Journal of Computational and Applied Mathematics 313 (2017) 515–535

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Methods for verified stabilizing solutions to continuous-time algebraic Riccati equations

Tayyebe Haqiri [a,b], Federico Poloni [c,*]

[a] Department of Applied Mathematics, Faculty of Mathematics and Computer, Shahid Bahonar University of Kerman, Kerman, Iran

[b] Young Researchers Society of Shahid Bahonar University of Kerman, Kerman, Iran

[c] Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy

ARTICLE INFO

Article history:
Received 7 September 2015
Received in revised form 23 August 2016

MSC:
65M32
35Kxx
65T60

Keywords:
Algebraic Riccati equation
Stabilizing solution
Interval arithmetic
Verified computation
Krawczyk's method

ABSTRACT

We describe a procedure based on the Krawczyk method to compute a verified enclosure for the stabilizing solution of a continuous-time algebraic Riccati equation $A^*X + XA + Q = XGX$ building on the work of Hashemi (2012) and adding several modifications to the Krawczyk procedure. We show that after these improvements the Krawczyk method reaches results comparable with the current state-of-the-art algorithm (Miyajima, 2015), and surpasses it in some examples. Moreover, we introduce a new direct method for verification which has a cubic complexity in term of the dimension of X, employing a fixed-point formulation of the equation inspired by the ADI procedure. The resulting methods are tested on a number of standard benchmark examples.

1. Introduction

Consider the continuous-time algebraic Riccati equation (*CARE*)

$$A^*X + XA + Q = XGX, \tag{1.1}$$

where A, G and $Q \in \mathbb{C}^{n\times n}$ are given, G and Q are Hermitian, and $X \in \mathbb{C}^{n\times n}$ is unknown. Here, the notation A^* denotes the conjugate transpose of a complex matrix A while A^T shows the transpose of A. CAREs have a variety of applications in the field of control theory and filter design, such as the linear–quadratic optimal control problem and Hamiltonian systems of differential equations. We refer the reader to the books [1,2] for further information on the theoretical properties, solution algorithms and applications of CAREs.

A solution X_s of (1.1) is called *stabilizing* if the *closed loop matrix* $A - GX_s$ is *Hurwitz stable*, i.e., if all its eigenvalues have strictly negative real part. If a stabilizing solution X_s exists, it is unique and Hermitian, i.e., $X_s = (X_s)^*$. The stabilizing solution is the one of interest in almost all applications.

The work presented here addresses the problem of *verifying* the stabilizing solution of (1.1), that is, determining an interval matrix which is guaranteed to contain X_s. The main tool used for verified computation is interval arithmetic. Following well-established principles (see e.g. [3, Section 1]), we do not implement a solution algorithm using interval

* Corresponding author.
E-mail addresses: Haqiri@math.uk.ac.ir, thaqiri@gmail.com (T. Haqiri), fpoloni@di.unipi.it (F. Poloni).

http://dx.doi.org/10.1016/j.cam.2016.09.021
0377-0427/

arithmetic, but rather we assume that an approximated solution $\check{X} \approx X_s$ is available (computed, for instance, with a traditional, non-verified numerical method in machine arithmetic), and we use interval arithmetic to prove that a suitable interval matrix $\mathbf{X} \ni \check{X}$ contains X_s.

The problem of computing verified solutions to matrix Riccati equations (AREs) has been addressed before in the literature: the algorithms in [4,5], based on the interval Newton method, are pioneering works in this context but their computational complexity is $\mathcal{O}(n^6)$. In [4], the authors apply Brouwer's fixed point theorem to calculate verified solutions of the ARE

$$A^T X + XA + Q = XBR^{-1}B^T X, \tag{1.2}$$

with real symmetric matrices Q and R, Q positive semi definite and R positive definite. They find an interval matrix including a positive definite solution of (1.2). The paper [6] decreases this cost to $\mathcal{O}(n^5)$ by using the Krawczyk method, which is a variant of the Newton method that it does not require the inversion of an interval matrix. A major improvement is the algorithm in [7], which is applicable when the closed-loop matrix $A - G\check{X}$ is diagonalizable where $\check{X}$ denotes a numerical computed solution of (1.1), and requires only $\mathcal{O}(n^3)$ operations. The recent paper [8] describes a more efficient algorithm based again on the diagonalization of $A - G\check{X}$, $\check{X}$ a Hermitian numerical solution of (1.1). The resulting method has cubic complexity as well. An important feature of this algorithm is that does not require iteration to find a suitable candidate interval solution, unlike the previous methods. Rather, it uses a clever mix of interval arithmetic and IEEE arithmetic with prescribed rounding to determine the optimal radius of the interval $\mathbf{X}$. Hence it is typically faster than the alternatives. The same paper [8] also includes a method to verify the uniqueness and the stabilizing property of the computed solution.

We propose here a variant of the Krawczyk method suggested in [7], introducing several modifications. Namely:

- We use the technique introduced in [9], which consists in applying the Krawczyk method not to the original equation, but to one obtained after a change of basis, in order to reduce the number of verified operations required, with the aim to reduce the wrapping effects.
- We exploit the invariant subspace formulation of a CARE to make another change of basis, following a technique introduced in [10] for the *non-verified* solution of Riccati equations. This technique employs a suitable permutation of the Hamiltonian matrix to transform (1.1) into a different CARE whose stabilizing solution Y_s has bounded norm.
- When applying the Krawczyk method, an enclosure for the so-called *slope* matrix is needed; the standard choice to compute it is using the interval evaluation of the Jacobian of the function at hand. Instead, we use a different algebraic expression which results in a smaller interval.

With these improvements, the Krawczyk method can be used to prove that a solution exists inside some interval matrix, but not that this solution is unique or stabilizing. Our strategy for proving uniqueness is indirect: after having verified the existence of a solution $X_s \in \mathbf{X}$, we check if all the matrices inside the interval matrix $A - G\mathbf{X}$ are Hurwitz stable. If this holds, then it is automatically verified that the interval matrix $\mathbf{X}$ contains only one solution, and that it is the stabilizing one.

In addition, we present a different algorithm, based on a reformulation of (1.1) as a fixed-point equation, which requires $\mathcal{O}(n^3)$ operations per step and does not require the diagonalizability of the closed-loop matrix $A - G\check{X}$ in which $\check{X}$ is the computed approximate stabilizing solution of CARE (1.1). This algorithm is generally less reliable than the Krawczyk-based ones, but it has the advantage of not breaking down in cases in which the closed-loop matrix is defective or almost defective.

The techniques presented here can be adapted with minor sign changes to *anti-stabilizing* solutions, i.e., solutions X_{as} for which all the eigenvalues of $A - GX_{as}$ have positive real part. The algorithms in [7,8], in contrast, do not restrict to verifying stabilizing solutions only; however, solutions which are neither stabilizing nor anti-stabilizing have very few applicative uses.

We conclude the paper by evaluating the proposed algorithms on a large set of standard benchmark problems [11,12] for Riccati equations, comparing them with the algorithms in [7,8]. Using all the improvements described here, the gap between the Krawczyk method and the current best method in [8] is essentially eliminated. The four methods each handle satisfactorily a slightly different set of problems, and none of them is beaten by the alternatives in all possible experiments.

The paper is organized as follows. In the next section we introduce some notation and standard results in linear algebra and interval analysis which are at the basis of our methods. In Section 3 we discuss various algorithms based on the Krawczyk method to compute a thin interval matrix enclosing a solution of (1.1) while in Section 4, a fixed point approach is presented. In Sections 5 and 6 we perform some numerical tests and draw the conclusions and outlook, respectively.

2. Preliminaries and notation

We try to follow the standard notation of interval analysis defined in [13]. Subsequently, we use boldface lower and upper case letters for interval scalars or vectors and matrices, respectively, whereas lower case stands for scalar quantities and point vectors and upper case represents matrices.

Complex intervals can be defined either as rectangles or as discs. We use here the definition as discs, i.e., $\mathbb{IC}_{\text{disc}}$: a circular complex interval $\mathbf{x}$, or circular disc or simply a complex interval, is a closed circular disc of radius $\text{rad}(\mathbf{x}) \in \mathbb{R}$ with $\text{rad}(\mathbf{x}) \geq 0$ and center $\text{mid}(\mathbf{x}) \in \mathbb{C}$, written as $\mathbf{x} = \langle \text{mid}(\mathbf{x}), \text{rad}(\mathbf{x}) \rangle$. Operations on circular complex intervals can be defined (see e.g. [14,3]) so that they provide inclusion intervals for the exact results, i.e.,

$$\mathbf{x} \circ \mathbf{y} \supseteq \{x \circ y \ : \ x \in \mathbf{x}, \ y \in \mathbf{y}\}, \quad \circ \in \{+, -, \cdot, /\}.$$

Operations between a complex interval and a complex number $z \in \mathbb{C}$ can be performed by identifying z with $\langle z, 0\rangle \in \mathbb{IC}_{\text{disc}}$. We shall also use the notation $\mathbf{x}^{-1} = 1/\mathbf{x}$.

The *interval hull* of two intervals $\mathbf{x}$ and $\mathbf{y}$ is denoted by $\square(\mathbf{x}, \mathbf{y})$ which is the smallest interval containing $\mathbf{x}$ and $\mathbf{y}$. The magnitude of $\mathbf{x} \in \mathbb{IC}_{\text{disc}}$ is defined as $\text{mag}(\mathbf{x}) := \max\{|x| : x \in \mathbf{x}\}$.

We denote by $\mathbf{A} = \langle \text{mid}(\mathbf{A}), \text{rad}(\mathbf{A})\rangle \in \mathbb{IC}_{\text{disc}}^{m\times n}$ the $m \times n$ interval matrix $\mathbf{A}$ whose (i, j) element is the complex interval $\langle \text{mid}(\mathbf{A}_{ij}), \text{rad}(\mathbf{A}_{ij})\rangle$, with $\text{rad}(\mathbf{A}_{ij}) \geq 0$; $1 \leq i \leq m$, $1 \leq j \leq n$. For interval vectors and matrices, mid, rad, mag, and $\square$ will be applied component-wise.

The *Frobenius norm* of a complex matrix $A = (A_{ij})$ is defined as $\|A\|_F := (\sum_{i,j} |A_{ij}|^2)^{1/2}$. This definition can be extended to complex interval matrices, providing an interval-valued function $\|\mathbf{A}\|_F$ defined as the smallest interval containing $\{x : x = \|A\|_F, A \in \mathbf{A}\}$.

The *Kronecker product* $A \otimes B$ of an $m \times n$ matrix $A = (A_{ij})$ and a $p \times q$ matrix B is an $mp \times nq$ matrix defined as the block matrix whose blocks are $A \otimes B := [A_{ij}B]$. For a point matrix $A \in \mathbb{C}^{m\times n}$, the vector $\text{vec}(A) \in \mathbb{C}^{mn}$ denotes column-wise vectorization whereby the successive columns of A are stacked one below the other, beginning with the first column and ending with the last. Moreover, $\bar{A}$ denotes the complex conjugate of A and if A is an invertible matrix, then $A^{-T} := (A^T)^{-1}$ and $A^{-*} := (A^*)^{-1}$. The element-wise division of a matrix $A = (A_{ij}) \in \mathbb{C}^{m\times n}$ by a matrix $B = (B_{ij}) \in \mathbb{C}^{m\times n}$, also known as the *Hadamard division*, denoted by $A./B$, results in an $m \times n$ matrix $C = (C_{ij})$ whose (i, j) element is given by $C_{ij} = A_{ij}/B_{ij}$ provided that $B_{ij} \neq 0$, for each $1 \leq i \leq m$ and $1 \leq j \leq n$. For a given vector $d = (d_1, d_2, \ldots, d_n)^T \in \mathbb{C}^n$, $\text{Diag}(d) \in \mathbb{C}^{n\times n}$ is the diagonal matrix whose (i, i) entry is d_i. Conversely, given a diagonal matrix D, $\text{diag}(D)$ is the vector whose elements are the diagonal entries of D. Most of these notions and operations are analogously defined for interval quantities.

The definition of inverse of an interval matrix may be problematic in general, but if $\mathbf{D} = \text{Diag}(\mathbf{d})$ is diagonal, with $\mathbf{d} = (\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_N)^T$ and $0 \notin \mathbf{d}_i$ for each $i = 1, 2, \ldots, N$, then we may define $\mathbf{D}^{-1} := \text{Diag}((\mathbf{d}_1^{-1}, \mathbf{d}_2^{-1}, \ldots, \mathbf{d}_N^{-1})^T)$.

The following lemmas contain simple arithmetical properties of the Kronecker product and the vec operator which we will use in the following. Most of them appear also e.g. in [9] or [15].

