *cost function* J defines the total cost incurred by following a control policy over time. Given $t_0 \in [0, t_{\text{end}}]$ and a continuous-time system subject to

$$\dot{x}(t) = f(t, x(t), u(t)),\ x(t_0) = x_0,$$

the cost function has the form

$$J(t_0, x_0, u) = \int_{t_0}^{t_{\text{end}}} L(t, x(t), u(t))\mathrm{d}t + \phi(x(t_{\text{end}})),$$

where $x(t)$ is the state of the system at time t, $u(t)$ the control input, $L(t, x, u)$ the instantaneous cost and $\phi(x(t_{\text{end}}))$ the terminal cost at final time t_{end}.
The *value function* or *cost-to-go function* V represents the minimum cost achievable from a given state and time onwards, assuming optimal control from that point. V has the form

$$V(t, x(t)) = \min_u \left[\int_t^{t_{\text{end}}} L(\tau, x(\tau), u(\tau))\mathrm{d}\tau + \phi(x(t_{\text{end}})) \right], \quad V(t_{\text{end}}, x(t_{\text{end}})) = \phi(x(t_{\text{end}})).$$

The *Hamilton-Jacobi-Bellman equation* is a nonlinear partial differential equation that characterizes the value function V for an optimal control problem as solution of

$$\min_u \left\{ L(t, x, u) + \frac{\partial V(t, x)}{\partial x}\dot{x} + \frac{\partial V(t, x)}{\partial t} \right\} = 0,$$

with $V(t_{\text{end}}, x(t_{\text{end}})) = \phi(x(t_{\text{end}}))$ and $\dot{x} = f(t, x, u)$.

Chapter 1

Matrix Equations

In this chapter, we present classical results and solvers for some of the most common matrix equations. In particular, we classify linear matrix equations into Sylvester, Lyapunov, and Stein equations. For each of these, we present a Kronecker-based direct solver and a version of the Bartels–Stewart algorithm. We then address the quadratic case, i.e. algebraic Riccati equations, describing an invariant subspace method based on the Schur decomposition for computing solutions.

1.1 Linear Matrix Equations

We start by introducing the simplest case one may try to solve: the linear case. Depending on their structure, linear matrix equations are classified into the following three groups: Sylvester, Lyapunov, and Stein equations.

1.1.1 Sylvester equations

Definition 1.1.1. A *Sylvester equation* is a matrix equation of the form

$$AX + XB = Q, \tag{1.1}$$

where $X \in \mathbb{R}^{m\times n}$ is the unknown, and the coefficient matrices are $A \in \mathbb{R}^{m\times m}, B \in \mathbb{R}^{n\times n}, Q \in \mathbb{R}^{m\times n}$.

By using the operator vec, see Property 4 of the Kronecker product, we can transform (1.1) into the linear system

$$(I_n \otimes A + B^{\mathsf{T}} \otimes I_m)\,\mathrm{vec}(X) = \mathrm{vec}(Q). \tag{1.2}$$

Remark 1.1.2 (Uniqueness of the solution)**.** From Property 5 of the Kronecker product, the eigenvalues of the system matrix are all the summations $\lambda_i + \mu_j$, where $\mathrm{spec}(A) = \{\lambda_i : i = 1, \ldots, m\}$ and $\mathrm{spec}(B) = \{\mu_j : j = 1, \ldots, n\}$, and it follows that the matrix is non singular if and only if $\lambda_i + \mu_j \neq 0$ for any i, j, i.e. $\mathrm{spec}(A) \cap \mathrm{spec}(-B) = \emptyset$. In this case, the Sylvester equation admits a unique solution.

A first naive method to solve (1.1) is computing the solution $\mathrm{vec}(X)$ of the linear system (1.2), and then recover X by reshaping. Note that (1.2) is a $mn \times mn$ system and solving it with the Gaussian elimination with partial pivoting costs $O((mn)^3)$ flops, i.e. $\mathcal{O}(n^6)$ if $m = n$. Therefore it is impractical for large-scale problems.
In Algorithm 1 we report the implementation of this approach.

Algorithm 1 Naive algorithm for the Sylvester equation

`function` $X =$ `sylv_naive`(A, B, Q)

1: $[m, n] =$ `size`(Q)
2: **% Construct the Kronecker linear system**
3: $H =$ `kron`$(I_n, A) +$ `kron`(B^T, I_m)
4: $q =$ `reshape`$(Q, m \cdot n, 1)$
5: **% Solve the linear system**
6: Solve $H \cdot x = q$ for x
7: **% Reshape back to matrix form**
8: $X =$ `reshape`(x, m, n)
9: **return** X

A more advanced scheme is the *Bartels and Stewart algorithm* [3] which exploits the complex Schur decomposition (see Preliminaries) of A and B^{T} in order to solve a Sylvester equation with upper and lower triangular coefficients. In particular, let us consider the following decompositions:

$$A = U\tilde{A}U^{\mathsf{H}},\ B^{\mathsf{T}} = V\tilde{B}^{\mathsf{T}}V^{\mathsf{H}},$$

where $\tilde{A}$ and $\tilde{B}^{\mathsf{T}}$ are upper triangular and U, V are unitary. Then we can rewrite (1.1) in terms of $\tilde{A}$ and $\tilde{B}$ as follows:

$$\tilde{A}\tilde{X} + \tilde{X}\tilde{B} = \tilde{Q}, \tag{1.3}$$

with $\tilde{X} = U^{\mathsf{H}} X \overline{V}$ and $\tilde{Q} = U^{\mathsf{H}} Q \overline{V}$, where $\overline{V} = \left(V^{\mathsf{H}}\right)^{\mathsf{T}}$. We exploit the fact that $\tilde{A}$ and $\tilde{B}$ are upper and lower triangular, respectively, by considering the linear-system version of (1.3), i.e.

$$(I_n \otimes \tilde{A} + \tilde{B}^{\mathsf{T}} \otimes I_m)\,\mathrm{vec}(X) = \mathrm{vec}(Q). \tag{1.4}$$

The latter can be written in the form

$$\tilde{A}_{k,k}\tilde{X}_{k,l} + \tilde{X}_{k,l}\tilde{B}_{l,l} = \tilde{Q}_{k,l} - \sum_{i=k+1}^{m} \tilde{A}_{k,i}\tilde{X}_{i,l} - \sum_{j=l+1}^{n} \tilde{X}_{k,j}\tilde{B}_{j,l}, \text{ for } k = m, \ldots, 1,\ l = n, \ldots, 1. \tag{1.5}$$

Once we solve (1.3), we recover X using $X = U\tilde{X}V^{\mathsf{T}}$. If $m = n$, the overall cost of this algorithm turns to $60n^3$ flops, i.e. $\mathcal{O}(n^3)$.
We report the implementation of this method in Algorithm 2.

1.1.2 Lyapunov equations

Definition 1.1.3. A *Lyapunov equation* is a particular Sylvester equation of the form

$$AX + XA^{\mathsf{T}} = Q, \tag{1.6}$$

where $X, A, Q \in \mathbb{R}^{n\times n}$ and $Q = Q^{\mathsf{T}}$.

By vectorizing the equation (1.6) with the vec operation, we can rewrite it in the form of the linear system

$$(I_n \otimes A + \overline{A} \otimes I_n)\,\mathrm{vec}(X) = \mathrm{vec}(Q). \tag{1.7}$$

Remark 1.1.4 (Uniqueness of the solution)**.** From Property 5 of the Kronecker product, it follows that the eigenvalues of the system matrix are all the summations $\lambda_i + \overline{\lambda}_j$ for $i, j = 1, \ldots, n$, where $\mathrm{spec}(A) = \{\lambda_i : i = 1, \ldots, n\}$. The system is invertible, i.e. equation (1.6) has a unique solution, if and only if A has no pairs of eigenvalues of the kind $(\lambda, -\overline{\lambda})$. Note that this is true when A is stable.

Algorithm 2 Bartels and Stewart algorithm for the Sylvester equation

`function` $X = \texttt{sylv\_bs}(A, B, Q)$

1: $[m, n] = \texttt{size}(Q)$
2: **% Complex Schur decomposition**
3: $[U, \tilde{A}] = \texttt{schur}(A, \texttt{'complex'})$
4: $[V, \tilde{B}] = \texttt{schur}(B^T, \texttt{'complex'})$
5: $\tilde{Q} = U^{\mathsf{H}} Q \overline{V}$
6: $\tilde{X} = \texttt{zeros}(m, n)$
7: **% Backward substitution for Sylvester recurrence**
8: Solve $(\tilde{A} + \tilde{B}(n, n) \cdot I_m) \cdot \tilde{X}(:, n) = \tilde{Q}(:, n)$ for $\tilde{X}(:, n)$
9: **for** $j = n - 1 : -1 : 1$ **do**
10: $\quad r = \sum_{k=j+1}^{n} \tilde{B}(k, j) \cdot \tilde{X}(:, k)$
11: $\quad v = \tilde{Q}(:, j) - \tilde{A} \cdot r$
12: $\quad$ Solve $(\tilde{A} + \tilde{B}(j, j) \cdot I_m) \cdot \tilde{X}(:, j) = v$ for $\tilde{X}(:, j)$
13: **end for**
14: **% Reconstruct the solution**
15: $X = U \tilde{X} V^{\mathsf{T}}$
16: **return** X

Remark 1.1.5 (Hermiticity of the solution)**.** Observe that if we apply the conjugate transpose operation on both sides of equation (1.6), we obtain

$$AX^{\mathsf{H}} + X^{\mathsf{H}}A^{\mathsf{T}} = Q. \tag{1.8}$$

Therefore, if X is solution of (1.6), then X^{H} is solution of the same equation. Under the assumption of existence and uniqueness, $X = X^{\mathsf{H}}$, i.e. the solution is Hermitian.

As before, one could compute the solution of (1.6) by solving the linear system (1.7). The implementation follows straightforwardly from Algorithm 1 by replacing line 3 with $H = I_n \otimes A + \overline{A} \otimes I_n$. However, solving this system using Gaussian elimination with partial pivoting generally requires $\mathcal{O}(n^6)$ flops. Let us see how we can exploit the symmetry of the Lyapunov equation by using the Bartels and Stewart algorithm. In this case, we need to compute the Schur decomposition only for one matrix, because once we know the decomposition of A, we also know it for A^{T}. By substituting

$$A = U\tilde{A}U^{\mathsf{H}}, A^{\mathsf{T}} = U\tilde{A}^{\mathsf{T}}U^{\mathsf{H}},$$

into (1.6), the latter becomes

$$\tilde{A}\tilde{X} + \tilde{X}\tilde{A}^{\mathsf{T}} = \tilde{Q}, \tag{1.9}$$

where $\tilde{X} = U^{\mathsf{H}}XU$, i.e. $X = U\tilde{X}U^{\mathsf{H}}$, and $\tilde{Q} = U^{\mathsf{H}}QU$. Finally, the following system

$$(I_n \otimes \tilde{A} + \overline{\tilde{A}} \otimes I_n)\,\mathrm{vec}(X) = \mathrm{vec}(Q), \tag{1.10}$$

can be written as

$$\tilde{A}_{k,k}\tilde{X}_{k,l} + \tilde{X}_{k,l}\overline{\tilde{A}}_{l,l} = \tilde{Q}_{k,l} - \sum_{i=k+1}^{n} \tilde{A}_{k,i}\tilde{X}_{i,l} - \sum_{j=l+1}^{n} \tilde{X}_{k,j}\overline{\tilde{A}}_{j,l}, \text{ for } l,k = n,\ldots,1. \tag{1.11}$$

Remark 1.1.6. Since $\tilde{X}$ is the solution of (1.9), it follows from Remark 1.1.5 that it is Hermitian. Moreover, note that the reconstruction of the solution $X = U\tilde{X}U^{\mathsf{H}}$ preserves Hermiticity. Indeed, since $\tilde{X} = \tilde{X}^{\mathsf{H}}$, we have $X^{\mathsf{H}} = U\tilde{X}^{\mathsf{H}}U^{\mathsf{H}} = U\tilde{X}U^{\mathsf{H}} = X$.

See Algorithm 3 for an implementation.

Algorithm 3 Bartels and Stewart algorithm for the Lyapunov equation

`function` $X = \texttt{lyap\_bs}(A, Q)$

1: $n = \texttt{size}(Q, 1)$
2: **% Complex Schur decomposition**
3: $[U, \tilde{A}] = \texttt{schur}(A, \texttt{'complex'})$
4: $\tilde{Q} = U^{\mathsf{H}} Q U$
5: $\tilde{X} = \texttt{zeros}(n)$
6: **% Backward substitution for Lyapunov recurrence**
7: Solve $(\tilde{A} + \overline{\tilde{A}(n,n)} \cdot I_n) \cdot \tilde{X}(:, n) = \tilde{Q}(:, n)$ for $\tilde{X}(:, n)$
8: **for** $i = n - 1 : -1 : 1$ **do**
9: $\quad v = \tilde{Q}(:, i) - \tilde{X}(:, i+1 : n) \cdot \tilde{A}(i, i+1 : n)^{\mathsf{H}}$
10: $\quad$ Solve $(\tilde{A} + \overline{\tilde{A}(i,i)} \cdot I_n) \cdot \tilde{X}(:, i) = v$ for $\tilde{X}(:, i)$
11: **end for**
12: **% Reconstruct the solution**
13: $X = U \tilde{X} U^{\mathsf{H}}$
14: **return** X

1.1.3 Stein equations

Definition 1.1.7. A *Stein equation* is a linear matrix equation of the form

$$X - AXB = Q, \tag{1.12}$$

where $X \in \mathbb{R}^{m\times n},\ Q \in \mathbb{R}^{m\times n},\ A \in \mathbb{R}^{m\times m}$ and $B \in \mathbb{R}^{n\times n}$.

Using Property 4 of the Kronecker product, we can transform (1.12) into

$$(I_{mn} - B^{\mathsf{T}} \otimes A)\,\mathrm{vec}(X) = \mathrm{vec}(Q), \tag{1.13}$$

since $I_{mn} = I_n \otimes I_m$.

Remark 1.1.8 (Uniqueness of the solution)**.** The eigenvalues of the system matrix are all the terms of the form $1 - \lambda_i\mu_j$ for $i = 1, \ldots, m, j = 1, \ldots, n$, where $\mathrm{spec}(A) = \{\lambda_i : i = 1, \ldots, m\}$ and $\mathrm{spec}(B) = \{\mu_j : j = 1, \ldots, n\}$, and it follows that the matrix is non singular if and only if $\lambda_i\mu_j \neq 1$ for any i, j.

A naive approach to solve (1.12) is considering the linear system (1.13). For this case, the code is the same of Algorithm 1 but we replace line 4 with $H = I_{mn} - B^{\mathsf{T}} \otimes A$ A more efficient algorithm is developed by adapting the Bartels and Stewart scheme to this case. Let

$$A = U\tilde{A}U^{\mathsf{H}}, B^{\mathsf{T}} = V\tilde{B}^{\mathsf{T}}V^{\mathsf{H}},$$

be the Schur form of A and B^{T} respectively. Then, we can rewrite (1.12) as

$$\tilde{X} - \tilde{A}\tilde{X}\tilde{B} = \tilde{Q}, \tag{1.14}$$

where $\tilde{X} = U^{\mathsf{H}}X\overline{V}$ and $\tilde{Q} = U^{\mathsf{H}}Q\overline{V}$. We present the implementation of this approach in Algorithm 4.

A particular case of the Stein equation is the symmetric Stein equation.

Definition 1.1.9. A *symmetric Stein equation* is a particular Stein equation of the form

$$X - AXA^{\mathsf{T}} = Q, \tag{1.15}$$

where $X \in \mathbb{R}^{n\times n}$ and $A, Q \in \mathbb{R}^{n\times n}$ with $Q = Q^{\mathsf{T}}$.

Algorithm 4 Bartels and Stewart algorithm for the Stein equation

function $X = \texttt{stein\_bs}(A, B, Q)$

1: $[m, n] = \texttt{size}(Q)$
2: **% Complex Schur Decomposition**
3: $[U, \tilde{A}] = \texttt{schur}(A, \texttt{'complex'})$
4: $[V, \tilde{B}] = \texttt{schur}(B^{\mathsf{T}}, \texttt{'complex'})$
5: $\tilde{Q} = U^{\mathsf{H}} Q \overline{V}$
6: $\tilde{X} = \texttt{zeros}(m, n)$
7: Solve $(I_m - \tilde{B}(n, n)\tilde{A})\tilde{X}(:, n) = \tilde{Q}(:, n)$ for $\tilde{X}(:, n)$
8: **for** $j = n - 1 : -1 : 1$ **do**
9: $\quad R_j = \sum_{k=j+1}^{n} \tilde{B}(k, j)\tilde{X}(:, k)$
10: $\quad$ Solve $(I_m - \tilde{B}(j, j)\tilde{A})\tilde{X}(:, j) = \tilde{Q}(:, j) + \tilde{A}R_j$ for $\tilde{X}(:, j)$
11: **end for**
12: **% Reconstruct the solution**
13: $X = U\tilde{X}V^{\mathsf{T}}$
14: **return** X

Note that using the vec operator, we can rewrite (1.15) as

$$(I_{n^2} - \overline{A} \otimes A)\,\mathrm{vec}(X) = \mathrm{vec}(Q). \tag{1.16}$$

Remark 1.1.10 (Uniqueness of the solution)**.** The eigenvalues of the system matrix are all the terms of the form $1 - \lambda_i\overline{\lambda}_j$ for $i, j = 1, \ldots, n$, where $\mathrm{spec}(A) = \{\lambda_i : i = 1, \ldots, n\}$. Therefore, the system (1.16) has a unique solution if and only if $\lambda_i\overline{\lambda}_j \neq 1$ for i, j.

Remark 1.1.11 (Hermiticity of the solution)**.** If we apply the conjugate transpose of both sides of equation (1.15), we obtain

$$X^{\mathsf{H}} - AX^{\mathsf{H}}A^{\mathsf{T}} = Q. \tag{1.17}$$

It follows that if X is a solution of (1.15), then also X^{H} is. Under the assumption of existence and uniqueness, the solution is Hermitian.

In order to solve (1.15), we can proceed simply by solving the system (1.16), but it has a cost of $\mathcal{O}(n^6)$ flops, or by adapting Algorithm 4 to this case, by considering only one Schur decomposition, and it has a cost of $\mathcal{O}(n^3)$ flops.
In the literature, one may find a slightly different and more general version of the previous equations. This is because, when discretizing a PDE via the finite element or finite difference method, the resulting matrix equations typically involve additional coefficient matrices. In the following, we report only the generalized definitions of these equations. For more details, see [19, 42].

Definition 1.1.12. A *generalized Sylvester equation* is a matrix equation of the form $AXB + CXD = Q$, where the coefficient dimension and the unknown X have consistent dimensions.

Definition 1.1.13. A *generalized Lyapunov equation* is a matrix equation of the form $AXE^{\mathsf{H}} + EXA^{\mathsf{H}} = Q$, where the coefficient dimension and the unknown X have consistent dimensions, and $Q = Q^{\mathsf{H}}$.

1.2 Algebraic Riccati Equations

A more general class of matrix equations includes those with a quadratic term involving the unknown. These equations are called Algebraic Riccati Equations, which

we refer to as AREs. The name Riccati comes from their similarity with the Riccati differential equation

$$x'(t) = ax(t)^2 + bx(t) + c, \tag{1.18}$$

where the unknown function $x(t)$ appears both in the linear and quadratic term.
Algebraic Riccati equations are encountered in many applications from different areas, including optimal control and robust control [40, 29], filtering problems [4] and differential games [1]. Solving algebraic Riccati equations is fundamental in many computational problems for model reduction and controller design of dynamical linear systems. The interest in algebraic Riccati equations is motivated not only by the increasing demand from a wide range of applications, especially in the engineering field, but also by the new mathematical framework that has been developed for their study. Since the 1950s and 1960s researchers have implemented many numerical algorithms to solve matrix equations, but it is only in the last few years that the attention shifted on the study of large-scale equations, coming from real-world problems. In this section, we introduce several useful theoretical concepts related to algebraic Riccati equations, along with a classical algorithm for computing their solutions.

1.2.1 Nonsymmetric algebraic Riccati equations

Definition 1.2.1. A *Nonsymmetric Algebraic Riccati Equation* (NARE) is a matrix equation of the form

$$C + XA + DX - XBX = 0, \tag{1.19}$$

where $X \in \mathbb{R}^{m\times n}$ is the unknown, and $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{m\times n}$, and $D \in \mathbb{R}^{m\times m}$ are the coefficients.

We associate this equation with the matrix

$$H = \begin{bmatrix} A & -B \\ -C & -D \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}.$$

The solutions of (1.19) can be put in one-to-one correspondence with certain invariant subspaces of H. In fact, one can easily prove that X is a solution of (1.19) if and only if

$$H \begin{bmatrix} I_n \\ X \end{bmatrix} = \begin{bmatrix} I_n \\ X \end{bmatrix} (A - BX).$$

In particular, the columns of $\begin{bmatrix} I_n \\ X \end{bmatrix}$ span an invariant subspace for H and the set of eigenvalues of $A - BX$ is a subset of the eigenvalues of H. This justifies the following result.

Theorem 1.2.2 ([20])**.** The matrix $X \in \mathbb{C}^{m\times n}$ is a solution of (1.19) if and only if there exists an n-dimensional invariant subspace $\mathcal{V}$ of H such that the columns of $\begin{bmatrix} I_n \\ X \end{bmatrix}$ span $\mathcal{V}$. Moreover, it holds that $\begin{bmatrix} I_n & 0 \\ X & I_m \end{bmatrix}^{-1} H \begin{bmatrix} I_n & 0 \\ X & I_m \end{bmatrix} = \begin{bmatrix} A - BX & -B \\ 0 & -(D - XB) \end{bmatrix}$.

Remark 1.2.3. A consequence of the previous theorem is that the spectrum of H is given by the union of the spectra of $A - BX$ and $XB - D$.

Corollary 1.2.4.

1. If the columns of $\begin{bmatrix} Y \\ Z \end{bmatrix}$ span an n-dimensional invariant subspace $\mathcal{V}$ of H such that $Y \in \mathbb{C}^{n\times n}$ is invertible, then $X := ZY^{-1}$ is a solution of (1.19).
2. Any $\begin{bmatrix} Y_1 \\ Z_1 \end{bmatrix}$, with $Y_1 \in \mathbb{C}^{n\times n}$, whose columns span $\mathcal{V}$, is such that Y_1 is invertible and $X = Z_1 Y_1^{-1}$.

Proof.

1. By the definition of invariant subspace, there exists a matrix $\Lambda \in \mathbb{C}^{n\times n}$ such that $H \begin{bmatrix} Y \\ Z \end{bmatrix} = \begin{bmatrix} Y \\ Z \end{bmatrix} \Lambda$. Since Y is invertible by hypothesis, we can rewrite this expression as

$$H \begin{bmatrix} I_n \\ ZY^{-1} \end{bmatrix} Y = \begin{bmatrix} I_n \\ ZY^{-1} \end{bmatrix} Y\Lambda.$$

By setting $X = ZY^{-1}$ and $T = Y\Lambda Y^{-1}$, we have that $H \begin{bmatrix} I_n \\ X \end{bmatrix} = \begin{bmatrix} I_n \\ X \end{bmatrix} T$. Thus, by Theorem 1.2.2, we can conclude that X is a solution of (1.19).

