SIAM J. OPTIM.
Vol. 17, No. 1, pp. 1–36
© 2006 Society for Industrial and Applied Mathematics

SOLVING MATRIX INEQUALITIES WHOSE UNKNOWNS ARE MATRICES*

JUAN F. CAMINO†, J. WILLIAM HELTON‡, AND ROBERT E. SKELTON§

Abstract. This paper provides algorithms for numerical solution of convex matrix inequalities in which the variables naturally appear as matrices. This includes, for instance, many systems and control problems. To use these algorithms, no knowledge of linear matrix inequalities is required. However, as tools, they preserve many advantages of the linear matrix inequality framework. Our method has two components: (1) a numerical algorithm that solves a large class of matrix optimization problems and (2) a symbolic "convexity checker" that automatically provides a region which, if convex, guarantees that the solution from (1) is a global optimum on that region. The algorithms are partly numerical and partly symbolic and since they aim at exploiting the matrix structure of the unknowns, the symbolic part requires the development of new computer techniques for treating noncommutative algebra.

Key words. matrix inequalities, convex optimization, semidefinite programming, noncommutative algebra, computer algebra

AMS subject classifications. 90C25, 90C22, 15A42, 15A45, 93A99

DOI. 10.1137/040613718

1. The basic idea.

Since the early 1990s, matrix inequalities (MIs) have become very important in engineering, particularly in control theory. If one has the ability to convert the MIs arising in a particular problem to a linear matrix inequality (LMI), then the problem can be solved up to substantial size. The wide acceptance of LMIs stems from the following advantages:

1. If a control problem is posed as an LMI, then any local solution is a global optimum.
2. Efficient numerical LMI solvers are readily available.
3. Once a control problem is posed as an LMI, adding constraints in the form of LMIs results in a LMI problem.

On the other hand, the LMI framework has the following disadvantages:

1. There is no systematic way to produce LMIs for general classes of problems.
2. There is no way of knowing whether it is possible to reduce a system problem to an LMI without actually doing it.
3. The user must possess the knowledge of manipulating LMIs, which takes considerable training. Indeed, if one does not have the ability to deal with LMIs, then it is not clear what one should do.
4. Transformations via Schur complements can lead to a large LMI representation.

*Received by the editors August 20, 2004; accepted for publication (in revised form) September 19, 2005; published electronically April 21, 2006.
http://www.siam.org/journals/siopt/17-1/61371.html

†Department of Computational Mechanics, School of Mechanical Engineering, State University of Campinas, 13083-970, Campinas, SP, Brazil (camino@fem.unicamp.br). This author was partly supported by CAPES/Brazil and by NSF grants DMS 0100576 and 0400794.

‡Department of Mathematics, University of California at San Diego, La Jolla, CA 92093-0112 (helton@math.ucsd.edu). This author was partly supported by NSF grants DMS 0100576 and 0400794 and the Ford Motor Company.

§Department of Mechanical and Aerospace Engineering, University of California at San Diego, La Jolla, CA 92093-0411 (bobskelton@ucsd.edu). This author was partly supported by DARPA.

1.1. Our method. The main objective for this paper is to provide a method for solving MIs that possesses similar advantages to the LMI framework but without its main disadvantages. Our method has two components:

1. a numerical algorithm, called NCSDP, that solves a large class of matrix optimization problems;
2. a symbolic "convexity checker" that automatically provides a region $\mathcal{G}$. If $\mathcal{G}$ is convex, then the solution from (1) is a global optimum on $\mathcal{G}$. Also, convexity ensures good numerical behavior of NCSDP on $\mathcal{G}$.

1.2. The convexity region algorithm. The symbolic convexity region algorithm receives as input a function $F(x)$ and gives as output a family of inequalities that determine a region $\mathcal{G}$ of x on which $F(x)$ is "matrix convex." Often, we just refer to matrix convexity as convexity and it is defined precisely in section 4.5. This algorithm produces sufficient conditions, which with some very weak hypotheses are necessary conditions for convexity. A concern is that the output might produce a "region of convexity $\mathcal{G}$" with several connected components, in which case the user must select one of them. (See section 3.1 for an example.)

1.3. The numerical solver for matrix inequalities. Our NCSDP solver can be used to solve optimization problems involving matrix inequalities. It is designed for situations where there are only a few unknown matrices and it attempts with symbolic manipulation (as well as numerics) to use the matrix structure to advantage. The solver has very reliable behavior in convex situations. The novel features of our algorithm that allow us to view the matrices as unknowns, rather than the entries of these matrices as unknowns, are discussed in sections 4.4, 7.2, and 8.

1.4. Combining the tools. Putting together the convexity checker and the NCSDP solver, we have a set of tools to solve many engineering problems that can be posed as matrix inequalities with matrix unknowns. Section 3 gives an example of these tools. Our method is effective on problems with few unknowns, but we reiterate that we can take each unknown to be a matrix. This is not a serious restriction for many system problems (e.g., most of the classics [23]).

1.5. LMI analogues. In some sense, there is a parallel between the conventional LMI approach and our approach. In the former, one needs to be able to convert the optimization problem over matrix functions into an equivalent LMI problem, so that some available LMI solver can be used. In our approach, the convexity checker provides a region $\mathcal{G}$ which, if convex, guarantees reliable behavior of our NCSDP solver and that a solution is a global optimum on $\mathcal{G}$.

1.6. Matrix unknowns. The advantage of dealing with matrices as single letters is that one letter z can stand for a matrix Z with n^2 commuting variables. In typical engineering situations, most problems have few matrix unknowns, often two or three, and few (not exceedingly complicated) constraints (usually fewer than 10). This contrasts with treating matrices in terms of their entries where one often has several thousand variables. A disadvantage is that matrix multiplication is not commutative and so we must develop computer tools for performing algebraic operations on noncommuting variables. The major focus of this research is how to use the matrix structure of the unknowns to advantage, and this will come out as the article unfolds. Our algorithms, including the NCSDP solver, combine symbolic and numerical manipulations and lead to several very natural open questions.

1.7. Software availability. The user interface of our NCSDP code is not polished and we do not yet distribute NCSDP. However, the convexity checker algorithm is well documented and available through NCAlgebra, a noncommutative algebra package that runs under Mathematica. This package provides a large number of useful commands and functions for symbolic computation. It can be downloaded from http://math.ucsd.edu/∼ncalg.

2. Nomenclature. We use uppercase letters (e.g., X) for matrices and lowercase letters (e.g., x) for symbolic variables. The notation $\mathcal{Q}, \mathcal{H}$ stands for the symbolic gradient and Hessian maps, and the notation $\mathbb{Q}, \mathbb{H}$ is used to indicate we have substituted matrices of compatible size for the symbolic variables in $\mathcal{Q}, \mathcal{H}$. The n-dimensional Euclidean space is denoted by $\mathbb{R}^n$. The space of $n \times m$ real matrices is denoted by $\mathbb{R}^{n\times m}$. The space of $n \times n$ symmetric matrices with real entries is denoted by $\mathbb{S}^n$. Let $(\mathbb{S}^n)^g$ stand for the direct product $\mathbb{S}^n \times \mathbb{S}^n \times \cdots \times \mathbb{S}^n$ of order g. The expression $A \geq B$ ($A > B$) means that $A - B$ is a positive semidefinite (positive definite) matrix. The associated spaces are respectively denoted by $\mathbb{S}_+$ and $\mathbb{S}_{++}$. The usual Kronecker product of two matrices A and B is denoted by $A \otimes B$ and the trace of A is $\mathrm{Tr}\,\{A\}$. To define the vec operation, let us associate the vector $\mathrm{vec}(X) \in \mathbb{R}^{nm}$ with each matrix $X \in \mathbb{R}^{n\times m}$ by listing the entries of the columns, column by column, that is, $\mathrm{vec}(X) = [X_{11}, X_{21}, \ldots, X_{n1}, X_{12}, \ldots, X_{n2}, \ldots, X_{1m}, \ldots, X_{nm}]^T$.

3. Introducing our approach by an example. Suppose one is given two matrices[1] A and S, where S is symmetric, and one needs to solve the following problem:

$$\max\{\mathrm{Tr}\,\{X\} \,:\, (X, Y, A, S) \in \mathrm{closure}(\mathcal{S})\}, \tag{P1}$$

where the domain $\mathcal{S}$ is given by

$$\mathcal{S} = \left\{(X, Y, A, S) \in \mathcal{V} \,:\, F(X, Y, A, S) < 0, \quad X^2 < I, \quad Y > 0, \quad Y^2 < I\right\}$$

with $\mathcal{V} = \mathbb{S}^n \times \mathbb{S}^n \times \mathbb{R}^{n\times n} \times \mathbb{S}^n$ and

$$\begin{aligned} F(X, Y, A, S) := &-AX(XA^TY^{-1}AX - Y)^{-1}XA^T \\ &- (Y^{-1}(XA^TY^{-1}AXY^{-1} - Y)Y^{-1})^{-1} - AX(Y^{-1}(XA^TY^{-1}AX - Y))^{-1} \\ &- ((XA^TY^{-1}AX - Y)Y^{-1})^{-1}XA^T + XA^TY^{-1}AX - S. \end{aligned}$$

To solve this problem, we apply our two-step methodology:

1. Determine a domain $\mathcal{G}$ on which the above problem is convex.
2. Solve numerically the optimization problem on $\mathcal{G}$ using NCSDP.

3.1. Step 1. Determining a region of convexity in problem (P1). This step is purely symbolic and we do not use the particular numerical values of A and S given in (3.1). We describe this step using standard TEX notation rather than displaying actual computer runs.

The problem (P1) is to maximize $\mathrm{Tr}\,\{X\}$ over the domain

$$\mathcal{S} := \mathcal{S}_1 \cap \mathcal{S}_2$$

[1] Ultimately, in our example we shall take the matrices A and S to be

$$A = \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix}, \qquad S = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}. \tag{3.1}$$

The matrices are chosen small to save space.

with

$$\mathcal{S}_1 := \{(X,Y,A,S) \in \mathcal{V} : Y^2 < I, \quad X^2 < I\}$$

and

$$\mathcal{S}_2 := \{(X,Y,A,S) \in \mathcal{V} : F(X,Y,A,S) < 0, \quad Y > 0\}.$$

This optimization problem will be convex whenever $\mathcal{S}$ is a convex domain, since the objective function $\mathrm{Tr}\{X\}$ is linear in X. It is clear that $\mathcal{S}_1$ is convex; we wish to show that $\mathcal{S}_2$ is convex so that we can conclude that $\mathcal{S}$ is convex. For this purpose, we use our symbolic package to find the region where $F(X,Y,A,S)$ is convex with respect to X, Y in the domain $\mathcal{S}_2$.