Lemma 2.1. *Assume that $A = (A_{ij})$, $B = (B_{ij})$, $C = (C_{ij})$ and $D = (D_{ij})$ be complex matrices with compatible sizes. Then,*

(1) $(A \otimes B)(C \otimes D) = AC \otimes BD$,
(2) $A \otimes (B + C) = (A \otimes B) + (A \otimes C)$,
(3) $(A \otimes B)^* = A^* \otimes B^*$,
(4) $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$, *if A and B are invertible,*
(5) $\text{vec}(ABC) = (C^T \otimes A)\,\text{vec}(B)$,
(6) $(\text{Diag}(\text{vec}(A)))^{-1}\,\text{vec}(B) = \text{vec}(B./A)$, *if $A_{ij} \neq 0$ for each (i, j).*

Lemma 2.2. *Let $\mathbf{A} = (\mathbf{A}_{ij})$, $\mathbf{B} = (\mathbf{B}_{ij})$ and $\mathbf{C} = (\mathbf{C}_{ij})$ be complex interval matrices of compatible sizes. Then,*

(1) $\left\{(C^T \otimes A)\,\text{vec}(B) : A \in \mathbf{A},\ B \in \mathbf{B},\ C \in \mathbf{C}\right\} \subseteq \begin{Bmatrix} \text{vec}\big(\mathbf{A}(\mathbf{BC})\big), \\ \text{vec}\big((\mathbf{AB})\mathbf{C}\big), \end{Bmatrix}$

(2) $\Big(\text{Diag}(\text{vec}(\mathbf{A}))\Big)^{-1}\,\text{vec}(\mathbf{B}) = \text{vec}(\mathbf{B}./\mathbf{A})$, *if $0 \notin \mathbf{A}_{ij}$ for all (i, j).*

The *interval evaluation* $\mathbf{f}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ of a function $f(x_1, x_2, \ldots, x_N)$ (defined by an explicit formula) is obtained by replacing (1) the variables $x_1, x_2, \ldots, x_N$ with interval variables $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ and (2) each arithmetic operation in the formula with the corresponding interval operation. The following *inclusion property* holds (see e.g. [14]):

$$f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) := \{f(x_1, x_2, \ldots, x_N) : x_1 \in \mathbf{x}_1, x_2 \in \mathbf{x}_2, \ldots, x_N \in \mathbf{x}_N\} \subseteq \mathbf{f}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N).$$

Note that, in principle, different equivalent formulas for the same ordinary function could give different interval evaluations (for instance, $\mathbf{x}(\mathbf{x} + 1)$ vs. $\mathbf{x} \cdot \mathbf{x} + \mathbf{x}$). Choosing the version which gives the tighter interval is an important detail.

In addition, one of the main difficulties in dealing with multivariate problems with interval arithmetic is the so-called *wrapping effect*: the image of an interval vector under a map (even a simple one such as matrix–vector multiplication $\mathbf{x} \mapsto A\mathbf{x}$) is in general *not* an interval vector; hence, in our computations we have to replace it with an enclosing interval. This may lead to a considerable increase of the size of the intervals, especially if it happens repeatedly during an algorithm. We refer the reader to the review article [3] for a thorough introduction.

3. Modified Krawczyk's methods

Enclosure methods using interval arithmetic are based on the following idea. Let $g : \mathbb{C}^N \to \mathbb{C}^N$ be some function of which we wish to find a zero. First find a function $h : \mathbb{C}^N \to \mathbb{C}^N$ whose fixed points are known to be the zeros of g. Assume that h is continuous and that an interval evaluation $\mathbf{h}$ is available. Then if $\mathbf{h}(\mathbf{x}) \subseteq \mathbf{x}$ we know that $h(\mathbf{x}) \subseteq \mathbf{x}$ and so h has a fixed point in $\mathbf{x}$ by Brouwer's theorem [16].

In this paper, often the functions $\mathbf{h}$ are variants of the Krawczyk operator. To define this operator, we first need the concept of a slope.

Definition 3.1 (*See e.g. [14]*)**.** Suppose $f : \psi \subseteq \mathbb{C}^N \to \mathbb{C}^N$ and $x, y \in \mathbb{C}^N$. Then, a *slope* $S(f; x, y)$ is a mapping from the Cartesian product $\psi \times \psi$ to $\mathbb{C}^{N\times N}$ such that

$$f(y) - f(x) = S(f; x, y)(y - x).$$

We are now ready to state the result which is at the basis of all the modified Krawczyk-type algorithms used in the rest of our paper.

Theorem 3.2 (*See e.g. [17]*)**.** *Assume that $f : \psi \subset \mathbb{C}^N \to \mathbb{C}^N$ is continuous. Let $\check{x} \in \psi$ and $\mathbf{z} \in \mathbb{IC}^N_{disc}$ be such that $\check{x} + \mathbf{z} \subset \psi$. Moreover, assume that $\mathcal{S} \subset \mathbb{C}^{N\times N}$ is a set of matrices such that $S(f; \check{x}, x') \in \mathcal{S}$ for every $x' \in \check{x} + \mathbf{z} =: \mathbf{x}$. Finally, let $R \in \mathbb{C}^{N\times N}$. Denote by $\mathcal{K}_f(\check{x}, R, \mathbf{z}, \mathcal{S})$ the set*

$$\mathcal{K}_f(\check{x}, R, \mathbf{z}, \mathcal{S}) := \{-Rf(\check{x}) + (I_N - RS)z \ : \ S \in \mathcal{S}, \ z \in \mathbf{z}\}.$$

If

$$\mathcal{K}_f(\check{x}, R, \mathbf{z}, \mathcal{S}) \subseteq \operatorname{int}(\mathbf{z}), \tag{3.1}$$

then the function f has a zero x_* in $\check{x} + \mathcal{K}_f(\check{x}, R, \mathbf{z}, \mathcal{S}) \subseteq \mathbf{x}$, in which $\operatorname{int}(\mathbf{z})$ is the topological interior of $\mathbf{z}$.

Moreover, if $S(f; y, y') \in \mathcal{S}$ for each $y, y' \in \mathbf{x}$, then x_* is the only zero of f contained in $\mathbf{x}$.

In computation, one defines the Krawczyk operator [18]

$$\mathbf{k}_f(\check{x}, R, \mathbf{z}, \mathbf{S}) := -Rf(\check{x}) + (I_N - R\mathbf{S})\mathbf{z}, \tag{3.2}$$

where $\mathbf{S}$ is an interval matrix containing all slopes $S(f; y, y')$ for $y, y' \in \mathbf{x}$. In many cases, a possible choice for $\mathbf{S}$ can be obtained from $\mathbf{f}'(\mathbf{x})$, an interval evaluation of the Jacobian f' on the interval $\mathbf{x}$. Indeed, in the case of real intervals it holds that $S(f; x, y) \in \mathbf{f}'(\mathbf{x})$ for all $x, y \in \mathbf{x}$, because of the mean value theorem. In the complex case, though, this inclusion does not always hold. By the inclusion property of interval arithmetic,

$$\mathbf{k}_f(\check{x}, R, \mathbf{z}, \mathbf{S}) \subset \operatorname{int}(\mathbf{z}) \tag{3.3}$$

implies (3.1). So, if (3.3) is satisfied then f has a zero in $\check{x} + \mathbf{k}_f(\check{x}, R, \mathbf{z}, \mathbf{S})$. In practice, one attempts to make the terms $-Rf(\check{x})$ and $I_N - R\mathbf{S}$ as small as possible, to obtain the crucial relation (3.3). The typical choice is taking as $\check{x}$ a good approximation of a zero of f and as R a good approximation of $(f'(\check{x}))^{-1}$, both obtained via a classic floating point algorithm, see for instance [9].

3.1. A residual form for the Krawczyk operator

We now introduce the concepts that are needed to apply the modified Krawczyk method to solve a matrix equation such as (1.1). The *Fréchet derivative* [19] of a Fréchet differentiable matrix function $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ at a point $X \in \mathbb{C}^{n\times n}$ is a linear mapping $L_F : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ such that for all $E \in \mathbb{C}^{n\times n}$

$$F(X + E) - F(X) - L_F(X, E) = o(\|E\|).$$

Since L_F is a linear operator, we can write

$$\operatorname{vec}(L_F(X, E)) = K_F(X) \operatorname{vec}(E),$$

for a matrix $K_F(X) \in \mathbb{C}^{n^2\times n^2}$ that depends on L but not E. One refers to $K_F(X)$ as the *Kronecker form* of the Fréchet derivative of F at X.

In the case of the continuous-time algebraic Riccati equation (1.1), we apply the Krawczyk method to the function $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ defined as

$$F(X) := A^*X + XA + Q - XGX,$$

which appeared before in [7]. For this function, one has

$$L_F(X, E) = E(A - GX) + (A^* - XG)E.$$

Lemma 2.1 part 5 turns out that its Kronecker form is

$$K_F(X) = I_n \otimes (A^* - XG) + (A - GX)^T \otimes I_n.$$

When $X = X^*$, we can write this expression in an alternate form as

$$K_F(X) = I_n \otimes (A - GX)^* + (A - GX)^T \otimes I_n. \tag{3.4}$$

We wish to use the modified Krawczyk algorithm on the function obtained by regarding F as a vector map $f : \mathbb{C}^N \to \mathbb{C}^N$, with $N = n^2$, defined by

$$f(x) := \operatorname{vec}(A^*X + XA + Q - XGX), \quad x = \operatorname{vec}(X). \tag{3.5}$$

The following result, which is a slight variation of a theorem in [7], shows that the Fréchet derivative can be used to obtain an enclosure for the slope in the modified Krawczyk method. We report it, with a different proof from the one in [7], because this presentation will be more convenient in the following development of our method. Due to this reformulation, we will get a weaker result with respect to uniqueness.

Theorem 3.3. *Let* $\mathbf{X}$ *be an interval matrix, and* $\mathbf{K}_F(\mathbf{X}) = I_n \otimes (A - G\mathbf{X})^* + (A - G\mathbf{X})^T \otimes I_n$ *be the interval evaluation of* $K_F(X)$ *in* (3.4)*. Then for each* $Y, Y' \in \mathbf{X}$ *such that* $Y = Y^*$*, it holds that* $S(f; y, y') \in \mathbf{K}_F(\mathbf{X})$*, where* $y = \operatorname{vec}(Y)$, $y' = \operatorname{vec}(Y')$.

Proof. We have

$$\begin{aligned}\operatorname{vec}(F(Y) - F(Y')) &= \operatorname{vec}((A^* - YG)(Y - Y') + (Y - Y')(A - GY')) \\ &= \operatorname{vec}((A - GY)^*(Y - Y') + (Y - Y')(A - GY')) \\ &= (I_n \otimes (A - GY)^* + (A - GY')^T \otimes I_n) \operatorname{vec}(Y - Y'),\end{aligned}$$

hence by the inclusion property of interval arithmetic

$$S(f; y, y') = (I_n \otimes (A - GY)^* + (A - GY')^T \otimes I_n) \in \mathbf{K}_F(\mathbf{X}). \quad \square \tag{3.6}$$

The next ingredient that we need to apply the Krawczyk algorithm is the matrix R. One would like to use $R \approx (K_F(\check{X}))^{-1}$, where $\check{X} = \check{X}^*$ is an approximation of the stabilizing solution to the CARE (1.1) computed in floating point arithmetic. However, this is the inverse of an $n^2 \times n^2$ matrix, whose computation would cost $\mathcal{O}(n^6)$ floating point operations in general. Even considering the Kronecker product structure of $K_F(\check{X})$, there is no algorithm in literature to compute R explicitly with less than $\mathcal{O}(n^5)$ arithmetic operations. The action of R, that is, computing the product Rv given a vector $v \in \mathbb{C}^{n^2}$, can be computed with $\mathcal{O}(n^3)$ operations with methods such as the Bartels–Stewart algorithm [20]. However, this method cannot be used effectively in conjunction with interval arithmetic due to excessive wrapping effects, as argued in [21].

The work [7] (and, earlier, on a similar equation, [9]) contains an alternative method to perform this computation with complexity $\mathcal{O}(n^3)$, in the case when $A - G\check{X}$ is diagonalizable, where $\check{X}$ is a numerical solution of CARE (1.1). Assume that an approximate eigendecomposition of $A - G\check{X}$ is available, that is,

$$A - G\check{X} \approx V \Lambda W \quad \text{with } V, W, \Lambda \in \mathbb{C}^{n\times n}, \tag{3.7a}$$

$$\Lambda = \operatorname{Diag}(\lambda_1, \lambda_2, \ldots, \lambda_n), \quad VW \approx I_n. \tag{3.7b}$$

We write $\approx$ instead of $=$ because $V, W \approx V^{-1}$ and λ_i, $i = 1, 2, \ldots, n$ are computed numerically with a standard method such as MATLAB's `eig`. So, equality does not hold (in general) in the mathematical sense. Furthermore, we assume the nonsingularity of

$$\Delta = I_n \otimes \Lambda^* + \Lambda^T \otimes I_n. \tag{3.7c}$$

Once these quantities are computed, we can factorize $K_F(\check{X})$ by replacing I_n in $I_n \otimes (A - G\check{X})^*$ with $V^{-T} I_n V^T$ and in $(A - G\check{X})^T \otimes I_n$ with $W^* I_n W^{-*}$ and then using Lemma 2.1, so that

$$\begin{aligned}K_F(\check{X}) &= I_n \otimes (A - G\check{X})^* + (A - G\check{X})^T \otimes I_n \\ &= (V^{-T} \otimes W^*)(I_n \otimes (W(A - G\check{X})W^{-1})^* + (V^{-1}(A - G\check{X})V)^T \otimes I_n)(V^T \otimes W^{-*}),\end{aligned}$$

and choose R as

$$R = (V^{-T} \otimes W^*)\Delta^{-1}(V^T \otimes W^{-*}). \tag{3.8}$$

Then, $R \approx (K_F(\check{X}))^{-1}$ holds since if $\check{X}$ is close enough to the stabilizing solution of (1.1), then one can expect that $W(A - G\check{X})W^{-1}$ and also $V^{-1}(A - G\check{X})V$ to be close to Λ. So, the computation of an enclosure, $\operatorname{vec}(\mathbf{L})$, for $l := -Rf(\check{x})$ in $\mathcal{K}_f(\check{x}, R, \mathbf{z}, \mathcal{S})$ can be done using exclusively the matrix–matrix operations, as shown in Lines 1–9 of Algorithm 1.