2. Assume now that the columns of $\begin{bmatrix} Y_1 \\ Z_1 \end{bmatrix}$ span the same subspace $\mathcal{V}$. Then, we can write its columns as a linear combination of the columns of $\begin{bmatrix} Y \\ Z \end{bmatrix}$, i.e. there exists

a nonsingular matrix T such that $\begin{bmatrix} Y_1 \\ Z_1 \end{bmatrix} = \begin{bmatrix} Y \\ Z \end{bmatrix} T$, i.e. $Y_1 = YT$, $Z_1 = ZT$. Therefore, Y_1 is invertible because it is the product of invertible matrices, and $Z_1 Y_1^{-1} = ZTT^{-1}Y^{-1} = X$.

□

Definition 1.2.5. We say that an n-dimensional invariant subspace of H is a *graph subspace* if it is spanned by the columns of an $(n+m) \times n$ matrix whose leading $n \times n$ submatrix is invertible.

From the previous results, it follows

Theorem 1.2.6. There is a one-to-one correspondence between the solutions of a NARE and the n-dimensional graph invariant subspaces of the matrix H.

There are several special cases derived from (1.19) that one can study, but we will focus only on the symmetric case, usually referred to as the continuous-time algebraic Riccati equation.

1.2.2 Continuous-time algebraic Riccati equation

Definition 1.2.7. A *Continuous-time Algebraic Riccati Equation* (CARE) is a special case of a NARE of the form

$$C + XA + A^{\mathsf{T}}X - XBX = 0, \tag{1.20}$$

where $X \in \mathbb{R}^{n\times n}$ is the unknown and $A, B, C \in \mathbb{R}^{n\times n}$ are the coefficients, with $B = B^{\mathsf{T}}, C = C^{\mathsf{T}}$.

Remark 1.2.8 (Hermiticity of the solution)**.** By taking the complex conjugate on both sides of (1.20), it follows that if X is a solution of the equation, then also X^{H} is. Therefore, under the assumption of existence and uniqueness, the solution is Hermitian.

Note that in this case the matrix H has the form $H = \begin{bmatrix} A & -B \\ -C & -A^{\mathsf{T}} \end{bmatrix}$ and it is a Hamiltonian matrix.

Definition 1.2.9. We say that a solution X of (1.20) is (*almost*) *stabilizing* if the spectrum of $A - BX$ is contained in the (closed) left half-plane $\mathbb{C}_<$. Similarly, we say that a solution X of (1.20) is (*almost*) *antistabilizing* if the spectrum of $A - BX$ is contained in the (closed) right half-plane $\mathbb{C}_>$.

Remark 1.2.10. From a direct computation, it is easy to observe that a solution X is Hermitian if and only if

$$\begin{bmatrix} I_n & X^{\mathsf{H}} \end{bmatrix} \mathcal{J} \begin{bmatrix} I_n \\ X \end{bmatrix} = 0,$$

where $\mathcal{J}$ is the matrix defined previously.

The following theorem presents a sufficient condition for the uniqueness of a Hermitian stabilizing solution for (1.20).

Theorem 1.2.11 ([18])**.** If H has no pure imaginary eigenvalues and there exists a solution X such that $A - BX$ is stable, then X is the unique stabilizing solution, in particular X is Hermitian.

In the study of CAREs, we are usually interested in the *maximal* and *minimal* Hermitian solutions, where the semiordering is meant with respect to positive definiteness. In the following, we list a few results concerning the existence and uniqueness of almost stabilizing and antistabilizing solutions for (1.20).

Theorem 1.2.12 ([5])**.** Assume $B \succeq 0$. There exists a unique Hermitian solution X_+ of (1.20) such that the eigenvalues of $A - BX_+$ have nonpositive real part if and only if the pair (A, B) is stabilizable and the partial multiplicities of the pure imaginary eigenvalues of H, if any, are all even.

Theorem 1.2.13 ([5])**.** Assume $B \succeq 0$. There exists a unique Hermitian solution X_- of (1.20) such that the eigenvalues of $A - BX_-$ have nonnegative real part if and only if the pair $(-A, B)$ is stabilizable and the partial multiplicities of the pure imaginary eigenvalues of H, if any, are all even.

Definition 1.2.14. We call the solutions X_+ and X_- of the previous two theorems *maximal* and *minimal* solutions of (1.20), respectively.

To better understand the origin of this definition, let us consider the following theorem:

Theorem 1.2.15 ([5])**.** Assume $B \succeq 0$. Suppose that the CARE (1.20) has X_+ and X_- among its solutions. If X is any other Hermitian solution, then $X_- \preceq X \preceq X_+$.

Let us now examine how the notions of stabilizability and detectability for a dynamical system are linked to the properties of the maximal and minimal solutions of the corresponding CARE.

Theorem 1.2.16 ([5])**.** Let $B, C \succeq 0$. The pair (A, B) is stabilizable if and only if the matrix X_+ is the unique Hermitian positive semidefinite solution of (1.20) such that the eigenvalues of $A - BX_+$ have nonpositive real part. The pair (A, B) is stabilizable and (A, C) is detectable if and only if the matrix X_+ is the unique Hermitian positive semidefinite solution of (1.20) such that the eigenvalues of $A - BX_+$ have negative real part.

A similar result holds for X_-.

Theorem 1.2.17 ([5])**.** Let $B, C \succeq 0$. The pair $(-A, B)$ is stabilizable if and only if the matrix X_- is the unique Hermitian negative semidefinite solution of (1.20) such that the eigenvalues of $A - BX_-$ have nonnegative real part. The pair $(-A, B)$ is stabilizable and $(-A, C)$ is detectable if and only if the matrix X_- is the unique Hermitian negative semidefinite solution of (1.20) such that the eigenvalues of $A - BX_-$ have positive real part.

For more details, see [5, 46].

From the previous theory, it follows that any solution of an algebraic Riccati equation can be obtained from an invariant subspace of a suitable matrix. The most straightforward way to find an invariant subspace is through eigenvectors, but this procedure may be ill-conditioned, so we will consider a more numerically stable scheme based on the Schur decomposition.

Theorem 1.2.6 shows that in order to compute the stabilizing solution of (1.20), it is sufficient to compute a basis of the stable invariant subspace of $H = \begin{bmatrix} A & -B \\ -C & -A^{\mathsf{T}} \end{bmatrix}$,

which is unique by the splitting of eigenvalues of H. To do that, we first compute the complex Schur decomposition of H, i.e. $H = UTU^{\mathsf{H}}$, with $U \in \mathbb{C}^{2n\times 2n}$ unitary and $T \in \mathbb{C}^{2n\times 2n}$ upper triangular. Then, we reorder T by permuting its columns so that the eigenvalues with negative real parts are in the top-left block, the stable subspace:

$$H = UVV^{\mathsf{H}}TVV^{\mathsf{H}}U* = U_{\mathbf{new}}T_{\mathbf{new}}U_{\mathbf{new}}^{\mathsf{H}},$$

where $V \in \mathbb{C}^{2n\times 2n}$ is a unitary matrix. By partitioning $U_{\mathbf{new}}$ and $T_{\mathbf{new}}$ into four $n \times n$ blocks

$$U_{\mathbf{new}} = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix}, \qquad T_{\mathbf{new}} = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix},$$

we can write

$$H \begin{bmatrix} U_{11} \\ U_{21} \end{bmatrix} = \begin{bmatrix} U_{11} \\ U_{21} \end{bmatrix} T_{11},$$

and, thus, the stabilizing solution is given by $X = U_{21}U_{11}^{-1}$. Note that since we have to compute the Schur decomposition of H to obtain X, the complexity cost of this method is $\mathcal{O}(n^3)$ flops.
The implementation is shown in Algorithm 5.

In the literature, one may also find a more general version of CAREs called generalized continuous-time algebraic Riccati equations.

Definition 1.2.18. A *Generalized Continuous-time Algebraic Riccati Equation* (GCARE) is an equation of the form

$$C + E^{\mathsf{T}}XA + A^{\mathsf{T}}XE - E^{\mathsf{T}}XBXE = 0, \tag{1.21}$$

where all the coefficient matrices belong to the space $\mathbb{R}^{n\times n}$, with $B = B^{\mathsf{T}}$ and $C = C^{\mathsf{T}}$.

Remark 1.2.19. Recall that if $\det(E) \neq 0$, then $\det(E^{\mathsf{T}}) \neq 0$. Therefore, under the assumption that E is nonsingular, by multiplying (1.21) on the left by E^{-*1} and on the right by E^{-1}, the GCARE is reduced to the CARE

$$\tilde{C} + X\tilde{A} + \tilde{A}^{\mathsf{T}}X - XBX = 0,$$

where $\tilde{A} = AE^{-1}$ and $\tilde{C} = E^{-*}CE^{-1}$.

[1]We denote $E^{-T} = \left(E^{\mathsf{T}}\right)^{-1}$.

Algorithm 5 Invariant subspace method for CARE

function $X = \texttt{inv\_sub}(A, B, C)$

1: $n = \texttt{size}(A, 1)$
2: **% Compute H and its Complex Schur Decomposition**
3: $H = \begin{bmatrix} A & -B \\ -C & -A^{\mathsf{T}} \end{bmatrix}$
4: $[U, T] = \texttt{schur}(H, \texttt{'complex'})$
5: $[V, \tilde{B}] = \texttt{schur}(B^{\mathsf{T}}, \texttt{'complex'})$
6: **% Reorder the Schur form to bring eigenvalues with $Re(\lambda) < 0$ to the top-left**
7: $eigs\_H = \texttt{ordeig}(T)$
8: $select = \texttt{real}(eigs\_H) < 0$
9: $[U_{\text{new}}, T_{\text{new}}] = \texttt{ordschur}(U, T, select)$
10: $U_{11} = U_{\text{new}}(1 : n, 1 : n)$
11: $U_{21} = U_{\text{new}}(n + 1 : end, 1 : n)$
12: Solve $XU_{11} = U_{21}$ for X
13: **return** X

Chapter 2

Large Scale Methods

In the previous chapter, we introduced the main concepts and results that will be used throughout this work, and we presented several classical solution methods for different types of matrix equations. However, when dealing with real-world data, the dimensions of the matrices are often very large. In such cases, the algorithms discussed earlier become impractical due to their high computational cost and memory requirements. To address large-scale problems efficiently, one typically represents the coefficient matrices in low-rank or sparse form and develops iterative schemes that exploit this structure.
In this chapter, we present recent methods for solving Lyapunov equations and algebraic Riccati equations. We begin with the Alternating Direction Implicit (ADI) method [4] for solving large Lyapunov equations, followed by two of its low-rank variants based on the ZZ^{H} [28] and LDL^{T} [27] decomposition of the solution X. Next, we recall Newton's method [38] and provide two Newton-based schemes for solving CAREs: the Newton–Kleinman method [25] and Newton's method with exact line search [14]. Finally, we introduce a more recent method for solving algebraic Riccati equations, the RADI method [9]. For each of these approaches, we provide MATLAB implementations along with observations on their properties and derivation.

2.1 Alternating Direction Implicit

The *alternating direction implicit* (ADI) method [4] is a very common scheme used to solve large and sparse Lyapunov equations of the form

$$AX + XA^{\mathsf{T}} = Q, \tag{2.1}$$

with $A,\ Q \in \mathbb{R}^{n\times n}$ and $Q \preceq 0$, where A is a sparse matrix and Q is a low-rank matrix. Assume that (2.1) has a unique solution X. The ADI iteration is defined by

$$(A + p_k I_n)X_{k-\frac{1}{2}} = Q - X_{k-1}(A^{\mathsf{T}} - p_k I_n), \tag{2.2}$$

$$(A + p_k I_n)X_k = Q - X_{k-\frac{1}{2}}^{\mathsf{H}}(A^{\mathsf{T}} - p_k I_n), \tag{2.3}$$

for $k = 1, 2, \ldots, k_{max}$, where $X_0 = 0$ is the initial guess and the shift parameters $p_k \in \mathbb{C}$ are called *ADI parameters*. The complete implementation is provided in Algorithm 6.

One crucial problem in the numerical implementation of the ADI method is the choice of the shift parameters, which can significantly affect the convergence of the method, as shown by the following theorem.

Theorem 2.1.1 ([4])**.** Let X be the solution of the Lyapunov equation (2.1). If A is diagonalizable, in particular if $T^{-1}AT$ is a diagonal matrix, and X_k is the matrix obtained at the k-th step of the ADI iteration with parameters $p_1, \ldots, p_k$, such that $\{p_1, \ldots, p_k\} = \{\overline{p_1}, \ldots, \overline{p_k}\}$ and $X_0 = 0$, then

$$||X - X_k||_2 \leq \mu_2^2(T) f(p_1, \ldots, p_k)^2 ||X||_2,$$

with $\mu_2(T) = ||T||_2||T^{-1}||_2$ and

$$f(p_1, \ldots, p_k) = \max_{x \,\in\, \mathrm{spec}(A)} \left| \prod_{i=1}^{k} \frac{x - \overline{p_i}}{x + p_i} \right| = \max_{x \,\in\, \mathrm{spec}(A)} \left| \mathcal{F}_{p_1}(x) \cdots \mathcal{F}_{p_k}(x) \right|,$$

where $\mathcal{F}_\gamma(z) := (z + \gamma)^{-1}(z - \overline{\gamma})$.

This result suggests an approach for selecting the optimal shift parameters to accelerate the convergence, based on solving an optimization problem. Let k_{max} be the fixed

Algorithm 6 ADI method

function $[X, \texttt{normres}, \texttt{iter}] = \texttt{adi}(A, Q, \texttt{p}, k_{\max}, \texttt{tol})$

$n = \texttt{size}(A, 1)$, $n_{\mathrm{p}} = \texttt{length}(\texttt{p})$
$R = -Q$
$\texttt{normres} = \|R\|_F$
$\texttt{iter} = 0$
$X_1 = 0$
$\texttt{cond} = \texttt{normres} > \texttt{tol}$ && iter$< k_{\max}$
while cond **do**
    $p = \texttt{p}(\texttt{mod}(\texttt{iter}, n_{\mathrm{p}}) + 1)$
    $T = A^{\mathsf{T}} - pI_n$
    $\texttt{LHS} = A + pI_n$, $\texttt{RHS}_1 = Q - X_1 T$
    % **Compute** $X_{k-\frac{1}{2}}$
    Solve $\texttt{LHS} \cdot X_{\mathrm{half}} = \texttt{RHS}_1$ for X_{half}
    $\texttt{RHS}_2 = Q - X_{\mathrm{half}}^{\mathsf{H}} T$
    % **Compute** X_k
    Solve $\texttt{LHS} \cdot X_2 = \texttt{RHS}_2$ for X_2
    $\Delta = X_2 - X_1$
    $R = R + A\Delta + \Delta A^{\mathsf{T}}$
    $\texttt{iter} = \texttt{iter} + 1$
    $\texttt{normres}(\texttt{iter}) = \|R\|_F$
    $X_1 = X_2$
    $\texttt{cond} = \texttt{normres} > \texttt{tol}$ && $\texttt{iter} < k_{\max}$
end while
return $X = X_1$

maximum number of ADI iterations. We aim to find the parameters that minimize the function $f(p_1, \ldots, p_{k_{max}})$ of Theorem 2.1.1, i.e. we need to solve the following min-max problem

$$\min_{p_1,\ldots,p_{k_{max}}} \max_{x\in\mathcal{R}} \left| \prod_{i=1}^{k_{max}} \frac{x-\overline{p_i}}{x+p_i} \right|, \tag{2.4}$$

where $\mathcal{R} \subset \mathbb{C}$ is either the spectrum of A or a set containing it, and, for $i = 1, \ldots, k$, either p_i is real or there exists $j \in \{1, \ldots, k_{max}\}$ such that $p_j = \overline{p_i}$. The solution of (2.4) is generally not known, but, in some cases, it is, e.g. when the eigenvalues of A are all real and negative [44, 45]. For large-scale matrices, the whole spectrum is usually unknown and, therefore, one uses, e.g., a small number of Ritz values which are used to solve the optimization problem in an approximate sense to get the so called *heuristic shift parameters*. Others approaches can be found in [43, 34, 28, 26].

Remark 2.1.2. From (2.2), it follows that we can express the solution X_k in terms of the solution at the previous iteration as follows:

$$X_k = \mathcal{F}_{p_k}(A) X_{k-1}^{\mathsf{H}} \mathcal{F}_{\overline{p_k}}(A^{\mathsf{T}}) + R_k, \ k \geq 1,$$

where $\mathcal{F}_\gamma(z)$ is the function defined in Theorem 2.1.1 and $R_k = (A + p_k I_n)^{-1} Q (I - \mathcal{F}_{\overline{p_k}}(A^{\mathsf{T}})) = 2\Re(p_k)(A + p_k I_n)^{-1} Q (A^{\mathsf{T}} + \overline{p_k} I_n)^{-1}$. Moreover, let us note that $\mathcal{F}_{p_k}(A)^{\mathsf{H}} = \mathcal{F}_{\overline{p_k}}\left(A^{\mathsf{T}}\right)$. In fact,

$$\begin{aligned} \mathcal{F}_{p_k}(A)^{\mathsf{H}} &= \left(A^{\mathsf{T}} - p_k I_n\right)\left(A^{\mathsf{T}} + \overline{p_k} I_n\right)^{-1} \\ &= I_n - 2\Re(p_k)\left(A^{\mathsf{T}} + \overline{p_k} I_n\right)^{-1} \\ &= \left(A^{\mathsf{T}} + \overline{p_k} I_n\right)^{-1}\left(A^{\mathsf{T}} - p_k I_n\right) = \mathcal{F}_{\overline{p_k}}\left(A^{\mathsf{T}}\right). \end{aligned}$$

Since X_0 and R_k are Hermitian for any k, then X_k is Hermitian for any iteration k.

2.1.1 Low-rank ADI

In the large-scale context, the structure of the Lyapunov equations we want to solve is slightly different from (2.1). That is because we assume that the coefficient matrices in the equation have special properties, such as sparsity or low-rank decomposition structure, in order to save computational cost and memory. Let us now introduce a variant of

the ADI method that exploits the low-rank representation of Q, called *Cholesky factor ADI* (CF-ADI) [28].
Let us consider the following Lyapunov equation:

$$AX + XA^{\mathsf{T}} = -BB^{\mathsf{T}}, \tag{2.5}$$

where $A \in \mathbb{R}^{n\times n}$ is assumed to be stable and $B \in \mathbb{R}^{n\times l}$ is full rank, with $l \ll n$. Thus, $Q = -BB^{\mathsf{T}}$ is negative semidefinite and low-rank. Let us assume $\Re(p_k) < 0$ for any k. From Remark 2.1.2, since $Q \preceq 0$ and $X_0 = 0$, it follows that $X_k \succeq 0$ for each k and

$$\texttt{rank}(X_k) \leq \texttt{rank}(X_{k-1}) + \texttt{rank}(B) \leq \texttt{rank}(X_{k-2}) + 2 \cdot \texttt{rank}(B) \leq \ldots \leq k \cdot l.$$

The idea of the CF-ADI method is to write $X_k = Z_k Z_k^{\mathsf{H}}$, where $Z_k \in \mathbb{C}^{n\times lk}$ is called *low-rank Cholesky factor* of X_k, and to update at each iteration k the factor Z_k without forming X_k explicitly. For given ADI shifts $\{p_1, \ldots, p_k\} \in \mathbb{C}_<$, the low-rank ADI method successively computes

$$V_1 = (A + p_1 I_n)^{-1} B \in \mathbb{C}^{n\times l} \tag{2.6}$$

$$V_k = V_{k-1} - (p_k + \overline{p_{k-1}})(A + p_k I_n)^{-1} V_{k-1} \in \mathbb{C}^{n\times l},\ k \geq 2. \tag{2.7}$$

At the k-th iteration, the approximate low-rank solution factor is

$$Z_k = \left[\sqrt{-2\Re(p_1)}V_1, \ldots, \sqrt{-2\Re(p_k)}V_k\right] \in \mathbb{C}^{n\times kl},$$

i.e.

$$Z_k = \left[Z_{k-1}, \sqrt{-2\Re(p_k)}V_k\right].$$

Let us now investigate the low-rank nature of the residual [26]. For $k \geq 2$, the identity (2.7) can be written as

$$\begin{aligned} V_k = (I_n - (p_k + \overline{p_{k-1}})(A + p_k I_n)^{-1})V_{k-1} &= (A - \overline{p_{k-1}} I_n)(A + p_k I_n)^{-1} V_{k-1} \\ &= \left(\prod_{j=2}^{k} (A - \overline{p_{j-1}} I_n)(A + p_j I_n)^{-1}\right)(A + p_k I_n)^{-1} B. \end{aligned} \tag{2.8}$$

Moreover, observe that the matrices $A \pm pI_n$ and $(A+qI_n)^{-1}$ commute for all $p, q \in \mathbb{C} \setminus \operatorname{spec}(A)$. In fact,

$$(A+pI_n)(A+qI_n)^{-1} = I_n + (p-q)(A+qI_n)^{-1} = (A+qI_n)^{-1}(A+pI_n) \tag{2.9a}$$

$$(A-pI_n)(A+qI_n)^{-1} = I_n - (p+q)(A+qI_n)^{-1} = (A+qI_n)^{-1}(A-pI_n). \tag{2.9b}$$

Let us denote $W_{k-1} := \left(\prod_{j=1}^{k-1}(A-\overline{p_j}I_n)(A+p_jI_n)^{-1}\right)B$ and set $W_0 = B$. Thus, (2.8) becomes

$$V_k = (A+p_kI_n)^{-1}\left(\prod_{j=1}^{k-1}(A-\overline{p_j}I_n)(A+p_jI_n)^{-1}\right)B = (A+p_kI_n)^{-1}W_{k-1}.$$

It follows that

$$\begin{aligned} W_k = (A-\overline{p_k}I_n)V_k &= (A-\overline{p_k}I_n)(A+p_kI_n)^{-1}W_{k-1} \\ &= (I_n - 2\Re(p_k)(A+p_kI_n)^{-1})W_{k-1} = W_{k-1} - 2\Re(p_k)V_k \in \mathbb{C}^{n\times l}. \end{aligned}$$

Therefore, we have found a recurrence relation to update the residual W_{k-1} using V_k. From (2.8), we obtain another expression to write W_k:

$$W_k = (A-\overline{p_k}I_n)V_k = \prod_{j=1}^{k}(A-\overline{p_j}I_n)(A+p_jI_n)^{-1}B = \mathcal{P}_kB, \tag{2.10}$$

with $\mathcal{P}_k = \mathcal{P}_k(A, p_1, \ldots, p_k) := \prod_{j=1}^{k}(A-\overline{p_j}I_n)(A+p_jI_n)^{-1}$. To justify the introduction of W_k, we recall [11], where it is proved that the Lyapunov residual at the k-th step R_k can be written as

$$R_k = AZ_kZ_k^{\mathsf{H}} + Z_kZ_k^{\mathsf{H}}A^{\mathsf{T}} + BB^{\mathsf{T}} = \mathcal{P}_kBB^{\mathsf{T}}\mathcal{P}_k = W_kW_k^{\mathsf{T}},$$

i.e. at iteration k, the residual has rank at most l. Actually, if $p_k \notin \operatorname{spec}(A)$ for any k, then the rank is exactly l.