Since matrix multiplication is not commutative, we must treat the matrices X, Y, A, and S symbolically as noncommutative variables. Thus, we load the Mathematica package NCAlgebra, which contains our convexity checker software. We type in the function F just as we see it in the definition of F and apply the convexity checker algorithm **NCConvexityRegion[]** (see section 5) using its default set of permutations. One of the outputs is the list

$$\{2y^{-1},\ -2(xa^Ty^{-1}ax - y)^{-1},\ 2y^{-1},\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0\}.$$

The interpretation of the output is that $F(X,Y,A,S)$ will be a convex function on the region consisting of all matrices that make each nonzero entry in the output list a positive definite expression, which, in this case, is the region given by

$$2Y^{-1} > 0 \qquad \text{and} \qquad -2(XA^TY^{-1}AX - Y)^{-1} > 0.$$

Thus, we conclude from this output that $F(X,Y,A,S)$ is simultaneously convex in X and Y whenever A, S, X, and Y are matrices of compatible dimension in the region $\mathcal{G}_{\mathcal{S}_2}$ given by

$$\mathcal{G}_{\mathcal{S}_2} := \{(X,Y,A,S) \in \mathcal{V} : Y > 0, \quad XA^TY^{-1}AX < Y\}. \tag{3.2}$$

To find if the above region $\mathcal{G}_{\mathcal{S}_2}$ is itself simultaneously convex in X and Y, we run the convexity checker once more on the function $G(X,Y,A) := XA^TY^{-1}AX - Y$,

NCConvexityRegion$[xa^Ty^{-1}ax - y, \{x,\ y\}]$.

This command outputs the list $\{2y^{-1},\ 0\}$. Thus, the region $\mathcal{G}$ is convex on matrices Y satisfying $Y > 0$. Thus, the region where the function G is convex consists of matrices Y satisfying $Y > 0$; consequently the region $\mathcal{G}_{\mathcal{S}_2}$ in (3.2) is convex. Thus, we can conclude that the optimization problem (P1) is convex inside the convex region

$$\mathcal{G} := \{(X,Y,A,S) \in \mathcal{V} : (X,Y,A,S) \in \mathcal{S} \cap \mathcal{G}_{\mathcal{S}_2}\}.$$

Equivalently

$$\mathcal{G} := \{(X,Y,A,S) \in \mathcal{V} : F(X,Y,A,S) < 0, \quad XA^TY^{-1}AX < Y, \quad Y > 0, \quad Y^2 < I, \quad X^2 < I\}.$$

Note that the region of convexity $\mathcal{G}$ for the optimization problem (P1) was determined without considering any specific numerical values for A and S. Thus, the set

of inequalities $\mathcal{G}$ characterizes a convex region for any arbitrary choice of two $n \times n$ matrices A and symmetric S no matter what value n is. Whether $\mathcal{G}$ is the biggest such region we have not said. In fact, the algorithm addresses this, requiring an interpretation, for which we refer to [5], or see section 5 for an abbreviated account. For the example above this gives that the largest subregion of matrix tuples (of large enough size) on which the Hessian is positive is the closure of $\mathcal{G}$.

3.2. Step 2. Invoking the NCSDP solver. Until this point, all calculations were symbolic. Now, we make the particular numerical choice for the matrices A, S given in (3.1). The optimization problem (P1) can now be solved with the NCSDP solver reliably and globally on the convex region of 2×2 matrices satisfying the constraints $\mathcal{G}$. We emphasize that this amounts to adding the following convex constraint

$$XA^TY^{-1}AX < Y \tag{3.3}$$

to the constraints defining $\mathcal{S}$. *Thus, we are not solving exactly the original problem, and the user must decide if this constraint meets his or her engineering needs.* Beware that declining to add the constraint (3.3) subjects one to the difficulties found in nonconvex situations, but one can still run numerical optimization routines.

To use NCSDP, we define the objective for this optimization problem as

$$\text{obj} := -\operatorname{Tr}\{X\},$$

subject to the constraint $G_i < 0$, where

$$\begin{aligned} G_1 &:= F(X, Y, A, S), \qquad G_2 := XA^TY^{-1}AX - Y, \\ G_3 &:= -Y, \qquad G_4 := YY - I, \qquad G_5 := XX - I, \end{aligned}$$

with

$$A = \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix}, \qquad S = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}.$$

Using this input, namely, (obj, $\{G_1, G_2, G_3, G_4, G_5\}$, $\{X, Y\}$), we run NCSDP. The solver finds the global optimizers over $\mathcal{G}$ to be

$$X^* = \begin{bmatrix} 0.3421 & 0.0263 \\ 0.0263 & 0.0788 \end{bmatrix}, \qquad Y^* = \begin{bmatrix} 0.8107 & 0.0016 \\ 0.0016 & 0.4255 \end{bmatrix}.$$

The optimal cost is therefore $\operatorname{Tr}\{-X^*\} = -0.4208$. We repeat that X^*, Y^* is a global optimum over the region

$$\mathcal{S} \cap \{(X, Y, A, S) \in \mathbb{S}^{2\times 2} \,:\, XA^TY^{-1}AX < Y, \quad Y > 0\}$$

with the matrices A and S as given above.

We emphasize there is no need to know much Mathematica to use NCSDP, unless one desires to use the convexity checker, and related symbolic algorithms, as we did in this example.

3.3. Scope of our methods. The method we shall describe here applies to problems of the form

$$\min_{X_i} \{\operatorname{Tr}\{X_1\} \,:\, X_i \in \text{closure}(\mathcal{G})\},$$

where the feasibility region $\mathcal{G}$ is given by

$$\mathcal{G} = \Big\{ (X_i, A_j) \,:\, F_1(X_i, A_j) > 0, \ldots, F_\ell(X_i, A_j) > 0 \Big\}$$

with $F_1(X_i, A_j), \ldots, F_\ell(X_i, A_j)$ rational expressions of noncommutative variables A_j, X_i, X_i^T. We assume the closure of the set $\mathcal{G}$ is compact. We can take some of the variables to be formally symmetric, like $X_7 = X_7^T$. The methods also apply to the feasibility problem, namely, determining if $\mathcal{G}$ is empty. We expect that (once refined) such methods will have advantages when the F_k are not highly complicated expressions.

An example of a problem we do not treat here is

$$\min \mathrm{Tr}\,\{X\} \qquad \text{subject to} \quad \mathrm{Tr}\,\big\{X^2\big\} \le 1.$$

However, we think our methods extend to such situations.

Space and expository considerations forced us to consider a single function $F(X)$ of a single symmetric variable $X = X^T$. The extension to the multivariate case stated above, found in [4], follows similar ideas, but it is too long to present here.

3.4. Comparing to the LMI approach. The optimization problem (P1) was actually selected to correspond to an LMI problem, so that we could compare approaches. There is not enough space to describe this in detail. (The corresponding LMI system has dimension 4×4.) We found that our approach produced exactly what was obtained using the LMI. Indeed our "extra condition" $XA^TY^{-1}AX < Y$ was a necessary condition for the LMI to be positive definite. Thus, from the LMI point of view, it is an essential constraint.

Since transformations via Schur complements can lead to an LMI representation with large constraint block matrices, the NCSDP solver has the potential to reduce the optimization time significantly compared to primal-dual solvers (see section 9).

3.5. Comparing to optimization over functions of commutative variables. If one has a complicated polynomial or rational function F, then there are typically many isolated regions on which the Hessian of F is positive definite. In our terminology, there are many "regions of convexity" for F. Thus, our technique requires selecting those convexity regions of interest and finding optima on them.

To those whose experience is with classical rational optimization, this seems odd, because there are many regions of convexity for F. However, our motivation comes from systems engineering problems, where we reemphasize that the number of matrix unknowns is small and that the rational functions are not terribly complicated; consequently F has a few connected regions of convexity. Moreover, the inequality constraints in a problem (e.g., $Y > 0$, $X^2 < I$) often select one convexity region.

4. Background on NC rational functions and convexity.

4.1. NC polynomials. We work with noncommutative (NC) polynomials with real numbers as coefficients in variables $x = \{x_1, \ldots x_g\}$. They cause little confusion, so a few examples suffice for an introduction:

$$p(x) = x_1x_2x_1 + x_1x_2 + x_2x_1, \qquad x_1^T = x_1, \quad x_2^T = x_2,$$

where the variables x_j are formally symmetric. In this next expression

$$p(x) = x_1^Tx_2x_1 + x_1^Tx_2 + x_2x_1, \qquad x_2^T = x_2,$$

the variable x_2 is formally symmetric but x_1 is not. Often, the term indeterminant is used instead of the term variable.

An NC polynomial p is symmetric provided that it is formally symmetric with respect to the involution ^{T}. Often, we shall substitute $n \times n$ matrices $X_1, \ldots, X_g$ into p for the variables $x_1, \ldots, x_g$. For a symmetric p, if the x_j are designated as symmetric variables, then the matrices X_j must be taken to be symmetric, and the resulting matrix $p(X_1, \ldots, X_g)$ is symmetric. The variables x_j which are not declared symmetric, if substituted by the matrix X_j, also result in the variables x_j^T being substituted by X_j^T.

4.2. NC rational functions. We shall discuss the notion of an NC rational function in terms of rational expressions. There is a technicality, "analytic at 0," which we include, since it makes formal definitions simpler. Casual readers can ignore it, since assuming analyticity elsewhere suffices.

An NC rational expression analytic at 0 is defined recursively. NC polynomials are NC rational expressions as are all sums and products of NC rational expressions. If r is a NC rational expression and $r(0) \neq 0$, then the inverse of r is a rational expression.

The notion of the formal domain of a rational expression r, denoted $\mathcal{F}_{r,\text{formal}}$, very roughly speaking, is

$$\mathcal{F}_{r,\text{formal}} := \{X \,:\, r(X) \text{ is defined } (\text{is not infinite})\}\,.$$

More precisely, the formal domain and the evaluation $r(X)$ of the rational expression at a tuple $X \in (\mathbb{S}^n)^g \bigcap \mathcal{F}_{r,\text{formal}}$ are both defined recursively.[2]

The following example illustrates it conveniently.

Example 4.1. Let the symmetric NC rational expressions $r(x)$ be given by

$$r(x_1, x_2) = (1 + x_1 - (3 + x_2)^{-1})^{-1}$$

with $x_1 = x_1^T$ and $x_2 = x_2^T$. The domain $\mathcal{F}_{r,\text{formal}}$ is

$$\bigcup_{n>0} \left\{ X_1, X_2 \in \mathbb{S}^n \,:\, I + X_1 - (3I + X_2)^{-1} \text{ and } 3I + X_2 \text{ are invertible} \right\}.$$

A difficulty is two different expressions, such as

$$r_1 = x_1(1 - x_2x_1)^{-1} \quad \text{and} \quad r_2 = (1 - x_1x_2)^{-1}x_1,$$

that can be converted into each other with algebraic manipulation. Thus they represent the same function and one needs to specify an equivalence relation on rational expressions to arrive at what are typically called NC rational functions. (This is standard and simple for commutative (ordinary) rational functions.) There are many alternate ways to describe NC rational functions and they go back 50 years or so in the algebra literature; cf. [17]. For engineering purposes, one need not be too concerned, since what happens is that two expressions r_1 and r_2 are equivalent whenever the usual manipulations one is accustomed to with matrix expressions convert r_1 to r_2. We say more on this in section 4.3.