For the latter term in each member of $\mathcal{K}_f(\check{x}, R, \mathbf{z}, \mathcal{S})$, i.e., $(I_{n^2} - RS)z$, however, we get

$$\begin{aligned}u &:= (I_{n^2} - RS)z = (I_{n^2} - (V^{-T} \otimes W^*)\Delta^{-1}(V^T \otimes W^{-*})(I_n \otimes (A - GY)^* + (A - GY')^T \otimes I_n))z \\ &= ((V^{-T} \otimes W^*)\Delta^{-1}(\Delta - I_n \otimes (W(A - GY)W^{-1})^* - (V^{-1}(A - GY')V)^T \otimes I_n)(V^T \otimes W^{-*}))z,\end{aligned}$$

in which I_{n^2} has replaced by $V^{-T}V^T \otimes W^*W^{-*}$, and $Y, Y' \in \mathbf{X}$ with $Y = Y^*$. Then, Algorithm 2 Lines 2–7 will compute an enclosure for this term as the interval matrix $\mathbf{U}$ whose vectorization contains u.

Another point to note is that we can transform the multiplication $\Gamma^{-1} \operatorname{vec}(M)$, for an $n \times n$ matrix M and a diagonal matrix Γ, into $M./N$, where N is defined by $N_{ij} = \bar{\Gamma}_{ii} + \Gamma_{jj}$, using point 6 of Lemma 2.1, and similarly for interval matrices using point 2 of Lemma 2.2. This point will appear in, for example, Algorithm 1 Line 8, and Algorithm 2 Line 6.

The standard method [3] to obtain an interval vector $\mathbf{z} = \operatorname{vec}(\mathbf{Z})$ that satisfies (3.3) is an iterative one. We start from the residual matrix $\mathbf{Z}_0 := \mathbf{F}(\check{\mathbf{X}})$, that is, the interval evaluation of $F(\check{X})$, and proceed alternating successive steps of enlarging

this interval with a technique known as ε-*inflation* [3], applying the Krawczyk operator to it, $\mathbf{z}_{i+1} = \mathbf{k}_f(\check{x}, R, \mathbf{z}_i, \mathbf{S})$. This procedure terminates when (and if) we find an interval for which (3.3) holds; it is ultimately a trial-and-error procedure, which is not guaranteed to succeed: the operator $\mathbf{k}_f$ may simply not contract its interval argument $\mathbf{z}_i$ sufficiently. This may be due to ill-conditioning of the original equation, to a bad choice of R, or to wrapping effects and other overestimations in the interval computations.

As we will show in the numerical experiments in Section 5, in all cases our algorithms based on the Krawczyk method either terminated after 1–2 steps or failed. So in practice the number of steps can be kept very small.

Several slightly different versions of the iterative procedure to obtain a valid interval for inclusion appear in literature; some involve intersecting the intervals obtained in different steps [9,17,22], and some involve two attempts at inclusion in each iteration [9,22]. We use here the simplest approach, following [7,3]. The exact strategy is shown in Algorithm 1 (and its subroutine Algorithm 2). The algorithm with these choices coincides with the algorithm presented in [7], except for the fact that [7] presents it for a generic Hermitian solution.

In all algorithms, whenever the evaluation order of an expression is not specified exactly due to missing brackets, we evaluate from left to right.

Algorithm 1 Computation of an interval matrix $\mathbf{X}$ containing a solution of CARE (1.1).

Compute an approximate stabilizing solution $\check{X}$ of CARE (1.1) using any floating point algorithm
Compute approximations V, W, Λ for the eigendecomposition of $A - G\check{X}$ in floating point {For instance, using the MATLAB command `eig`}
Compute with floating point arithmetic $D := (D_{ij})$ such that $D_{ij} \approx \bar{\Lambda}_{ii} + \Lambda_{jj}$
Compute interval matrices $\mathbf{I}_V$ and $\mathbf{I}_W$ containing V^{-1} and W^{-1}, respectively {For instance, using `verifylss.m` from INTLAB} If this fails, or if D has any zero elements, return **failure**
$\check{\mathbf{X}} = \langle \check{X}, 0 \rangle$ {To ensure that operations involving $\check{X}$ are performed in a verified fashion with interval arithmetic}
$\mathbf{F} = A^*\check{\mathbf{X}} + \check{\mathbf{X}}A + Q - \check{\mathbf{X}}G\check{\mathbf{X}}$ {Using verified interval arithmetic}
$\mathbf{G} = \mathbf{I}_W^* \mathbf{F} V$
$\mathbf{H} = \mathbf{G}./D$
$\mathbf{L} = -W^* \mathbf{H} \mathbf{I}_V$
$\mathbf{Z} = \mathbf{L}$
for $k = 1, 2, \ldots, k_{max}$ **do**
  Set $\mathbf{Z} = \square(0, \mathbf{Z} \cdot \langle 1, 0.1 \rangle + \langle 0, \texttt{realmin} \rangle)$ {ε-inflation technique}
  Compute $\mathbf{K}$ using Algorithm 2
  if $\mathbf{K} \subset \mathrm{int}(\mathbf{Z})$ {successful inclusion} **then**
    Return $\mathbf{X} = \check{X} + \mathbf{K}$
  end if
  $\mathbf{Z} = \mathbf{K}$
end for
Return **failure** {Maximum number of iterations reached}

Algorithm 2 Computation of an interval matrix $\mathbf{K}$ such that $\mathrm{vec}(\mathbf{K}) = \mathbf{k}_f(\check{x}, R, \mathbf{z}, \mathbf{S})$ encloses $\mathcal{K}_f(\check{x}, R, \mathbf{z}, \mathcal{S})$.

Input $A, G, Q, \check{X}, \mathbf{Z}$
{Additionally, in this subfunction we use $V, W, \mathbf{I}_V, \mathbf{I}_W, \Lambda, D, \mathbf{L}$ which are already computed in Algorithm 1}
$\mathbf{M} = \mathbf{I}_W^* \mathbf{Z} V$
$\mathbf{N} = W(A - G(\check{X} + \mathbf{Z}))\mathbf{I}_W$
$\mathbf{O} = \mathbf{I}_V(A - G(\check{X} + \mathbf{Z}))V$
$\mathbf{P} = (\Lambda - \mathbf{N})^* \mathbf{M} + \mathbf{M}(\Lambda - \mathbf{O})$
$\mathbf{Q} = \mathbf{P}./D$
$\mathbf{U} = W^* \mathbf{Q} \mathbf{I}_V$
$\mathbf{K} = \mathbf{L} + \mathbf{U}$
Return $\mathbf{K}$

Notice the ε-inflation, which is performed by enlarging the computed interval by 10% and adding $\langle 0, \texttt{realmin} \rangle$. Throughout the paper, `realmin` denotes the smallest positive normalized floating point number.

All the operations in Algorithm 1 are matrix–matrix computations requiring $\mathcal{O}(n^3)$ arithmetic operations, so its total cost is $\mathcal{O}(n^3 s)$, where s is the number of steps needed before success.

In the following Sections 3.2–3.4, we describe three modifications that improve the reliability of the Krawczyk algorithm by temporarily neglecting the issue of uniqueness. We shall show later, in Section 3.5, how the uniqueness of the solution inside $\mathbf{X}$ can be recovered *a posteriori*.

3.2. An affine transform enclosure

In this section we describe a technique for reducing the wrapping effect in the modified Krawczyk method, which has already been successfully applied to several matrix equations [9,17]. The main idea is applying the verification algorithm to a modified function $\hat{f}$ obtained from f via an affine transformation; in this way, we reduce the number of interval operations to perform inside the verification procedure.

Assuming again that V, W and Δ defined in (3.7) are nonsingular, we define the function

$$\hat{f}(\hat{x}) := (V^T \otimes W^{-*}) f((V^{-T} \otimes W^*)\hat{x}). \tag{3.9}$$

If $\check{x} = \mathrm{vec}(\check{X})$ is an approximate solution to $f(x) = 0$, then $\hat{\check{x}} := (V^T \otimes W^{-*})\check{x}$ is an approximate solution to $\hat{f}(\hat{x}) = 0$. The Kronecker form of the Fréchet derivative of $\hat{F}(\hat{X})$, matrix formulation of $\hat{f}(\hat{x})$, is given by

$$K_{\hat{F}}(\hat{X}) = (V^T \otimes W^{-*}) K_F(X)(V^{-T} \otimes W^*), \quad X = W^* \hat{X} V^{-1}.$$

Moreover, let $\hat{\mathbf{x}} = \mathrm{vec}(\hat{\mathbf{X}}) := \hat{\check{x}} + \hat{\mathbf{z}}$, where $\hat{\mathbf{z}} = \mathrm{vec}(\hat{\mathbf{Z}})$. A set of slopes for $\hat{f}$ on $\hat{\mathbf{x}}$ can be defined as

$$\hat{\mathcal{S}} := \{S(\hat{f}; \hat{y}, \hat{y}') : \hat{y}, \hat{y}' \in \hat{\mathbf{x}}\}.$$

Defining $y := (V^{-T} \otimes W^*)\hat{y}$, $y' := (V^{-T} \otimes W^*)\hat{y}'$, we have

$$\begin{aligned} S(\hat{f}; \hat{y}, \hat{y}')(\hat{y} - \hat{y}') &= \hat{f}(\hat{y}) - \hat{f}(\hat{y}') \\ &= (V^T \otimes W^{-*})(f(y) - f(y')) \\ &= (V^T \otimes W^{-*}) S(f; y, y')(y - y') \\ &= (V^T \otimes W^{-*}) S(f; y, y')(V^{-T} \otimes W^*)(\hat{y} - \hat{y}'). \end{aligned}$$

Hence

$$S(\hat{f}; \hat{y}, \hat{y}') = (V^T \otimes W^{-*}) S(f; y, y')(V^{-T} \otimes W^*).$$

In particular, if we combine this result with Theorem 3.3, we can take $\hat{\mathbf{S}}$ in the Krawczyk operator $\mathbf{k}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathbf{S}})$ as

$$\hat{\mathbf{S}} := I_n \otimes (W(A - G\check{\mathbf{X}})W^{-1})^* + (V^{-1}(A - G\check{\mathbf{X}})V)^T \otimes I_n. \tag{3.10}$$

where

$$\check{\mathbf{X}} = W^* \hat{\check{\mathbf{X}}} V^{-1}, \qquad \hat{\check{\mathbf{X}}} = \hat{\check{X}} + \hat{\mathbf{Z}},$$

as long as $\check{X}$ is Hermitian.

Observe that

$$I_n \otimes (W(A - G\check{X})W^{-1})^* + (V^{-1}(A - G\check{X})V)^T \otimes I_n \approx I_n \otimes \Lambda^* + \Lambda^T \otimes I_n,$$

so a natural choice for $\hat{R}$ is the diagonal matrix

$$\hat{R} = \Delta^{-1},$$

in which Δ is defined as in (3.8).

Now, we compute an enclosure for $\mathcal{K}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathcal{S}}) := \{-\hat{R}\hat{f}(\hat{\check{x}}) + (I_{n^2} - \hat{R}S)\hat{z}, S \in \hat{\mathcal{S}}, \hat{z} \in \hat{\mathbf{z}}\}$ which can be written as $\mathbf{k}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathbf{S}})$ in which $\hat{\check{x}}$ is an approximate solution for (3.9), $\hat{R}$ is Δ^{-1}, $\hat{\mathcal{S}} = \{S(\hat{f}; \hat{y}, \hat{y}'), \hat{y}, \hat{y}' \in \hat{\mathbf{x}} := (V^T \otimes W^{-*})\check{x} + \hat{\mathbf{z}}\}$, and $\hat{\mathbf{z}} := \mathrm{vec}(\hat{\mathbf{Z}})$. As in Algorithm 1, we also take care that the quantities which are not available exactly are enclosed into computable quantities in interval forms, for instance $\mathbf{I}_V$ and $\mathbf{I}_W$ are interval matrices which are known to contain the exact value of V^{-1} and W^{-1}, appropriately. More details for computing the superset

$$\begin{aligned} \mathbf{k}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathbf{S}}) &= -\hat{R}\hat{f}(\hat{\check{x}}) + (I_{n^2} - \hat{R}\hat{\mathbf{S}})\hat{\mathbf{z}} \\ &= -\Delta^{-1}((V^T \otimes W^{-*}) f(\check{x}) - (\Delta - I_n \otimes (W(A - G\check{\mathbf{X}})W^{-1})^* - (V^{-1}(A - G\check{\mathbf{X}})V)^T \otimes I_n)\hat{\mathbf{z}}), \end{aligned}$$

for $\mathcal{K}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathcal{S}})$, are displayed in Algorithm 4. The complete algorithm is shown in Algorithm 3.