Remark 2.1.3 (Real arithmetic)**.** As we have already enlightened, the choice of the ADI shifts is crucial for improving the convergence of the method. One aspect we should also

care about is that complex shifts require complex arithmetic computations and more memory. For this purpose, Benner et al. [10] introduced a reformulated low-rank ADI iteration, where they exploit the fact that the ADI shifts need to occur either as a real number $p_k \in \mathbb{R}_<$ or as a pair of complex conjugate numbers $p_k \in \mathbb{C}_<$, $p_{k+1} = \overline{p_k}$.

Remark 2.1.4 (Complexity)**.** The CF-ADI method results in considerable savings in computational time and memory, and the complexity of this method is considerably less than that of standard ADI as shown in Table 2.1 [28].

Table 2.1: Complexity of CF-ADI and ADI.
J is the total number of ADI iterations.

	CF-ADI	**ADI**
Sparse A	$\mathcal{O}(Jln)$	$\mathcal{O}(Jn^2)$
Full A	$\mathcal{O}(Jln^2)$	$\mathcal{O}(n^3) + \mathcal{O}(Jn^2)$

In Algorithm 7, we provide the implementation of the CF-ADI in a more general case, where the coefficient of the linear term has the form $A + UV^\mathsf{T}$, for $U,\ V$ low-rank matrices. In this case, we exploit this structure for the solution of the linear systems by using the Sherman-Morrison-Woodbury identity [23].

As we will see later, the scheme we have just presented can still be improved, especially when solving differential Riccati equations with backward differentiation formulas (see Chapter 4). In that case, the decomposition $X = ZZ^\mathsf{H}$ of the solution may lead to complex arithmetic, which in turn makes complex storage unavoidable. In the following, we present a recent ADI iteration based on an LDL^T decomposition that keep the computations in real arithmetic [27].
Let us consider a Lyapunov equation of the form

$$A^\mathsf{T}X + XA = -GSG^\mathsf{T}, \tag{2.11}$$

where $G \in \mathbb{R}^{n\times l},\ l \ll n$ and $S = S^\mathsf{T} \in \mathbb{R}^{l\times l}$. The main idea of the scheme is to split the right-hand side of the equation in the form GSG^T and the solution as $X = LDL^\mathsf{T}$, where L will be of low rank and D is a symmetric and block-diagonal matrix. The ADI

Algorithm 7 ZZ^{H}-factorization based ADI method

function $[Z, \texttt{normres}, \texttt{iter}] = \texttt{cf\_adi}(A, T, U, B, \texttt{p}, k_{\max}, \texttt{tol})$

$n_{\mathrm{p}} = \texttt{length}(\texttt{p})$, $n = \texttt{size}(A, 1)$, $m = \texttt{size}(T, 2)$
$W = B$, $Z = [\,]$
if $\|W^{\mathsf{H}}W\|_F < 1\mathrm{e}-14$ **then**
    $Z = \texttt{zeros}(n, 0)$
    $\texttt{normres} = 0$, $\texttt{iter} = 0$
else
    $\texttt{iter} = 1$, $\texttt{normres} = \|W^{\mathsf{H}}W\|_F$
    $\texttt{cond} = \texttt{normres}(\texttt{iter}) > \texttt{tol}$ && $\texttt{iter} < k_{\max}$
    while cond **do**
        $p_1 = \texttt{p}(\texttt{mod}(\texttt{iter}-1, n_{\mathrm{p}}) + 1)$
        $\widetilde{A} = A + p_1 I$
        if $T = 0$ **and** $U = 0$ **then**
            $V = \Re(\widetilde{A}^{-1}W)$
        else
            $M = \widetilde{A}^{-1}W$, $N = \widetilde{A}^{-1}T$, $L = I + U^{\top}N$
            $V = M - N(L^{-1}(U^{\top}M))$
        end if
        if $\Im(p_1) == 0$ **then**
            $W = W - 2p_1 V$
            $V = \sqrt{-2\Re(p_1)}V$
        else
            $\gamma = 2\sqrt{-\Re(p_1)}$, $\delta = \Re(p_1)/\Im(p_1)$
            $W = W + \gamma^2(\Re(V) + \delta \cdot \Im(V))$
            $V = [\gamma(\Re(V) + \delta \cdot \Im(V)), \gamma\sqrt{\delta^2+1} \cdot \Im(V)]$
        end if
        $Z = [Z, V]$
        $\texttt{iter} = \texttt{iter} + 1$, $\texttt{normres}(\texttt{iter}) = \|W^{\mathsf{H}}W\|_F$
        $\texttt{cond} = \texttt{normres}(\texttt{iter}) > \texttt{tol}$&&$\texttt{iter} < k_{\max}$
    end while
end if
return Z

iteration becomes

$$L_k D_k L_k^{\mathsf{T}} = -2\Re(\mu_k)(A^{\mathsf{T}} + \mu_k I_n)^{-1} G S G^{\mathsf{T}} (A + \overline{\mu_k} I_n)^{-1} \tag{2.12}$$

$$+ (A^{\mathsf{T}} + \mu_k I_n)^{-1}(A^{\mathsf{T}} - \mu_k I_n) L_{k-1} D_{k-1} L_{k-1}^{\mathsf{T}} (A - \mu_k I_n)(A + \mu_k I_n)^{-1}, \tag{2.13}$$

with $L_0,\ D_0 = [\,]$ and ADI shift parameters $\mu_k \in \mathbb{C}$. Therefore, we can compute L_k and D_k as follows:

$$L_k := [(A^{\mathsf{T}} + \mu_k I_n)^{-1} G,\ (A^{\mathsf{T}} + \mu_k I_n)^{-1}(A^{\mathsf{T}} - \mu_k I_n) L_{k-1}],$$
$$D_k := \begin{bmatrix} -2\Re(\mu_k) S & \\ & D_{k-1} \end{bmatrix}.$$

Let us simplify the denotation by setting $R_k := (A^{\mathsf{T}} + \mu_k I_n)^{-1}$ and $T_k := A^{\mathsf{T}} - \mu_k I_n$. From the recurrence formula above and the commutativity of the R_k's and T_k's, it holds

$$L_k = [R_1 G,\ R_2 T_1 (R_1 G), \ldots, R_{k+1} T_k (R_k T_{k-1} \ldots R_2 T_1 R_1 G)],$$
$$D_k = \begin{bmatrix} -2\Re(\mu_1) S & & & \\ & -2\Re(\mu_2) S & & \\ & & \ddots & \\ & & & -2\Re(\mu_k) S \end{bmatrix} = -2\mathrm{diag}(\Re(\mu_1), \ldots, \Re(\mu_k)) \otimes S.$$

Note that the introduction of the potentially indefinite matrices S and D_k in the decomposition of the right-hand side and the solution, respectively, avoids the need for complex storage and arithmetic. We report an implementation of this scheme in Algorithm 8.

Remark 2.1.5 (Column compression)**.** As the iterations proceed, the number of columns of Z in Algorithm 7, or of L_k and D_k in Algorithm 8, increases. This results in increased memory requirements for storing the solution. One way to keep the factors as small as possible is to perform a column compression operation at the end of the while loop. Given a prescribed tolerance `tol` as a truncation criterion, a naive column compression for the ZZ^{H} decomposition is given by Algorithm 9. For the LDL^{T} decomposition, the procedure is analogous [27].

Algorithm 8 LDL^{T}-factorization based ADI method

function $[L, D] = \texttt{ldlt\_adi}(\mu_1, \ldots, \mu_k, A, G, S, \texttt{tol})$

% ADI shifts: $\mu_1, \ldots, \mu_k \in \mathbb{C}$

$W_0 = G$, $j = 1$
while $\|W_{j-1} S W_{j-1}^{\mathsf{T}}\|_2 \ \geq \ \texttt{tol} \cdot \|GSG^{\mathsf{T}}\|_2$ **do**
  Solve $(A + \mu_j I_n)V_j = W_{j-1}$ for V_j
  if $\mu_j \in \mathbb{R}$ **then**
    $W_j = W_{j-1} - 2\mu_j V_j$
    $L_j = [L_{j-1},\ V_j]$
  else
    $\eta_j = \sqrt{2}, \quad \delta_j = \dfrac{\Re(\mu_j)}{\Im(\mu_j)}$
    $W_{j+1} = W_{j-1} - 4\Re(\mu_j)\left(\Re(V_j) + \delta_j \Im(V_j)\right)$
    $L_{j+1} = [L_{j-1},\ \eta_j(\Re(V_j) + \delta_j \Im(V_j)),\ \eta_j \sqrt{\delta_j^2 + 1}\, \Im(V_j)]$
    $j = j + 1$
  end if
  $j = j + 1$
end while
$D_j = -2\,\mathrm{diag}(\Re(\mu_1), \ldots, \Re(\mu_j)) \otimes S^{-1}$
return L, D

Algorithm 9 Column Compression

function $Z = \texttt{col\_comp}(Z, \texttt{tol})$

$[A, S, \_] = \texttt{svd}(Z, 0)$
Choose last index i such that $S(i,i)/S(1,1) \geq \texttt{tol}$
% **Truncation**
$Z = A(:, 1\!:\!i)\, S(1\!:\!i, 1\!:\!i)$
return Z

2.2 Newton's Method

In the previous section, we presented a highly efficient scheme for solving large-scale Lyapunov equations. In what follows, we introduce a classic iterative approach for solving nonlinear equations: Newton's method [38]. We apply this method to compute the solution of algebraic Riccati equations, which are nonlinear due to the quadratic term involving the solution. The idea, in this case, is to linearize the ARE using Newton's method and then solve the resulting Lyapunov equation with the CF-ADI algorithm.
Let us now recall the general Newton's procedure to compute the solution X of the problem $\mathcal{F}(X) = 0$, where $\mathcal{F} : \mathcal{V} \to \mathcal{V}$ is a differentiable operator in a Banach space. The iteration is defined by

$$X_{k+1} = X_k - (\mathcal{F}'_{X_k})^{-1}[\mathcal{F}(X_k)], \qquad X_0 \in \mathcal{V}, \tag{2.14}$$

where $\mathcal{F}'_X$ is the Fréchet derivative of $\mathcal{F}$ at the point X. To avoid the explicit construction of $\mathcal{F}'_{X_k}$, which can be unstable and expensive, at each step we solve the equivalent problem

$$\mathcal{F}'_{X_k}[H_k] = -\mathcal{F}(X_k), \qquad X_{k+1} = X_k + H_k, \tag{2.15}$$

where $H_k := X_{k+1} - X_k$ is called *Newton increment*. Once we compute H_k, there are different ways to update the solution. In the following, we present two of these methods: Newton-Kleinman and Newton with exact line search.

2.2.1 Newton-Kleinman

Assume we want to solve the CARE

$$\mathcal{R}(X) := Q + XA + A^{\mathsf{T}}X - XSX = 0, \tag{2.16}$$

where $A,\ Q,\ S,\ X \in \mathbb{R}^{n\times n},\ Q = Q^{\mathsf{T}},\ S = S^{\mathsf{T}}$, and $\mathcal{R}(X)$ is the Riccati operator. As we have already highlighted, it is nonlinear. Thus, to linearize it following (2.15), we first have to compute the Fréchet derivative $\mathcal{R}'_X$ of $\mathcal{R}$ at $X \in \mathcal{V}$, where, in this case, $\mathcal{V} = \mathbb{C}^{n\times n}$. Let $H \in \mathbb{C}^{n\times n}$ and $h \in \mathbb{R}$. Then,

$$\mathcal{R}(X + hH) := Q + XA + hHA + A^{\mathsf{T}}X + hA^{\mathsf{T}}H - XSX - hXSH - hHSX - h^2HSH.$$

Therefore, by applying the definition of Fréchet derivative,

$$\begin{aligned}\lim_{h\to 0}\left[\mathcal{R}(X+hH)-\mathcal{R}(X)\right] &= \lim_{h\to 0}\left(hHA+hA^{\mathsf{T}}H-hXSH-hHSX-h^2HSH\right)\\ &= (A-SX)^{\mathsf{T}}H+H(A-SX),\end{aligned}$$

we conclude that the Riccati operator $\mathcal{R}$ is differentiable at X and $\mathcal{R}'_X[H] = (A-SX)^{\mathsf{T}}H + H(A-SX)$, for any $H \in \mathbb{C}^{n\times n}$. The Newton iteration (2.15) applied to (2.16) is given by

$$\mathcal{R}'_{X_k}[H_k] = -\mathcal{R}(X_k), \qquad\qquad X_{k+1} = X_k + H_k. \tag{2.17}$$

To compute the solution H_k, we need to solve a Lyapunov equation. This can be done efficiently using either the ADI or CF-ADI method, depending on the structure of the coefficient matrices, see Algorithms 6 and 7. In particular, let us notice that H_k is Hermitian, as it is solution of a Lyapunov equation. Moreover, if X_0 is Hermitian, then X_{k+1} is also Hermitian for any k. Let us now recall that Newton's method requires an initial guess X_0 to start [21], and its choice is crucial for the convergence of the method: an initial guess that is too far from the exact solution may cause the method to fail, whereas one that is close to the solution can significantly accelerate convergence. In the following, we state a few results that characterize good choices for X_0 when solving CAREs.

Theorem 2.2.1 ([25, 20])**.** Assume that $S,\ Q \succeq 0$ and the pairs (A,S) and (A^{T},Q) are stabilizable. If X_0 is any Hermitian matrix such that $\operatorname{spec}(A-SX_0) \subset \mathbb{C}_<$, then Newton's method applied to (2.16) yields a sequence of Hermitian matrices $\{X_k\}_{k\geq 0}$ such that

1. $\operatorname{spec}(A-SX_k) \subset \mathbb{C}_<$ for every k;
2. $X_1 \succeq X_2 \succeq \ldots \succeq X_+$;
3. $\lim_{k\to\infty} X_k = X_+$;
4. for any matrix norm $||\cdot||$, there exists a constant $c > 0$ such that $||X_{k+1}-X_+|| \leq c||X_k-X_+||^2$ for $k \geq 0$, i.e. the convergence is globally quadratic;

where X_+ is assumed to be the unique stabilizing solution of the equation.

Under the hypotheses of the previous theorem, it's sufficient to choose a Hermitian stabilizing matrix X_0 in order to guarantee the quadratic convergence of the method. In what follows, we investigate a numerical procedure to choose such an X_0, so that the method becomes self-contained. We begin by recalling the following lemma.

Lemma 2.2.2 ([37])**.** Let $F,\ Q \in \mathbb{C}^{n\times n}$. If either Q is positive definite or $Q = -GG^{\mathsf{H}},\ G \in \mathbb{C}^{n\times m}$ and $m \leq n$, with the pair (F, G) controllable, then the Lyapunov equation $FX + XF^{\mathsf{H}} = -Q$ has a positive definite solution if and only if F is a stable matrix.

Assume now that A is stable and $S \succeq 0$. Let (A, S) also be controllable, i.e. rank($[A - \lambda I_n,\ S]$)=n for any $\lambda \in \mathbb{C}$.

Remark 2.2.3. (A, S) is controllable if and only if rank($[A-\lambda I_n,\ S]$)=n for any $\lambda \in$ spec(A). In fact, if $\lambda \notin$ spec(A), then $A-\lambda I_n$ is invertible, thus rank$(A-\lambda I_n)$=n and rank($[A-\lambda I_n,\ S]$)=n. The converse also holds.

Let $\lambda \in$ spec(A) and $\beta > 0$. Then $-(\lambda + \beta) \in$ spec(-$(A + \beta I_n)$). We want to choose β such that, for any $\lambda \in$ spec(A), spec$(-(A + \beta I_n)) \subset \mathbb{C}_<$, i.e.

$$\Re(-(\lambda + \beta)) < 0,$$

i.e.

$$\beta > -\Re(\lambda).$$

For this purpose, we can choose $\beta > 0$ and $\beta > \max\limits_{\lambda\in \text{ spec}(A)} -\Re(\lambda) = -\min\limits_{\lambda\in \text{ spec}(A)} \Re(\lambda)$. Observe now that $(A + \beta I_n,\ S)$ is controllable. In fact, using Remark 2.2.3, if $\mu \in$ spec$(A + \beta I_n)$, then there exists $\lambda \in$ spec(A) such that $\mu = \lambda + \beta$. Therefore, for any $\mu \in$ spec$(A + \beta I)$, rank($[(A + \beta I_n)-\mu I_n,\ S]$)=rank($[(A + \beta I_n - \lambda I_n - \beta I_n,\ S]$)=rank($[A - \lambda I_n,\ S]$)=$n$, for any $\lambda \in$ spec(A). Similarly, we can prove that $(-A-\beta I_n,\ \sqrt{2}S)$ is controllable. Now, using Lemma 2.2.2 with $F = -(A + \beta I_n)$ and $G = \sqrt{2}S$, we can conclude that there exists a unique positive definite solution $\hat{X}$ to

$$(A + \beta I_n)X + X(A + \beta I_n)^{\mathsf{T}} = 2SS^{\mathsf{T}}. \tag{2.18}$$

Let us define $X_{-1} := S^{\mathsf{T}}\hat{X}^{-1}$, which is well-defined since $\hat{X} \succ 0$, thus invertible. Then, we can rewrite equation (2.18) as

$$(A - SX_{-1})\hat{X} + \hat{X}(A - SX_{-1})^{\mathsf{T}} = -2\beta\hat{X}. \tag{2.19}$$

Again, from Lemma 2.2.2 with $F = A - SX_{-1}$ and $Q = 2\beta\hat{X} \succ 0$, and using that $\hat{X}$ is the positive define solution of $FX + XF^{\mathsf{T}} = -Q$, we can conclude that $F = A - SX_{-1}$ is stable, i.e. X_{-1} is a stabilizing matrix. However, in general, X_{-1} is not Hermitian, but Theorem 2.2.1 suggests to choose an initial guess which is both stabilizing and Hermitian. To do that, let us introduce a matrix M such that $M \succ X_{-1}^{\mathsf{T}}SX_{-1}$. Note that M is positive definite,

because we assumed $S \succeq 0$. In fact, for any $x \in \mathbb{R}^n \setminus \{0\}$, we have $x^\mathsf{T} M x > x^\mathsf{T} X_{-1}^\mathsf{T} S X_{-1} x = (X_{-1}x)^\mathsf{T} S (X_{-1}x) \geq 0$. Hence, $x^\mathsf{T} M x > 0$ for any $x \in \mathbb{R}^n \setminus \{0\}$, so M is positive definite. If we apply Lemma 2.2.2 to the equation

$$(A - SX_{-1})^\mathsf{T} X + X(A - SX_{-1}) = -M, \tag{2.20}$$

using the fact that $A - SX_{-1}$ is stable, and so it is $(A - SX_{-1})^\mathsf{T}$, and $M \succ 0$, we can deduce that there exists a unique solution X_0 that is positive definite. By adding and subtracting suitable terms, and setting $X = X_0$, we can rewrite (2.20) in this form

$$(A - SX_0)^\mathsf{T} X_0 + X_0(A - SX_0) = -M - (X_0 - X_{-1})^\mathsf{T} S (X_0 - X_{-1}) - X_0 S X_0 + X_{-1}^\mathsf{T} S X_{-1}. \tag{2.21}$$

Finally, applying Lemma 2.2.2 again, we get that $A - SX_0$ is stable, i.e. X_0 is the Hermitian stabilizing solution we were looking for.

Remark 2.2.4. In the more general case in which (A, S) is just stabilizable, it can happen that the matrix $\hat{X}$ we previously obtained is just positive semidefinite, and not positive definite. In that case, a stabilizing matrix X_{-1} is given by $X_{-1} = S^\mathsf{T} \hat{X}^\dagger$, where $\hat{X}^\dagger$ is the Moore-Penrose inverse of $\hat{X}$, often called *pseudoinverse* of $\hat{X}$. This case is particularly relevant for large-scale problems, where the system is often only stabilizable. In such settings, the existence of a positive semidefinite solution is actually desirable: it ensures stabilizing properties while allowing for a low-rank representation. Conversely, if the solution were positive definite, i.e. a full-rank matrix, no exact low-rank decomposition would exist. Therefore, the semidefinite nature of $\hat{X}$ is not a limitation, but rather an inherent and advantageous feature of large-scale stabilizable systems.

In the following, we provide the algorithm structure to compute a suitable initial guess X_0, see Algorithm 10.

The Newton–Kleinman method refers to Kleinman's reformulation of Newton's method for algebraic Riccati equations [25], in which each Newton step is expressed as the solution of a Lyapunov equation. In large-scale settings, these Lyapunov equations are often solved efficiently using iterative schemes such as the ADI method. Algorithm 11 solves

$$Q + XA + A^\mathsf{T} X - XSX = 0, \qquad A,\ Q,\ S \in \mathbb{R}^{n \times n}, \tag{2.22}$$

while Algorithm 12 solves

$$C^\mathsf{T} C + ZZ^\mathsf{H} A + A^\mathsf{T} ZZ^\mathsf{H} - ZZ^\mathsf{H} BB^\mathsf{T} ZZ^\mathsf{H} = 0, \quad A \in \mathbb{R}^{n \times n},\ B \in \mathbb{R}^{n \times r},\ C \in \mathbb{R}^{p \times n}, \tag{2.23}$$

such that $p + r \ll n$.

Algorithm 10 Initial Guess for Newton's Method for CAREs

function $X = \texttt{init\_newton}(A, S)$

% Output: X, initial guess for Newton's method

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

$[U, T_A] = \texttt{schur}(A)$ ▷ Real Schur decomposition

$T_D = U^\mathsf{T} S$

$\beta = -\min\left(\Re(\texttt{ordeig}(T_A))\right)$

$\beta = \max(0, \beta) + 0.5$

Solve $(T_A + \beta I_n)\hat{X} + \hat{X}(T_A + \beta I_n)^\mathsf{T} = 2SS^\mathsf{T}$ using Algorithm 6.