[2]The formal domain of a polynomial p is all of $(\mathbb{S}^n)^g$ and $p(X)$ is defined as before. The formal domain of sums and products of rational expressions is the intersection of their respective formal domains. If r is an invertible rational expression analytic at 0 and $r(X)$ is invertible, then X is in the formal domain of r^{-1}.

For $\mathfrak{r}$ a rational function, that is, an "equivalence class of rational expressions r," we define its domain by

$$\mathcal{F}_{\mathfrak{r}} := \bigcup_{\{r \text{ represents } \mathfrak{r}\}} \mathcal{F}_{r,\text{formal}}.$$

Henceforth we do not distinguish between rational functions $\mathfrak{r}$ and rational expressions r, since this causes no confusion.

4.3. Partial fraction expansion of an NC rational. A computer algebra package must have a way to put functions into a canonical form. For example, if two rational expressions r_1 and r_2 represent the same rational function, then the canonical form of $r_1 - r_2$ would be 0. In NCAlgebra we have the command NCSimplifyRational, which in principle, when applied to a rational expression r, outputs what one might think of as a noncommutative partial fraction expansion of r; in practice, our command gives the true canonical form on a broad class of NC rational expressions but not all, since doing all of them is an infinite process. The theory behind producing this kind of canonical form is found in [11] and [24]. The idea is to generate what is called a Gröbner basis (GB) from the defining equations for inverses and store key elements of the GB as replacement rules in NCSimplifyRational. This is well suited to systems whose input operators B are left invertible, output operators C are right invertible, and state operators are generically invertible. Indeed they naturally lie in what is called a path algebra. It is not hard (for a GB expert) to prove that GB production respects the path algebra structure; thus, for example, the right inverse of B will never occur. See [10] for an extensive treatment of GBs in a path algebra.

4.4. Symbolic differentiation of noncommutative functions. Since our goal is to use symbolic computation to determine the gradient and the Hessian of functions in our optimization problems and to preserve the matrix structure of the unknowns, we need the notion of derivatives of function of variables which are symbolic noncommutative elements.

Noncommutative rational functions of x are polynomials in x and in inverses of polynomials in x. An example of a symmetric noncommutative function is

$$F(a, b, x) = ax + xa^T - \frac{3}{4}xbb^Tx, \quad x = x^T. \tag{4.1}$$

It is also assumed there is an involution on these rational functions which is denoted by the superscript ^{T} and which will play the role of transpose later when we substitute matrices for the indeterminates.

The *first directional derivative* of a noncommutative rational function $F(x)$ with respect to x in the direction δ_x is defined in the usual way

$$DF(x)[\delta_x] := \lim_{t\to 0} \frac{1}{t}\left(F(x + t\delta_x) - F(x)\right) = \frac{d}{dt}\, F(x + t\delta_x)\Big|_{t=0}.$$

For example, with F in (4.1),

$$DF(x)[\delta_x] = a\delta_x + \delta_x a^T - \frac{3}{4}\delta_x bb^T x - \frac{3}{4}xbb^T\delta_x,$$

and if $p(x) = x^4$,

$$Dp(x)[\delta_x] = \delta_x xxx + x\delta_x xx + xx\delta_x x + xxx\delta_x.$$

It is easy to check that derivatives of symmetric noncommutative rational functions always have the form

$$DF(x)[\delta_x] = \operatorname{sym}\left\{\sum_{i=1}^{k} a_i \delta_x b_i\right\}. \tag{4.2}$$

The sym operator is defined as $\operatorname{sym}\{M\} = M + M^T$.

The *second directional derivative* of a noncommutative rational function $F(x)$ with respect to x in the direction δ_x is defined by

$$D^2F(x)\left[\delta_x, \delta_x\right] = \frac{d^2}{dt^2} F(x + t\delta_x)\bigg|_{t=0}.$$

For example, if $p(x) = x^4$, then

$$D^2p(x)\left[\delta_x, \delta_x\right] = 2(\delta_x\delta_x xx + \delta_x x\delta_x x + \delta_x xx\delta_x + x\delta_x\delta_x x + x\delta_x x\delta_x + xx\delta_x\delta_x).$$

One can easily show that the second directional derivative of a symmetric noncommutative rational functions has the form

$$\begin{aligned} D^2F(x)\left[\delta_x, \delta_x\right] = \operatorname{sym}\Bigg\{ & \sum_{j=1}^{w_1} m_j \delta_x n_j \delta_x t_j \\ & + \sum_{j=1+w_1}^{w_2} m_j {\delta_x}^T n_j \delta_x t_j + \sum_{j=1+w_2}^{w_3} m_j \delta_x n_j {\delta_x}^T t_j \Bigg\}. \end{aligned} \tag{4.3}$$

For $r(x)$ given by $r(x_1, x_2) = (1 + x_1 - (3 + x_2)^{-1})^{-1}$, we have

$$Dr(x)[\delta_{x_1}] = -(1 + x_1 - (3 + x_2)^{-1})^{-1}\delta_{x_1}(1 + x_1 - (3 + x_2)^{-1})^{-1}$$

and

$$\begin{aligned} D^2r(x)\left[\delta_{x_1}, \delta_{x_1}\right] = 2(1 + x_1 - (3 + x_2)^{-1})^{-1} \cdots \\ \delta_{x_1}(1 + x_1 - (3 + x_2)^{-1})^{-1}\delta_{x_1}(1 + x_1 - (3 + x_2)^{-1})^{-1}. \end{aligned}$$

4.4.1. Symbolic NC differentiator algorithm. Derivatives of rational expressions can be defined recursively from the following rules:

1. If $r(x)$ is a polynomial, use the standard formula.
2. The product rule is, if $r(x) = r_1(x)r_2(x)$, then $Dr(x)[\delta_x] = Dr_1(x)[\delta_x]r_2(x) + r_1(x)Dr_2(x)[\delta_x]$.
3. The sum rule is, if $r(x) = r_1(x) + r_2(x)$, then $Dr(x)[\delta_x] = Dr_1(x)[\delta_x] + Dr_2(x)[\delta_x]$.
4. If $r(x)$ is the inverse $r(x) = f^{-1}(x)$ of an NC rational expression satisfying $f(0) \neq 0$, then $Dr(x)[\delta_x] := -f^{-1}(x)Df(x)[\delta_x]f^{-1}(x)$.

Our differentiation algorithm applies these rules (in a natural order) to an NC rational expression $r(x)$ and gives a new NC rational expression $Dr(x)[\delta_x]$, the directional derivative of the rational expression $r(x)$ in direction δ_x. Similarly, there are the natural formulas for the second directional derivative $D^2r(x)\left[\delta_x, \delta_x\right]$, for sums products and inverses. Our algorithm uses these recursively to compute our symbolic Hessian of $r(x)$.

4.5. Matrix convex functions. It will be shown that the definition just presented for the Hessian of a symmetric noncommutative rational function F is the key to determine the region of convexity for F. Therefore, it is the main ingredient of our NCConvexityRegion algorithm. There are several (almost equivalent) notions of noncommutative convexity; thus we define matrix convex functions as it is the definition used throughout the paper. For formal definitions, a detailed presentation, and an substantial theory behind the algorithm, see [5].

Let us suppose that F is the symmetric noncommutative rational function to be analyzed. Say F is a function of the noncommutative variables $x_1, \ldots, x_k$. Then, the function F is said to be matrix convex with respect to the variables $x_1, \ldots, x_k$ on a certain domain $\mathcal{G}$ provided its Hessian, denoted by $D^2F(X_1, \ldots, X_k)[\delta_{X_1}^2, \ldots, \delta_{X_k}^2]$, is a positive semidefinite matrix for all $X_1, \ldots, X_k$ in[3] $\mathcal{G}$ and all $\delta_{X_1}, \ldots, \delta_{X_k}$.

It is known (cf. [5]) that if $\mathcal{G}$ is a convex set, then this definition is equivalent to the usual notion of convexity, the geometrically matrix convex functions, which states that

$$F(\alpha X + (1-\alpha)Y) \leq \alpha F(X) + (1-\alpha)F(Y)$$

with $X := \{X_1, \ldots, X_k\}$ and $Y := \{Y_1, \ldots, Y_k\}$ tuples of matrices of compatible dimensions, and $0 \leq \alpha \leq 1$ a scalar. Of course, $\mathcal{G}$ might have separate components; then one often can focus on the component of primary interest with convexity on that component alone.

5. How the convexity checker algorithm works. With these notions of convexity, we now briefly introduce the algorithm underlying the command

NCConvexityRegion$[F, \{x\}]$

that provides a region $\mathcal{G}$ on which $F(X)$ is matrix convex. The main steps of the algorithm are as follows:

1. The second directional derivative with respect to $x_1, \ldots, x_k$, called the Hessian $D^2F[\delta_x, \delta_x]$ of the function F, is computed.
2. As the Hessian is always a quadratic function of the δ_x directions, it can be associated with a symmetric matrix $M_{D^2F[\delta_x, \delta_x]}$ with entries which are NC rational functions of x but not δ_x.
3. The noncommutative LDL$^{\mathrm{T}}$ factorization is applied to the coefficient matrix $M_{D^2F[\delta_x, \delta_x]}$.
4. And finally specifying positive definiteness of the resulting diagonal matrix $D(x_1, \ldots, x_k)$ gives inequalities describing a region $\mathcal{G}$ of variables on which F is matrix convex.
5. If a linear independence condition which is usually true holds and if the region $\mathcal{G}$ is nonempty for matrices of large enough size, then the closure of $\mathcal{G}$ is the largest domain on which $D^2F[\delta_x, \delta_x]$ is positive semidefinite. (See [5] for details on this rather complicated fact.)

Our implementation assumes that each pivot in the LDL$^{\mathrm{T}}$ decomposition is invertible. Possibly the most informative thing that could be done briefly is to give a simple example, presented in [5].

Example 5.1. Define the function $F(x)$ by

$$F(x) = g^T x^T a x g + x^T b x + g^T x^T c x + x^T c^T x g,$$

where $b = b^T$ and $a = a^T$.

[3] More precisely, for all X in the set $\mathcal{G}$ that intersect the domain of the rational function F.

1. The Hessian of $F(x)$ is given by
$$D^2F(X)\,[\delta_x,\delta_x] = 2({\delta_x}^T b\delta_x + {\delta_x}^T c^T\delta_x G + g^T{\delta_x}^T a\delta_x g + g^T{\delta_x}^T c\delta_x).$$
2. Equivalently, this quadratic expression takes the form
$$D^2F(X)\,[\delta_x,\delta_x] = V[\delta_x]^T M_{D^2F} V[\delta_x] = 2({\delta_x}^T, g^T{\delta_x}^T)\begin{pmatrix} b & c^T \\ c & a \end{pmatrix}\begin{pmatrix}\delta_x \\ \delta_x g\end{pmatrix}.$$
3. The LDL$^{\mathrm{T}}$ decomposition with no permutation applied to M_{D^2F} is
$$\begin{pmatrix} 1 & 0 \\ cb^{-1} & 1 \end{pmatrix}\begin{pmatrix} b & 0 \\ 0 & a - cb^{-1}c^T \end{pmatrix}\begin{pmatrix} 1 & b^{-1}c^T \\ 0 & 1 \end{pmatrix},$$
provided that b is invertible.[4]
4. Therefore, when b is invertible, sufficient conditions for the Hessian to be positive semidefinite are
$$b > 0 \quad \text{and} \quad a - cb^{-1}c^T > 0.$$
5. If the Hessian is "positive," then for large enough dimension, a, b, c are in the closure of the set described in step 4.