Note that computing $\hat{\mathbf{L}}$ in Algorithm 3 requires fewer dense $n \times n$ interval matrix multiplications than computing $\mathbf{L}$ in Algorithm 1 as well as computing $\hat{\mathbf{U}}$ in Algorithm 4 versus computing $\mathbf{U}$ in Algorithm 2, so the impact of the wrapping effect is reduced. This is the reason why one expects Algorithm 3 to work in more cases than Algorithm 1.

An important observation is that the last transformation $\mathbf{X} = \check{X} + W^* \hat{\mathbf{K}} \mathbf{I}_V$ happens after the Krawczyk verification procedure. So, while the procedure guarantees that only one zero $\hat{x}_s$ of $\hat{f}$ is contained in $W^{-*}\check{X}V + \hat{\mathbf{K}}$, when we return to the

Algorithm 3 Computation of an interval matrix $\mathbf{X}$ containing a solution of CARE (1.1).

1: Compute an approximate stabilizing solution $\check{X}$ of CARE (1.1) using any floating point algorithm
2: Compute approximations V, W, Λ for the eigendecomposition of $A-G\check{X}$ in floating point {For instance, using the MATLAB command `eig`}
3: Compute with floating point arithmetic $D := (D_{ij})$ such that $D_{ij} \approx \bar{\Lambda}_{ii} + \Lambda_{jj}$
4: Compute interval matrices $\mathbf{I}_V$ and $\mathbf{I}_W$ containing V^{-1} and W^{-1}, respectively {For instance, using `verifylss.m` from INTLAB} If this fails, or if D has any zero elements, return **failure**
5: $\check{\mathbf{X}} = \langle \check{X}, 0 \rangle$ {To ensure that operations involving $\check{X}$ are performed in a verified fashion with interval arithmetic}
6: $\mathbf{F} = A^*\check{\mathbf{X}} + Q + \check{\mathbf{X}}(A - G\check{\mathbf{X}})$
7: $\hat{\mathbf{F}} = \mathbf{I}_W^* \mathbf{F} V$
8: $\hat{\mathbf{L}} = -\hat{\mathbf{F}}./D$
9: $\hat{\mathbf{Z}} = \hat{\mathbf{L}}$
10: **for** $k = 1, 2, \ldots, k_{max}$ **do**
11: Set $\hat{\mathbf{Z}} = \square(0, \hat{\mathbf{Z}} \cdot \langle 1, 0.1 \rangle + \langle 0, \texttt{realmin} \rangle$ {ε-inflation technique}
12: Compute $\hat{\mathbf{K}}$ using Algorithm 4 (or Algorithm 5)
13: **if** $\hat{\mathbf{K}} \subset \operatorname{int}(\hat{\mathbf{Z}})$ {successful inclusion} **then**
14: Return $\mathbf{X} = \check{X} + W^* \hat{\mathbf{K}} \mathbf{I}_V$
15: **end if**
16: $\hat{\mathbf{Z}} = \hat{\mathbf{K}}$
17: **end for**
18: Return **failure** {Maximum number of iterations reached}

Algorithm 4 Evaluating $\hat{\mathbf{K}}$ with $\operatorname{vec}(\hat{\mathbf{K}}) = \mathbf{k}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathbf{S}})$ encloses $\mathcal{K}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathcal{S}})$.

1: Input $A, G, Q, \check{X}, \hat{\mathbf{Z}}$
{Additionally, in this sub-function we use $V, W, \mathbf{I}_V, \mathbf{I}_W, \Lambda, D, \hat{\mathbf{L}}$ which are already computed in Algorithm 3}
2: $\hat{\mathbf{M}} = W^* \hat{\mathbf{Z}} \mathbf{I}_V$
3: $\hat{\mathbf{N}} = \mathbf{I}_W^* (A - G(\check{X} + \hat{\mathbf{M}}))^* W^*$
4: $\hat{\mathbf{O}} = \mathbf{I}_V (A - G(\check{X} + \hat{\mathbf{M}}))V$
5: $\hat{\mathbf{P}} = (\Lambda^* - \hat{\mathbf{N}})\hat{\mathbf{M}} + \hat{\mathbf{M}}(\Lambda - \hat{\mathbf{O}})$
6: $\hat{\mathbf{U}} = \hat{\mathbf{P}}./D$
7: $\hat{\mathbf{K}} = \hat{\mathbf{L}} + \hat{\mathbf{U}}$
8: Return $\hat{\mathbf{K}}$

original setting and compute an enclosure for $\mathbf{X} = \check{X} + W^* \hat{\mathbf{K}} \mathbf{I}_V$, other solutions of (1.1) may fall into this enclosure. Hence, Algorithm 3 alone does *not* guarantee that there is a unique solution of (1.1) in $\mathbf{X}$, *nor* that this solution is the stabilizing one. We resolve with this issue in Section 3.5.

Another small improvement introduced in this algorithm is gathering $\check{\mathbf{X}}$ in Line 6 of Algorithm 3, in order to reduce the wrapping effect.

3.3. Verifying a different Riccati equation

Another possible modification to the verification process consists in modifying the equation into one with (possibly) better numerical properties. The idea stems from the following classical formulation of a CARE as an invariant subspace problem.

Lemma 3.4 (*See e.g. [1]*). *The stabilizing solution X_s of* CARE (1.1) *is the only matrix $X_s \in \mathbb{C}^{n\times n}$ such that*

$$H \begin{bmatrix} I_n \\ X_s \end{bmatrix} = \begin{bmatrix} I_n \\ X_s \end{bmatrix} R, \qquad H = \begin{bmatrix} A & -G \\ -Q & -A^* \end{bmatrix} \in \mathbb{C}^{2n\times 2n} \tag{3.11}$$

for some Hurwitz stable matrix R. Moreover, it holds that $R = A - GX_s$.

We use this formulation to relate the solution X_s to the one of a different CARE. The following result is a natural result of the literature on algebraic Riccati equations (see e.g. [1]), and the idea used here is certainly not original, but we prove it explicitly because we do not have a reference with this exact statement.

Lemma 3.5. *Let X_s be the stabilizing solution of (1.1). Suppose that $P \in \mathbb{C}^{2n\times 2n}$ be a nonsingular matrix such that $P^{-1}HP$ has the same structure as H, i.e.,*

$$P^{-1}HP = \begin{bmatrix} A_P & -G_P \\ -Q_P & -A_P^* \end{bmatrix} \in \mathbb{C}^{2n\times 2n}, \tag{3.12}$$

for some matrices $A_P, G_P = G_P^*, Q_P = Q_P^* \in \mathbb{C}^{n\times n}$. Let Y_s be the stabilizing solution of the CARE

$$A_P^*Y + YA_P + Q_P = YG_PY, \tag{3.13}$$

and $U_1, U_2 \in \mathbb{C}^{n\times n}$ be defined by

$$P\begin{bmatrix} I_n \\ Y_s \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}.$$

If U_1 is invertible, then $X_s = U_2U_1^{-1}$.

Proof. We have

$$P^{-1}HP\begin{bmatrix} I_n \\ Y_s \end{bmatrix} = \begin{bmatrix} I_n \\ Y_s \end{bmatrix} R_P,$$

for the Hurwitz stable matrix $R_P = A_P - G_PY_s$. Multiplying both sides by P on the left we get

$$H\begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} R_P,$$

and then multiplying on the right by U_1^{-1}

$$H\begin{bmatrix} I_n \\ U_2U_1^{-1} \end{bmatrix} = \begin{bmatrix} I_n \\ U_2U_1^{-1} \end{bmatrix} U_1R_PU_1^{-1}.$$

Since $U_1R_PU_1^{-1}$ is Hurwitz stable, Lemma 3.4 gives us the thesis. □

The paper [10] contains a convenient strategy to construct a matrix P with a particularly simple form (a permutation matrix with some sign changes) for which all the required assumptions hold and in addition Y_s is bounded. Define for each $k = 1, 2, \ldots, n$

$$S_k := \begin{bmatrix} I_n - E_{kk} & E_{kk} \\ -E_{kk} & I_n - E_{kk} \end{bmatrix} \in \mathbb{C}^{2n\times 2n},$$

where E_{kk} is the matrix which has 1 in position (k, k) and zeros elsewhere; in other words, S_k swaps the entries k and $n + k$ of a vector in $\mathbb{C}^{2n}$, and changes sign to one of them. The matrices S_k are orthogonal and commute with each other.

Theorem 3.6 (*[10, Theorem 3.4]*). *Let $\mathcal{I} = \{i_1, i_2, \ldots, i_k\}$ be a subset of $\{1, 2, \ldots, n\}$, and $P = S_{i_1}S_{i_2}\ldots S_{i_k}$. Then*

(1) *For each choice of $\mathcal{I}$, the matrix $P^{-1}HP$ has the structure (3.12).*
(2) *For each $\tau \geq \sqrt{2}$, one can find $\mathcal{I}$ such that U_1 is nonsingular and Y_s has all its elements bounded in modulus by τ (referring to the definitions of U_1 and Y_s in Lemma 3.5).*

These results suggest an alternative verification strategy:

(1) Compute P satisfying Theorem 3.6.
(2) Form the coefficients A_P, G_P and Q_P, which can be obtained from the entries of H only using permutations and sign changes.
(3) Using one of the various verification methods for CAREs, compute an interval matrix $\mathbf{Y}$ containing the stabilizing solution Y_s.
(4) Compute

$$\begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix} = P\begin{bmatrix} I_n \\ \mathbf{Y} \end{bmatrix},$$

which, again, requires only rearranging the entries and changing their signs, and hence can be done without wrapping effects.
(5) Compute using interval arithmetic a solution $\mathbf{X}$ to the linear system $\mathbf{X}\mathbf{U}_1 = \mathbf{U}_2$.

Then, clearly, $\mathbf{X}$ contains the true stabilizing solution X_s of (1.1). Again, since the interval matrix $\mathbf{X}$ computed in the last step is only a solution enclosure and suffers from wrapping effect, it might be the case that other solutions of the CARE (1.1) are contained in $\mathbf{X}$ in addition to X_s.

The MATLAB toolbox [23] contains algorithms to compute a subset $\mathcal{I}$ (and hence a matrix P) satisfying the conditions of Theorem 3.6, for every $\tau > \sqrt{2}$, in time bounded by $\mathcal{O}(n^3 \log_\tau n)$. The factor $\log_\tau n$ is a worst-case factor only, and in our experience for most matrices fine-tuning the choice of τ does not have a big impact on neither performance nor stability. Here, we always use the method with its default value $\tau = 3$.

With this method, one transforms the problem of verifying (1.1) into the one of verifying (3.13); this latter Riccati equation has a stabilizing solution Y_s whose entries are bounded in modulus by τ, hence one may expect that less cancellation can take place in the algorithms. While there is no formal guarantee that this must happen, in practice, in most cases the eigenvector matrix V_P of $R_P = A_P - G_P Y_s$ has a lower condition number than the one V of $A - GX_s$, as we report in the experiments (see Fig. 5 in the following), and verification of (3.13) is often easier than verification of (1.1). Ultimately, this is only a heuristic approach, though.

Let us analyze the computational complexity of Algorithm 6.

Theorem 3.7. *Algorithm* 6 *requires at most* $\mathcal{O}(n^3 \log_\tau n + n^3 s)$ *floating point operations, where* s *is the number of steps required by the inner verification algorithm in Line* 5 .

Proof. Computing $\check{X}$ in Line 2 requires $\mathcal{O}(n^3)$ operations, using for instance the algorithm mentioned in [8] (based on the ordered Schur form of H and an additional Newton step with the residual computation performed in emulated quadruple-precision arithmetic). Forming P in Theorem 3.6 via the approach explored in [23] costs $\mathcal{O}(n^3 \log_\tau n)$ floating point operations. Computing $\mathbf{Y}$ by using Algorithm 3 has cost $\mathcal{O}(n^3)$ per step (and the same will hold for Algorithm 8 that we will introduce later): the cost for the eigendecomposition and the enclosures $\mathbf{I}_V$ and $\mathbf{I}_W$ is again cubic in n, and all the other matrix–matrix operations (including the Hadamard divisions) in Algorithms 3 and 5 have again cost $\mathcal{O}(n^3)$ at most, as they only involve $n \times n$ matrices. □

3.4. A new superset

According to Theorem 3.2, the computed interval matrix is guaranteed to contain a unique solution if the set $\mathbf{S}$ contains the slopes $S(f; y, y')$ for all $y, y' \in \mathbf{x}$. On the other hand, if we employ an interval matrix containing only the slopes $S(f; \check{x}, y')$ for all $y' \in \mathbf{x}$, existence can be proved, but not uniqueness. Since we have already decided to forgo (for now) uniqueness, it makes sense to let go of it also when choosing the superset $\mathbf{S}$.