% **Recover** X

$X = T_D^\mathsf{T} \hat{X}^{-1} U^\mathsf{T}$

% **Check whether** X **is Hermitian or not**

if $\|X - X^\mathsf{T}\| > 10^{-13}$ **then**

    $M = X^\mathsf{T} S X + 0.5 \cdot I_n$

    Solve $(A - SX)^\mathsf{T} X + X(A - SX) = -M$ for X

end if

return X

Algorithm 11 Newton-Kleinman with ADI

function $[X, \texttt{normres}, \texttt{iter}, \texttt{iter\_ADI}] = \texttt{nw\_kl}(Q, A, S, X_0, k_{\max}, k_{\max}^{\text{ADI}}, \texttt{tol}, \texttt{tol}_{\text{ADI}})$

$n = \texttt{size}(A, 1), \quad X_1 = X_0$

$R_{\text{temp}} = Q + AX_0 + X_0A^{\mathsf{T}} - X_0SX_0$

$\texttt{normres} = \|R_{\text{temp}}\|_F, \quad \texttt{iter} = 0$

$\texttt{cond} = \texttt{normres} > \texttt{tol}$ && $\texttt{iter} < k_{\max}$

while cond **do**

  $A_{\text{hat}} = A^{\mathsf{T}} - X_1S$

  $X_{\text{ADI}}^{(0)} = 0$

  % **Compute the vector** params **with the shift parameters**

  Solve $A_{\text{hat}}H + HA_{\text{hat}}^{\mathsf{H}} = -R$ using Algorithm 6:

    $[H, \sim, \texttt{iter\_adi}] = \texttt{adi}(A_{\text{hat}}, -R_{\text{temp}}, \texttt{params}, X_{\text{ADI}}^{(0)}, k_{\max}^{\text{ADI}}, \texttt{tol}_{\text{ADI}})$

  $X_2 = X_1 + H$

  `iter = iter + 1`

  $\texttt{iter\_ADI}(\texttt{iter}) = \texttt{iter\_adi}$

  $R = Q + AX_2 + X_2A^{\mathsf{T}} - X_2SX_2$

  $\texttt{normres}(\texttt{iter}) = \|R\|_F$

  $R_{\text{temp}} = R$

  $X_1 = X_2$

  $\texttt{cond} = \texttt{normres(iter)} > \texttt{tol}$ && $\texttt{iter} < k_{\max}$

end while

return X_1

Algorithm 12 Newton-Kleinman with CF-ADI

function $[Z, \texttt{normres}, \texttt{iter}, \texttt{iter\_ADI}] = \texttt{nw\_kl\_lr}(C, A, B, Z_0, k_{\max}, k_{\max}^{\mathrm{ADI}}, \texttt{tol}, \texttt{tol}_{\mathrm{ADI}})$

$n = \texttt{size}(A, 1), \quad p = \texttt{size}(C, 1)$

$X_0 = Z_0 Z_0^{\mathsf{H}}$

$R = C^{\mathsf{T}} C + A^{\mathsf{T}} X_0 + X_0 A - X_0 B B^{\mathsf{T}} X_0$

$\texttt{normres} = \|R\|_F, \quad \texttt{iter} = 0, \quad Z_1 = Z_0$

$\texttt{cond} = \texttt{normres} > \texttt{tol}$ && $\texttt{iter} < k_{\max}$

while cond **do**

  Compute the vector params with the shift parameters

  $T = -Z_1 (Z_1^{\mathsf{H}} B), \quad U = B, \quad G = [C^{\mathsf{T}}, -T]$

  Solve $(A^{\mathsf{T}} + TU^{\mathsf{T}})X + X(A^{\mathsf{T}} + TU^{\mathsf{T}})^{\mathsf{H}} = -GG^{\mathsf{H}}$ using Algorithm 7:

    $[Z_2, \sim, \texttt{iter\_adi}] = \texttt{cf\_adi}(A^{\mathsf{T}}, T, U, G, \texttt{params}, k_{\max}^{\mathrm{ADI}}, \texttt{tol}_{\mathrm{ADI}})$

  $\texttt{iter} = \texttt{iter} + 1, \quad \texttt{iter\_ADI}(\texttt{iter}) = \texttt{iter\_adi}$

  % Compute residual via LDL^{T} factorization

  $l = \texttt{size}(Z_2, 2)$

  $L = [A^{\mathsf{T}} Z_2,\ Z_2,\ C^{\mathsf{T}}], \quad Y = Z_2^{\mathsf{H}} B$

  $D = \begin{bmatrix} & I_l & \\ I_l & -YY^{\mathsf{H}} & \\ & & I_p \end{bmatrix}$

  $[\sim, R] = \texttt{qr}(L, 0)$

  $\texttt{normres}(\texttt{iter}) = \|RDR^{\mathsf{H}}\|_F$

  $\texttt{cond} = \texttt{normres} > \texttt{tol}$ && $\texttt{iter} < k_{\max}$

  $Z_1 = Z_2$

end while

$Z = Z_1$

return Z

2.2.2 Exact Line search

Previously, we have described the canonical way to update the solution X_k once we find H_k, i.e. by simply computing $X_{k+1} = X_k + H_k$. A more efficient procedure to compute X_{k+1} is given by the so called Newton's iteration with *exact line search* [14]. The idea is to introduce, at each step, a Newton's step size t_k such that $X_{k+1} = X_k + t_k H_k$, where t_k is computed through the minimization problem

$$\min_t ||\mathcal{R}(X_k + tH_k)||_F^2.$$

From (2.16), we obtain

$$\mathcal{R}(X_k + tH_k) = \mathcal{R}(X_k) + t[(A - BX_k)^{\mathsf{H}} H_k + H_k(A - BX_k)] - t^2 H_k B H_k.$$

If $V_k := H_k B H_k$ and H_k solves (2.17), then

$$\mathcal{R}(X_k + tH_k) = (1-t)\mathcal{R}(X_k) - t^2 V_k.$$

Thus, the quantity we want to minimize can be rewritten as

$$f_k(t) := ||\mathcal{R}(X_k + tH_k)||_F^2 = \operatorname{trace}(\mathcal{R}(X_k + tH_k)^2) = \alpha_k(1-t)^2 - 2\beta_k(1-t)t^2 + \gamma_k t^4,$$

where $\alpha_k = \operatorname{trace}(\mathcal{R}(X_k)^2),\ \beta_k = \operatorname{trace}(\mathcal{R}(X_k)V_k),\ \gamma_k = \operatorname{trace}(V_k^2)$.
Let us note that if $\gamma_k \neq 0$, then $f_k(t)$ has at most two local minima, one of which is the global minimum. Conversely, if $\gamma_k = 0$, the function $f_k(t)$ attains its global minimum value (zero) at $t = 1$. In this case, the increment V_k vanishes, meaning that any choice of t yields the same result. This situation corresponds to stagnation of the iteration, which can occur only when $H_k = 0$ or $BH_k = 0$, i.e. when X (or equivalently BX) no longer changes, indicating that the solution has been reached.

Remark 2.2.5. By differentiating f, we obtain

$$\begin{aligned}\dot{f}_k(t) &= -2\operatorname{trace}((\mathcal{R}(X_k) + 2tV_k)(\mathcal{R}(X_k + tH_k)) \\ &= -2\operatorname{trace}((\mathcal{R}(X_k) + 2tV_k)((1-t)\mathcal{R}(X_k) - t^2 V_k)).\end{aligned}$$

Since $\dot{f}_k(0) = -2\operatorname{trace}(\mathcal{R}\left(X_k\right)^2) \leq 0$, and $\dot{f}_k(2) = 2\operatorname{trace}\left((\mathcal{R}(X_k) + 4V_k)^2\right) \geq 0$, there exists a local minimum of f_k at some value of $t_k \in [0,2]$. Moreover, if $\mathcal{R}(X_k) \neq 0$, i.e. if X_k is not a solution of (2.16), then $\dot{f}_k(0) < 0$, so the Newton step is a descent direction of $||\mathcal{R}(X_k + tH_k)||_F$. Therefore, for the minimizing $t_k \in [0,2]$, we have $||\mathcal{R}(X_k + tH_k)||_F \leq ||\mathcal{R}(X_k)||_F$ and $||\mathcal{R}(X_k + tH_k)||_F = ||\mathcal{R}(X_k)||_F$ if and only if $\mathcal{R}(X_k) = 0$.

See Algorithm 13 for the implementation. In the low-rank case, the procedure is analogous, but `cf_adi` is used instead.

2.3 RADI method

In this section, we present the RADI method [9], a scheme for solving continuous-time algebraic Riccati equations (CAREs). It can be viewed as a generalization of low-rank methods for Lyapunov equations and is among the fastest-converging algorithms [8]. Consider the following CARE

$$Q + A^{\mathsf{T}}X + XA - XSX = 0, \tag{2.24}$$

where $Q = C^{\mathsf{T}}C,\ S = BB^{\mathsf{T}},\ A \in \mathbb{R}^{n\times n},\ B \in \mathbb{R}^{n\times r},\ C \in \mathbb{R}^{p\times n}$, such that $p + r \ll n$. Given a numerical solution $\hat{X}$ for (2.24), a common way to measure the quality of the approximation is to consider the norm of the residual matrix

$$\mathcal{R}(\hat{X}) := Q + A^{\mathsf{T}}\hat{X} + \hat{X}A - \hat{X}S\hat{X}.$$

The derivation of RADI is based on the following theorem:

Theorem 2.3.1 ([9])**.** Let $\hat{X} \in \mathbb{C}^{n\times n}$ be an approximation to a solution of (2.24).

- Let $X = \hat{X} + \tilde{X}$ be an exact solution of (2.24). Then $\tilde{X}$ is a solution to the residual equation
$$\tilde{A}^{\mathsf{H}}\tilde{X} + \tilde{X}\tilde{A} + \tilde{Q} - \tilde{X}S\tilde{X} = 0, \tag{2.25}$$
where $\tilde{A} = A - S\hat{X}$ and $\tilde{Q} = \mathcal{R}(\hat{X})$;
- Conversely, if $\tilde{X}$ is a solution to (2.25), then $X = \hat{X} + \tilde{X}$ is a solution to the original Riccati equation (2.24). Moreover, if $\hat{X} \succeq 0$ and $\tilde{X}$ is a stabilizing solution to (2.25), then $X = \hat{X} + \tilde{X}$ is the stabilizing solution to (2.24);
- If $\hat{X} \succeq 0$ and $\mathcal{R}(\hat{X}) \succeq 0$, then the residual equation (2.25) has a unique stabilizing solution;
- If $\hat{X} \succeq 0$ and $\mathcal{R}(\hat{X}) \succeq 0$, then $\hat{X} \preceq X$, where X is the stabilizing solution of (2.24).

Therefore, the scheme of the algorithm is the following:

1. Let $\hat{X} = 0$ be the initial guess;

Algorithm 13 Newton's method with exact line search

function $[X, \texttt{normres}, \texttt{iter}, \texttt{iter\_ADI}] = \texttt{nw\_ls}(Q, A, S, X_0, k_{\max}, k_{\max}^{\mathrm{ADI}}, \texttt{tol}, \texttt{tol}_{\mathrm{ADI}})$

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

$R_{\mathrm{temp}} = Q + X_0 A + A^{\mathsf{T}} X_0 - X_0 S X_0$

$\texttt{normres} = \|R_{\mathrm{temp}}\|_F, \quad \texttt{iter} = 0, \quad X_1 = X_0$

$\texttt{cond} = \texttt{normres} > \texttt{tol}$ && $\texttt{iter} < k_{\max}$

while cond **do**

  % Fréchet derivative: $R'_X(H) = (A - SX)^{\mathsf{H}} H + H(A - SX)$

  $\hat{A} = A - SX_1$

  $X_{\mathrm{ADI}}^{(0)} = 0$

  Compute the vector params with the shift parameters

  Solve $\hat{A}^{\mathsf{H}} H + H\hat{A} = -R_{\mathrm{temp}}$ using Algorithm (6):

    $[H, \sim, \texttt{iter\_adi}] = \texttt{adi}(\hat{A}^{\mathsf{H}}, -R_{\mathrm{temp}}, \texttt{params}, X_{\mathrm{ADI}}^{(0)}, k_{\max}^{\mathrm{ADI}}, \texttt{tol}_{\mathrm{ADI}})$

  $\texttt{iter} = \texttt{iter} + 1, \quad \texttt{iter\_ADI}(\texttt{iter}) = \texttt{iter\_adi}$

  % Compute coefficients for line search

  $V = HSH$

  $\alpha = \texttt{trace}(R_{\mathrm{temp}} R_{\mathrm{temp}}), \quad \beta = \texttt{trace}(R_{\mathrm{temp}} V), \quad \gamma = \texttt{trace}(VV)$

  Find t **minimizing**

$$f(t) = \alpha(1-t)^2 - 2\beta(1-t)t^2 + \gamma t^4, \quad t \in [0, 2]$$

  $X_2 = X_1 + tH, \;\; R = Q + X_2 A + A^{\mathsf{T}} X_2 - X_2 S X_2$

  $\texttt{normres}(\texttt{iter}) = \|R\|_F$

  $R_{\mathrm{temp}} = R$, $X_1 = X_2$

  $\texttt{cond} = \texttt{normres}(\texttt{iter}) > \texttt{tol}$ && $\texttt{iter} < k_{\max}$

end while

$X = X_1$

return X

2. Construct the residual equation (2.25);

3. Compute an approximation $\tilde{X}_1$ of the stabilizing solution $\tilde{X}$ for (2.25);

4. Accumulate $\hat{X} = \hat{X} + \tilde{X}_1$.

In step 3, we need to choose $\tilde{X}_1 \succeq 0$ and ensure that $\mathcal{R}(\hat{X} + \tilde{X}) \succeq 0$ in order to guarantee uniqueness of the solution and monotonicity of the sequence of approximate solutions. With appropriate choices of the shift parameters and by exploiting the recursive construction of the equation factors, one obtains Algorithm 14.

Algorithm 14 RADI method with reduced use of complex arithmetic

function $[Z, Y] = \texttt{radi}(A, B, C, \texttt{tol})$

% Output: $X \approx ZY^{-1}Z^{\mathsf{T}}$ solving (2.24) with Z, Y real

$R = C^{\mathsf{T}}, \quad K = 0, \quad Y = [\,], \quad Z = [\,]$

while $\|R^{\mathsf{H}}R\| \ \geq \ \texttt{tol} \cdot \|CC^{\mathsf{T}}\|$ **do**

  Obtain the next shift σ

  if first iteration **then**

    $V = \sqrt{-2\Re(\sigma)} \cdot (A^{\mathsf{T}} + \sigma I_n)^{-1} R$

  else

    $V = \sqrt{-2\Re(\sigma)} \cdot (A^{\mathsf{T}} - KB^{\mathsf{T}} + \sigma I_n)^{-1} R$ ▷ Use SMW if needed

  end if

  if $\sigma \in \mathbb{R}$ **then**

    $Z = [Z, V]$

    $\widetilde{Y} = I - \frac{1}{2\Re(\sigma)} \left(V^{\mathsf{H}}B\right) \left(V^{\mathsf{H}}B\right)^{\mathsf{H}}$

    $Y = \begin{bmatrix} Y & \\ & \widetilde{Y} \end{bmatrix}$

    $R = R + \sqrt{-2\Re(\sigma)}\, V\widetilde{Y}^{-1}$, $K = K + (V\widetilde{Y}^{-1}) \left(V^{\mathsf{H}}B\right)$

  else

    $Z = [Z, \Re(V), \Im(V)]$

    $V_r = (\Re(V))^{\mathsf{H}}B, \quad V_i = (\Im(V))^{\mathsf{H}}B$

    $F_1 = \begin{bmatrix} -\Re(\sigma)V_r - \Im(\sigma)V_i \\ \Im(\sigma)V_r - \Re(\sigma)V_i \end{bmatrix}, \quad F_2 = \begin{bmatrix} V_r \\ V_i \end{bmatrix}, \quad F_3 = \begin{bmatrix} \Im(\sigma)I_p \\ \Re(\sigma)I_p \end{bmatrix}$

    $\widetilde{Y} = \begin{bmatrix} I_p & \\ & \frac{1}{2}I_p \end{bmatrix} - \frac{1}{4|\sigma|^2\Re(\sigma)} F_1F_1^{\mathsf{H}} - \frac{1}{4\Re(\sigma)} F_2F_2^{\mathsf{H}} - \frac{1}{2|\sigma|^2} F_3F_3^{\mathsf{H}}$

    $Y = \begin{bmatrix} Y & \\ & \widetilde{Y} \end{bmatrix}$

    $R = R + \sqrt{-2\Re(\sigma)}\, [\Re(V) \ \ \Im(V)]\widetilde{Y}^{-1}(:, 1:p)$

    $K = K + [\Re(V) \ \ \Im(V)]\widetilde{Y}^{-1} \begin{bmatrix} V_r \\ V_i \end{bmatrix}$

  end if

end while

return Z, Y

Chapter 3

New extension of the ADI method

In chapter 2, we introduced the standard ADI method along with two different low-rank approaches, the ZZ^{H}-based one and the LDL^{T}-based one. Moreover, in Theorem 2.1.1 we presented an upper bound of the error of the numerical approximation depending on the choice of the shift parameters. What we want to highlight now is that all the previous results and implementation regarding the alternating direction implicit scheme have been done by assuming a zero initial guess X_0 as starting approximation for the algorithm. In this chapter, we are presenting an extension of the ADI method in order to support a non-zero initial guess. Note that we consider this extension only for the LDL^{T} low-rank case.

This novelty has been formulated in [33] by considering the LDL^{T}-type low-rank case and it has been derived introducing the notion of *fully commuting splitting schemes* to solve arbitrary linear systems. In Algorithm 15, we provide the implementation of the scheme presented in the paper to solve

$$AX + XA^{\mathsf{T}} = -GSG^{\mathsf{T}},$$

with a large and sparse coefficient matrix $A \in \mathbb{R}^{n\times n}$ and a low-rank term comprised of the factors $G \in \mathbb{R}^{n\times l}$ and $S \in \mathbb{R}^{l\times l}$, where $l \ll n$.

We present now the generalization of Theorem 2.1.1 for the case of an initial guess $X_0 \neq 0$.

Theorem 3.0.1. Let X be the solution of the Lyapunov equation (2.1) If A is diagonalizable, in particular if $T^{-1}AT$ is a diagonal matrix, and X_k is the matrix obtained at the k-th step of the ADI iteration with parameters $p_1, \ldots, p_k$, such that $\{p_1, \ldots, p_k\} = \{\overline{p_1}, \ldots, \overline{p_k}\}$, then

$$||X - X_k||_2 \leq \mu_2^2(T) f(p_1, \ldots, p_k)^2 ||X - X_0||_2,$$

Algorithm 15 ADI with a non-zero initial guess

function $[V_0, V_1, \ldots, T] = \texttt{nonzero\_adi}(A, G, S, Z_0, Y_0, \{\alpha_k\})$

Assemble initial residual factors:

$$R_0 = \begin{bmatrix} G & Z_0 & AZ_0 \end{bmatrix}, \quad T = \begin{bmatrix} S & & \\ & & Y_0 \\ & Y_0 & \end{bmatrix}$$

$k = 0$

repeat

  if $\alpha_k \in \mathbb{R}$ **then** ▷ single step

    $V_k = (A + \alpha_k I_n)^{-1} R_k$

    $R_{k+1} = R_k - 2\Re(\alpha_k) V_k$

    $k = k + 1$

  else ▷ double step; requires $\alpha_{k+1} = \overline{\alpha_k}$

    $\widehat{V}_k = (A + \alpha_k I_n)^{-1} R_k$

    $\delta_k = \Re(\alpha_k) / \Im(\alpha_k)$

    $V_k = \sqrt{2}\left(\Re(\widehat{V}_k) + \delta_k \Im(\widehat{V}_k)\right)$

    $V_{k+1} = \sqrt{2(\delta_k^2 + 1)}\, \Im(\widehat{V}_k)$

    $R_{k+2} = R_k - 2\sqrt{2}\Re(\alpha_k) V_k$

    $k = k + 2$

  end if

until converged

Assemble solution factors (if needed):

$$Z = \begin{bmatrix} Z_0 & V_0 & V_1 & \ldots \end{bmatrix}, \quad Y = \operatorname{blockdiag}\big(Y_0,\, -2\Re(\alpha_0)T,\, -2\Re(\alpha_1)T,\, \ldots\big)$$

return Z, Y

with $\mu_2(T) = ||T||_2||T^{-1}||_2$ and

$$f(p_1, \ldots, p_k) = \max_{x \in \text{spec}(A)} \left| \prod_{i=1}^{k} \frac{x - \overline{p_i}}{x + p_i} \right| = \max_{x \in \text{spec}(A)} |\mathcal{F}_{p_1}(x) \cdots \mathcal{F}_{p_k}(x)|,$$

where $\mathcal{F}_\gamma(z) := (z + \gamma)^{-1}(z - \overline{\gamma})$.

Proof. Let us recall from Remark 2.1.2 that we can express the solution X_k at iteration k in terms of the solution X_{k-1} as follows:

$$X_k = \mathcal{F}_{p_k}(A) X_{k-1}^{\mathsf{H}} \mathcal{F}_{\overline{p_k}}\left(A^{\mathsf{T}}\right) + R_k,$$

where $R_k = 2\Re(p_k)(A + p_k I_n)^{-1} Q (A^{\mathsf{T}} + \overline{p_k} I_n)^{-1}$ and X_k is Hermitian for any k. Therefore, it holds

$$\begin{aligned} X_k - X &= \mathcal{F}_{p_k}(A) X_{k-1} \mathcal{F}_{p_k}(A)^{\mathsf{H}} - X + R_k \\ &= \mathcal{F}_{p_k}(A)[X_{k-1} - X]\mathcal{F}_{p_k}(A)^{\mathsf{H}} + R_k - X + \mathcal{F}_{p_k}(A) X \mathcal{F}_{p_k}(A)^{\mathsf{H}}. \end{aligned}$$

Now, observe that

$$R_k - X + \mathcal{F}_{p_k}(A) X \mathcal{F}_{p_k}(A)^{\mathsf{H}} = 2\Re(p_k)(A + p_k I_n)^{-1}(Q - AX - XA^{\mathsf{T}}) = 0,$$

where the last equality holds because X is the solution of (2.1) by hypothesis. Thus, we obtain the recurrence relation

$$X_k - X = \mathcal{F}_{p_k}(A)[X_{k-1} - X]\mathcal{F}_{p_k}(A)^{\mathsf{H}},$$

which leads to

$$X_k - X = \mathcal{F}_{p_k}(A) \cdots \mathcal{F}_{p_1}(A)[X_0 - X]\mathcal{F}_{p_1}(A)^{\mathsf{H}} \cdots \mathcal{F}_{p_k}(A)^{\mathsf{H}}.$$

Taking the 2-norm and using the submultiplicative property of matrix norms, we obtain

$$||X_k - X||_2 \leq ||\mathcal{F}_{p_k}(A) \cdots \mathcal{F}_{p_1}(A)||_2 \cdot ||X_0 - X||_2 \cdot ||\mathcal{F}_{\overline{p_1}}\left(A^{\mathsf{T}}\right) \cdots \mathcal{F}_{\overline{p_k}}\left(A^{\mathsf{T}}\right)||_2. \tag{3.1}$$

From (2.9), it follows that $\mathcal{F}_p$ and $\mathcal{F}_q$ commute, so the order of the shifts is irrelevant. Moreover, if $\phi(z)$ is a function defined on the spectrum of $A = TDT^{-1}$, then the corresponding matrix function $\phi(A)$ is defined as

$$\phi(A) = T\phi(D)T^{-1}.$$

Therefore, by setting

$$\phi(z) := \mathcal{F}_{p_1}(z) \cdots \mathcal{F}_{p_k}(z),$$

we get

$$||\phi(A)||_2 \leq ||T||_2 \cdot ||\phi(D)||_2 \cdot ||T^{-1}||_2 = \mu_2(T) \cdot \max_{x \in \text{ spec}(A)} |\phi(x)|. \tag{3.2}$$

Since $\{p_1, \ldots, p_k\} = \{\overline{p_1}, \ldots, \overline{p_k}\}$, it follows that

$$\mathcal{F}_{\overline{p_1}}\left(A^{\mathsf{T}}\right) \cdots \mathcal{F}_{\overline{p_k}}\left(A^{\mathsf{T}}\right) = \phi\left(A^{\mathsf{T}}\right) = \left(T^{\mathsf{T}}\right)^{-1} \phi(D) T^{\mathsf{T}}.$$

Then, it holds

$$||\phi\left(A^{\mathsf{T}}\right)||_2 \leq ||T^{\mathsf{T}}||_2 \cdot ||\phi(D)||_2 \cdot ||\left(T^{\mathsf{T}}\right)^{-1}||_2 = \mu_2(T) \max_{x \in \text{ spec}(A)} |\phi(x)|, \tag{3.3}$$

where the last inequality follows from the identity $\mu_2(T^{\mathsf{T}}) = \mu_2(T)$. Finally, using the bounds in (3.2) and (3.3) in (3.1), we conclude that

$$||X_k - X||_2 \leq \mu_2^2(T) \left(\max_{x \in \text{ spec}(A)} |\phi(x)|\right)^2 ||X_0 - X||_2, \tag{3.4}$$

which proves the claim. □

In the following, we present a result that provides an upper bound on the rank of the numerical solution X_k at iteration k.