A finer property of our algorithm (and implementation) is that it includes the possibility of permutations. Thus, if we know that a (instead of b) in the example above is invertible, then a permutation can be applied before applying the LDL$^{\mathrm{T}}$ decomposition. This makes **NCConvexityRegion[]** somewhat delicate. In practice, the runs will finish with some choices of permutation and not with others. Also, the expressions that appear in the outputs from successful runs using different permutations can be different; however, the theory behind NCConvexityRegion tells us that the sets they describe are all the same.

Example 5.2. Let $p(x)$ be given by $p(x) = a^Tx^2b + b^Tx^2a$ with $x = x^T$. Its Hessian is $a^T{\delta_x}^2b + b^T{\delta_x}^2a$. Represent this as v^TMv with
$$v^T := \begin{pmatrix} a^T\delta_x & b^T\delta_x \end{pmatrix} \quad \text{and} \quad M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$
whose eigenvalues are 1 and -1. Thus, we conclude from steps 4 and 5 that (for large enough n) there is no set $\mathcal{G}$ of matrices a and b, open in the set of pairs of $n \times n$ matrices, on which the Hessian is positive definite.

A related example is $p(x) = a^Tx^2a + a^Tx^2a$. Its Hessian is $a^T{\delta_x}^2a + a^T{\delta_x}^2a$; the representation above still works with $v^T := (a^T\delta_x, a^T\delta_x)$. However, the linear independence condition of step 4 on $v^T := (a^T\delta_x, a^T\delta_x)$ does not hold, that is, the condition $a^T\delta_x$ and $a^T\delta_x$ are linearly independent fails, and thus no definitive conclusion is possible. Another (more natural) representation is $a^T{\delta_x}^2a + a^T{\delta_x}^2a = 2a^T\delta_x 1\delta_x a$, that is, M is the 1×1 matrix 1. Thus step 4 implies, the Hessian of $p(x)$ is everywhere positive; note that in this case, linear independence holds. NCConvexityRegion does the later not the former calculation.

Our convexity region algorithm was new with [5], where it is described in detail, together with the proofs of its properties. The guarantee it produces for the nonnegativity of the NC Hessian (as in step 4) is straightforward to prove, while the converse (as in step 5) is quite surprising and not at all easy to prove.

[4]The list returned by **NCConvexityRegion** is $\{b, \quad a - cb^{-1}c^T\}$.

5.1. Noncommutativity is essential. A great advantage of our framework is that treating matrices as single letters is likely the only practical necessary and sufficient approach available for checking convexity of rational functions on matrices of large dimension. For a commutative rational function F, one might imagine a Parrilo type of sum of squares algorithm [22], which could affirm positivity of the Hessian of F, and thus convexity of F. Unfortunately, it would be practical with only a few dozen variables. If the unknown X and Y were symmetric matrices on even a 10-dimensional state-space, the entrywise representation would give about 100 commuting variables. This is prohibitive. On the other hand, our convexity checking method is insensitive to the dimension of the state-space.

6. Convex optimization over matrix functions. In this paper, the presentation of the numerical NCSDP optimization solver for matrix functions is limited to the single variable case. The extension to the multivariate case, found in [4], follows similar ideas, but it is too long to present here. This solver is based on an implementation of the method of centers.

There are in the literature few papers on solving nonlinear matrix inequalities. In [15], the authors presented and analyzed a numerical interior point trust region algorithm that can be used for solving a class of nonlinear (nonconvex) semidefinite programming problem. For MI problems that concern the minimization of the largest eigenvalue of a matrix (this is a convex but highly nonsmooth problem), the work by [21, 16] is a good source. See [1, 26, 6, 27, 28, 29] for general SDP problems and [3] for a comprehensive introduction to convex optimization.

The main distinctions between these approaches and ours is that our method focus on unknowns which themselves are treated as matrices. In our research, we deal with the entire matrix structure instead of dealing with the individual entries of the matrix unknowns. Also, the user does not need to calculate first and second derivatives by hand, since this is done automatically in our method.

The outline of our method is as follows. We compute the first and second derivatives of a potential function noncommutatively (symbolically) in a way that keeps the matrix structure and does not split up the matrices (see section 4.4). This step provides the Hessian map $\mathcal{H}(\delta_x)$ and the gradient map $\mathcal{Q}$. It is this step whose efficiency is improved by our MinimumSylvesterIndex algorithm for symbolically obtaining an efficient form for $\mathcal{H}(\delta_x)$ (this is described in section 8). After this, our algorithm turns numerical (by substituting matrices for the indeterminate that appears in the expressions for $\mathcal{H}(\delta_x)$ and $\mathcal{Q}$) and the code aims to solve the respective numerical linear system of equations $\mathbb{H}(\delta_X) = \mathbb{Q}$ in the direction δ_X. We find that the numerical linear subproblem has an elegant form, and it is an interesting open question how to fully exploit this form. We numerically solve this linear system for δ_X in a conventional way. The method successively iterates, at the numerical level, until the algorithm converges to an optimal solution.

As described above, one needs to compute derivatives of an auxiliary potential function. The formula for this potential function depends on which member of the family of penalty/barrier methods one wishes to adhere to. The approach we have selected is known as the *analytic method of centers*. Before describing this method in section 6.2, we characterize the optimization problem we are interested in section 6.1.

6.1. Constrained optimization problem. Throughout the paper, we shall be primarily concerned with the convex optimization problem

$$\text{(COP)} \qquad f^{\text{opt}} = \min \{\text{Tr}\,\{X\} \,:\, X \in \text{closure}(\mathcal{G})\}$$

with the feasibility domain $\mathcal{G}$ given by

$$\mathcal{G} = \Big\{ X \in \mathcal{V} \,:\, F(X) > 0 \Big\},$$

where we assume the closure of $\mathcal{G}$ is compact, the set $\mathcal{V}$ is a subspace of $\mathbb{R}^{p\times q}$, and $F : \mathcal{V} \to \mathbb{S}^n$ is a concave function. This type of problem incorporates the eigenvalue minimization problem as a particular case.

6.2. Review of the method of centers. The idea behind the method of centers ([2, 19] and references therein) is to replace the above constrained problem by a sequence of unconstrained minimization problems whose solutions eventually tend to the set of optimal solutions of (COP). This occurs in the context of *interior penalty methods*. It follows therefore, that under certain hypotheses, the original problem (COP) can be approximated by a sequence of unconstrained convex optimization problems of the form

$$X^*(\gamma) = \operatorname{argmin} \{ \phi_\gamma(X) \,:\, X \in \mathcal{G}_\gamma \} \tag{UOP}$$

with the auxiliary potential function $\phi_\gamma : \mathcal{G}_\gamma \to \mathbb{R}$ given by

$$\phi_\gamma(X) = \zeta \log \left(1/(\gamma - \operatorname{Tr}\{X\})\right) - \log\det F(X),$$

where ζ is a scalar satisfying $\zeta \geq 1$ and $\mathcal{G}_\gamma$ is the domain given by

$$\mathcal{G}_\gamma = \Big\{ X \in \mathcal{G} \,:\, \operatorname{Tr}\{X\} < \gamma \Big\}.$$

The decrease of the parameter γ has to be done in such a way that the method maintains feasibility at each iteration and that the sequence $\{\gamma^k\}$ is guaranteed to converge to f^{opt} (the minimum values of the objective function). The formula for updating γ at some iteration k is given by

$$\gamma^{k+1} = (1-\theta)\operatorname{Tr}\left\{X^k\right\} + \theta\gamma^k, \quad 0 < \theta < 1, \tag{6.1}$$

where X^k denotes $X^*(\gamma^k)$. Under mild conditions, the solution $X^*(\gamma)$ of (UOP) approaches the set of optimal solutions of (COP) for an appropriate sequence of decreasing centralization parameter γ (see [2, 7, 19]).

Using these facts, one possible algorithm based on the method of centers can be described by the following algorithm.

Algorithm 6.1. Method of centers.
Fix θ such that $0 < \theta < 1$;
Choose X^0 and γ^0 such that $X^0 \in \mathcal{G}_{\gamma^0}$;
$k \leftarrow 0$;
while not converged **do**
  $\gamma^{k+1} \leftarrow (1-\theta)\operatorname{Tr}\left\{X^k\right\} + \theta\gamma^k$;
  $X^{k+1} \leftarrow \operatorname{argmin}\left\{\phi_{\gamma^{k+1}}(X^k) \,:\, X^k \in \mathcal{G}_{\gamma^{k+1}}\right\}$;
  $k \leftarrow k+1$;
end while

There are some important comments concerning this algorithm:

1. The bound γ^{k+1} from (6.1), used in the determination of the analytic center of the potential $\phi_{\gamma^{k+1}}(X^k)$, never produces infeasible starting points X^k, γ^{k+1}.

2. It will be necessary to find feasible starting points X^0 and γ^0 to be used in Algorithm 6.1. This is a *feasibility problem* that can be solved by the same method of centers.
3. Evidently, the expensive part of the algorithm is the *inner loop*, the part that computes the *analytic center*.

6.3. An algorithm to solve the inner loop. This section sketches briefly a standard algorithm to solve the inner loop. The algorithm implemented in the NCSDP code to find the analytic center

$$X^{k+1} = \operatorname{argmin} \left\{ \phi_\gamma(X^k) \,:\, X^k \in \mathcal{G}_\gamma \right\}$$

for fixed scalar γ is based on a conventional modified Newton's method [19, 3, 20], as follows:

while not converged **do**
$X^{k+1} \leftarrow X^k + \sigma {\delta_X}^k$;
end while

where ${\delta_X}^k$ is the Newton direction (see section 7.1). The step length used in this algorithm [19] is given by

$$\sigma = \begin{cases} 1/(1+\tau) & \text{if } \tau > 1/4, \\ 1 & \text{otherwise} \end{cases}$$

for $\tau = \sqrt{g^T H^{-1} g}$ with g and H, respectively, the gradient and the Hessian of $\phi_\gamma(X^k)$. The stopping criteria used in our experiments was, stop as soon as $\sigma = 1$. In practice Newton's method works better with a line search instead of the above fixed step length. Certainly, we shall consider implementing a line search in a more elaborate version of the code.