A simple modification to our proof of Theorem 3.3 gives a tighter inclusion for the slope superset by reducing the wrapping effect.

Theorem 3.8. *Let* f *be as in* (3.5), $\mathbf{X} \in \mathbb{IC}^{n\times n}$ *be an interval matrix, and* $\check{X} \in \mathbf{X}$ *be Hermitian. Then, the interval matrix*

$$I_n \otimes (A - G\check{X})^* + (A - G\mathbf{X})^T \otimes I_n$$

contains the slopes $S(f, \check{x}, y')$ *for each* $Y' \in \mathbf{X}$ *where* $\check{x} = \mathrm{vec}(\check{X})$ *and* $y' = \mathrm{vec}(Y')$.

Proof. We repeat the proof of Theorem 3.3, with $y = \check{x}$, and replace the term $\mathbf{K}_F(\mathbf{X})$ in (3.6) with the tighter inclusion $(I_n \otimes (A - G\check{X})^* + (A - G\mathbf{X})^T \otimes I_n)$. □

As a consequence of Theorem 3.8, we can replace (3.10) with

$$\hat{\mathbf{S}} = I_n \otimes (W(A - G\check{X})W^{-1})^* + (V^{-1}(A - G\check{\mathbf{X}})V)^T \otimes I_n \tag{3.14}$$

in our modified Krawczyk algorithm applied to $\hat{f}$, and it will still yield an interval matrix containing a (possibly non-unique) solution of (1.1).

Algorithm 5 Evaluating $\hat{\mathbf{K}}$ with $\mathrm{vec}(\hat{\mathbf{K}}) = \mathbf{k}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathbf{S}})$ encloses $\mathcal{K}_{\hat{f}}(\hat{\check{x}}, \hat{R}, \hat{\mathbf{z}}, \hat{\mathcal{S}})$ with a tighter superset that does not guarantee solution uniqueness.

{This algorithm is identical to Algorithm 4, apart from Line 3 which is replaced by the following}

3: $\hat{\mathbf{N}} = \mathbf{I}_W^*(A - G\check{X})^* W^*$

3.5. Verification of uniqueness and stabilizability

As noted before, the modifications to the Krawczyk method introduced here do not ensure that the found interval matrix contains only one solution to (1.1). However, the following result holds.

Theorem 3.9 (*[24, Theorem 23.3]*)**.** *The* CARE (1.1) *has at most one stabilizing solution.*

Algorithm 6 Computation of an interval matrix $\mathbf{X}$ containing a solution of (1.1) using permuted Riccati bases.

1: Input A, G, Q
2: Compute an approximate stabilizing solution $\check{X}$ of (1.1) using any floating point algorithm
3: Compute a matrix P satisfying approximately point 2 of Theorem 3.6 {For instance, using the function `canBasisFromSubspace` in the toolbox [23] on the subspace $\check{U} = \begin{bmatrix} I_n \\ \check{X} \end{bmatrix}$}
4: Compute A_P, G_P, Q_P satisfying (3.12)
5: Compute a verified solution $\mathbf{Y}$ to (3.13) using either Algorithm 3 o r Algorithm 8. If the verification fails, return **failure**
6: Set $\begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix} = P \begin{bmatrix} I_n \\ \mathbf{Y} \end{bmatrix}$
7: Compute an enclosure $\mathbf{X}$ for the solution of the interval system $\mathbf{X}\mathbf{U}_1 = \mathbf{U}_2$ {For instance, using `verifylss` from INTLAB} If this fails, return **failure**
8: Return $\mathbf{X}$

A proof using the facts in [1] can be obtained by considering the eigenvalues of H in (3.11). Let X_s be a stabilizing solution, and let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of $A - GX_s$. Because of the formula (3.11), $\lambda_1, \lambda_2, \ldots, \lambda_n$ are also eigenvalues of H (see [1, Section 2.1.1]). Moreover, the eigenvalues of H have Hamiltonian symmetry, (see [1, Section 1.5]), so there are n more eigenvalues with positive real part. We have identified $2n$ eigenvalues of H, counted with multiplicity, and none of them is purely imaginary; hence H has no purely imaginary eigenvalue and [1, Theorem 2.17] holds.

Theorem 3.9 gives a simple method to check the uniqueness of the solution in $\mathbf{X}$.

Corollary 3.10. *Let $\mathbf{X} \in \mathbb{IC}_{disc}^{n\times n}$ be an interval matrix containing a solution X of the* CARE (1.1)*. If every matrix in $A - G\mathbf{X}$ is Hurwitz stable, then $X = X_s$, the stabilizing solution, and it is the only solution of* (1.1) *inside $\mathbf{X}$.*

To verify stability, we can use the method described in [25], which is summarized in [8, Lemma 2.4]. The resulting method is described in Algorithm 7. In the algorithm, we use the notation $\Re z$ to mean the real part of the complex number z.

Algorithm 7 Verifying the Hurwitz stability of an interval matrix $\mathbf{M}$.

1: Input $\mathbf{M}$
2: Compute approximations V, W, Λ for the eigendecomposition of mid($\mathbf{M}$) in floating point {For instance, using the MATLAB command `eig`}
3: $\mathbf{V} = \langle V, 0 \rangle$
4: $R = \mathrm{mag}(W(\mathbf{M}\mathbf{V} - \mathbf{V}\Lambda))$
5: $S = \mathrm{mag}(I_n - \mathbf{V}W)$
6: $e =$ the $n \times 1$ matrix with $e_{i,1} = 1$ for each i
7: $u = Re$ {This line and the successive ones are performed in floating point arithmetic, with rounding upward}
8: $t = Se$
9: $\mu = \max(u./ - (t - e))$
10: $r = u + \mu t$
11: **if** $(\max(t) < 1$ and $r + \max(\Re(\mathrm{diag}(\Lambda)))e < 0)$ **then**
12: Return **success** {Every matrix $M \in \mathbf{M}$ is Hurwitz stable}
13: **else**
14: Return **failure**
15: **end if**

Notice one subtle point: when we apply Algorithm 7 to $A - G\mathbf{X}$ we recompute V, Λ and W from the eigendecomposition of mid$(A - G\mathbf{X})$; this differs slightly from using the values computed previously, which came from the eigendecomposition of $A - G\check{X}$ (because $\mathbf{X}$ was not available at that point). This choice gives better results in our experiments. The cost for this verification is again $\mathcal{O}(n^3)$ floating point operations.

Hence, if the verification in Algorithm 7 succeeds for the solution enclosure $\mathbf{X}$ returned by either Algorithm 1 or Algorithm 3, then $\mathbf{X}$ contains exactly one solution of (1.1), and it is the stabilizing one.

4. A direct fixed-point method

While the methods described in the previous sections work for many examples of Riccati equations, an essential limitation is that all of them require the closed-loop matrix $A - G\check{X}$ to be diagonalizable. Products with the eigenvector matrix V and its inverse are required along the algorithm, and if these are ill-conditioned then the wrapping effects are more pronounced and the required inclusion $\mathbf{K} \subset \mathrm{int}(\mathbf{Z})$ or $\hat{\mathbf{K}} \subset \mathrm{int}(\hat{\mathbf{Z}})$ is less likely to hold. A striking example of this phenomenon is the first example in the benchmark set [11]. This is a simple 2×2 problem which appears in [11] as nothing more than a "warm-up example", and yet all the verification methods described here (including those from [7,8]) fail.

To solve this issue, we would like to propose a different method for verification. The procedure is based on some ideas which appear in the context of ADI methods [26]. While this method is somehow more primitive and works on a lower number of examples, it does not require that the closed-loop matrix be diagonalizable.

We rewrite the CARE (1.1) as follows. Given any Hermitian $\check{X} \in \mathbb{C}^{n\times n}$, one can write the exact stabilizing solution X_s of the CARE (1.1) as $X_s = \check{X} + Z_s$ for an unknown Hermitian correction matrix Z_s, and rewrite (1.1) as a Riccati equation in Z,

$$\tilde{A}^*Z + Z\tilde{A} + \tilde{Q} = ZGZ, \quad \text{with } \tilde{A} = A - G\check{X},\ \tilde{Q} = A^*\check{X} + \check{X}A + Q - \check{X}G\check{X}. \tag{4.1}$$

The stabilizing solution of this equation is Z_s, since $\tilde{A} - GZ_s = A - GX_s$ is Hurwitz stable. Note that the degree-two coefficient G is unchanged. For any $s \in \mathbb{C}$ such that $\tilde{A} - sI_n$ is nonsingular, (4.1) is equivalent to the fixed point equation

$$Z = (\tilde{A} - sI_n)^{-*}(ZGZ - \tilde{Q} - Z(\tilde{A} + sI_n)).$$

Thus, if we find an interval $\mathbf{Z}$ such that $(\tilde{A} - sI_n)^{-*}(\mathbf{Z}G\mathbf{Z} - \tilde{Q} - \mathbf{Z}(\tilde{A} + sI_n)) \subseteq \mathbf{Z}$, it follows from the Brouwer fixed-point theorem that (4.1) has a solution $Z_* \in \mathbf{Z}$, and that (1.1) has a solution $X_* \in \check{X} + \mathbf{Z}$.

This simple iterative method is effective when $(\tilde{A} - sI_n)^{-*}\mathbf{Z}(\tilde{A} + sI_n)$ does not suffer excessively from wrapping effects, since we can expect $\tilde{Q}$ and the quadratic term $\mathbf{Z}G\mathbf{Z}$ to be small.

Are there any preconditioning transformations that we can make to improve the method? A possibility is applying a change of basis to the whole problem. Let $V \in \mathbb{C}^{n\times n}$ be invertible; we set

$$Z_V = V^*ZV, \qquad A_V = V^{-1}\tilde{A}V, \qquad Q_V = V^*\tilde{Q}V, \qquad G_V = V^{-1}GV^{-*}, \tag{4.2}$$

so that (4.1) is transformed into

$$A_V^*Z_V + Z_VA_V + Q_V = Z_VG_VZ_V.$$

Continuing as above, we obtain the fixed-point equation

$$Z_V = (A_V - sI_n)^{-*}(Z_VG_VZ_V - Q_V - Z_V(A_V + sI_n)). \tag{4.3}$$

If $\tilde{A}$ is diagonalizable, we can set V as its computed approximate eigenvector matrix, as in (3.7). One can see then that the resulting method has several steps in common with the Krawczyk method described in the previous sections. This time, though, we are free to choose the matrix V without the risk of our method turning into a $\mathcal{O}(n^6)$ one, since everything in (4.3) is computable explicitly with standard linear algebra operations.

Some heuristic experimentation led us to the following choices: we take s equal to the approximation of $-\min\{\Re\lambda : \lambda \text{ is an eigenvalue of } \tilde{A}\}$ computed in floating-point arithmetic (motivated by the idea to make $A_V + sI_n$ small and $A_V - sI_n$ large), and V as the orthogonal factor of the (computed) Schur factorization of $\tilde{A} \approx VTV^{-1}$ (motivated by the idea to concentrate most of the "weight" of $V^{-1}\tilde{A}V$ on its diagonal). The matrix $\tilde{A}$ is an approximation of $A - GX_s$, which is Hurwitz stable, so in exact arithmetic we would have $s > 0$ and $A_V - sI_n = V^{-1}(\tilde{A} - sI_n)V$ invertible, since its eigenvalues are $\lambda_i - s$, where λ_i are the eigenvalues of $A - GX_s$, and thus have strictly negative real part. Hence these properties are likely to hold also for its computed approximation $\tilde{A}$.

The resulting algorithm is described in Algorithm 8.

Theorem 4.1. *Algorithm* 8 *has a cost of* $\mathcal{O}(n^3s)$ *arithmetic operations, if the verification succeeds in* s *steps.*

Proof. Again, all the required operations in every step are matrix–matrix operations between $n \times n$ matrices. The Schur decomposition requires $\mathcal{O}(n^3)$ operations as well, in practice [27]. □

Once again, uniqueness is not guaranteed, but it can be deduced *a posteriori* if the verification of the stabilizing property of the computed inclusion interval $\mathbf{X}$ succeeds.