Theorem 3.0.2. Let us consider the ADI iteration described in (2.2) and let X_0 be a Hermitian initial guess. Then,

$$\text{rank}(X_k) \leq \text{rank}(X_0) + k \cdot \text{rank}(Q), \qquad \text{for any iteration } k.$$

In particular, if $Q = BB^{\mathsf{T}}$, with $B \in \mathbb{R}^{n \times l}$ is full rank, then

$$\text{rank}(X_k) \leq \text{rank}(X_0) + k \cdot l, \qquad \text{for any iteration } k.$$

Proof. From Remark 2.1.2, it follows that

$$\text{rank}(X_k) \leq \text{rank}\left(\mathcal{F}_{p_k}(A) X_{k-1}^{\mathsf{H}} \mathcal{F}_{p_k}(A)^{\mathsf{H}}\right) + \text{rank}(R_k),$$

for any iteration k. Let us also note that

$$\begin{aligned}
&\text{rank}\left(\mathcal{F}_{p_k}(A) X_{k-1}^{\mathsf{H}} \mathcal{F}_{p_k}(A)^{\mathsf{H}}\right) = \text{rank}(X_{k-1}^{\mathsf{H}}), \\
&\text{rank}(R_k) = \text{rank}\left((A + p_k I_n)^{-1} Q \left[(A + p_k I_n)^{-1}\right]^{\mathsf{H}}\right) = \text{rank}(Q),
\end{aligned}$$

since the relation of congruence between matrices preserves the rank. Moreover, since X_0 is Hermitian by hypothesis, it follows from Remark 2.1.2 that $X_{k-1}^{\mathsf{H}} = X_{k-1}$ for any k. Therefore, we obtain the recurrence relation

$$\operatorname{rank}(X_k) \leq \operatorname{rank}(X_{k-1}) + \operatorname{rank}(Q),$$

that leads to

$$\operatorname{rank}(X_k) \leq \operatorname{rank}(X_0) + k \cdot \operatorname{rank}(Q),$$

for any k. □

Chapter 4

Differential Riccati Equations

Differential Riccati equations are matrix-valued nonlinear differential equations with applications in various domains of engineering and science, such as optimal control [29], model reduction of linear time-varying (LTV) systems [35], damping optimization in mechanical systems [12], control of shear flows [24], and the numerical solution of stochastic differential equations [41]. In the previous chapters, we considered only algebraic Riccati equations, but we now turn to their differential counterpart, i.e., the case where the solution—and possibly also the coefficients—depends on time. In particular, in this chapter we provide the definition of the differential Riccati equation, along with various discretization methods for computing its numerical solution [27, 15]. At the end of the chapter, we present two classical yet important applications in engineering that motivate our interest in this topic: the Linear Quadratic Regulator problem and the Tracking problem [29].

Definition 4.0.1. A *Differential Riccati Equation* (DRE) is a differential equation of the form

$$\begin{cases} \dot{X} = Q + A^{\mathsf{T}}X + XA - XSX, & t \in [t_0, t_{\text{end}}], \\ X(t_0) = X_0. \end{cases} \tag{4.1}$$

where the solution X as well as the coefficient matrices A, Q, and S may depend on time. The equation is called *autonomous* if the coefficients are constant in time, and *non-autonomous* otherwise.

In the following, we consider several common yet highly effective approaches for solving differential Riccati equations (DREs). The main idea behind these methods is to apply an appropriate discretization to the differential equation, thereby reducing it to a sequence of

algebraic Riccati equations. In particular, we begin with the Midpoint and Trapezoidal rules, and then proceed to introduce Backward Differentiation Formulas [27] and Rosenbrock schemes [15].

4.1 Discretization schemes

4.1.1 Midpoint rule

The Midpoint rule is an implicit scheme derived by applying a quadrature formula to the integral form of equation (4.1). Given the problem (4.1), we first discretize the domain $[t_0, t_{\text{end}}]$ into the points $\{t_0, t_1, \ldots, t_n\}$. For each step k, let $\tau_k := t_{k+1} - t_k$ denote the time-step size. The Midpoint method applied to (4.1) [27] yields

$$X_{k+1} = X_k + \tau_k \mathcal{R}\left(t_k + \frac{\tau_k}{2}, \frac{1}{2}(X_k + X_{k+1})\right),$$

where $\mathcal{R}(t, X) := Q + A^{\mathsf{T}}X + XA - XSX$ and $X_{k+1} \approx X(t_{k+1})$. This scheme leads to the algebraic Riccati equation for X_{k+1}

$$\begin{aligned}\left[\tau_k Q_{k'} + X_k + \frac{\tau_k}{2}\left(A_{k'}^{\mathsf{T}} X_k + X_k A_{k'} - \frac{X_k S_{k'} X_k}{2}\right)\right] \\ + \left(\frac{\tau_k}{2}A_{k'} - \frac{\tau_k}{4}S_{k'}X_k - \frac{1}{2}I_n\right)^{\mathsf{H}} X_{k+1} + X_{k+1}\left(\frac{\tau_k}{2}A_{k'} - \frac{\tau_k}{4}S_{k'}X_k - \frac{1}{2}I_n\right) \\ - X_{k+1}\left(\frac{\tau_k}{4}S_{k'}\right)X_{k+1} = 0,\end{aligned}$$

where $A_{k'} \equiv A\left(t_k + \frac{\tau_k}{2}\right)$, $Q_{k'} \equiv Q\left(t_k + \frac{\tau_k}{2}\right)$ and $S_{k'} \equiv S\left(t_k + \frac{\tau_k}{2}\right)$.

4.1.2 Trapezoidal rule

Following the same philosophy as the Midpoint rule, and applying a different quadrature formula, one can derive the Trapezoidal scheme [27]

$$X_{k+1} = X_k + \frac{\tau_k}{2}\left(\mathcal{R}(t_k, X_k) + \mathcal{R}(t_{k+1}, X_{k+1})\right).$$

Re-arranging the terms in the equation, we end up with the ARE

$$\left[\frac{\tau_k}{2}Q_{k+1} + X_k + \frac{\tau_k}{2}\left(Q_k + A_k^{\mathsf{T}}X_k + X_kA_k - X_kS_kX_k\right)\right] + \left(\frac{\tau_k}{2}A_{k+1} - \frac{1}{2}I_n\right)^{\mathsf{T}} X_{k+1} + X_{k+1}\left(\frac{\tau_k}{2}A_{k+1} - \frac{1}{2}I_n\right) - X_{k+1}\left(\frac{\tau_k}{2}S_{k+1}\right)X_{k+1} = 0,$$

where $Q_k \equiv Q(t_k),\ A_k \equiv A(t_k),\ S_k \equiv S(t_k)$.

4.1.3 Backward Differentiation Formulas

The *Backward Differentiation Formulas* (BDFs) [27] are a family of implicit methods used to numerically solve ordinary differential equations. The idea behind these methods is to express the solution at a given time in terms of the already-computed solutions from previous time steps.

The general BDF of order p allows us to discretize (4.1) as

$$X_{k+1} = \sum_{j=1}^{p} -\alpha_j X_{k+1-j} + \tau_k\beta\mathcal{R}(t_{k+1}, X_{k+1}).$$

The expressions α_j, β denote the determining coefficients for the p-step BDF formula given in Table 4.2 (see [2]).

p	β	α_1	α_2	α_3	α_4	α_5	α_6
1	1	-1					
2	$\frac{2}{3}$	$-\frac{4}{3}$	$\frac{1}{3}$				
3	$\frac{6}{11}$	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$			
4	$\frac{12}{25}$	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$		
5	$\frac{60}{137}$	$-\frac{300}{137}$	$\frac{300}{137}$	$-\frac{200}{137}$	$\frac{75}{137}$	$-\frac{12}{137}$	
6	$\frac{60}{147}$	$-\frac{360}{147}$	$\frac{450}{147}$	$-\frac{400}{147}$	$\frac{225}{147}$	$-\frac{72}{147}$	$\frac{10}{147}$

(4.2)

This discretization leads to the algebraic Riccati equation

$$\left(\tau_k\beta Q_{k+1} - \sum_{j=1}^{p}\alpha_j X_{k+1-j}\right) + \left(\tau_k\beta A_{k+1} - \frac{1}{2}I_n\right)^{\mathsf{T}} X_{k+1} + X_{k+1}\left(\tau_k\beta A_{k+1} - \frac{1}{2}I_n\right) - X_{k+1}(\tau_k\beta S_{k+1})X_{k+1} = 0, \quad (4.3)$$

where $Q_{k+1} \equiv Q(t_{k+1}),\ A_{k+1} \equiv A(t_{k+1}),\ S_{k+1} \equiv S(t_{k+1})$ are the coefficient matrices and X_{k+1} is the unknown. For large-scale applications the data are usually given in the low-rank form

$$Q_k = C_k^{\mathsf{T}} C_k, \qquad C_k \in \mathbb{R}^{q\times n}, \tag{4.4a}$$

$$S_k = B_k B_k^{\mathsf{T}}, \qquad B_k \in \mathbb{R}^{n\times m}, \tag{4.4b}$$

so that also the solution appears to be of low numerical rank. If we consider the decomposition of the solution $X_k = Z_k Z_k^{\mathsf{H}}$ described in the second chapter, the BDF discretization of (4.1) in the large-scale context becomes

$$\hat{C}_{k+1}^{\mathsf{H}}\hat{C}_{k+1} + \hat{A}_{k+1}^{\mathsf{T}} Z_{k+1} Z_{k+1}^{\mathsf{H}} + Z_{k+1} Z_{k+1}^{\mathsf{H}} \hat{A}_{k+1} - Z_{k+1} Z_{k+1}^{\mathsf{H}} \hat{B}_{k+1}\hat{B}_{k+1}^{\mathsf{T}} Z_{k+1} Z_{k+1}^{\mathsf{H}} = 0, \tag{4.5}$$

with

$$\begin{aligned}
\hat{A}_{k+1} &= \tau_k \beta A_{k+1} - \frac{1}{2} I_n, \\
\hat{B}_{k+1} &= \sqrt{\tau_k \beta}\, B_{k+1}, \\
\hat{C}_{k+1}^{\mathsf{H}} &= \left[\sqrt{\tau_k \beta}\, C_{k+1}^{\mathsf{T}},\ \sqrt{-\alpha_1}\, Z_k,\ \ldots,\ \sqrt{-\alpha_p}\, Z_{k+1-p} \right].
\end{aligned}$$

At this point, the main idea is to solve the algebraic Riccati equation (4.5). If we linearize this ARE by applying Newton's method, we obtain the Lyapunov equation

$$\tilde{A}_{k+1}^{(l)\,*} X_{k+1}^{(l)} + X_{k+1}^{(l)} \tilde{A}_{k+1}^{(l)} = -G_{k+1}^{(l)} G_{k+1}^{(l)\,*},$$

with $\tilde{A}_{k+1}^{(l)} = \hat{A}_{k+1} - \tau_k \beta B_{k+1} B_{k+1}^{\mathsf{T}} X_{k+1}^{(l-1)}$ and $G_{k+1}^{(l)} = \left[\hat{C}_{k+1}^{\mathsf{H}},\ \sqrt{\tau_k \beta} X_{k+1}^{(l-1)} B_{k+1} \right]$ for $X_{k+1}^{(l)}$ at the l-th Newton step. Let us observe that for BDF schemes of order $p \geq 2$, some of the coefficients $\alpha_j,\ j = 1, \ldots, p$, are positive, so that the scalars $\sqrt{-\alpha_j}$ in the definition of $\hat{C}_{k+1}$ are complex. This leads to a Lyapunov equation with a complex right-hand side factor G, which makes complex storage unavoidable. However, by using the LDL^{T} factorization $X_{k+1} = L_{k+1} D_{k+1} L_{k+1}^{\mathsf{T}}$ for the solution of the DRE, we can avoid complex data and arithmetic. The reason is that the coefficients $\alpha_j,\ j = 1, \ldots, p$, appear in the diagonal block S of the right-hand side $-GSG^{\mathsf{T}}$, so that we do not need to take the square root of the non-positive coefficients. The LDL^{T}-based implementation for solving (4.1) with BDF schemes is provided in Algorithm 16.

Remark 4.1.1 (Initialization)**.** Observe that to start a BDF method of order $p > 1$, the initial values $X_0, \ldots, X_{p-1}$ are required with sufficient accuracy to obtain the desired order of convergence. If they are not directly available, as usually happens, we can compute them

Algorithm 16 LDL^{T}-factored BDF method of order p to solve (4.1)

function $[L, D, t] = \texttt{bdf\_ldlt}(A(t),\ B(t),\ C(t),\ a,\ b,\ k_{\max},\ \texttt{tol}_{\text{ADI}},\ \tau)$

% Inputs: $A(t),\ B(t),\ C(t)$ coefficient matrix functions, $t \in [a, b]$, maximum number of Newton's iterations, $k_{\max}$, ADI tolerance $\texttt{tol}_{\text{ADI}}$, step size τ

% Output: For each time step $t = t_k$, (L_k, D_k, t_k) such that $X_k \approx L_k D_k L_k^{\mathsf{T}}$

$t_0 = a$, $m = \texttt{size}(B, 2)$, $q = \texttt{size}(C, 1)$

for $k = 0$ to $\frac{b-a}{\tau}$ **do**

  $t_{k+1} = t_k + \tau$

  $\hat{A}_{k+1} = \tau\beta A_{k+1} - \frac{1}{2} I_n$

  $\hat{C}_{k+1}^{\mathsf{T}} = \left[C_{k+1}^{\mathsf{T}},\ L_k,\ \ldots,\ L_{k+1-p} \right]$

  for $l = 1$ to $k_{\max}$ **do**

    $\tilde{A}^{(l)} = \hat{A}_{k+1} - \tau_k \beta B_{k+1} \left(B_{k+1}^{\mathsf{T}} L_{k+1}^{(l-1)} \right) D_{k+1}^{(l-1)} L_{k+1}^{(l-1)\ T}$

    $G^{(l)} = \left[\hat{C}_{k+1}^{\mathsf{T}},\ K^{(l-1)} \right]$

    $S^{(l)} = \begin{bmatrix} \tau\beta I_q & & & & \\ & -\alpha_1 D_k & & & \\ & & \ddots & & \\ & & & -\alpha_p D_{k+1-p} & \\ & & & & \tau\beta I_m \end{bmatrix}$

    Compute the shift parameters $\{\mu_k\}$

    Solve $\hat{A}^{(l)\ T} X^{(l)} + X^{(l)} \hat{A}^{(l)} = -G^{(l)} S^{(l)} G^{(l)\ T}$ using Algorithm 8:

      $[L^{(l)}, D^{(l)}] = \texttt{ldlt\_adi}(\mu_1, \ldots, \mu_k, \hat{A}^{(l)}, G^{(l)}, S^{(l)}, \texttt{tol}_{\text{ADI}})$

    $K^{(l)} = L^{(l)} \left(D^{(l)} \left(L^{(l)\ T}\ B_{k+1} \right) \right)$

  end for

  $L_{k+1} = L^{(l_{\max})}, \quad D_{k+1} = D^{(l_{\max})}$

end for

return $L = \{L_k\},\ D = \{D_k\},\ t = \{t_k\}$

using the method of order $p-1$ with sufficiently small time steps. This process is then applied recursively to the order $p-1$ method in order to generate the initial values required for its start. Therefore, starting with X_0 and applying lower order methods, we can compute all the initial guesses we need to apply order p method. See Algorithm 2 in [7] for the implementation of these extra time steps.

4.1.4 Rosenbrock Methods

Rosenbrock methods [31] form a class of implicit schemes that approximate the solution of differential equations through the solution of a sequence of linear systems. Their application to differential Riccati equations is presented in [15, 27]. For order p, we have to solve p linear systems. The general p-stage Rosenbrock method applied to (4.1) yields

$$\left(\frac{1}{\tau_k\gamma_{ii}}I_n - \frac{\partial\mathcal{R}}{\partial X}(t_k, X_k)\right)K_i = \mathcal{R}\left(t_{k,i}, X_k + \sum_{j=1}^{i-1} a_{i,j}K_j\right) + \sum_{j=1}^{i-1}\frac{c_{i,j}}{\tau_k}K_j + \gamma_i\tau_k\mathcal{R}_{t_k}, \tag{4.6}$$

$$X_{k+1} = X_k + \sum_{j=1}^{p} m_j K_j, \tag{4.7}$$

where $t_{k,i} = t_k + \alpha_i\tau_k,\ i = 1,\ldots,p$ and $\gamma_{i,i}, a_{i,j},\ c_{i,j},\ \gamma_i,\ m_j$ and α_i are the method coefficients, see [22], $\mathcal{R}_{t_k} = \frac{\partial\mathcal{R}}{\partial t}(t_k, X(t_k))$ and $\frac{\partial\mathcal{R}}{\partial X}(t_k, X_k)$ is the Fréchet derivative

$$\frac{\partial\mathcal{R}}{\partial X}(t_k, X_k) : \mathbb{R}^{n\times n} \ni U \to (A_k - S_kX_k)^{\mathsf{H}}U + U(A_k - S_kX_k),$$

of $\mathcal{R}$ at X_k. Therefore, we can reformulate (4.6) as

$$\hat{A}_k^{\mathsf{H}}K_i + K_i\hat{A}_k = -\mathcal{R}\left(t_{k,i}, X_k + \sum_{j=1}^{i-1} a_{i,j}K_j\right) - \sum_{j=1}^{i-1}\frac{c_{i,j}}{\tau_k}K_j - \gamma_i\tau_k\mathcal{R}_{t_k},$$

$$X_{k+1} = X_k + \sum_{j=1}^{p} m_j K_j,$$

with $\hat{A}_k := A_k - S_kX_k - \frac{1}{2\tau_k\gamma_{i,i}}I_n,\ i = 1,\ldots,p$. Let us focus now on the first and second order Rosenbrock schemes for an autonomous DRE [16]. Note that, since we are considering the autonomous case, $\mathcal{R}_{t_k} = 0$. From (4.6), it follows that we can write the 1-stage Rosenbrock scheme in the ZZ^{H}-type representation as

$$\hat{A}_k^{\mathsf{H}}X_{k+1} + X_{k+1}\hat{A}_k = -G_kG_k^{\mathsf{H}}, \tag{4.8}$$

with $\gamma_{1,1} = 1$, $\hat{A}_k = A_k - S_k X_k - \frac{1}{2\tau_k} I_n$ and the right-hand side factor

$$G_k = \left[C_k^{\mathsf{T}}, Z_k Z_k^{\mathsf{H}} B_k, \sqrt{\frac{1}{\tau_k}} Z_k \right] \in \mathbb{R}^{n \times (q+m+z_k)}.$$

In the second-order Rosenbrock method we have to solve two Lyapunov equations, and the scheme can be reformulated as follows

$$\tilde{A}_k^{\mathsf{H}} K_1 + K_1 \tilde{A}_k = -\mathcal{R}(X_k), \tag{4.9a}$$

$$\tilde{A}_k^{\mathsf{H}} K_{21} + K_{21} \tilde{A}_k = -\tau_k^2 K_1 B_k B_k^{\mathsf{T}} K_1 - \left(2 - \frac{1}{\gamma}\right) K_1, \tag{4.9b}$$

$$K_2 = -K_{21} + \left(1 - \frac{1}{\gamma}\right) K_1, \tag{4.9c}$$

$$X_{k+1} = X_k + \frac{3}{2}\tau_k K_1 + \frac{1}{2}\tau_k K_2, \tag{4.9d}$$

with $\tilde{A}_k = \gamma\tau_k(A_k - S_k X_k) - \frac{1}{2} I_n$. Considering the low-rank representation of the data (4.4), we have that

$$-\mathcal{R}(X_k) = -C_k^{\mathsf{T}} C_k - A_k^{\mathsf{T}} Z_k Z_k^{\mathsf{H}} - Z_k Z_k^{\mathsf{H}} A_k + Z_k Z_k^{\mathsf{H}} B_k B_k^{\mathsf{T}} Z_k Z_k^{\mathsf{H}}. \tag{4.10}$$

In [16, 30] are described two possible splittings of (4.10) of the form $-G_k G_k^{\mathsf{H}}$. The first one consider the partitioning

$$G_k = \left[C_k^{\mathsf{T}},\ A_k^{\mathsf{T}} Z_k + Z_k,\ i Z_k Z_k^{\mathsf{H}} B_k,\ i A_k^{\mathsf{T}} Z_k,\ i Z_k \right] \in \mathbb{C}^{n \times (q+m+3z_k)},$$

but in this way we introduce complex arithmetic. The idea to avoid complex data is to use a superposition approach for the splitting (4.9a) into the two equations

$$\tilde{A}_k^{\mathsf{H}} \hat{K}_1 + \hat{K}_1 \tilde{A}_k = -N_k N_k^{\mathsf{H}}, \qquad \tilde{A}_k^{\mathsf{H}} \tilde{K}_1 + \tilde{K}_1 \tilde{A}_k = -U_k U_k^{\mathsf{H}},$$

such that $K_1 := \hat{K}_1 - \tilde{K}_1$ and $-G_k G_k^{\mathsf{H}} := -N_k N_k^{\mathsf{H}} + U_k U_k^{\mathsf{H}}$, with

$$N_k = \left[C_k^{\mathsf{T}},\ A_k^{\mathsf{T}} Z_k + Z_k \right] \in \mathbb{R}^{n \times (q+z_k)}, \qquad U_k = \left[Z_k Z_k^{\mathsf{H}} B_k,\ A_k^{\mathsf{T}} Z_k,\ Z_k \right] \in \mathbb{R}^{n \times (m+2z_k)}.$$