7. Solving for the analytic center. In sections 7.1, 7.2, and 7.3 we discuss in depth the linear subproblem that provides the update direction δ_X, which is the core of the modified Newton's algorithm presented in section 6.3. Therefore, in these sections we show how to exploit the matrix structure of the unknowns to find an elegant formula for the linear subproblem.

7.1. Describing the main steps. The original convex optimization problem (COP) has now been replaced by a sequence of unconstrained convex minimization problems of the form (UOP) for a decreasing sequence of scalars $\{\gamma^k\}$ provided by formula (6.1). To find the update directions which lead toward the central path for fixed values of γ, Newton's method is applied by minimizing an approximation, the second-order Taylor series expansion, of the potential function $\phi_\gamma(X)$. In a vague sense, these procedures can be summarized as follows:

1. Compute symbolically the second-order Taylor expansion of the potential function $\phi_\gamma(x + \delta_x)$ in some direction δ_x

$$\phi_\gamma(x) + D\phi_\gamma(x)\,[\delta_x] + \frac{1}{2} D^2\phi_\gamma(x)\,[\delta_x, \delta_x]\,.$$

2. The Newton step ${\delta_x}^*$ must satisfy the necessary optimality conditions for the following quadratic minimization problem:

$$\min_{\delta_x} \; D\phi_\gamma(x)\,[\delta_x] + \frac{1}{2} D^2\phi_\gamma(x)\,[\delta_x, \delta_x]\,.$$

3. This first-order necessary optimality condition is algebraically[5] given by

$$0 = D\left[D\phi_\gamma(x)\,[\delta_x] + \frac{1}{2}D^2\phi_\gamma(x)\,[\delta_x,\delta_x]\right]\Big[\delta_v\Big] \quad \text{for all symmetric } \delta_v. \tag{7.1}$$

Which will be shown (in Theorem 7.1) to be equivalently[6] written as

$$\mathrm{Tr}\left\{\delta_v(\mathcal{H}(\delta_x)-\mathcal{Q})^T + (\mathcal{H}(\delta_x)-\mathcal{Q}){\delta_v}^T\right\} = 0 \quad \text{for all symmetric } \delta_v. \tag{7.2}$$

4. Finally, find a Newton update ${\delta_X}^*$ satisfying (7.1) or (7.2) for all δ_v.

Section 7.2 concerns steps 1 through 3, which are performed symbolically. On the other hand, step 4, presented in section 7.3, is completely numerical.

7.2. Obtaining the formulas for the linear subproblem. The main ingredient of our approach is how we use symbolic computation to determine the algebraic linear system of equations that provides the update direction δ_X toward the central paths. At the outset of this work, it was not obvious that we could find a clean symbolic formula for the linear subproblem which treated both known and unknown matrices as a whole and did not break them into entries. Fortunately, this is possible, as the next theorem shows.

THEOREM 7.1. *Let $\mathcal{V}$ be a subspace of $\mathbb{R}^{p\times q}$, and let the map $F : \mathcal{V} \to \mathbb{S}$ be concave. Consider the unconstrained auxiliary potential function $\phi_\gamma : \mathcal{G}_\gamma \to \mathbb{R}$ given by*

$$\phi_\gamma(X) = \zeta \log\Big(1/(\gamma - \mathrm{Tr}\,\{X\})\Big) - \log\det F(X),$$

where ζ is a scalar satisfying $\zeta \geq 1$ and the feasibility domains $\mathcal{G}$ and $\mathcal{G}_\gamma$ are respectively given by

$$\mathcal{G} = \Big\{X \in \mathcal{V} \,:\, F(X) > 0\Big\} \quad \textit{and} \quad \mathcal{G}_\gamma = \Big\{X \in \mathcal{G} \,:\, \mathrm{Tr}\,\{X\} < \gamma\Big\},$$

where we assume the closure of $\mathcal{G}$ is compact. Then, the update direction ${\delta_X}^*$ toward the central path for the above potential is the solution of the following symbolically[7] computable algebraic linear equation:

$$\mathrm{Tr}\left\{\delta_v(\mathcal{H}(\delta_x)-\mathcal{Q})^T + (\mathcal{H}(\delta_x)-\mathcal{Q}){\delta_v}^T\right\} = 0 \quad \textit{for all } \delta_v \in \mathcal{V}, \tag{7.3}$$

where $\mathcal{H}(\delta_x)$ is linear as regarded as a function of δ_x. Moreover, $\mathcal{Q}$ and $\mathcal{H}(\delta_x)$ are given by

$$\mathcal{Q} = \sum_{i=1}^{k} a_i^T F(x)^{-1} b_i^T - \frac{1}{2}\zeta\,(\gamma - \mathrm{Tr}\,\{x\})^{-1}\,\mathfrak{I}_d$$

[5] Assuming that all x belong to some space $\mathcal{V}$, then $\delta_x, \delta_v \in \mathcal{V}$. Cf. footnotes 6, 7.

[6] When we say $\mathrm{Tr}\,\{x\}$, we mean that the operation $\mathrm{Tr}\,\{\}$ is to be performed after x has been replaced by a matrix. Cf. footnote 7.

[7] When we say $\delta_v \in \mathcal{V}$, we mean that we will substitute matrices in $\mathcal{V}$ for the indeterminate δ_v. The same is true for $\mathrm{Tr}\,\{x\}$. Cf. footnote 6.

and

$$\begin{aligned}\mathcal{H}(\delta_x) = &\sum_{i=1}^{k}\sum_{j=1}^{k} a_i^T F(x)^{-1} a_j \delta_x b_j F(x)^{-1} b_i^T + \sum_{i=1}^{k}\sum_{j=1}^{k} a_i^T F(x)^{-1} b_j^T \delta_x^T a_j^T F(x)^{-1} b_i^T \\ &- \frac{1}{2}\sum_{j=1}^{w_1} n_j^T \delta_x^T m_j^T F(x)^{-1} t_j^T + m_j^T F(x)^{-1} t_j^T \delta_x^T n_j^T \\ &- \frac{1}{2}\sum_{j=1+w_1}^{w_2} n_j \delta_x t_j F(x)^{-1} m_j + n_j^T \delta_x m_j^T F(x)^{-1} t_j^T \\ &- \frac{1}{2}\sum_{j=1+w_2}^{w_3} t_j F(x)^{-1} m_j \delta_x n_j + m_j^T F(x)^{-1} t_j^T \delta_x n_j^T \\ &+ \frac{1}{2}\zeta \left(\gamma - \operatorname{Tr}\{x\}\right)^{-2} \operatorname{Tr}\{\delta_x\}\, \mathfrak{I}_d,\end{aligned}$$

where the terms a_i, b_i, m_i, n_i, t_i are obtained from the first and second directional derivatives of $F(x)$ as given by (4.2) *and* (4.3). *The term $\mathfrak{I}_d$ stands for the symbolic analogue of the identity matrix.*

Proof. The theorem follows from manipulations of (7.1). For a detailed presentation see [4]. □

The result of Theorem 7.1, the algebraic linear equation (7.3), can be further specialized depending upon the structure of the underlying subspace $\mathcal{V}$; in other words, if there is or is not some restriction imposed on X. Specifying various structures for the underlying subspace $\mathcal{V}$ is the subject of Corollary 7.2 which is the main result of this section.

COROLLARY 7.2. *Let $\mathcal{V}$ be a subspace of $\mathbb{R}^{p\times q}$ and $\mathcal{C}$ be a convex domain in $\mathcal{V}$. Let the map $F : \mathcal{C} \to \mathbb{S}$ be concave. Consider the unconstrained auxiliary potential function $\phi_\gamma : \mathcal{G}_\gamma \to \mathbb{R}$ given by*

$$\phi_\gamma(X) = \zeta \log\Big(1/(\gamma - \operatorname{Tr}\{X\})\Big) - \log\det F(X),$$

where ζ is a scalar satisfying $\zeta \geq 1$ and the feasibility domains $\mathcal{G}$ and $\mathcal{G}_\gamma$ are respectively given by

$$\mathcal{G} = \Big\{X \in \mathcal{C} \subset \mathcal{V} \,:\, F(X) > 0\Big\} \quad \textit{and} \quad \mathcal{G}_\gamma = \Big\{X \in \mathcal{G} \,:\, \operatorname{Tr}\{X\} < \gamma\Big\}.$$

Then, depending upon the structure of the underlying subspace $\mathcal{V}$, the update direction $\delta_x{}^*$ toward the central path for the above potential is the solution of one of the following symbolically computable algebraic linear equations:

1. *The subspace $\mathcal{V}$ equals $\mathbb{R}^{p\times q}$ so that the unknown X can be any matrix in $\mathbb{R}^{p\times q}$:*

$$\sum_{i=1}^{c_1} \mathfrak{a}_i \delta_x \mathfrak{b}_i + \sum_{j=c_1+1}^{c_2} \mathfrak{a}_j \delta_x^T \mathfrak{b}_j + \varrho \operatorname{Tr}\{\delta_x\} = \mathfrak{Q}.$$

2. *The subspace $\mathcal{V}$ equals $\mathbb{S}^p$ so that the unknown X is restricted to being symmetric:*

$$\sum_{i=1}^{c_2} \mathfrak{b}_i^T \delta_x \mathfrak{a}_i^T + \mathfrak{a}_i \delta_x \mathfrak{b}_i + \varrho \operatorname{Tr}\{\delta_x\} = \mathfrak{Q} + \mathfrak{Q}^T.$$

3. *The unknown X is restricted to being a scalar multiple of the identity, that is, $X = \sigma I$, for some scalar σ:*

$$\mathrm{Tr}\left\{\sum_{i=1}^{c_2} \mathfrak{a}_i \mathfrak{b}_i + \varrho \mathrm{Tr}\left\{\mathfrak{I}_d\right\}\right\} \delta_\sigma = \mathrm{Tr}\left\{\mathfrak{Q}\right\}, \qquad \delta_x = \delta_\sigma \mathfrak{I}_d, \quad \delta_\sigma \in \mathbb{R}.$$

For these expressions, $\mathfrak{Q}$ is the gradient term given by

$$\mathfrak{Q} = \sum_{i=1}^{k} a_i^T F(x)^{-1} b_i^T - \frac{1}{2}\zeta \left(\gamma - \mathrm{Tr}\left\{x\right\}\right)^{-1} \mathfrak{I}_d.$$

The term ϱ is the cost term given by $\varrho = \frac{1}{2}\zeta(\gamma - \mathrm{Tr}\left\{x\right\})^{-2}\mathfrak{I}_d$. And, by an appropriate relabeling, the terms $\mathfrak{a}_i$ and $\mathfrak{b}_i$ are obtained from the Hessian map $\mathcal{H}(\delta_x)$ presented in Theorem 7.1*.*

Proof. The corollary follows from Theorem 7.1 by expressing the linear system of equations (7.3) considering the structure of the underlying subspace $\mathcal{V}$. (See [4].) ∎

The above results provide the necessary conditions that the update δ_X must satisfy in order to be a Newton direction toward the central path of the unconstrained auxiliary potential function $\phi_\gamma(X)$.