5. Numerical experiments

This section presents numerical experiments to validate the algorithms. We compare four different approaches:

(1) The modified Krawczyk approach described in [7] and in Section 3.1. This corresponds to Algorithm 1. When the algorithm is successful, we check afterwards whether $A - G\mathbf{X}$ is Hurwitz stable using Algorithm 7. We call this approach *Method H* in the following.
(2) The method described in [8] (using the MATLAB implementation `Mn.m` published by its author). The method already includes running Algorithm 7 to check if the computed solution is Hurwitz stable, so we do not need any additional steps. We call this procedure *Method M*.
(3) Algorithm 6, choosing as its subroutine to solve the transformed CARE (3.13) the Krawczyk-based Algorithm 3 and the modified superset trick used in Algorithm 5. This is a combination of all the improvements to Method H described in Section 3. We call this procedure *Method K* (where K stands for Krawczyk). When the algorithm is successful, we check afterwards whether $A - G\mathbf{X}$ is Hurwitz stable using Algorithm 7.

Algorithm 8 Computation of an interval matrix $\mathbf{X}$ containing a solution of (1.1) using a simple fixed-point algorithm.

Input A, G, Q
Compute an approximate stabilizing solution $\check{X}$ of (1.1) using any floating point algorithm
Compute $\tilde{A}$ (in floating point) as in (4.1)
Choose s and V in (4.3) {For instance, $s \approx -\min\{\Re\lambda : \lambda \text{ is an eigenvalue of } \tilde{A}\}$ and V as the (approximate) orthogonal Schur factor of $\tilde{A}$}
Compute an interval matrix $\mathbf{I}_V$ containing V^{-1} {For instance, using `verifylss` from INTLAB}
Compute interval matrices $\mathbf{A}_V, \mathbf{G}_V, \mathbf{Q}_V$ containing A_V, G_V, Q_V, respectively {Replacing $\check{X}$ and V in (4.1) and (4.2) with $\check{\mathbf{X}} = \langle \check{X}, 0\rangle$ and $\mathbf{V} = \langle V, 0\rangle$, respectively}
Compute an interval matrix $\mathbf{I}_s$ containing $(\mathbf{A}_V^* - sI_n)^{-1}$ {For instance, using `verifylss` from INTLAB}
Set $k = 0$ and $\mathbf{Z}_V = -\mathbf{I}_s\mathbf{Q}_V$
for $k = 1, 2, \ldots, k_{max}$ **do**
  Set $\mathbf{Z}_V = \square(0, \mathbf{Z}_V \cdot \langle 1, 0.1\rangle + \langle 0, \texttt{realmin}\rangle)$ {ε−inflation technique}
  Set $\mathbf{Y} = \mathbf{I}_s(-\mathbf{Q}_V - \mathbf{Z}_V(\mathbf{A}_V + sI_n - \mathbf{G}_V\mathbf{Z}_V))$
  if $\mathbf{Y} \subset \operatorname{int}(\mathbf{Z}_V)$ **then**
    Return $\mathbf{X} = \check{X} + \mathbf{I}_V^*\mathbf{Z}_V\mathbf{I}_V$
  end if
end for
Return **failure** {Maximum number of iterations reached}

(4) Algorithm 6 again, but using the fixed-point Algorithm 8 to solve the transformed CARE (3.13). This is a combination of the techniques described in Sections 3.3 and 4. We call this procedure *Method F* (where F stands for fixed-point). When the algorithm is successful, we check afterwards whether $A - G\mathbf{X}$ is Hurwitz stable using Algorithm 7.

The algorithms were tested in MATLAB 2015b with INTLAB v6, using unit round off $u = 2^{-53} \approx 1.1 \times 10^{-16}$, and run on a computer with an Intel core Duo 2.66 GHz processor and 6 GB main memory.

The required stabilizing solutions of CAREs are computed using the method described in [8] (ordered Schur method followed by one step of Newton refinement in simulated quadruple precision).

In order to assess the quality of the enclosures computed in each experiment we use the norm-wise relative error `nre` and the geometric average relative precision `garp`. The first error measure is defined as

$$\texttt{nre}(\mathbf{X}) := \operatorname{mag} \frac{\|\operatorname{rad}(\mathbf{X})\|_F}{\|\mathbf{X}\|_F}.$$

This is the simplest possible bound for the (norm-wise) relative error

$$\frac{\|X_s - \operatorname{mid}(\mathbf{X})\|_F}{\|X_s\|_F}$$

obtained by taking $\operatorname{mid}(\mathbf{X})$ as an approximation of the solution.

Following previous works (see e.g. [21]), we also report a component-wise error indicator `garp` based on the relative precision of an interval, $\operatorname{rp}(\mathbf{x})$, defined as

$$\operatorname{rp}(\mathbf{x}) := \min(\operatorname{relerr}(\mathbf{x}), 1),$$

where relerr is the relative error of the interval $\mathbf{x} = \langle \operatorname{mid}(\mathbf{x}), \operatorname{rad}(\mathbf{x})\rangle$ defined by

$$\operatorname{relerr}(\mathbf{x}) := \begin{cases} \left|\dfrac{\operatorname{rad}(\mathbf{x})}{\operatorname{mid}(\mathbf{x})}\right|, & \text{if } 0 \notin \mathbf{x}, \\ \operatorname{rad}(\mathbf{x}), & \text{otherwise.} \end{cases}$$

We define our residual measure as the geometric average of $\operatorname{rp}(\mathbf{X}_{ij})$

$$\texttt{garp}(\mathbf{X}) := \left(\prod_{i,j=1}^{n} \operatorname{rp}(\mathbf{X}_{ij})\right)^{\frac{1}{n^2}}, \quad \mathbf{X} = (\mathbf{X}_{ij}).$$

The quantity $-\log(\operatorname{rp}(\mathbf{x}))$ can be interpreted as the number of known correct digits of an *exact* value contained in $\mathbf{x}$; so, loosely speaking, $-\log(\texttt{garp}(\mathbf{X}))$ represents the average number of known correct digits [17].

5.1. CAREX Benchmark problems

We ran these algorithms on all the equations from the benchmark set described in [12], which contains experiments taken from the test suite CAREX [11], run with both default and non-default arguments. The results are reported in Tables 1–4, and a visualization of the results is in Fig. 6.

Table 1
Comparison between various proposed methods.

Experiment number in [12]	Problem property		Method H		Method M		Method K		Method F	
	size		nre	k	nre	k	nre	k	nre	k
	cond(V)	cond(V_P)	garp	time	garp	time	garp	time	garp	time
1	2		NaN	*	NaN	–	NaN	*	3.75e−15	2
	7.75e+07	3.17e+07	NaN	*	NaN	*	NaN	*	4.18e−15	2.93e−02
2	2		9.67e−14	1	4.65e−15	–	1.21e−14	1	1.00e−14	3
	1.01e+01	1.15e+00	1.04e−13	2.00e−02	4.97e−15	7.59e−03	1.27e−14	2.28e−02	1.06e−14	2.70e−02
3	4		3.93e−14	1	2.99e−15	–	3.70e−14	1	8.04e−14	6
	9.73e+00	5.11e+00	2.80e−14	2.28e−02	2.12e−15	1.03e−02	5.02e−14	2.90e−02	1.05e−13	4.28e−02
4	8		1.02e−14	1	2.34e−15	–	7.76e−14	1	1.03e−13	15
	1.23e+00	2.18e+00	1.49e−14	1.74e−02	3.42e−15	9.05e−03	1.05e−13	2.44e−02	1.38e−13	5.72e−02
5	9		6.73e−14	1	1.10e−14	–	4.34e−13	1	2.06e−12	42
	7.54e+01	6.52e+01	4.34e−14	1.78e−02	1.05e−14	9.37e−03	7.57e−13	2.45e−02	4.70e−12	1.23e−01
6	30		4.79e−13	2	3.35e−14	–	9.20e−09	2	NaN	*
	1.11e+05	3.48e+03	2.92e−11	5.15e−02	1.87e−12	1.88e−02	1.15e−08	6.64e−02	NaN	*
7	2		2.35e−16	2	2.12e−16	–	5.57e−16	1	7.47e−16	3
	1.62e+00	3.31e+00	5.48e−16	2.18e−02	4.36e−16	7.58e−03	6.17e−16	2.27e−02	8.72e−16	2.69e−02
8	2		3.22e−08	1	1.78e−08	–	3.67e−16	1	8.55e−16	2
	1.01e+00	2.42e+00	4.75e−10	1.68e−02	3.04e−10	1.08e−02	4.83e−16	2.34e−02	1.88e−10	2.52e−02
9	2		6.41e−16	1	1.92e−16	–	3.34e−10	1	2.25e−09	7
	1.22e+00	6.99e+01	2.59e−15	1.69e−02	4.87e−16	8.31e−03	3.36e−10	2.28e−02	2.26e−09	3.64e−02
10	2		NaN	*	5.36e−12	–	3.00e−08	1	1.59e−08	45
	6.80e+01	1.11e+00	NaN	*	5.36e−12	8.49e−03	3.00e−08	2.15e−02	1.59e−08	1.34e−01
11	2		9.65e−16	1	6.29e−16	–	2.45e−15	1	4.53e−15	2
	3.74e+00	1.01e+00	1.04e−15	2.20e−02	6.75e−16	7.71e−03	2.63e−15	2.90e−02	4.87e−15	3.00e−02

The `Experiment number` follows the order used in [12]. Note that this set of problems is designed to be challenging for non-verified CARE solvers in machine arithmetic, so it is not surprising that the verification algorithms cannot deal with all of them with perfect accuracy.

When the algorithms are successful, we report in Tables 1–3 the number k of required iterations of the outer Krawczyk loop. If an algorithm breaks down or does not converge within the maximum number of steps (which we took to be 50 for the iterative Methods H, K and F), then we write a star in the corresponding column. Method M is not iterative, therefore for it we put − in the column containing the number of iterations.

The size of the problem (value of n) and the total time (in seconds, including the time required to verify the stabilizing property) taken on our test machine are reported, as well as the norm-2 condition number of V (used by Methods H, M and K) and the same quantity for the eigenvector matrix V_P of the closed-loop matrix $A_P - G_P Y_s$ used in the two Methods K and F. All these details are given in Tables 1–3 in the column named `Problemproperty`.

Table 4 reports the result of checking the stabilization property; a plus sign means that the property is verified, a minus sign means failure to verify the property, and a star means that the algorithm had already failed to compute an inclusion interval. As one can see, there is only a very limited number of cases in which the stabilization procedure fails.

Remarks are in order on some of the problems.

Experiment 1: This is an example of the phenomenon described in the beginning of Section 4: the closed-loop matrix $A - GX_s$ is not diagonalizable. The coefficient matrices for this example are

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, G = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad Q = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}.$$

The exact value of the closed loop matrix for the original and transformed equations are respectively

$$A - GX_s = \begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix} \quad \text{and} \quad A_P - G_P Y_s = \begin{bmatrix} -2/3 & 1/3 \\ -1/3 & -4/3 \end{bmatrix},$$

Table 2
Comparison between various proposed methods.

Experiment number in [12]	Problem property		Method H		Method M		Method K		Method F	
	size		nre	k	nre	k	nre	k	nre	k
	cond(V)	cond(V_P)	garp	time	garp	time	garp	time	garp	time
12	2		7.96e−16	1	3.22e−16	–	1.53e−15	1	3.90e−11	2
	1.42e+03	1.01e+00	8.94e−16	2.20e−02	3.74e−16	7.50e−03	2.29e−15	2.82e−02	6.56e−11	2.95e−02
13	2		9.41e−09	1	2.23e−09	–	6.99e−16	1	NaN	*
	2.42e+00	1.01e+00	3.01e−10	1.69e−02	7.20e−11	8.29e−03	1.15e−15	2.28e−02	NaN	*
14	2		1.65e−15	1	2.69e−16	–	9.49e−16	1	4.37e−15	3
	1.01e+00	1.01e+00	1.92e−15	1.65e−02	3.14e−16	7.24e−03	1.09e−15	2.29e−02	5.09e−15	2.71e−02
15	2		4.72e−11	1	3.36e−12	–	2.92e−15	1	NaN	*
	1.01e+00	1.29e+00	4.72e−11	1.67e−02	3.36e−12	8.09e−03	2.39e−15	2.27e−02	NaN	*
16	2		NaN	*	5.87e−10	–	9.38e−12	1	NaN	*
	1.00e+00	1.29e+00	NaN	*	5.87e−10	8.07e−03	5.63e−12	2.15e−02	NaN	*
17	2		2.42e−15	1	2.23e−16	–	4.80e−15	1	1.25e−14	5
	1.01e+00	2.62e+00	2.69e−15	2.23e−02	2.23e−16	9.29e−03	4.98e−15	2.83e−02	1.35e−14	3.92e−02
18	2		NaN	*	NaN	–	NaN	*	NaN	*
	1.01e+00	2.62e+00	NaN	*	NaN	*	NaN	*	NaN	*
19	3		3.53e−15	1	2.73e−16	–	3.67e−15	1	6.64e−15	3
	1.01e+00	1.01e+00	1.20e−14	1.98e−02	8.76e−16	7.88e−03	1.19e−14	2.29e−02	2.16e−14	2.72e−02
20	3		9.95e−05	3	3.87e−05	–	4.77e−15	1	1.73e−14	3
	1.01e+00	1.00e+00	1.18e−04	2.78e−02	4.53e−05	8.36e−03	6.08e−12	2.30e−02	2.08e−14	2.70e−02
21	4		1.29e−14	1	4.76e−15	–	1.26e−13	1	3.81e−13	11
	9.01e+00	3.58e+00	1.73e−14	2.25e−02	6.41e−15	8.79e−03	1.39e−13	2.92e−02	4.32e−13	5.84e−02
22	4		7.28e−05	2	3.52e−06	–	1.70e−04	1	NaN	*
	1.22e+01	5.94e+00	4.81e−06	3.02e−02	2.96e−07	1.04e−02	8.34e−06	2.90e−02	NaN	*

both with a double (defective) eigenvalue in -1. The approximation $\check{X}$ computed with the Schur method satisfies $\|X_s - \check{X}\| = 1.68e-15$. The matrix $A - G\check{X}$ is diagonalizable with two very close eigenvalues. Hence, the computed condition numbers of V and V_P are both large, and the first three algorithms, which are based on the diagonalization of an approximation of $A - GX_s$, fail. On the other hand, the fixed-point algorithm does not encounter any difficulty and returns a tight interval $\mathbf{X}$ containing the stabilizing solution. The condition number of the eigenvector matrix of mid$(A - G\mathbf{X})$ is 7.75e7, but the verification with Algorithm 7 succeeds nevertheless.