Even if we avoid complex arithmetic, this second splitting is not better than the first one, because experiments have shown that the solution $K_1 = \hat{K}_1 - \tilde{K}_1$ is affected by numerical inaccuracies, and therefore the scheme is not stable. For completeness, the classical low-rank representation $-G_k G_k^{\mathsf{H}}$ of the right-hand side of (4.9b) is done by setting

$$G_k = \left[\tau_k T_1 T_1^{\mathsf{T}} B_k,\ \sqrt{2 - \frac{1}{\gamma}} T_1 \right] \in \mathbb{R}^{n \times (m+t_k)},$$

where $K_1 = T_1 T_1^{\mathsf{T}} \in \mathbb{R}^{n \times t_k}$. Let us consider now the LDL^{T}-type factorization to tackle these issues [27]. In this case, the factors of the right-hand side $-\tilde{G}_k \tilde{S}_k \tilde{G}_k^{\mathsf{H}}$ in (4.8) can be written as

$$\tilde{G}_k = \begin{bmatrix} C_k^{\mathsf{T}}, & L_k \end{bmatrix} \in \mathbb{R}^{n \times (q+l_k)},$$
$$\tilde{S}_k = \begin{bmatrix} I_q & \\ & D_k L_k^{\mathsf{T}} B_k B_k^{\mathsf{T}} L_k D_k + \frac{1}{\tau_k} D_k \end{bmatrix} \in \mathbb{R}^{(q+l_k) \times (q+l_k)},$$

where $X_k = L_k D_k L_k^{\mathsf{T}}$ is the approximate solution. Observe that the number of columns of G_k and $\tilde{G}_k$ equals the number of linear systems that has to be solved when constructing the factors Z_k and L_k for the solution of the Lyapunov equation (4.8). Therefore, with the ZZ^{H} approach we have to solve

$$(q + m + z_k) - (q + l_k) = m + z_k - l_k$$

linear systems more than in the LDL^{T} approach. Actually, since it can be shown that $l_z \leq z_k$, with the latter method we can save at least m system solves in every step. Therefore, if we assume that n_{lyap} is the constant number of Lyapunov solver steps per time step and n_{ODE} is the number of time steps, we can avoid the computation of $m \cdot n_{\text{lyap}} \cdot n_{\text{ODE}}$ during the solution of the DRE. For the second-order, we split the right-hand side in (4.9a)

$$-C_k^{\mathsf{T}} C_k - A_k^{\mathsf{T}} L_k D_k L_k^{\mathsf{T}} - L_k D_k L_k^{\mathsf{T}} A_k + L_k D_k L_k^{\mathsf{T}} B_k B_k^{\mathsf{T}} L_k D_k L_k^{\mathsf{T}}$$

as $-\tilde{G}_k \tilde{S}_k \tilde{G}_k^{\mathsf{T}}$ with

$$\tilde{G}_k = \begin{bmatrix} C_k^{\mathsf{T}}, & A_k^{\mathsf{T}} L_k, & L_k \end{bmatrix} \in \mathbb{R}^{n \times (q+2l_k)},$$
$$\tilde{S}_k = \begin{bmatrix} I_q & & \\ & & D_k \\ & D_k & -D_k L_k^{\mathsf{T}} B_k B_k^{\mathsf{T}} L_k D_k \end{bmatrix} \in \mathbb{R}^{(q+2l_k) \times (q+2l_k)}.$$

Observe that for equation (4.9a), we can save

$$(q + m + 3z_k) - (q + 2l_k) = m + 3z_k - 2l_k \geq m + z_k,$$

linear system solves using the LDL^{T}-type factorization instead of the ZZ^{H} one. Assume now $K_1 = \tilde{T}_1 D_1 \tilde{T}_1^{\mathsf{T}}$. The right-hand side factors $\tilde{G}_k$ and $\tilde{S}_k$ of (4.9b) can be written as

$$\tilde{G}_k = \tilde{T}_1 \in \mathbb{R}^{n \times \tilde{t}_k},$$
$$\tilde{S}_k = \tau_k^2 \tilde{D}_1 \tilde{T}_1^{\mathsf{T}} B_k B_k^{\mathsf{T}} \tilde{T}_1 \tilde{D}_1 + \left(2 - \frac{1}{\gamma}\right) \tilde{D}_1 \in \mathbb{R}^{\tilde{t}_k \times \tilde{t}_k}.$$

Note that t_k and $\tilde{t}_k$ satisfy $\tilde{t}_k \leq t_k$. Therefore, from (4.9b), we can avoid the computation of

$$(m + t_k) - \tilde{t}_k \geq m,$$

linear systems. Thus, with the LDL^T low-rank representation, we can save at least $(m + z_k) + m = 2m + z_k$ linear system solves in each step of the Lyapunov solver. For the implementation of first and second order of the Rosenbrock scheme see [15].

4.2 Applications

In the following, we describe and solve two classical problems that usually arise in control theory: the linear quadratic regulator problem and the tracking problem [29].

4.2.1 Linear Quadratic Regulator

The *Linear Quadratic Regulator* (LQR) problem is a fundamental optimal control problem in systems and control theory that seeks to find the best control input for a linear system to minimize a quadratic cost function. As we will see, this problem can be formulated in both the infinite-horizon and finite-horizon cases, and depending on the formulation, its solution will be given by solving an ARE or a DRE, respectively.

Infinite-horizon formulation

Let us consider the continuous-time linear dynamical system in state-space form

$$\dot{x}(t) = Ax(t) + Bu(t),\ x(0) = x_0, \tag{4.11}$$

with $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$, where $x(t) \in \mathbb{R}^n$ is the state vector and $u(t) \in \mathbb{R}^m$ is the control vector.
The goal is to find the *optimal control* $u(t)$ that minimizes the cost function

$$J = \int_0^{+\infty} \left(x^\mathsf{T} Q x + u^\mathsf{T} R u \right) \mathrm{d}t,$$

where $Q \succeq 0$ is the state penalty matrix and $R \succ 0$ is the control penalty matrix. The classic procedure is to optimize the cost-to-go function $V(x)$, which satisfies the Hamilton-Jacobi-Bellman equation:

$$\min_{u\in\mathbb{R}^m} \left[x^\mathsf{T} Q x + u^\mathsf{T} R u + \frac{\partial V}{\partial x} \cdot \dot{x} \right] = 0,\ \text{for any } x \in \mathbb{R}^n.$$

It is well known that for the infinite-horizon problem the optimal cost-to-go function is quadratic and does not depend on time. For simplicity, we choose

$$V(x) = x^\mathsf{T} S x,\ S \succeq 0.$$

Thus $\frac{\partial V}{\partial x} = 2x^\mathsf{T} S$. By construction, the terms inside the min are quadratic and convex (because $R \succ 0$), so we can take the minimum explicitly by finding the solution where the gradient of those terms vanishes:

$$\frac{\partial \left[x^\mathsf{T} Q x + u^\mathsf{T} R u + \frac{\partial V}{\partial x} \cdot \dot{x}\right]}{\partial u} = 2u^\mathsf{T} R + 2x^\mathsf{T} S B = 0. \tag{4.12}$$

This yields to

$$u^\mathsf{T} = -x^\mathsf{T} S B R^{-1},$$

and, thus, to the optimal control law

$$u = -R^{-1} B S x = -K x.$$

The matrix K is generally called *feedback gain matrix*. By inserting this quantity into (4.12) and simplifying, one obtains

$$x^\mathsf{T} \left(Q + SA + A^\mathsf{T} S - S B R^{-1} B^\mathsf{T} S\right) x = 0,$$

and since this must hold for all x, then the matrix inside the parentheses must be zero, i.e.

$$Q + SA + A^\mathsf{T} S - S B R^{-1} B^\mathsf{T} S = 0.$$

Finite-horizon formulation

Let us consider a linear time-invariant system governed by a state-space equation of the form

$$\dot{x} = Ax(t) + Bu(t),\ x(t_0) = x_0,$$

with A, B, x, u as before and let

$$J = \int_{t_0}^{t_\text{end}} (x^\mathsf{T} Q x + u^\mathsf{T} R u)\mathrm{d}t + x^\mathsf{T}(t_\text{end}) Q_\text{end} x(t_\text{end}),$$

the finite-horizon cost function, with $Q_\text{end} \succeq 0$, $Q \succeq 0$ and $R \succ 0$. To proceed with the formulation, we also need to investigate a particular form for the cost-to-go function $V(x, t)$, that now is time-dependent.

We will consider a solution of the form $V(x,t) = x^\mathsf{T} S(t) x$, with $S(t) \succ 0$, and the HJB equation in this case has the additional term $\frac{\partial V}{\partial t}$, so that we have to solve

$$\min_{u \in \mathbb{R}^m} \left[x^\mathsf{T} Q x + u^\mathsf{T} R u + \frac{\partial V}{\partial x} \dot{x} + \frac{\partial V}{\partial t} \right] = 0, \text{ for any } x \in \mathbb{R}^n, \text{ for any } t \in [t_0, t_{\text{end}}]. \tag{4.13}$$

As before, we can find the minimum by settling the gradient to zero:

$$\frac{\partial \left[x^\mathsf{T} Q x + u^\mathsf{T} R u + \frac{\partial V}{\partial x} \dot{x} + \frac{\partial V}{\partial t} \right]}{\partial u} = 2 u^\mathsf{T} R + \frac{\partial V}{\partial x} \cdot B = 0.$$

This yields to

$$u^\mathsf{T} = -\frac{1}{2} \frac{\partial V}{\partial x} B R^{-1},$$

and, in particular, to the optimal control

$$u = -\frac{1}{2} R^{-1} B^\mathsf{T} \left(\frac{\partial V}{\partial x} \right)^\mathsf{T}.$$

Recalling that $V = x^\mathsf{T} S(t) x$, we have that $\frac{\partial V}{\partial x} = 2 x^\mathsf{T} S(t)$ and $\frac{\partial V}{\partial t} = x^\mathsf{T} \dot{S}(t) x$. Thus $u = -R^{-1} B^\mathsf{T} S(t) x = -Kx$ and by substituting this quantity into (4.13), we obtain

$$x^\mathsf{T} \Big(Q - S(t) B R^{-1} B^\mathsf{T} S(t) + S(t) A + A^\mathsf{T} S(t) + \dot{S}(t) \Big) x = 0.$$

Therefore, $S(t)$ must satisfy the continuous-time differential Riccati equation

$$-\dot{S}(t) = Q + S(t) A + A^\mathsf{T} S(t) - S(t) B R^{-1} B^\mathsf{T} S(t), \tag{4.14}$$

subject to the terminal condition

$$S(t_{\text{end}}) = Q_{\text{end}}.$$

Note that the solution of the previous example is exactly the steady-state solution of this equation, defined by $\dot{S}(t) = 0$.

Remark 4.2.1. Let us highlight that (4.14) is not subject to an initial condition, but rather to the terminal condition $S(t_{\text{end}}) = Q_{\text{end}}$. This suggests that the appropriate approach to solve the problem, in order to exploit the knowledge of the terminal condition, is to integrate the equation backward in time. To this end, we perform the change of variable $t \mapsto t_0 + t_{\text{end}} - t$ [7] to reverse time, and define $\hat{S}(\tau) := S(t_0 + t_{\text{end}} - t)$, where $\tau = t_0 + t_{\text{end}} - t$. The corresponding problem to be solved is then given by

$$\dot{\hat{S}}(\tau) = Q + \hat{S}(\tau) A + A^\mathsf{T} \hat{S}(\tau) - \hat{S}(\tau) B R^{-1} B^\mathsf{T} \hat{S}(\tau), \qquad \hat{S}(\tau_0) = Q_{\text{end}},$$

where $\tau_0 = t_{\text{end}}$. Finally, the original solution $S(t)$ is recovered by reversing time again, i.e. $S(t) = \hat{S}(t_0 + t_{\text{end}} - t)$. See Algorithm 1 in [6] for an implementation of BDF methods backward in time.

4.2.2 The Tracking problem

The *Linear Quadratic Tracking* (LQT) problem [29] is a classical optimal control problem where the goal is to make a system follow a desired reference trajectory as closely as possible over time, while possibly minimizing control effort.
Let us consider the linear time-invariant (LTI) system

$$\dot{x}(t) = Ax(t) + Bu(t),\ x(t_0) = x_0, \tag{4.15}$$

and let $u(t)$ be the control input we want to design such that $x(t) \approx x_{\text{ref}}(t)$ and $u(t) \approx u_{\text{ref}}(t)$ over $t \in [t_0, t_{\text{end}}]$, where $x_{\text{ref}}(t), u_{\text{ref}}(t)$ is the nominal trajectory. Let

$$J = \int_{t_0}^{t_{\text{end}}} \Big[(x(t) - x_{\text{ref}}(t))^\mathsf{T} Q (x(t) - x_{\text{ref}}(t)) + (u(t) - u_{\text{ref}}(t))^\mathsf{T} R (u(t) - u_{\text{ref}}(t))\Big]\, \mathrm{d}t \\ + (x(t_{\text{end}}) - x_{\text{ref}}(t_{\text{end}}))^\mathsf{T} Q_{\text{end}} (x(t_{\text{end}}) - x_{\text{ref}}(t_{\text{end}})), \tag{4.16}$$

be the cost index we want to minimize, where $Q \succeq 0$ penalizes the tracking error, $R \succ 0$ penalizes the control effort and $Q_{\text{end}} \succeq 0$. The strategy to solve this problem is to reformulate (4.15) and (4.16) in terms of the state error $e(t)$ and to reduce the original problem to an LQR problem.
Let $e(t) := x(t) - x_{\text{ref}}(t)$ be the state error and $\overline{u}(t) := u(t) - u_{\text{ref}}(t)$. Then,

$$\dot{e}(t) \;=\; \dot{x}(t) \;-\; \dot{x}_{\text{ref}}(t) \;=\; Ax(t) \;+\; Bu(t) \;-\; \dot{x}_{\text{ref}}(t) \;=\; Ae(t) \;+\; B\overline{u}(t) \;+\; d(t),$$

where $d(t) = Ax_{\text{ref}}(t) + Bu_{\text{ref}}(t) - \dot{x}_{\text{ref}}(t)$. Moreover, we can rewrite J as

$$J = \int_{t_0}^{t_{\text{end}}} \Big[e(t)^\mathsf{T} Q e(t) + \overline{u}(t)^\mathsf{T} R \overline{u}(t)\Big]\, \mathrm{d}t + e(t_{\text{end}})^\mathsf{T} Q_{\text{ref}} e(t_{\text{end}}).$$

This is now a standard LQR problem with an affine disturbance $d(t)$, which is known.
Let's consider a cost-to-go function of the form

$$V(t, x) = x^\mathsf{T} S(t) x + 2x^\mathsf{T} s(t) + s_0(t),$$

with $S \succ 0$. In this case, we have

$$\frac{\partial V}{\partial x} = 2x^\mathsf{T} S(t) + 2s^\mathsf{T}(t), \qquad \frac{\partial V}{\partial t} = x^\mathsf{T} \dot{S}(t) x + 2x^\mathsf{T} \dot{s}(t) + \dot{s}_0(t),$$

and using the HJB (4.13) we obtain

$$\min_u \left[e(t)^\mathsf{T} Q e(t) + \overline{u}(t)^\mathsf{T} R \overline{u}(t) + \frac{\partial V}{\partial x}(Ax + Bu) + \frac{\partial V}{\partial t}\right] = 0.$$

To find the optimal controller, we set

$$\frac{\partial \left[e(t)^{\mathsf{T}} Q e(t) + \overline{u}(t)^{\mathsf{T}} R \overline{u}(t) + \frac{\partial V}{\partial x}(Ax + Bu) + \frac{\partial V}{\partial t}\right]}{\partial u} = 0,$$

i.e.

$$\begin{aligned}
\frac{\partial \left[e(t)^{\mathsf{T}} Q e(t) + \overline{u}(t)^{\mathsf{T}} R \overline{u}(t) + \frac{\partial V}{\partial x}(Ax + Bu) + \frac{\partial V}{\partial t}\right]}{\partial u} &= 2\overline{u}(t)^{\mathsf{T}} R + \left(2x^{\mathsf{T}} S(t) + 2s^{\mathsf{T}}(t)\right) B = 0, \\
\overline{u}^{\mathsf{T}} &= -\left(x^{\mathsf{T}} S(t) + s^{\mathsf{T}}(t)\right) B R^{-1}, \\
\overline{u} &= -R^{-1} B^{\mathsf{T}} \left(S(t)x + s(t)\right).
\end{aligned}$$

Therefore, we can conclude that the optimal control is given by

$$u(t) = u_{\text{ref}}(t) - R^{-1} B^{\mathsf{T}} (S(t)x + s(t)),$$

where the functions S, s, s_0 satisfy

$$\begin{aligned}
-\dot{S}(t) &= Q - S(t) B R^{-1} B^{\mathsf{T}} S(t) + S(t)A + A^{\mathsf{T}} S(t), \\
-\dot{s}(t) &= -Q x_{\text{ref}}(t) + \left[A^{\mathsf{T}} - S B R^{-1} B^{\mathsf{T}}\right] s(t) + S(t) B u_{\text{ref}}(t), \\
-\dot{s}_0(t) &= x_{\text{ref}}(t)^{\mathsf{T}} Q x_{\text{ref}}(t) - s^{\mathsf{T}}(t) B R^{-1} B^{\mathsf{T}} s(t) + 2 s(t)^{\mathsf{T}} B u_{\text{ref}}(t),
\end{aligned}$$

with terminal conditions

$$\begin{aligned}
S(t_{\text{end}}) &= Q_{\text{end}}, \\
s(t_{\text{end}}) &= -Q_{\text{end}} x_{\text{ref}}(t_{\text{end}}), \\
s_0(t_{\text{end}}) &= x_{\text{ref}}^{\mathsf{T}}(t_{\text{end}}) Q_{\text{end}} x_{\text{ref}}(t_{\text{end}}).
\end{aligned}$$

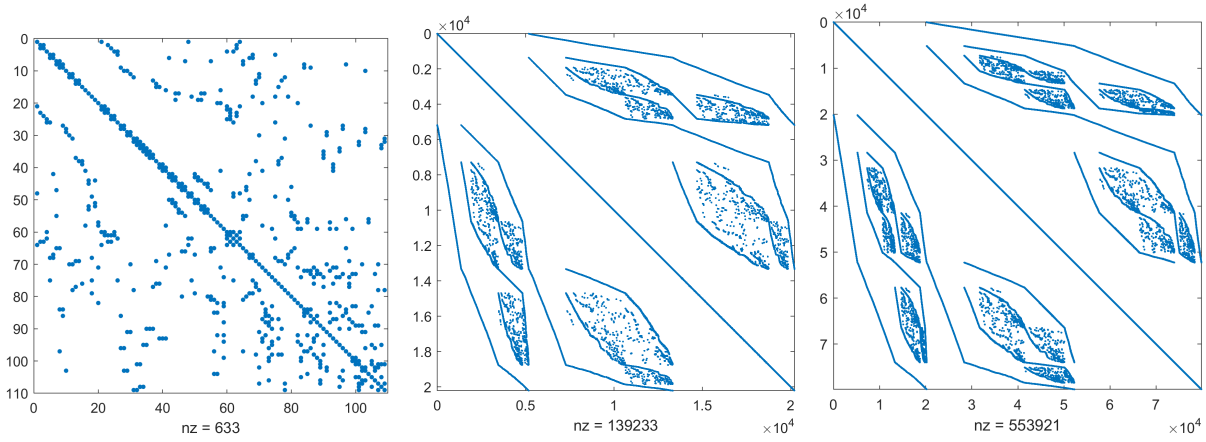
Chapter 5

Numerical Results

In this final chapter, we present various numerical experiments and results concerning the solution of differential Riccati equations (DREs). In particular, we employ the BDF method of orders 1, 2, and 3 to discretize a differential Riccati equatino, and consider three different schemes for solving the corresponding algebraic Riccati equations: the Newton–Kleinman method with a zero initial guess, the Newton–Kleinman method initialized with the solution from the previous time step, and the RADI method with a zero initial guess. Our focus is on solving large-scale DREs, as typically arises in real-world problems where classical solvers are not applicable. For this reason, we consider large, low-rank, and sparse coefficient matrices, such that the corresponding equations must be solved using the iterative methods described in Chapter 2.

We begin this chapter by introducing the problem on which our experiments are based: the rail profile cooling problem. In brief, this problem consists of determining the optimal cooling strategy, such as water spray intensities, locations, or timing, so that the rail's temperature distribution evolves as desired, while satisfying physical and technological constraints. As we will see, the problem is modeled by (5.6) [13, 36]. In Table 5.1, we report the number of nonzero elements in the coefficient matrices of the equation. It is evident that both matrices are sparse and that, as the problem dimension increases, the proportion of nonzero entries becomes tiny. Therefore, employing classical dense solvers for such problems is inefficient. Moreover, the advantage of using low-rank methods becomes particularly clear for large-scale problems, e.g. $n \geq 10^4$. For smaller dimensions, such as $n \in \{109,\ 1357\}$, `ode45` may require less computational time than our low-rank schemes. However, this is not a meaningful comparison, as the strength of low-rank approaches lies in their ability to handle large-scale problems efficiently.

When `ode45` is applied to a matrix differential equation, it operates on the vectorized form of the unknown matrix, effectively solving a system of dimension n^2. As a consequence, both the computational and memory requirements grow quadratically with n, quickly becoming prohibitive. In contrast, low-rank methods exploit the low-rank structure of the solution, allowing them to efficiently compute and store compact factors even for very large n, where classical dense solvers such as `ode45` become infeasible. In Figure 5.1, we plot the sparsity pattern of $\hat{A}$.



(a) $n = 109$ (b) $n = 20\ 209$ (c) $n = 79\ 841$

Figure 5.1: Sparsity pattern of $\hat{A}$. Nonzero values are colored while zero values are white.

Next, we show that for different orders of the BDF method, the convergence of the solvers follows the expected theoretical behavior, i.e., $\mathcal{O}(h^p)$ where h is the time-step size and p is the order of the discretization scheme. Finally, we present several tables comparing the different solvers in terms of runtime, average number of iterations, and average rank of the solution. We also provide graphs illustrating the evolution of the solution rank over time.

5.1 Cooling problem

In all of our experiments, we solve DREs that arise from optimal rail profile cooling problems. This problem arises in rolling mills because different stages of the production process require the raw material to be at specific temperatures. To achieve a high production rate while minimizing economic costs, it is desirable to decrease the temperature to the required level as quickly as possible before entering the next production phase. The cooling process is carried out

n	nnz $\hat{A}$	nnz $\hat{E}$	% nnz $\hat{A}$	% nnz $\hat{E}$
109	633	633	5.328	5.328
1 375	8 985	8 997	0.488	0.489
5 177	35 185	35 241	0.131	0.031
20 209	139 233	139 473	0.034	0.034
79 841	553 921	554 913	0.009	0.009

Table 5.1: Number of nonzero (nnz) elements in the coefficient matrices of (5.6).

by spraying cooling fluids onto the surface; however, it must be carefully controlled to ensure that material properties such as durability and porosity meet the required quality standards. Large temperature gradients within the rail profile can lead to unwanted deformations, loss of rigidity, and other undesirable material characteristics. Therefore, the actual goal is to achieve an even temperature distribution. The heat distribution is modeled by the following equation [13, 36]:

$$c\rho\frac{\partial x(t,\xi)}{\partial t} - \lambda\Delta_x x(t,\xi) = 0 \qquad \text{in } \mathbb{R}_{>0}\times\Omega, \tag{5.1}$$

$$\lambda\frac{\partial x(t,\xi)}{\partial \nu} = g_i \qquad \text{on } \mathbb{R}_{>0}\times\Gamma_i,\ \partial\Omega = \bigcup_i \Gamma_i, \tag{5.2}$$

$$x(0,\xi) = x_0(\xi) \qquad \text{in } \Omega, \tag{5.3}$$

where x is the temperature distribution, $c = 7620\ \frac{m^2}{s^2\,^\circ C}$ is the specific heat capacity, $\lambda = 26.4\ \frac{kg\ m}{s^3\,^\circ C}$ is the thermal conductivity and $\rho = 654\ \frac{kg}{m^3}$ is the density of the rail profile. The boundary $\partial\Omega$ is divided into several parts Γ_i, on which different boundary functions g_i are defined, allowing independent control over different surface regions. The vector ν denotes the outer normal of the boundary. In Figure 5.2, we provide a picture of the rail profile.