We provide a tutorial example in appendix A to illustrate Theorem 7.1 and Corollary 7.2 and to give an idea of how they are proved.

7.3. Solving the linear subproblem. The algebraic linear subproblem[8] provided in Corollary 7.2 always has the form

$$\sum_{i}^{N} \mathfrak{a}_i \delta_x \mathfrak{b}_i + \varrho\, \mathrm{Tr}\left\{\delta_x\right\} = \bar{\mathfrak{Q}}, \tag{7.4}$$

where the $\mathfrak{a}_i$, $\mathfrak{b}_i$, and $\bar{\mathfrak{Q}}$ are rational functions of the known noncommutative variables given in the problem formulation and δ_x is the unknown variable (the update direction). The notation $\mathfrak{a}_i, \mathfrak{b}_i$ stands for symbolic Sylvester terms, and the notation $\mathcal{A}_i, \mathcal{B}_i$ will indicate we have substituted matrices of compatible size for the symbolic variables in $\mathfrak{a}_i, \mathfrak{b}_i$.

The integer N has been called the Sylvester index in [14]. A key point is that the same linear system can have several representations of the form (7.4), that is, the representation of the Hessian map $\mathcal{H}(\delta_x)$ in Theorem 7.1 is not unique. We will see later in section 8 that there is a substantial advantage to obtaining a representation with a small Sylvester index.

There are two main costs in treating the linear subproblem (7.4):

FE: evaluating the matrices $\mathfrak{a}_i$, $\mathfrak{b}_i$, and $\bar{\mathfrak{Q}}$ at each iteration, that is, converting them from symbols to numeric matrices whose entries are numbers is time-consuming (see section 8);

NLS: solving numerically the resulting linear system for δ_X.

[8] (a) Depending upon the structure of the underlying subspace $\mathcal{V}$, the term $\bar{\mathfrak{Q}}$ will be either $\bar{\mathfrak{Q}} = \mathfrak{Q}$ or $\bar{\mathfrak{Q}} = \mathfrak{Q} + \mathfrak{Q}^T$. (b) The third case in Corollary 7.2 behaves in a similar way, so we do not go through it.

After the evaluation step FE has been performed, we rewrite the linear subproblem (7.4) as

$$\sum_{i}^{N} \mathcal{A}_i \delta_X \mathcal{B}_i + \varrho \operatorname{Tr}\{\delta_X\} = \mathbb{Q}, \tag{7.5}$$

indicating that the indeterminate have already been substituted by matrices of compatible dimension. Then, using the vec operation (see [12]), the matrix system (7.5) can be transformed into the equivalent vector form

$$Hv = g, \tag{7.6}$$

where H is the Hessian matrix given by

$$H = \sum_{i}^{N} \mathcal{B}_i^T \otimes \mathcal{A}_i + \operatorname{vec}(\varrho)\operatorname{vec}(I)^T,$$

where the vector g is given by $g = \operatorname{vec}(\mathbb{Q})$, and v is the vector of unknowns given by $v = \operatorname{vec}(\delta_X)$. The symbol $\otimes$ denotes the Kronecker product.

Therefore, the cost of numerically solving the linear subproblem can be split into two distinct costs:

KP: applying Kronecker products to build the Hessian matrix H;

LS: numerically solving $Hv = g$ for the unknown vector v.

The above "brute force" procedure does not take advantage of the particular structure of $\mathcal{H}(\delta_x)$. Of course, Lyapunov equations are very special cases for which there are extremely fast algorithms (see [9, 14]). Naturally, an open question highly motivated by this research is how one uses this special Sylvester structure to solve efficiently (7.5).

Iterative methods are attractive for solving Sylvester-type linear equations. Related to this is [13] and references therein. However, in our paper, we do not investigate numerical linear solvers special to Sylvester forms. It is a separate topic and our focus was on our new noncommutative symbolic methodology. Consequently, we just used our brute force Kronecker product approach since it is reliable. However, in order to speed up the implementation of our linear solver, we plan a careful study of iterative methods like conjugate gradient in a separate project.

8. Improving the evaluation time for the linear subproblem. In this section, we illustrate by examples that for a system of linear equations, the Sylvester form (7.4) is not unique. Moreover, we show that the Sylvester index has a great influence on the evaluation cost given in Step FE of section 7.3.

Consider an expression in the Sylvester form

$$\mathcal{H}(\delta_x) = a\delta_x a^T + x^T\delta_x x + b\delta_x b^T - a\delta_x x - x^T\delta_x a^T + b\delta_x a^T + a\delta_x b^T. \tag{8.1}$$

The Sylvester index in this case is seven. This expression can be written in at least two different ways, having the same number of terms. One possibility is

$$\mathcal{H}(\delta_x) = (a - x^T)\delta_x(a - x^T)^T + (a + b)\delta_x(a + b)^T - a\delta_x a^T = \sum_{i=1}^{N=3} \mathfrak{a}_i \delta_x \mathfrak{b}_i$$

for $\mathfrak{a}_i$ and $\mathfrak{b}_i$ given by

$$\begin{aligned} \mathfrak{a}_1 &= (a - x^T), \qquad \mathfrak{a}_2 = (a + b), \qquad \mathfrak{a}_3 = -a, \\ \mathfrak{b}_1 &= (a - x^T)^T, \quad \mathfrak{b}_2 = (a + b)^T, \quad \mathfrak{b}_3 = a^T. \end{aligned}$$

Another one is

$$\mathcal{H}(\delta_x) = (a + b - x^T)\delta_x(a + b - x^T)^T + b\delta_x x + x^T\delta_x b^T = \sum_{i=1}^{N=3} \mathfrak{a}_i\delta_x\mathfrak{b}_i$$

for $\mathfrak{a}_i$ and $\mathfrak{b}_i$ given by

$$\begin{aligned} \mathfrak{a}_1 &= (a + b - x^T), \qquad \mathfrak{a}_2 = b, \qquad \mathfrak{a}_3 = x^T, \\ \mathfrak{b}_1 &= (a + b - x^T)^T, \quad \mathfrak{b}_2 = x, \qquad \mathfrak{b}_3 = b^T. \end{aligned}$$

In both cases, the Sylvester index is now three, going down by over one half. Thus, for a given Hessian map $\mathcal{H}(\delta_x)$, the Sylvester index is not unique. Moreover, the $\mathcal{H}(\delta_x)$ may have different representations for a specific Sylvester index (as illustrated above). It is also easy to see that a significant reduction in the Sylvester index might happen for an expression which contains a large number of Sylvester terms. Based on those ideas, a few natural questions can be formulated:

1. Given an expression for the Hessian map $\mathcal{H}(\delta_x)$, what is the minimum Sylvester index associated with this expression?
2. Is there a symbolic algorithm to compute a minimum Sylvester index representation?
3. How many different expressions which achieve this minimal Sylvester index are possible?
4. Does the evaluation time in Step FE vary substantially for small versus large Sylvester index N?

This section addresses the first two questions. We describe preliminarily a symbolic algorithm which is fast and which often reduces the Sylvester index N dramatically. Later, we describe a more powerful (but slower) symbolic algorithm which gives the minimal Sylvester index when the coefficients $\mathfrak{a}_j$ and $\mathfrak{b}_j$ are polynomials. For our problems, the coefficients are not polynomials, but this algorithm applies with no restrictions. However, we can no longer guarantee that we obtain the minimal Sylvester index.

As to question 4, we have found through examples (see section 8.3.1) that the overall computational time spent on numerically solving an optimization problem using our NCSDP code dramatically reduces when the Sylvester index of the Hessian map $\mathcal{H}(\delta_x)$ is reduced by one of these two algorithms. However, we should consider the time consumed at the symbolic level by the algorithm itself. We found that the first algorithm to be presented is faster than the second algorithm. (The second provides the minimal Sylvester index).

8.1. A Sylvester index reducing algorithm. We now describe our first Sylvester index reducing algorithm, which is denoted by

NCCollectSylvester[exp, var].

The implementation used in our NCSDP optimization code is a command that sequentially applies two commands, called NCRightSylvester[] and NCLeftSylvester[], to the expression. These two "sided" commands have analogous implementation,

which uses a pattern match that collects similar terms on the right (respectively, on the left) side of the expression. We now present the idea behind these commands.

ALGORITHM 8.1 (NCRIGHTSYLVESTER ALGORITHM).

1. IDENTIFY THE TERMS IN WHICH THE EXPRESSION SHOULD BE COLLECTED. In the example given by expression (8.1), this term is δ_x.
2. BUILD A RIGHT LIST. This list contains the terms that multiplies δ_x from the right side (including δ_x itself). For the expression (8.1), we would obtain
$$\text{RightList} = \{\delta_x a^T,\ \delta_x x,\ \delta_x b^T\}.$$
3. BUILD A COLLECTLIST. For each element inside RightList, we add together all the terms that multiply this element from the left side. For our example we obtain
$$\text{CollectList} = \{(a + b - x^T),\ (x^T - a),\ (a + b)\}.$$
4. COMBINE THE COLLECTLIST AND THE RIGHTLIST. This gives the answer. For our example it is
$$\mathcal{H}(\delta_x) = (a + b - x^T)\delta_x a^T + (x^T - a)\delta_x x + (a + b)\delta_x b^T.$$

The above right-sided implementation of the collecting algorithm begins by building a list of multipliers from the right side of δ_x. Clearly, a similar implementation can also be done by obtaining a left list of terms that multiplies δ_x from the left side, instead of the right side. In this way, we can implement two collect commands that differ only by the side in which the process of collecting begins; thus, we can have an **NCRightSylvester[]** command (described above) and an **NCLeftSylvester[]** command. As already mentioned, our implementation encompasses these two commands into a single command

NCCollectSylvester[exp, var]
:= NCRightSylvester[NCLeftSylvester[exp, vars], vars].

These algorithms, when applied to an expression in the Sylvester form, in practice provide a large reduction on the Sylvester index. However, these algorithms do not guarantee that one can obtain the lowest possible Sylvester index. On the other hand, in the next section, we provide an algorithm which under some hypothesis provides the lowest possible Sylvester index.

8.2. The minimum Sylvester index algorithm. Consider a function $L(\delta)$ of a noncommutative variable δ in the Sylvester form

$$L(\delta) := \sum_{j=1}^{N} \mathfrak{a}_j \, \delta \, \mathfrak{b}_j,$$

with the terms $\mathfrak{a}_j$ and $\mathfrak{b}_j$ polynomials in noncommutative variables. The algorithm proposed in this section has property that if the $\mathfrak{a}_j$ and $\mathfrak{b}_j$ are restricted to be polynomials, then it always gives the lowest possible Sylvester index. This fact is presented in Theorem 8.3. We now present the steps of the algorithm and give an example. For

this purpose, consider the following expression:

$$\begin{aligned}(8.2)\quad L(\delta) &= (xb+axb)\,\delta\,(-2axb+bxb)+(xb+axb)\,\delta\,(xb-axb+bxb)\\ &-(xb+axb)\,\delta\,(xb-axb+2xax)+(xb+axb)\,\delta\,(xb-2axb+bxb+xax)\\ &+(c-xb-bxb+xax)\,\delta\,(-axb+xax)+(c+axb-bxb+xax)\,\delta\,(xb+xax)\\ &+(c+xb+2axb-bxb+xax)\,\delta\,(2xb+bxb).\end{aligned}$$

The Sylvester index associated with this expression is $N = 7$. Using our algorithm, we will see that the minimum Sylvester index is $N^* = 2$.