Experiments 30 and 31: In Method F for problem 30 and Method K for problem 31, we report termination in a finite number of iterations, but NaN for the error. In these problems, the verification algorithm succeeds for the Riccati equation (3.13), but the resulting interval $\mathbf{Y}$ cannot be converted into a solution interval $\mathbf{X}$ for (1.1) using Lemma 3.5, because the interval matrix $\mathbf{U}_1$ computed as described in Section 3.3 contains singular matrices, hence the solution set $\mathbf{X}$ is unbounded. So the method fails to produce a solution enclosure for (1.1).

Another interesting observation is that Method K needs only one iteration in all experiments when it works apart from one case (Experiment 6), i.e., the crucial relation (3.3) is already fulfilled for $k = 1$ in all the other cases. So, while technically it is an iterative algorithm, it seems that Method K can be safely used with a very small $k_{\max}$.

When they are successful, Methods H and K are comparable with respect to execution time as well as with respect to the quality of the enclosure. However, there are cases in which Method H is not successful, and this comprises cases with small dimensions (e.g., 2 in Example 10) as well as cases with large dimensions (e.g., 397 in Example 27).

Method M is significantly faster than the other algorithms. We remark, though, that MATLAB, being an interpreted language, is often not reliable in evaluating computational times. In particular, INTLAB is implemented entirely in MATLAB code, and its running time does not always match the theoretical complexity, especially when dealing with small matrices. For Methods K and F, which rely on Algorithm 6, another consideration is that the computation of the matrix P using the toolbox [23] requires in its default implementation a tight double `for` loop on the matrix entries. MATLAB executes loops of this kind much more slowly than operations on full matrices; hence comparing running times may show a larger discrepancy than the actual difference in performance between the algorithms.

Table 3
Comparison between various proposed methods.

Experiment number in [12]	Problem property		Method H		Method M		Method K		Method F	
	size		nre	k	nre	k	nre	k	nre	k
	cond(V)	cond(V_P)	garp	time	garp	time	garp	time	garp	time
23	4		3.25e−14	1	1.78e−15	–	3.43e−14	1	1.30e−13	11
	1.43e+01	1.77e+00	3.00e−14	2.23e−02	1.71e−15	7.96e−03	4.61e−14	2.92e−02	1.73e−13	5.84e−02
24	4		NaN	*	NaN	–	NaN	*	NaN	*
	1.74e+00	1.74e+00	NaN	*	NaN	*	NaN	*	NaN	*
25	77		4.08e−12	1	3.66e−13	–	4.70e−11	1	3.16e−10	12
	4.98e+01	1.87e+01	3.50e−11	2.11e−01	3.14e−12	1.13e−01	2.64e−10	3.03e−01	1.63e−09	4.34e−01
26	237		4.75e−11	1	4.35e−12	–	2.27e−09	1	1.26e−08	17
	2.41e+02	8.93e+01	8.89e−10	3.05e+00	8.21e−11	1.53e+00	1.41e−08	4.56e+00	7.09e−08	6.79e+00
27	397		NaN	*	6.71e−12	–	8.73e−09	1	6.69e−08	19
	1.31e+02	4.84e+01	NaN	*	2.27e−10	5.26e+00	5.50e−08	1.82e+01	3.87e−07	2.87e+01
28	8		4.35e−15	1	1.67e−15	–	4.35e−15	1	1.30e−14	4
	1.01e+00	1.01e+00	8.95e−15	1.74e−02	3.44e−15	7.72e−03	8.95e−15	2.37e−02	2.66e−14	3.04e−02
29	64		4.12e−13	1	4.99e−14	–	4.12e−13	1	7.12e−13	4
	1.01e+00	1.01e+00	1.97e−07	4.56e−02	2.38e−08	3.63e−02	1.97e−07	7.01e−02	3.40e−07	8.96e−02
30	21		NaN	*	NaN	–	3.90e−04	1	NaN	38
	2.42e+09	2.78e+00	NaN	*	NaN	*	3.79e−04	4.34e−02	NaN	2.08e−01
31	21		NaN	*	NaN	–	NaN	1	NaN	*
	2.42e+09	2.88e+02	NaN	*	NaN	*	NaN	5.06e−02	NaN	*
32	100		6.57e−12	1	1.13e−12	–	6.57e−12	1	NaN	*
	1.01e+00	1.01e+00	2.27e−11	1.52e−01	3.90e−12	1.36e−01	2.27e−11	2.33e−01	NaN	*
33	60		2.04e−14	1	2.67e−13	–	2.77e−10	1	NaN	*
	1.91e+01	1.55e+01	4.67e−14	1.12e−01	6.10e−13	4.97e−02	4.17e−10	1.58e−01	NaN	*

Methods K and M are the most reliable, and fail only on very ill-conditioned examples. Interestingly, the errors obtained by the two approaches differ by orders of magnitude on several problems, in both directions; there are also examples in which either one fails while the other succeeds. So there is no clear winner among the two.

Method F has the largest number of failures. Despite that, it is useful in special cases (such as in Experiment 1) in which the other algorithms have difficulties, particularly when the closed-loop matrix is not diagonalizable.

In many of the examples the performance of the methods based on diagonalizing the closed-loop matrix is (loosely) related to the condition number of V (or V_P, when it is used). To visualize this relationship, we show in Figs. 1–4 scatter plots of the obtained accuracy vs. the value of this condition number in the various examples. When the magnitude of cond(V) is moderate, cond(V_P) has typically the same order of magnitude, but in some cases when cond(V) is large cond(V_P) seems to be considerably lower, as shown in Fig. 5. There is only one case in which cond(V_P) is considerably larger than cond(V), that is, Experiment 9 (1.22 vs. 69.8). This shows experimentally that switching from the formulation (1.1) to (3.13) is beneficial.

5.2. Experiments with varying sizes

In view of the fact that the three Methods H, K and F are iterative taking an unspecified number of steps, and that the last two require a factorization which may require $\mathcal{O}(n^3 \log_\tau n)$ in the worst case, when n is the size of X in (1.1), the reader may wonder how the time taken by the various algorithm scales with the dimension n in practice. We have tested all algorithms on [11, Problem 15], which is a problem designed explicitly to check how Riccati solvers scale with the dimension of the equation. We have generated the test problem in 30 different sizes equally distributed in logarithmic scale between 10 and 1000, and we have tested the four algorithms on these examples. The resulting CPU times are reported in Fig. 7.

Overall, the results show that all methods scale essentially with $\mathcal{O}(n^3)$, and in particular that Methods K and F stay within a moderate factor of the time taken by Method M. In the two largest experiments $n = 853$, $n = 1000$, Method K is the only one to succeed: Method M fails, while Method F delivers a solution enclosure for which the stabilizing property cannot be

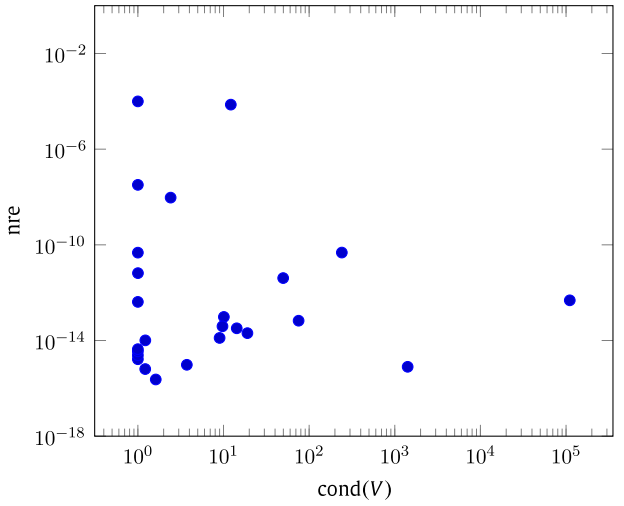

Fig. 1. nre of Method H vs. cond(V).

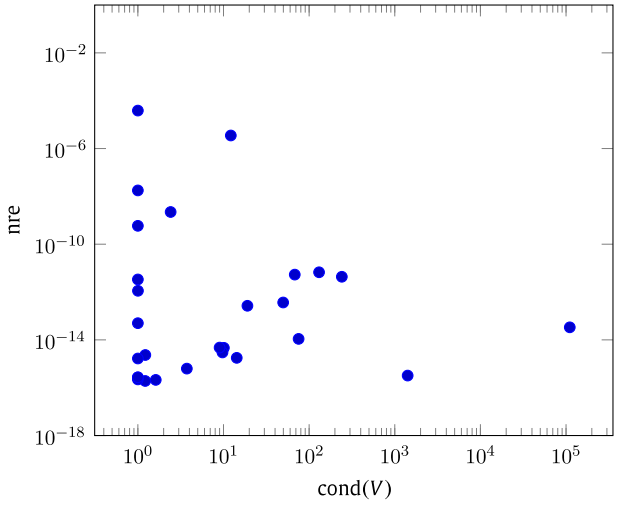

Fig. 2. nre of Method M vs. cond(V).

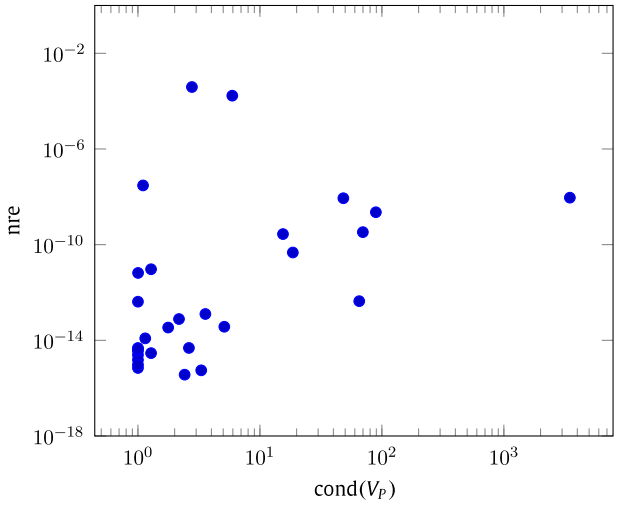

Fig. 3. nre of Method K vs. cond(V_P).

Table 4
Results for stabilizing property in all methods.

Experiment number	Method H	Method M	Method K	Method F
1	*	*	*	+
2	+	+	+	+
3	+	+	+	+
4	+	+	+	+
5	+	+	+	+
6	+	+	+	*
7	+	+	+	+
8	+	+	+	+
9	+	+	+	+
10	*	+	−	−
11	+	+	+	+
12	+	+	+	+
13	+	+	+	*
14	+	+	+	+
15	+	+	+	*
16	*	+	+	*
17	+	+	+	+
18	*	*	*	*
19	+	+	+	+
20	+	+	+	+
21	+	+	+	+
22	+	−	+	*
23	+	+	+	+
24	*	*	*	*
25	+	+	+	+
26	+	+	+	+
27	*	+	+	+
28	+	+	+	+
29	+	+	+	+
30	*	*	−	−
31	*	*	−	*
32	+	+	+	*
33	+	+	+	*

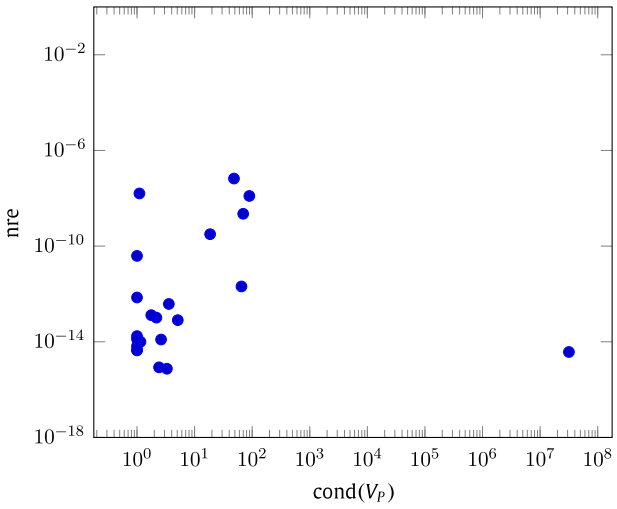

Fig. 4. nre of Method F vs. cond(V_P).

proved. Method H fails for each $n \geq 204$. Verification of the stabilizing property succeeds in all other cases apart from the two mentioned above for Method F.