After spatial discretization of (5.1), we obtain the state-space system

$$E\dot{x} = \left(\frac{\lambda}{c\rho}S + \frac{\gamma}{c\rho}E_\Gamma\right)x + \frac{\gamma}{c\rho}Bu, \qquad \text{with } t>0, x(0)=x_0, \tag{5.4}$$

$$y = Cx, \tag{5.5}$$

where $\gamma = 7.0164\ \frac{kg}{s^3\,^\circ C}$ is the coefficient of thermal conductivity at the input boundary regions, and E_Γ is a matrix specific to this problem. At this point, one can formulate an LQR problem, as discussed at the end of the previous chapter, and the solution of the following experiments

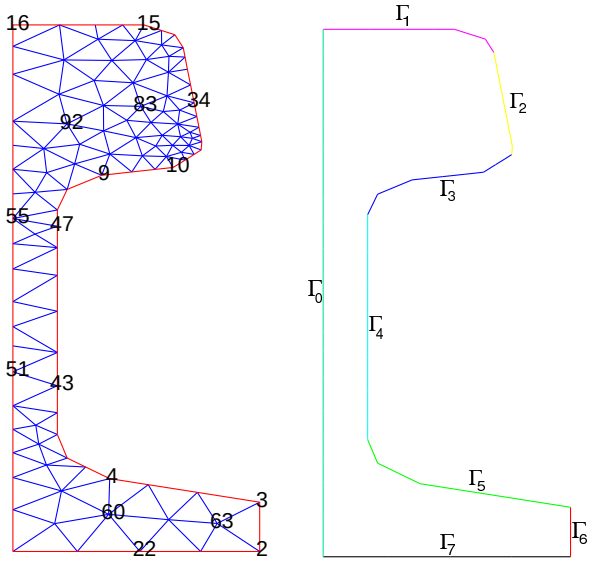


Figure 5.2: Initial mesh (left) and partitioning of the boundary (right)

corresponds to the solution of that control problem. In particular, we seek the matrix X that satisfies the generalized DRE

$$\frac{\mathrm{d}E^{\mathsf{T}}XE}{\mathrm{d}t} = -C^{\mathsf{T}}QC - E^{\mathsf{T}}X\hat{A} - \hat{A}XE + E^{\mathsf{T}}X\hat{B}R^{-1}XE, \tag{5.6}$$

$$X(t_{\mathrm{end}}) = X_{\mathrm{end}}, \tag{5.7}$$

where X_{end} is the terminal value of the solution, $\hat{A} = \frac{\lambda}{c\rho}S + \frac{\gamma}{c\rho}E_\gamma$ and $\hat{B} = \frac{\gamma}{c\rho}B$. For more details, see [17].

5.2 Experiments

All numerical experiments were conducted on the EIKE computational server (Ubuntu 22.04.5 LTS, AMD EPYC 9554 CPU with 128 cores, 2 TB RAM) using MATLAB R2025a. The coefficient matrices of the differential Riccati equation (5.6) for different problem dimensions are provided by the function `mess_get_linear_rail.m` from the `M.E.S.S.` Toolbox [39]. For more information, see `https://modelreduction.org/morwiki/FEniCS_Rail`. Moreover, to compute the shift parameters used in the algorithms, we employed the functions `mess_mnmx`, `mess_projection_shifts`, `mess_para`, and `mess_wachspress`, which are available in the same

toolbox. To perform the experiments, I built upon the existing functions in `M-M.E.S.S.-3.1`, such as `mess_bdf_dre` for solving DREs via BDF discretization methods. My extensions and modifications will be included in the next release of `M-M.E.S.S.-3.1`.

5.2.1 Order of Convergence

From the theory, it is known that BDF methods of order $p > 6$ are unstable, and that for any $p \in \{1, \dots, 6\}$, the global error of the corresponding BDF scheme satisfies

$$\max_{k \in \{0,\dots,n\}} ||X_k - X(t_k)|| = \mathcal{O}(h^p), \tag{5.8}$$

where X_k denotes the numerical solution at time t_k, and X is the exact solution. Let us now clarify how we can verify numerically that the BDF method used exhibits the expected order of convergence. From (5.8), it follows that

$$E(h) = Ch^p + o(h^p),$$

where $E(h) = \max_{k \in \{0,\dots,n\}} ||X_k - X(t_k)||$, $C > 0$ is a constant, and p is the order of the scheme. This implies that, for sufficiently small step sizes h,

$$E(h) \approx Ch^p,$$

so we expect $E(h)$ to behave like $f(h) = h^p$. Taking the natural logarithm of both $E(h)$ and $f(h)$, we obtain

$$\log(E(h)) \approx \log(C) + p\log(h), \qquad \log(f(h)) = p\log(h).$$

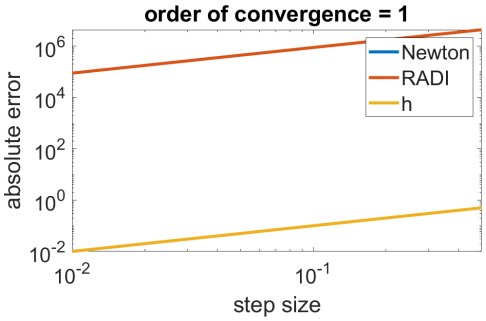
It follows that, in a $\log - \log$ plot, the two functions appear as straight lines with the same slope p; hence, they are parallel. Therefore, the idea is to compute $E(h)$ for small values of h, plot the corresponding $\log - \log$ graph, and check whether the curves are parallel. One main issue in this procedure is that computing the global error $E(h)$ requires knowledge of the exact solution X of the equation, which is generally not available. To address this problem, we vectorized the DRE (5.6) and solved the corresponding system of ordinary differential equations (ODEs) using MATLAB's built-in function `ode45`, which is based on an explicit Runge–Kutta (4,5) pair. In particular, we considered step sizes $h^{\text{BDF}} \in \{5 \cdot 10^{-1},\ 10^{-1},\ 5 \cdot 10^{-2},\ 10^{-2}\}$ for the BDF discretization, and $h^{\text{ode45}} = 10^{-5}$ for the `ode45` solver, in order to obtain a more accurate reference solution. The choice of these parameters is motivated by the observation that using

smaller values of h^{BDF} leads to a significant global error, causing the $E(h)$ curve to lose its linear behavior in the $\log - \log$ plot and thus distort the expcted convergence trend. This effect occurs because the solution computed by `ode45` is itself numerical, obtained by combining fourth- and fifth-order Runge–Kutta formulas, and is therefore also affected by discretization errors. Consequently, for small h or large n, such errors become noticeable and, especially for BDF orders $p \geq 3$, they significantly affect the quantity $||X_k^{\mathrm{BDF}} - X_k^{\mathrm{ode45}}||$, resulting in deviations from the expected slope in the $\log - \log$ plot. Therefore, the chosen range $h^{\mathrm{BDF}} \in \{5 \cdot 10^{-1},\ 10^{-1},\ 5 \cdot 10^{-2},\ 10^{-2}\}$ represents a good compromise that preserves the expected convergence behavior. We tested the BDF schemes for orders $p \in \{1, 2, 3\}$ and for system dimensions $n \in \{109, 1357\}$. Using larger values of n in these experiments becomes problematic because `ode45` is a dense solver, which can easily lead to memory issues. To perform these tests, we decided to use two methods as the inner ARE solver within the BDF scheme: the Newton–Kleinman method and RADI, both initialized with a zero initial guess at each iteration. In Figure 5.3, we consider coefficient matrices of dimension $n = 109$. We observe that Newton–Kleinman and RADI exhibit the same convergence behavior for different values of h; in fact, their corresponding lines overlap and cannot be distinguished. For $p = 1$ and $p = 2$, the line representing the theoretical behavior and those of the numerical methods have exactly the same slope. For $p = 3$, however, the lines are slightly no longer parallel for sufficiently small h. This deviation is due to the discretization errors affecting both X^{BDF} and X^{ode45}, as discussed previously.

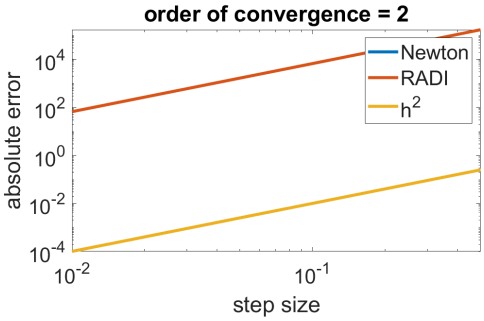
A similar effect is observed in Figure 5.4, but it is amplified due to the larger dimension $n = 1357$. It should be noted that the fact that the theoretical line does not have exactly the same slope as the others does not imply that the BDF method fails to achieve its expected order. Rather, it reflects that X^{ode45} is itself a numerical solution obtained via a discretization method, whose accuracy is insufficient compared to that of Newton–Kleinman or RADI to provide a fully reliable reference for assessing convergence. Even in this case, the lines corresponding to Newton–Kleinman and RADI still overlap.

5.2.2 Large-Scale Comparison

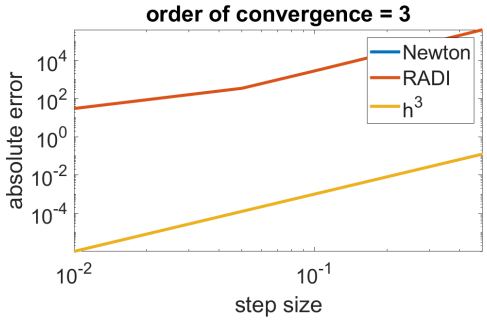
In the previous subsection, we verified that the convergence of the BDF scheme for orders 1, 2, and 3 follows the expected theoretical behavior, using both the Newton–Kleinman and RADI methods as inner solvers. In the following, we compare different approaches for solving (5.6) in terms of runtime, average number of iterations, and average solution rank. The inner solvers considered are: Newton–Kleinman and RADI with a zero initial guess, and New-



(a) $p = 1$

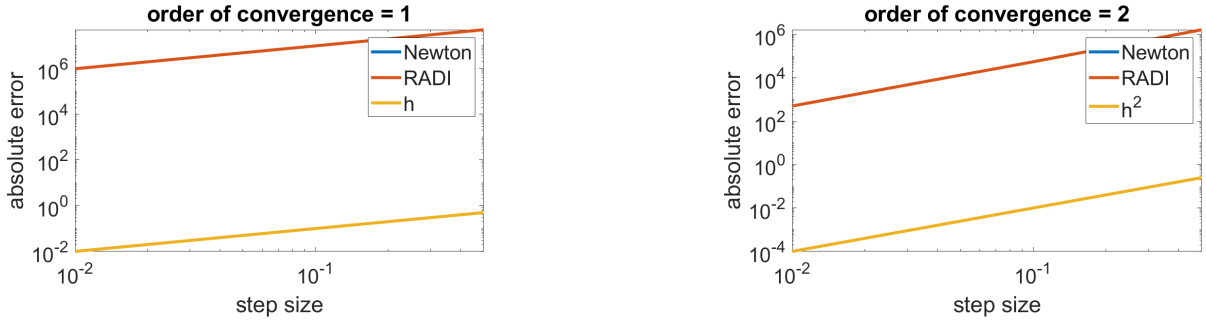


(b) $p = 2$



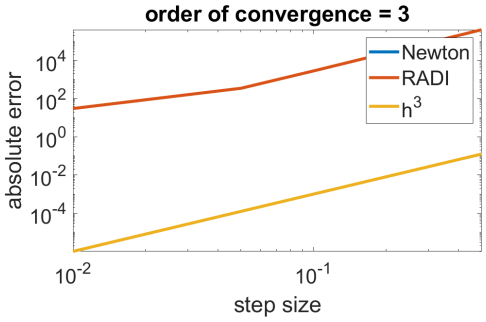
(c) $p = 3$

Figure 5.3: Convergence behavior using as inner solvers Newton-Kleinman and RADI for $n = 109$.



(a) $p = 1$

(b) $p = 2$



(c) $p = 3$

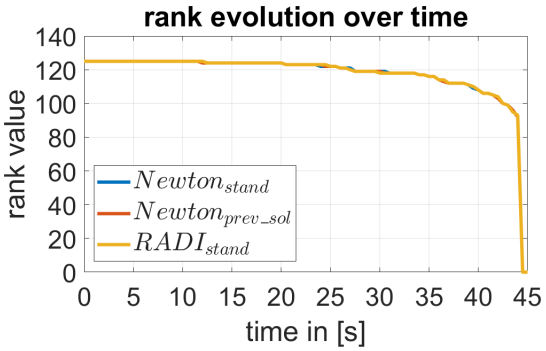
Figure 5.4: Convergence behavior using as inner solvers Newton-Kleinman and RADI for $n = 1357$.

BDF order	Inner solver	Runtime	Avg n. iter	Avg solution rank
$p = 1$	$Newton_{stand}$	7 min 40 sec	2	116.90
	$Newton_{prev\_sol}$	4 min 52 sec	1.3778	116.89
	$RADI_{stand}$	2 min 20 sec	21.144	116.92
$p = 2$	$Newton_{stand}$	6 min 56 sec	2	115.52
	$Newton_{prev\_sol}$	4 min 15 sec	1.3667	115.49
	$RADI_{stand}$	2 min 45 sec	19.367	115.45
$p = 3$	$Newton_{stand}$	6 min 1 sec	2	109.36
	$Newton_{prev\_sol}$	3 min 47 sec	1.4021	109.36
	$RADI_{stand}$	2 min 33 sec	17.443	109.29

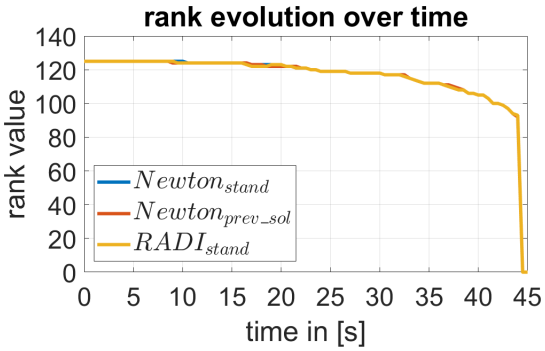
Table 5.2: Comparison for the solution of (5.6) with $n = 1357$.

ton–Kleinman initialized with the solution from the previous iteration. We refer to these solvers as $Newotn_{stand}$, $RADI_{stand}$ and $Newton_{prev\_sol}$, respectively. The solution is computed on the time interval $[0, 45]$ s. In the simulation model, the time axis is scaled by 10^2, corresponding to a model time interval of 4500 seconds. For the following experiments, we focus on large-scale DREs with dimensions $n \in \{1357, 5277,\ 20209,\ 79841\}$. In Table 5.2, we present a comparison of the three different methods for the problem of size $n = 1357$. It is evident that the RADI scheme converges significantly faster than the Newton–Kleinman method with a zero initial guess. In particular, we observe a speedup of approximately 2.37 times for orders 1 and 3, and about 2.52 for order 2. A similar trend is observed when comparing $RADI_{stand}$ with $Newton_{prev\_sol}$: for all BDF orders, $RADI_{stand}$ is roughly 1.5 times faster than $Newton_{prev\_sol}$. As expected, $Newton_{prev\_sol}$ outperforms $Newton_{stand}$, since in the context of differential equations the analytical solution is generally smooth, or at least differentiable. This implies that solutions at consecutive time steps do not differ drastically for sufficiently small time refinements. Consequently, initializing Newton–Kleinman with the solution from the previous time step provides an initial guess already close to the current solution, thereby reducing the number of iterations required for convergence.

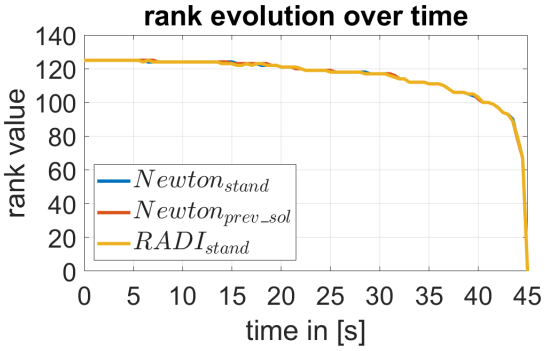
It is worth noting that the average number of iterations of $RADI_{stand}$ decreases as the BDF order increases. This is likely due to the fact that, for higher-order BDF schemes, more accurate initial approximations are used, leading to more precise solutions of the corresponding discretized AREs and consequently fewer iterations needed to reach the desired residual



(a) $p = 1$



(b) $p = 2$



(c) $p = 3$

Figure 5.5: Rank evolution for $n = 1357$.

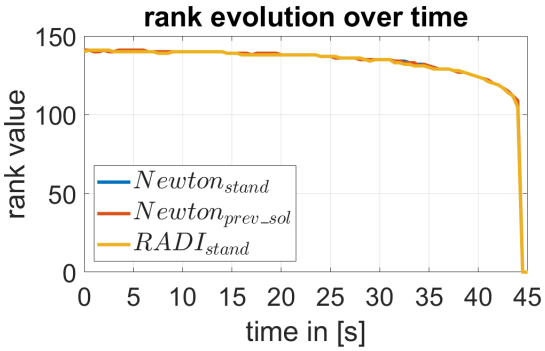
tolerance. In contrast, the two Newton–Kleinman–based methods exhibit almost the same average number of iterations across all three BDF orders. Moreover, for all methods, the average rank of the computed solution decreases as the BDF order increases. In particular, for $p = 1$, the three methods have approximately the same average solution rank, whereas for $p = 2$ and $p = 3$, the average rank of $RADI_{stand}$ becomes slightly smaller compared to that of the two Newton–Kleinman methods, which share nearly identical ranks. Since the computational complexity of the problem depends on the rank of the solution, this observation further explains why RADI converges significantly faster than the others. In Figure 5.5, we illustrate the evolution of the solution rank over time for the three methods. As can be seen, the rank profiles of the different schemes are very similar, almost identical. The curves decrease because we solved (5.6) backward in time, as the terminal condition is typically prescribed. Consequently, at the end of the time horizon, the solution has zero rank since we start from a zero initial value, and the rank gradually increases as we move backward in time. In particular, the rank grows rapidly during the initial phase of the computation and then nearly stabilizes. It is also worth highlighting that, for $p = 3$, the three curves start increasing earlier compared to those for $p = 1$ and $p = 2$. This behavior results from the initialization process described in Remark 4.1.1. Specifically, for $p = 3$, additional intermediate steps are introduced during which the solution is computed using the second-order BDF scheme. As a result, while for $p = 1$ and $p = 2$ the solution begins to increase after the first few time steps, for $p = 3$ the computation starts earlier, leading to nonzero values in the curves sooner. In Table 5.3, we present the comparison of the three methods for the problem size $n = 5177$. Here, the speedup achieved by using $RADI_{stand}$ becomes even clearer. In particular, for all three BDF orders, the speedup of $RADI_{stand}$ over $Newton_{stand}$ is approximately 2. As n increases, we observe that both $Newton_{stand}$ and $RADI_{stand}$ require more time when using $p = 3$ compared to $p = 1$ or $p = 2$. This behavior is expected, since a higher BDF order involves taking into account solutions from more previous time steps. As discussed in the previous algorithms, this implies that the low-rank factors involved are larger, leading to higher computational cost. A similar effect is expected between $p = 1$ and $p = 2$; however, in this case, the problem size may not be large enough for the difference to be clearly visible. We also note that, as before, the average number of iterations of RADI decreases when higher BDF orders are used, whereas this trend is not observed for $Newton_{prev\_sol}$, whose average iteration count is lowest for $p = 2$ but increases for $p = 1$ and even more for $p = 3$. Similarly, the average solution rank decreases with increasing order p, with a more pronounced drop from $p = 2$ to $p = 3$ compared to that from $p = 1$ to $p = 2$. As the problem dimension

BDF order	Inner solver	Runtime	Avg n. iter	Avg solution rank
$p = 1$	$Newton_{stand}$	20 min 38 sec	2	132.35
	$Newton_{prev\_sol}$	14 min 3 sec	1.3	131.97
	$RADI_{stand}$	10 min 32 sec	24.333	131.67
$p = 2$	$Newton_{stand}$	20 min 12 sec	2	130.63
	$Newton_{prev\_sol}$	12 min 52 sec	1.2889	130.60
	$RADI_{stand}$	9 min 51 sec	23.256	130.37
$p = 3$	$Newton_{stand}$	22 min 45 sec	2	124.42
	$Newton_{prev\_sol}$	13 min 25 sec	1.3402	124.48
	$RADI_{stand}$	11 min 23 sec	22.742	124.41

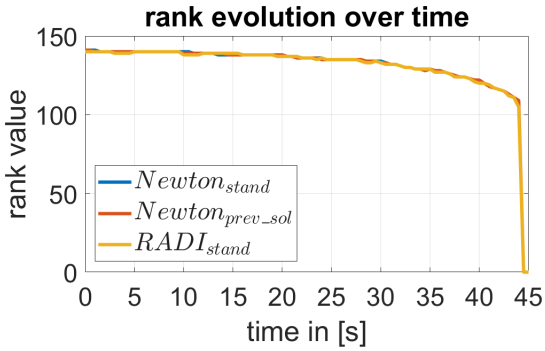
Table 5.3: Comparison for the solution of (5.6) with $n = 5177$.

n increases, the average number of RADI iterations grows compared to the values reported in Table 5.2, whereas for the two Newton–Kleinman methods, it remains nearly unchanged. In Figure 5.6, we show the evolution of the solution rank over time. The overall trend is very similar to that observed in Figure 5.5, with the only difference being that the curves now reach higher rank values.

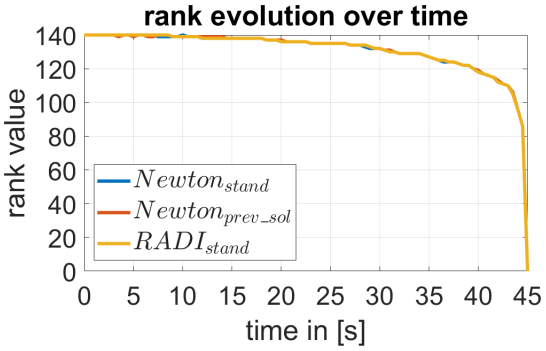
In Table 5.4, we compare the methods for a problem of size $n = 20209$. As before, RADI converges significantly faster than the Newton–Kleinman method with a zero initial guess. In particular, we observe a speedup of approximately 2.50 times for orders 1 and 3, and about 1.60 for order 2. When compared with $Newton_{prev\_sol}$, $RADI_{stand}$ remains faster, achieving a time saving of up to 32 minutes for BDF3. It is also worth highlighting the different runtimes observed for order 3. As discussed earlier, employing a higher BDF order translates, for our algorithms, into handling matrices with larger rank, since more solution factors must be stored at each time step. Consequently, the convergence can slow down due to the increased computational complexity. Let us note that, even though the current problem is approximately $\frac{20209^2}{5177^2} \approx 15$ times larger than the one previously presented for $n = 5177$, the average number of iterations per time step does not differ significantly. In particular, for the two Newton-based methods, it is slightly smaller, while for RADI it is slightly higher. For orders 1 and 2, RADI requires on average six additional iterations per time step, and for order 3 only four more. This increase is relatively minor compared to the substantial growth in problem dimension, confirming that the three low-rank implementations considered are highly suitable for solving large-scale problems.