Algorithm 8.2 (MinimumSylvesterIndex Algorithm).

1. Identify the terms in which the expression should be collected. In the example given by expression (8.2), this term is δ.
2. Build a right list. This list contains the terms that multiplies δ from the right side in $L(\delta)$. Denote this list by $\mathfrak{b}$. For our example, this list is

$$\begin{aligned}\mathfrak{b} = \{&(-2axb+bxb),\ (xb-axb+bxb),\ (xb-axb+2xax),\\ &(xb-2axb+bxb+xax),\ (-axb+xax),\ (xb+xax),\ (2xb+bxb)\}.\end{aligned}$$

3. Build a monomial list. This list contains the terms that appears in $\mathfrak{b}$. This list, denoted by $\mathfrak{m}$, contains only monomial that are linearly independents:

$$\mathfrak{m} = \{xb,\ axb,\ bxb,\ xax\}.$$

4. Find a matrix G such that $\mathfrak{b} = G\mathfrak{m}$. For our example, G is given by

$$G^T = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 2\\ -1 & -2 & -1 & -2 & -1 & 0 & 0\\ 0 & 1 & 1 & 1 & 0 & 0 & 1\\ 2 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

5. Build a left list. This list contains the terms that multiplies δ from the left side in $L(\delta)$. Denote this list by $\mathfrak{a}$:

$$\begin{aligned}\mathfrak{a} = \{&(xb+axb),\ (xb+axb),\ (xb+axb),\ -(xb+axb),\\ &(c-xb-bxb+xax),\ (c+axb-bxb+xax),\ (c+xb+2axb-bxb+xax)\}.\end{aligned}$$

6. The expression $L(\delta)$ can now be rewritten as

$$L(\delta) = \sum_{j=1}^{4} \mathfrak{c}_j\,\delta\,\mathfrak{m}_j$$

with $\mathfrak{c} = G^T\mathfrak{a}$ given by

$$\begin{aligned}\mathfrak{c} = \{&3(c+xb+2axb-bxb+xax),\ (-c-3xb-4axb+bxb-xax),\\ &(c+4xb+5axb-bxb+xax),\ 2(c-xb-bxb+xax)\}.\end{aligned}$$

7. Build a monomial list from $\mathfrak{c}$. For the above example:

$$\bar{\mathfrak{m}} = \{c,\ xb,\ axb,\ bxb,\ xax\}.$$

8. Find a transformation matrix $\bar{G}$ such that $\mathfrak{c} = \bar{G}\bar{\mathfrak{m}}$:

$$\bar{G} = \begin{bmatrix} 3 & 3 & 6 & -3 & 3 \\ -1 & -3 & -4 & 1 & -1 \\ 1 & 4 & 5 & -1 & 1 \\ 2 & -2 & 0 & -2 & 2 \end{bmatrix}.$$

9. Decompose matrix $\bar{G}$ as $\bar{G} = QR$, with Q and R full rank matrices:

$$Q^T = \frac{1}{5}\begin{bmatrix} 15 & -5 & 5 & 10 \\ 3 & -11 & 16 & -18 \end{bmatrix}, \quad R = \frac{1}{5}\begin{bmatrix} 5 & 4 & 9 & -5 & 5 \\ 0 & 5 & 5 & 0 & 0 \end{bmatrix}.$$

10. The minimal Sylvester index N^* is the rank of $\bar{G}$. Thus, the final expression is

$$L(\delta) = \sum_j^{N^*} [R\bar{\mathfrak{m}}]_j \, \delta \, [Q^T \mathfrak{m}]_j \,. \tag{8.3}$$

For our example (8.2), the result is

$$L(\delta) = \frac{1}{5}(5\,c + 4\,xb + 9\,axb - 5\,bxb + 5\,xax)\,\delta\,(3\,xb - axb + bxb + 2\,xax) + \frac{1}{5}(xb + axb)\,\delta\,(3\,xb - 11\,axb + 16\,bxb - 18\,xax)$$

with the minimum Sylvester index guaranteed to be $N^* = 2$.

The implementation of our **MinimumSylvesterIndex**$[L(\delta)$**,** $\delta]$ command is described by these steps. When the original expression $L(\delta)$ contains a large number of Sylvester terms, the time spent on generating the matrix G in step 4 might be long. However, we emphasize that the expression $L(\delta)$ provided in step 6 can alternatively be provided by **NCRightSylvester[],** which is significantly faster than steps 1 through 6. In fact, this is how the MinimumSylvesterIndex command was implemented.

Theorem 8.3. *Provided that the $\mathfrak{a}_j$ and $\mathfrak{b}_j$ are polynomials, the lowest Sylvester index for $L(\delta)$ is given by N^*, which is the dimension of the span of $\mathfrak{c}_j$ for $j = 1, \ldots, d_{\mathfrak{b}}$, i.e., the rank of $\bar{G}$.*

Proof. This theorem follows immediately from Lemmas 8.4 and 8.6. □

Lemma 8.4. *The representation* (8.3) *produced by the MinimumSylvesterIndex algorithm has the property that the polynomials $[R\bar{\mathfrak{m}}]_1, \ldots, [R\bar{\mathfrak{m}}]_{N^*}$ are linearly independent and that the polynomials $[Q^T\mathfrak{m}]_1, \ldots, [Q^T\mathfrak{m}]_{N^*}$ are also linearly independent.*

Proof. Since the vectors $\mathfrak{m}_j$ are linearly independent and Q has full rank, the vectors $[Q^T\mathfrak{m}]_j$ for $j = 1, \ldots, N^*$ are linearly independent. Similarly, since the vectors $\bar{\mathfrak{m}}_j$ are linearly independent and R has full rank, the vectors $[R\bar{\mathfrak{m}}]_j$ for $j = 1, \ldots, N^*$ are linearly independent. □

Definition 8.5. *We call a dependence free Sylvester representation any Sylvester expression $L(\delta) = \sum \mathfrak{a}_j \, \delta \, \mathfrak{b}_j$ with $\mathfrak{a}_j$ linearly independent and $\mathfrak{b}_j$ also linearly independent.*

Lemma 8.6. *Let $L(\delta)$ and $\widetilde{L}(\delta)$ be Sylvester representations such that*

$$L(\delta) := \sum_{j=1}^{N} \mathfrak{a}_j \, \delta \, \mathfrak{b}_j = \sum_{k=1}^{\widetilde{N}} \widetilde{\mathfrak{a}}_k \, \delta \, \widetilde{\mathfrak{b}}_k =: \widetilde{L}(\delta).$$

If the polynomials $\mathfrak{a}_j$ are linearly independent, and if the polynomials $\mathfrak{b}_j$ are linearly independent, then for each $k = 1, \ldots, \widetilde{N}$ we have

$$\mathfrak{a}_j \in \operatorname{span}\{\widetilde{\mathfrak{a}}_k\}_1^{\widetilde{N}} \qquad \textit{and} \qquad \mathfrak{b}_j \in \operatorname{span}\left\{\widetilde{\mathfrak{b}}_k\right\}_1^{\widetilde{N}}.$$

Consequently $N \leq \widetilde{N}$, and if $\widetilde{L}(\delta)$ is also a dependence free Sylvester representation, then their Sylvester indexes are the same, $\widetilde{N} = N$.

Proof. Let β denote the maximum of the degrees of all of the polynomials $\mathfrak{a}_j, \mathfrak{b}_j, \widetilde{\mathfrak{a}}_k, \widetilde{\mathfrak{b}}_k$ for $j = 1, \ldots, N$ and $k = 1, \ldots, \widetilde{N}$. Let $\mathcal{P}(y)$ denote the space of all polynomials of degree less than or equal to β in $y = \{y_1, \ldots, y_g\}$. Let $\mathcal{P}(y)\delta$ denote all polynomials in the variables $\{y_1, y_2, \ldots, y_g, \delta\}$ of the form $p(y)\delta$ for $p \in \mathcal{P}(y)$. Since $\{\mathfrak{b}_j\}$ for $j = 1, \ldots, N$ is a linearly independent subset of the finite dimensional vector space $\mathcal{P}(y)$, there is an inner product $(\ ,\ )$ defined on $\mathcal{P}(y)$ satisfying

$$(\mathfrak{b}_i, \mathfrak{b}_j) = \begin{cases} 0, & i \neq j, \\ 1, & i = j\,. \end{cases} \tag{8.4}$$

For each $p \in \mathcal{P}(y)$, let us define a map $E : \mathcal{P}(y) \to \mathcal{P}(y)\delta$ for any Sylvester form by

$$E(L(\delta), p) := \sum_{j=1}^{N} \mathfrak{a}_j\, \delta\, (\mathfrak{b}_j, p) = \sum_{j=1}^{N} (\mathfrak{b}_j, p)\mathfrak{a}_j\, \delta\,.$$

With this notation, for each $\ell \leq N$ we obtain

$$E(L(\delta), \mathfrak{b}_\ell) = \sum_{j=1}^{N} \mathfrak{a}_j\, \delta\, (\mathfrak{b}_j, \mathfrak{b}_\ell) = \mathfrak{a}_\ell\, \delta\,.$$

Since $L(\delta) = \widetilde{L}(\delta)$ we have

$$E(L(\delta), \mathfrak{b}_\ell) = E(\widetilde{L}(\delta), \mathfrak{b}_\ell) \longrightarrow \mathfrak{a}_\ell\, \delta = \sum_{k=1}^{\widetilde{N}} (\widetilde{\mathfrak{b}}_k, \mathfrak{b}_\ell)\widetilde{\mathfrak{a}}_k\, \delta\,.$$

Thus, the polynomial $\mathfrak{a}_\ell$ is a linear combination of the polynomials $\widetilde{\mathfrak{a}}_k$, i.e., $\mathfrak{a}_\ell \in \operatorname{span}\{\widetilde{\mathfrak{a}}_k\}$. In a similar way, we can define an inner product $(\mathfrak{a}_i, \mathfrak{a}_j)$ satisfying property (8.4) and apply it to $L(\delta)$ to obtain that $\mathfrak{b}_\ell \in \operatorname{span}\{\widetilde{\mathfrak{b}}_k\}$. ∎

8.2.1. Rational coefficients. We have presented an algorithm which has the property that if the $\mathfrak{a}_j$ and $\mathfrak{b}_j$ are polynomials, then it always gives the lowest possible Sylvester index. However, in our optimization application the $\mathfrak{a}_j$ and $\mathfrak{b}_j$ may be rational functions. Thus, we shall describe how one can extend the algorithm to rational functions rather than polynomials.