The MATLAB code used for the experiments is available online on https://bitbucket.org/fph/verifiedriccati.

6. Summary and outlook

We have introduced several improvements to the method in [7], borrowing ideas from both the verification methods and the matrix equations literature. The resulting method has been tested on several standard benchmark experiments, and is

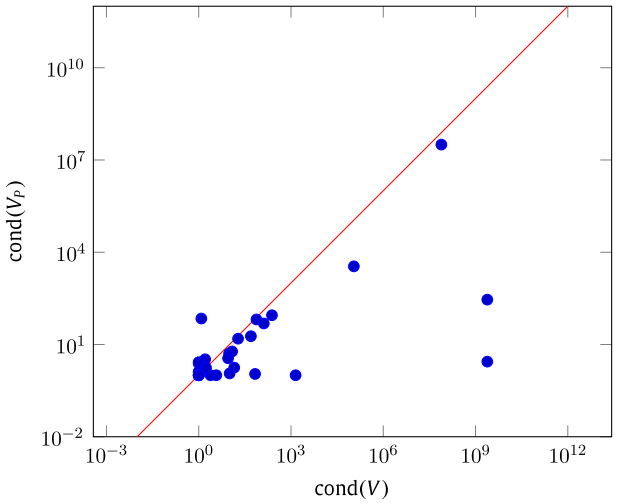

Fig. 5. cond(V_P) vs. cond(V). Most of the points lie below the axes bisector (drawn in red), which means that the condition number of V_P is generally lower than the one of V.

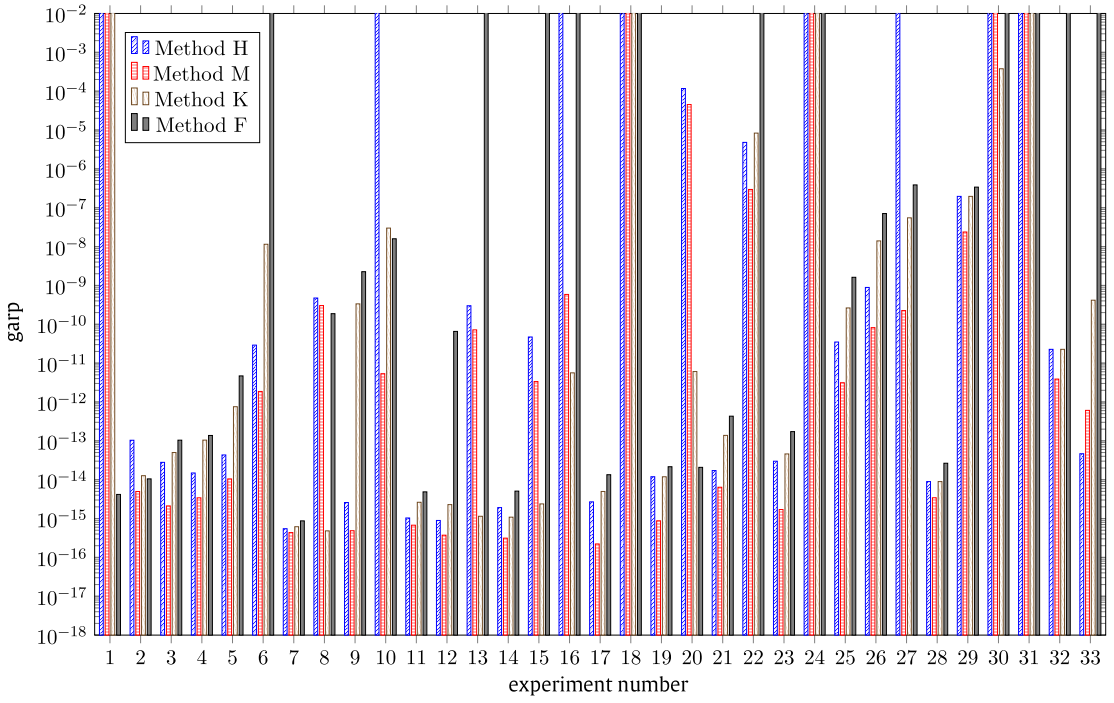

Fig. 6. Values of garp for each experiment number. A full bar means that the method failed to compute an enclosure.

competitive with the one introduced in [8], returning a smaller solution enclosure in several of the experiments. Moreover, the new fixed-point method described in Section 4 is a useful addition to the battery of existing verification methods; it is especially useful in the cases in which the closed-loop matrix is not diagonalizable.

There is no single algorithm that beats all the others on all the benchmark problems; hence it is important to have several methods available, each with its strengths and drawbacks. Overall, all but two of the problems in this challenging set of experiments could be verified with success.

A number of open problems remain: first of all reducing to zero the number of remaining failures in the methods. Of particular interest would be a method more effective than Method F that does not rely on the closed loop matrix being diagonalizable. Other possible research lines are applying these approaches to discrete-time Riccati equations (*DARE*) or more generally to non-symmetric algebraic Riccati equations (*NARE*).

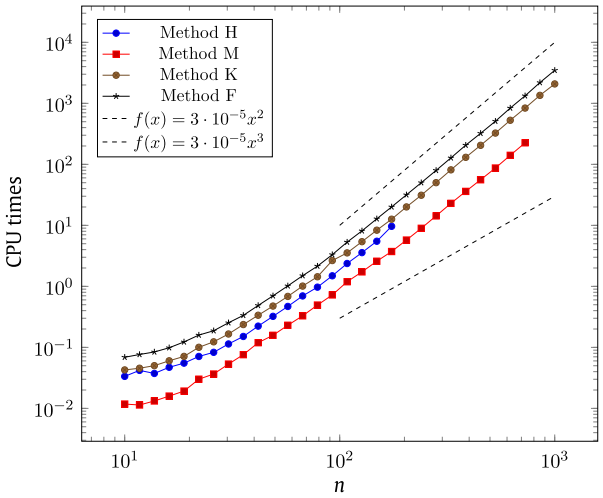

Fig. 7. CPU times for verification on a scaled version of Experiment 15 vs. dimension n.

Acknowledgments

The authors thank Prof. Dr. Wolfram Luther for providing them the technical report related to the Ref. [4] and also the paper [5]. They are also thankful to the anonymous referees for their remarks, which helped us improving the exposition of these results.

T. Haqiri acknowledges the support by the Ministry of Science, Research and Technology of the Islamic Republic of Iran (no. 42/1/301981) for her abroad research scholarship.

F. Poloni acknowledges the support of INDAM (Istituto Nazionale di Alta Matematica) and of the PRA 2014 project "Mathematical models for complex networks and systems" of the university of Pisa.

References

[1] Dario A. Bini, Bruno Iannazzo, Beatrice Meini, Numerical Solution of Algebraic Riccati Equations, in: Fundamentals of Algorithms, vol. 9, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2012.
[2] Peter Lancaster, Leiba Rodman, Algebraic Riccati Equations, Oxford Science Publications. The Clarendon Press, Oxford University Press, New York, 1995.
[3] Siegfried M. Rump, Verification methods: rigorous results using floating-point arithmetic, Acta Numer. 19 (2010) 287–449. http://dx.doi.org/10.1017/S096249291000005X.
[4] W. Luther, W. Otten, H. Traczinski, Verified Calculation of Solutions of Continuous and Discrete Time Algebraic Riccati Equation, in: Schriftenreihe des Fachbereichs Mathematik, Number 422, Universität Duisburg, Fachbereich Mathematik, 1998.
[5] Wolfram Luther, Werner Otten, Verified calculation of the solution of algebraic Riccati equation, in: Developments in Reliable Computing, Budapest, 1998, Kluwer Acad. Publ., Dordrecht, 1999, pp. 105–118.
[6] K. Yano, M. Koga, Verified numerical computation in lq control problem, Trans. Soc. Instrum. Control Eng. 45 (2011) 261–267.
[7] Behnam Hashemi, Verified computation of symmetric solutions to continuous-time algebraic Riccati matrix equations, in: SCAN, 15'th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Verified Numerical Computations, Russian Academy of Sciences, 2012, pp. 54–56. With accompanying slides available online. URL: http://conf.nsc.ru/files/conferences/scan2012/139586/Hashemi-scan2012.pdf.
[8] Shinya Miyajima, Fast verified computation for solutions of continuous-time algebraic Riccati equations, Jpn. J. Ind. Appl. Math. 32 (2) (2015) 529–544. http://dx.doi.org/10.1007/s13160-015-0178-4.
[9] Andreas Frommer, Behnam Hashemi, Verified computation of square roots of a matrix, SIAM J. Matrix Anal. Appl. 31 (3) (2009) 1279–1302. http://dx.doi.org/10.1137/090757058.
[10] Volker Mehrmann, Federico Poloni, Doubling algorithms with permuted Lagrangian graph bases, SIAM J. Matrix Anal. Appl. 33 (3) (2012) 780–805. http://dx.doi.org/10.1137/110850773.
[11] P. Benner, A. Laub, V. Mehrmann, A collection of benchmark examples for the numerical solution of algebraic Riccati equations I: the continuous-time case. Technical Report SPC 95-22, Forschergruppe 'Scientific Parallel Computing', Fakultät für Mathematik, TU Chemnitz-Zwickau, 1995, Version dated February 28, 1996.
[12] Delin Chu, Xinmin Liu, Volker Mehrmann, A numerical method for computing the Hamiltonian Schur form, Numer. Math. 105 (3) (2007) 375–412. http://dx.doi.org/10.1007/s00211-006-0043-0.
[13] R.B. Kearfott, M.T. Nakao, A. Neumaier, S.M. Rump, S.P. Shary, P.V. Hentenryck, Standardized notation in interval analysis, in: Proc. XIII Baikal International School-seminar — Optimization Methods and Their Applications, Vol. 4, 2005, pp. 106–113.
[14] Ramon E. Moore, R. Baker Kearfott, Michael J. Cloud, Introduction to Interval Analysis, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009, http://dx.doi.org/10.1137/1.9780898717716.
[15] Roger A. Horn, Charles R. Johnson, Topics in Matrix Analysis, Cambridge University Press, Cambridge, 1994, Corrected reprint of the 1991 original.
[16] Andreas Frommer, Proving conjectures by use of interval arithmetic, in: Perspectives on Enclosure Methods, Karlsruhe, 2000, Springer, Vienna, 2001, pp. 1–13.
[17] Andreas Frommer, Behnam Hashemi, Thomas Sablik, Computing enclosures for the inverse square root and the sign function of a matrix, Linear Algebra Appl. 456 (2014) 199–213. http://dx.doi.org/10.1016/j.laa.2013.11.047.
[18] R. Krawczyk, Newton-algorithms for evaluation of roots with error bounds, Computing (4) 3, 187–201 http://dx.doi.org/10.1007/BF02234767.
[19] Nicholas J. Higham, Theory and computation, in: Functions of Matrices, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2008, http://dx.doi.org/10.1137/1.9780898717778.

[20] R.H. Bartels, G.W. Stewart, Solution of the matrix equation $AX + XB = C$ [F4], Commun. ACM 15 (9) (1972) 820–826. http://dx.doi.org/10.1145/361573.361582, URL: http://doi.acm.org/10.1145/361573.361582.
[21] Andreas Frommer, Behnam Hashemi, Verified error bounds for solutions of Sylvester matrix equations, Linear Algebra Appl. 436 (2) (2012) 405–420. http://dx.doi.org/10.1016/j.laa.2010.12.002.
[22] Behnam Hashemi, Mehdi Dehghan, Efficient computation of enclosures for the exact solvents of a quadratic matrix equation, Electron. J. Linear Algebra 20 (2010) 519–536.
[23] Federico Poloni, PGDoubling — a MATLAB package to solve algebraic Riccati equations and optimal control problems using permuted graph bases, 2012. URL: https://bitbucket.org/fph/pgdoubling.
[24] R.W. Brockett, Finite Dimensional Linear Systems, in: Series in Decision and Control, Wiley, 1970.
[25] Shinya Miyajima, Fast enclosure for all eigenvalues and invariant subspaces in generalized eigenvalue problems, SIAM J. Matrix Anal. Appl. 35 (3) (2014) 1205–1225. http://dx.doi.org/10.1137/140953150.
[26] Eugene L. Wachspress, Iterative solution of the Lyapunov matrix equation, Appl. Math. Lett. 1 (1) (1988) 87–90. http://dx.doi.org/10.1016/0893-9659(88)90183-8.
[27] Gene H. Golub, Charles F. Van Loan, Matrix Computations, fourth ed., in: Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 2013.