(a) $p = 1$



(b) $p = 2$



(c) $p = 3$

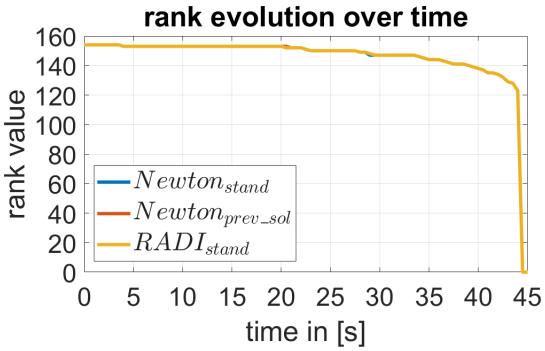
Figure 5.6: Rank evolution for $n = 5177$.

BDF order	Inner solver	Runtime	Avg n. iter	Avg solution rank
$p = 1$	$Newton_{stand}$	2h 38 min	2	145.16
	$Newton_{prev\_sol}$	1h 24 min	1.2111	145.16
	$RADI_{stand}$	1h 6 min	30.467	145.16
$p = 2$	$Newton_{stand}$	2h 13 min	2	143.94
	$Newton_{prev\_sol}$	1h 22 min	1.2	143.94
	$RADI_{stand}$	1h 23 min	29.222	143.93
$p = 3$	$Newton_{stand}$	3h	2	138.11
	$Newton_{prev\_sol}$	1h 45 min	1.2577	138.10
	$RADI_{stand}$	1h 13 min	26.722	138.07

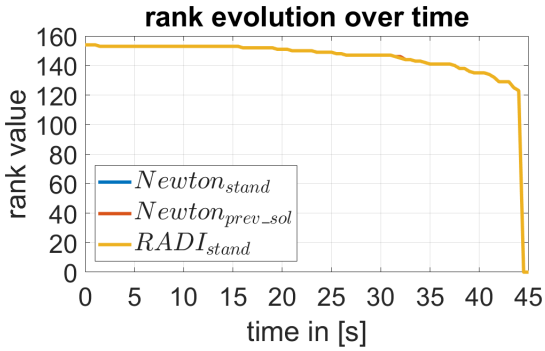
Table 5.4: Comparison for the solution of (5.6) with $n = 20209$.

Furthermore, for all methods, the average rank of the computed solution decreases as the BDF order increases. In Figure 5.7, we illustrate the evolution of the solution rank over time for the three methods.

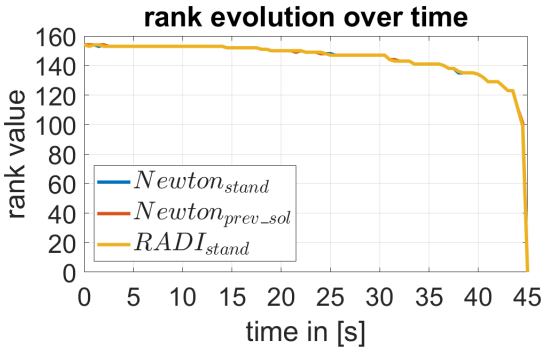
Finally, the largest example we consider corresponds to $n = 79841$, and the results are reported in Table 5.5. As in the previous tests, $RADI_{stand}$ is the fastest scheme, with a speedup of approximately 2.5 and 1.3 when compared to $Newton_{stand}$ and $Newton_{prev_sol}$, respectively. It is worth emphasizing that for large-scale problems such as this one, even a modest speedup of 1.3 is highly significant, as it can reduce the total computational time by up to one and a half hours. As observed before, the average number of RADI iterations per time step increases with n, but this increase remains reasonable compared to the corresponding growth in problem size. Indeed, this problem is about $\frac{79841^2}{20209^2} \approx 16$ times larger than the previous one, yet RADI requires only about five additional iterations per time step for each BDF order. Naturally, this leads to a longer runtime, since larger matrices make each iteration more computationally demanding. However, this is an acceptable trade-off given the substantial increase in problem size. The evolution of the solution rank for the different methods is shown in Figure 5.8, and its behavior is almost identical to that observed in the previous experiments.



(a) $p = 1$



(b) $p = 2$

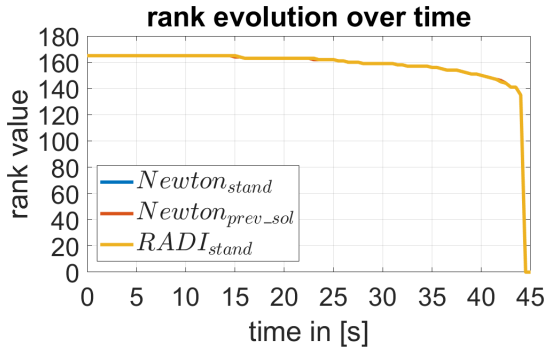


(c) $p = 3$

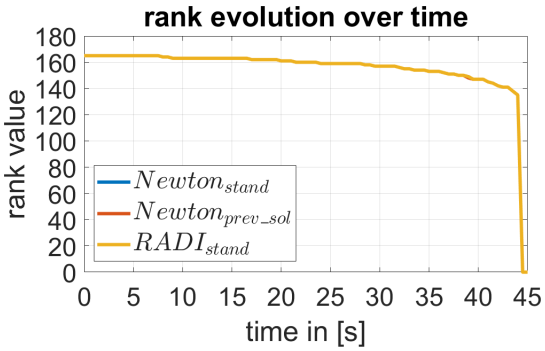
Figure 5.7: Rank evolution for $n = 20209$.

BDF order	Inner solver	Runtime	Avg n. iter	Avg solution rank
$p = 1$	$Newton_{stand}$	12h 10 min	2	156.30
	$Newton_{prev\_sol}$	6h 15 min	1.3111	156.30
	$RADI_{stand}$	4h 45 min	35.111	156.31
$p = 2$	$Newton_{stand}$	11h 18 min	2	154.73
	$Newton_{prev\_sol}$	5h 46 min	1.1333	154.72
	$RADI_{stand}$	4h 33 min	33.722	154.73
$p = 3$	$Newton_{stand}$	12h 11 min	2	148.15
	$Newton_{prev\_sol}$	6h 30 min	1.1959	148.16
	$RADI_{stand}$	4h 54 min	31.041	148.14

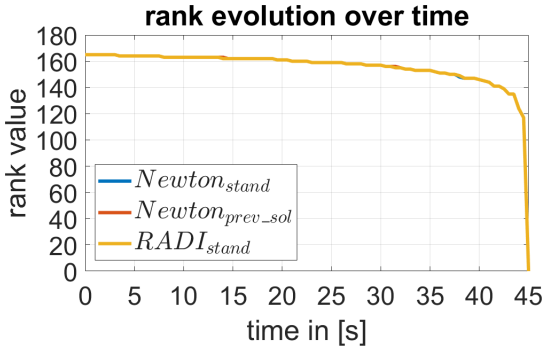
Table 5.5: Comparison for the solution of (5.6) with $n = 79841$.



(a) $p = 1$



(b) $p = 2$



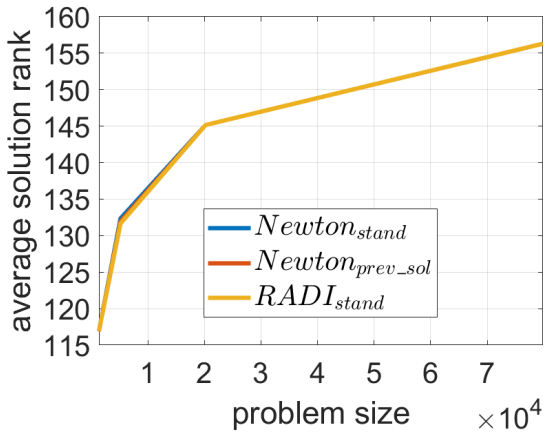
(c) $p = 3$

Figure 5.8: Rank evolution for $n = 79841$.

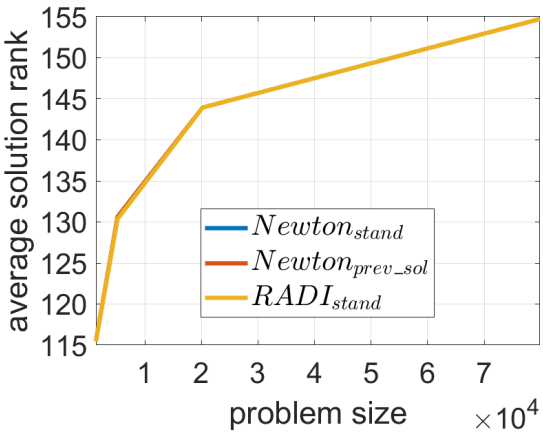
5.2.3 Conclusions

We compared the three methods for problem sizes $n \in \{1357,\ 5177,\ 20209, 79841\}$, and for different BDF orders $p \in \{1, 2, 3\}$. Across all experiments, we observed consistent trends. First, $Newton_{stand}$ is always the slowest scheme among the three. This is expected, since it is the least sophisticated method. As discussed earlier, $Newton_{prev\_sol}$ uses the solution from the previous time step as its initial guess, starting closer to the exact solution. When the time discretization is sufficiently fine, the smoothness of the DRE solution ensures that consecutive solutions differ only slightly. Consequently, reusing the previous solution significantly reduces the number of required iterations and the total runtime. The RADI method, on the other hand, can be seen as a natural extension of the low-rank ADI scheme to quadratic matrix equations. It is particularly efficient due to its specific low-rank structure $X = ZY^{-1}Z^{\mathsf{T}}$, which allows one to avoid certain expensive matrix operations. An interesting future investigation would be to analyze the performance of the Newton–Kleinman method when using the modified ADI algorithm introduced in Chapter 3, where the ADI iteration starts from the solution computed in the previous Newton step. With appropriate parameter tuning, this approach could further accelerate convergence. In such a case, one might wonder: could $Newton_{stand}$ and $Newton_{prev\_sol}$ become faster than RADI? Another potential direction for improvement would be to initialize RADI with the solution from the previous time step when solving DREs. Based on the results obtained from comparing $Newton_{stand}$ and $Newton_{prev\_sol}$, we expect that this modification could yield a more substantial speedup, possibly making it the most efficient BDF solver among those tested. A further common trend observed across all experiments is that the average solution rank decreases as the BDF order increases. Conversely, the average solution rank increases with the problem dimension n, as shown in Figure 5.9, which is perfectly reasonable. In fact, for larger problem sizes, higher-rank solutions are required to accurately capture the underlying dynamics, and thus the average rank naturally increases with n. For all BDF methods, the curves exhibit a similar trend: a steep rise for smaller values of n, which gradually slows as n becomes larger. From the rank-evolution curves, we consistently observe a rapid initial growth, corresponding to the beginning of the computation, i.e. the end of the time horizon, followed by a stabilization phase. This behavior is likely due to the smoothness of the exact solution, which prevents large variations over time. Finally, it is important to highlight the runtime differences between BDF orders. The computational cost for $p = 3$ is noticeably higher than for $p = 1$ or $p = 2$, and this gap becomes more significant as n increases. This behavior aligns with theoretical expectations: higher BDF

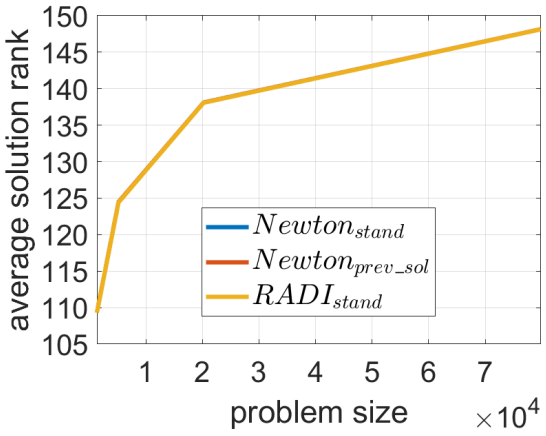
orders require storing and manipulating additional low-rank factors from previous time steps, thereby increasing computational complexity and runtime. To conclude, one may ask: what would happen for even larger problem sizes? Based on the results obtained for the tested dimensions, we expect the same qualitative behavior to persist, with RADI remaining the most efficient method overall.



(a) $p = 1$



(b) $p = 2$



(c) $p = 3$

Figure 5.9: Dependence of average solution rank on problem size for different BDF orders (p).

Bibliography

[1] H. Abou-Kandil, G. Freiling, V. Ionescu, and G. Jank, *Coupled and Generalized Riccati Equations*, in: *Matrix Riccati Equations in Control and Systems Theory*, Systems & Control: Foundations & Applications, Birkhäuser, Basel, 2003. https://doi.org/10.1007/978-3-0348-8081-7_6.

[2] U. M. Ascher and L. R. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*, Proceedings, 1998. https://api.semanticscholar.org/CorpusID:32366732.

[3] R. H. Bartels and G. W. Stewart, *Algorithm 432: Solution of the Matrix Equation $AX + XB = C$*, Communications of the ACM, vol. 15, 1972. https://doi.org/10.1145/361573.361582.

[4] D. A. Bini, B. Iannazzo, and B. Meini, *Numerical Solution of Algebraic Riccati Equations*, SIAM, Philadelphia, 2011. https://doi.org/10.1137/1.9781611972092.

[5] S. Bittanti, A.J. Laub, and J.C. Willems (eds.), *The Riccati Equation*, Communications and Control Engineering, Springer-Verlag, Berlin Heidelberg, 1991. https://doi.org/10.1007/978-3-642-58223-3.

[6] B. Baran, P. Benner, and J. Saak, *Riccati-feedback Control of a Two-dimensional Two-phase Stefan Problem*, Preprint (2022). https://arxiv.org/abs/2209.05476.

[7] B. Baran, P. Benner, J. Saak, and T. Stillfjord, *Numerical methods for closed-loop systems with non-autonomous data*, Preprint (2024). https://arxiv.org/abs/2402.13656.

[8] P. Benner, Z. Bujanović, P. Kürschner, and J. Saak, *A Numerical Comparison of Different Solvers for Large-Scale, Continuous-Time Algebraic Riccati Equations*

and LQR Problems, SIAM J. Sci. Comput., 42 (2018), pp. A957–A996. https://api.semanticscholar.org/CorpusID:119175112.

[9] P. Benner, Z. Bujanović, P. Kürschner, and J. Saak, *RADI: a low-rank ADI-type algorithm for large scale algebraic Riccati equations*, Numer. Math., 138 (2018), pp. 301–330. https://doi.org/10.1007/s00211-017-0907-5.

[10] P. Benner, P. Kürschner, and J. Saak, *Efficient handling of complex shift parameters in the low-rank Cholesky factor ADI method*, Numer. Algor., 62 (2013), pp. 225–251. https://doi.org/10.1007/s11075-012-9569-7.

[11] P. Benner, P. Kürschner, and J. Saak, *An improved numerical method for balanced truncation for symmetric second-order systems*, Math. Comput. Model. Dyn. Syst., 19 (2013), pp. 593–615. https://doi.org/10.1080/13873954.2013.794363.

[12] P. Benner, Z. Tomljanović, and N. Truhar, *Optimal damping of selected eigenfrequencies using dimension reduction*, Numerical Linear Algebra with Applications, vol. 20, pp. 1–17, 2013. https://doi.org/10.1002/nla.833.

[13] P. Benner and J. Saak, *A Semi-Discretized Heat Transfer Model for Optimal Cooling of Steel Profiles*, in: P. Benner, D.C. Sorensen, and V. Mehrmann (eds.), *Dimension Reduction of Large-Scale Systems*, Lecture Notes in Computational Science and Engineering, vol. 45, Springer, Berlin, Heidelberg, 2005. https://doi.org/10.1007/3-540-27909-1_19.

[14] P. Benner and R. Byers, *An exact line search method for solving generalized continuous-time algebraic Riccati equations*, IEEE Transactions on Automatic Control, vol. 43, no. 1, pp. 101–107, Jan. 1998. https://doi.org/10.1109/9.654908.

[15] P. Benner and H. Mena, *Rosenbrock Methods for Solving Riccati Differential Equations*, IEEE Trans. Autom. Control, vol. 58, no. 11, pp. 2950–2956, 2013. https://doi.org/10.1109/TAC.2013.2258495.

[16] P. Benner and H. Mena, *Numerical solution of the Infinite-Dimensional LQR-Problem and the associated Differential Riccati Equations*, Preprint MPIMD/12-13, Max Planck Institute Magdeburg, 2012. Available from http://www.mpi-magdeburg.mpg.de/preprints/.

[17] K. Eppler and F. Tröltzsch, *Fast Optimization Methods in the Selective Cooling of Steel*, in: M. Grötschel, S.O. Krumke, and J. Rambau (eds.), *Online Optimization of Large Scale Systems*, Springer, Berlin, Heidelberg, 2001. https://doi.org/10.1007/978-3-662-04331-8_13.

[18] Z. Bai and J. W. Demmel and A. Ruhe, *Existence, Uniqueness, and Parametrization of Lagrangian Invariant Subspaces*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 1089–1107. https://doi.org/10.1137/S0895479800377228.

[19] J.D. Gardiner, A.J. Laub, J.J. Amato, and C.B. Moler, *Solution of the Sylvester matrix equation* $AXB^{\mathsf{T}} + CXD^{\mathsf{T}} = E$, ACM Transactions on Mathematical Software, vol. 18, no. 2, pp. 223–231, 1992. https://doi.org/10.1145/146847.146929.

[20] I. Gohberg, P. Lancaster, and L. Rodman, *Algebraic Riccati Equations*, Indefinite Linear Algebra and Applications, Birkhäuser, Basel, 2005, pp. 289–318. https://doi.org/10.1007/3-7643-7350-4_14.

[21] C.-H. Guo and A. J. Laub, *On a Newton-Like Method for Solving Algebraic Riccati Equations*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 694–698. https://doi.org/10.1137/S0895479898348519.

[22] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer Series in Computational Mathematics, vol. 14, 2nd ed., Springer-Verlag, Berlin Heidelberg, 1996. https://doi.org/10.1007/978-3-642-05221-7.

[23] W. W. Hager, *Updating the Inverse of a Matrix*, SIAM Rev., 31 (1989), pp. 221–239. https://doi.org/10.1137/1031049.

[24] J. Hoepffner, *Stability and control of shear flows subject to stochastic excitations*, PhD dissertation, KTH Royal Institute of Technology, 2006. https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-3929.

[25] D. Z. Kleinman, *On an Iterative Technique for Riccati Equation Computations*, IEEE Trans. Automat. Control, 13 (1968), pp. 114–115. https://api.semanticscholar.org/CorpusID:121115745.

[26] P. Kürschner, *Efficient Low-Rank Solution of Large-Scale Matrix Equations*, Ph.D. dissertation, Shaker Verlag, Aachen, 2016. https://hdl.handle.net/11858/00-001M-0000-0029-CE18-2.

[27] N. Lang, H. Mena, and J. Saak, *On the benefits of the LDL^T factorization for large-scale differential matrix equation solvers*, Linear Algebra Appl., 480 (2015), pp. 44–71. https://doi.org/10.1016/j.laa.2015.04.006.

[28] J.-R. Li and J. White, *Low Rank Solution of Lyapunov Equations*, SIAM J. Matrix Anal. Appl., vol. 24, no. 1, pp. 260–280, 2002. https://doi.org/10.1137/S0895479801384937.

[29] V.L. Mehrmann (ed.), *The Autonomous Linear Quadratic Control Problem: Theory and Numerical Solution*, Lecture Notes in Control and Information Sciences, Springer-Verlag, Berlin Heidelberg, 1991. https://doi.org/10.1007/BFb0039443.

[30] H. Mena, *Numerical solution of differential Riccati equations arising in optimal control problems for parabolic partial differential equations*, Ph.D. dissertation, Escuela Politécnica Nacional, 2007. https://bibdigital.epn.edu.ec/handle/15000/8434.

[31] H. H. Rosenbrock, *Some general implicit processes for the numerical solution of differential equations*, The Computer Journal, vol. 5, no. 4, pp. 329–330, 1963. https://doi.org/10.1093/comjnl/5.4.329.

[32] J. Schulze and J. Saak, *An extension of the low-rank Lyapunov ADI to non-zero initial values and its applications*, Preprint (2024). https://doi.org/10.48550/arXiv.2406.13477.

[33] J. Schulze and J. Saak, *A unifying framework for ADI-like methods for linear matrix equations and beneficial consequences*, arXiv:2406.13477 [math.NA], 2025. https://arxiv.org/abs/2406.13477.

[34] J. Sabino, *Solution of Large-Scale Lyapunov Equations via the Block Modified Smith Method*, Ph.D. dissertation, Rice University, 2007. https://hdl.handle.net/1911/20641.

[35] H. Sandberg, *A case study in model reduction of linear time-varying systems*, Automatica, vol. 42, no. 3, pp. 467–472, 2006. https://doi.org/10.1016/j.automatica.2005.10.016.

[36] J. Saak, *Efficient Numerical Solution of Large Scale Algebraic Matrix Equations in PDE Control and Model Order Reduction*, Dissertation, Technische Universität Chemnitz, Chemnitz, 2009. https://nbn-resolving.org/urn:nbn:de:bsz:ch1-200901642.

[37] V. Sima, *Algorithms for Linear-Quadratic Optimization*, Chapman and Hall/CRC, 1996. https://doi.org/10.1201/9781003067450.

[38] J. J. Sopka, *Functional Analysis in Normed Spaces (L. V. Kantorovich and G. P. Akilov)*, SIAM Rev., 11 (1969), pp. 412–413. https://doi.org/10.1137/1011077.

[39] J. Saak, M. Köhler, and P. Benner, *M-M.E.S.S. – the Matrix Equations Sparse Solvers library*, https://doi.org/10.5281/zenodo.632897. See also: https://www.mpi-magdeburg.mpg.de/projects/mess.

[40] F.L. Lewis, D.L. Vrabie, and V.L. Syrmos, *Optimal Control*, John Wiley & Sons, 2012. https://doi.org/10.1002/9781118122631.

[41] C. Penland and P.D. Sardeshmukh, *The optimal growth of tropical sea surface temperature anomalies*, Journal of Climate, vol. 8, no. 8, pp. 1999–2024, 1995. https://doi.org/10.1175/1520-0442(1995)008<1999:TOGOTS>2.0.CO;2.

[42] T. Penzl, *Numerical solution of generalized Lyapunov equations*, Advances in Computational Mathematics, vol. 8, no. 1, pp. 33–48, 1998. https://doi.org/10.1023/A:1018979826766.

[43] P. Benner, H. Mena, and J. Saak, *On the parameter selection problem in the Newton-ADI iteration for large-scale Riccati equations*, ETNA. Electronic Transactions on Numerical Analysis, 29 (2007), pp. 136–149. http://eudml.org/doc/130671.

[44] E. L. Wachspress, *Iterative Solution of Elliptic Systems: And Applications to the Neutron Diffusion Equations of Reactor Physics*, Prentice-Hall, Englewood Cliffs, N.J., 1966.

[45] E. Wachspress, *The ADI Model Problem*, Springer, New York, NY, 2013. https://doi.org/10.1007/978-1-4614-5122-8.

[46] J.C. Willems, *Least squares stationary optimal control and the algebraic Riccati equation*, IEEE Transactions on Automatic Control, vol. 16, no. 6, pp. 621–634, December 1971. https://doi.org/10.1109/TAC.1971.1099831.