The conceptual idea is to think of inverses of expressions as new variables, say, w_j. Then any rational expression is a polynomial in the original variables together with the new letters w_j. In this way, one can apply directly the Sylvester index minimizing algorithm. As an example, suppose that $L(\delta)$ is given by

$$L(\delta) = x(1-x)^{-1}\,\delta\, x - (1-x)^{-1}\,\delta\, x + \delta\, x. \tag{8.5}$$

Using the change of variable

$$w = (1-x)^{-1} \tag{8.6}$$

this expression can be written as

$$L(\delta) = xw\delta x - w\delta x + \delta x.$$

Now, one can apply the MinimumSylvesterIndex command to obtain

$$L(\delta) = (xw - w + 1)\delta x.$$

In this way, the Sylvester index for $L(\delta)$ was reduced to $N = 1$.

That is how the MinimumSylvesterIndex command is used in NCSDP. Unpleasantly, the algorithm did not take into account the "side relationships" that w_j and the other variables might satisfy, which for the above example is

$$(1 - x)^{-1} \equiv x(1 - x)^{-1} + 1$$

or in terms of w

$$xw - w + 1 = 0.$$

Consequently $L(\delta)$ is identically zero. Thus, our algorithm when applied to rational functions fails to produce a minimal Sylvester representation.

To some extent, we are not optimistic about finding a practical exact algorithm for $L(\delta)$ having rational coefficients, because noncommutative Gröbner basis algorithms are very time-consuming. However, we are looking into more empirical methods. One effective test for linear dependence is as follows. Suppose that

$$L(\delta) = \sum_{j=1}^{N} \mathfrak{a}_j \delta \mathfrak{b}_j$$

has already been reduced with the command MinimumSylvesterIndex. Then we replace the symbols appearing in the expressions for $\mathfrak{a}_j$ and for $\mathfrak{b}_j$ by matrices of large dimensions generated randomly. In this way, we obtain random large matrices $\mathcal{A}_j$ and $\mathcal{B}_j$. After, we build numerically the matrices A and B as follows:

$$A = \begin{bmatrix} \operatorname{vec}(\mathcal{A}_1) & \operatorname{vec}(\mathcal{A}_2) & \cdots & \operatorname{vec}(\mathcal{A}_N) \end{bmatrix},$$
$$B = \begin{bmatrix} \operatorname{vec}(\mathcal{B}_1) & \operatorname{vec}(\mathcal{B}_2) & \cdots & \operatorname{vec}(\mathcal{B}_N) \end{bmatrix}.$$

Naturally, N is the number of columns of A and B. Denoting by r_A the rank of the matrix A (respectively, r_B for the rank of B), then the minimum Sylvester index for $L(\delta)$ will be $\min(r_A, r_B)$. If the $\mathcal{A}_j$ and $\mathcal{B}_j$ are polynomials, then we know that r_A and r_B remains N. However, when the $\mathcal{A}_j$ and $\mathcal{B}_j$ are rational functions rather than polynomial, this might not be the case, as described by the example (8.5).

An optimization problem will be presented in section 8.3.1 in which the Sylvester index of the Hessian map is $N = 1043$. After applying the MinimumSylvesterIndex command, the Sylvester index was reduced to $N = 26$. However, $N = 26$ is not the lowest possible index for this Hessian map. When we apply the empirical procedure just described, we found that $r_A = 22$ and $r_B = 22$. Therefore, we know that the minimum Sylvester index is less than or equal to $N = 22$. An empirical algorithm along these lines for actually computing the dependences is under investigation.

Remark 8.7. Another step is taken in order to improve the overall timing, and it is not related to the idea of simplifying expressions by collecting terms, but it is valuable. At the symbolic level we look for inverses of matrices which appear repeatedly inside the symbolic expressions for the Hessian map and we replace each occurrence of an inverse by a new variable. In this way, all numerical inverses are evaluated only once at the beginning of the linear subproblem. This can considerably improve the overall run times.

8.3. Experiments with MinimumSylvesterIndex. The previous examples were presented to illustrate methods for reducing the Sylvester index. Now, we present numerical evidence validating the usefulness of these ideas. For this purpose, let us consider the following eigenvalue minimization problem, whose numerical behavior is to be presented in section 9:

$$\text{(P2)} \qquad \begin{array}{l} \inf \lambda_{\max}(CXC^T) \\ \text{subject to} \end{array}$$
$$0 < X,$$
$$0 < G(X) := A_3X + XA_3^T - XR_3^{-1}X + S_3,$$
$$0 < F(X) := A_1X + XA_1^T - XR_1^{-1}X + S_1 - (A_2^TX + XA_2)G(X)^{-1}(A_2^TX + XA_2)$$

with all the matrices having dimension $n \times n$.

As already described, we need to compute symbolically the Hessian and the gradient of an auxiliary potential function. For the above example, this potential function is given by the symbolic formula

$$\phi_\gamma(x) = -\log\det x - \log\det F(x) - \log\det G(x) - \log\det(\gamma\mathfrak{I}_d - cxc^T),$$

where $\mathfrak{I}_d$ stand for the symbolic analogue of the identity matrix and γ is a scalar which is not relevant here. The expression $\phi_\gamma(x)$ is a function of the unknown x. If the update direction is taken to be δ_x, the Hessian map $\mathcal{H}(\delta_x)$ as a function of δ_x will have a structure of the form

$$\mathcal{H}(\delta_x) = \sum_i^N \mathfrak{a}_i\delta_x\mathfrak{b}_i,$$

where the $\mathfrak{a}_i$ and $\mathfrak{b}_i$ are noncommutative rational functions of the variables c, a_1, a_2, a_3, r_1, r_3, s_1, s_3, x. At this stage, one can apply the MinimumSylvesterIndex command to reduce the Sylvester index N. The gradient map $\mathcal{Q}$ is obtained from the first directional derivative of $\phi_\gamma(x)$ along the direction δ_x. Thus, for this symmetric case, one obtains a "symbolic" system given by

$$\mathcal{H}(\delta_x) = \mathcal{Q} + \mathcal{Q}^T. \tag{8.7}$$

The next step is to substitute for matrices of compatible dimensions the symbols appearing in $\mathcal{H}(\delta_x)$ and $\mathcal{Q}$. Thus, the code becomes numerical, and to find numerically the update direction δ_X, we must be able to solve the linear system of equations given by

$$\mathbb{H}(\delta_X) = \mathbb{Q} + \mathbb{Q}^T.$$

Using the vec operation, the above system can be equivalently written as

$$Hv = g, \tag{8.8}$$

where $H = \sum_i^N \mathcal{B}_i^T \otimes \mathcal{A}_i$, $g = \mathrm{vec}(\mathbb{Q} + \mathbb{Q}^T)$, and $v = \mathrm{vec}(\delta_X)$.

Therefore, in order to solve numerically the linear system given in (8.7), one needs

FE: to substitute matrices for the symbols appearing in $\mathcal{H}(\delta_x)$ and $\mathcal{Q}$;
KP: to evaluate the Hessian matrix H by applying N Kronecker products;
LS: to solve the system $Hv = g$ for the update direction v.

TABLE 8.1
Timing (seconds): formulas evaluation, Kronecker products, and linear solver.

SIZE	16			32			64		
	MSI	CS	UNT	MSI	CS	UNT	MSI	CS	UNT
FE	0.071	0.096	0.409	0.258	0.287	1.09	1.47	1.74	6.1
KP	0.039	0.061	1.341	0.603	0.974	21.47	9.74	15.71	344.3
LS	0.029	0.028	0.029	0.397	0.410	0.41	9.65	9.87	9.8
TOT	0.139	0.185	1.779	1.258	1.671	22.96	20.85	27.31	360.1

Ratio	UNT / MSI	UNT / MSI	UNT / MSI
FE	5.8	4.2	4.1
KP	34.4	35.6	35.3
TOT	12.8	18.3	17.3

The first two steps, namely, FE (formula evaluations) and KP (Kronecker products), are the two main steps where reducing the Sylvester index N of the expression for $\mathcal{H}(\delta_x)$ can significantly affect the evaluation time. We do not show the formulas for $\mathcal{H}(\delta_x)$ and $\mathcal{Q}$, since these expressions are quite large and would consume several pages. What is important is the fact that the formula for $\mathcal{H}(\delta_x)$ as computed originally, before applying any simplification rule, has $N = 1014$ Sylvester terms. However, after applying the NCCollectSylvester command to the original expression, the Sylvester index decreases to $N = 43$, and after applying our MinimumSylvesterIndex command to the original expression, we obtain the index $N = 26$ for this Hessian.

8.3.1. Time saved by applying MinimumSylvesterIndex. To find out how much time is actually saved at the numerical level, the NCSDP code is executed using the collected formulas for $\mathcal{H}(\delta_x)$ with $N = 26$ (MinimumSylvesterIndex command) and with $N = 43$ (NCCollectSylvester command) and the not collected formula for $\mathcal{H}(\delta_x)$ with $N = 1014$. For this set of experiments, the size n of the matrices involved assume the values $n = 16, 32, 64$. For each case, we execute the inner loop where the linear system (8.8) is numerically solved 20 times. Thus, we measure the CPU time per call (average over 20 iterations) spent on the above items, FE, KP, and LS (linear solver). In this way, we can analyze how the time spent on formula evaluations behaves as a function of the size of the matrices involved in the expressions, as well as the Sylvester index N.

The results are presented in Table 8.1, where MSI stands for the Hessian simplified by MinimumSylvesterIndex (the Sylvester index is $N = 26$), CS stands for the Hessian simplified by NCCollectSylvester (the Sylvester index is $N = 43$), and UNT stands for the untreated Hessian (the Sylvester index is $N = 1014$). In this table, the row labeled Ratio is the ratio between the untreated column and the MSI column. The time spent on solving the linear system, presented in the row labeled LS, is not affected by the expression being or not being collected. SIZE stands for matrix size, and TOT for the total time FE+KP+LS.

The results provided in Table 8.1 show that collecting terms in the expression for the Hessian map $\mathcal{H}(\delta_x)$ represents a huge saving, since the average time spent on substituting matrices for the symbols that appear in the expressions for the $\mathfrak{a}_i$ and $\mathfrak{b}_i$ when the expressions are not collected (UNT case) is approximately four to five times longer than the time for the MSI collected case (row FE in Ratio). Moreover, collecting the expressions significantly improved the time spent on evaluating Kronecker products, as seen from row KP under Ratio, where this timing improved by a factor ranging from 34.4 to 35.6.

For matrices of dimension 16 and 32, the time per call spent (over 20 iterations) on numerically solving the equation $Hv = g$ for the unknown v was relatively small, as seen from row LS. On the other hand, for matrices of dimension 64, the (LS) cost becomes significantly large. To understand this fact better, suppose the dimension of the matrices involved is chosen to be $n = 64$. Thus, the symmetric unknown matrix X having size 64×64 implies that the unknown vector v and the system to be solved will have size approximately $64^2/2 = 2048$.

Kronecker products are also extremely expensive, as seen from row KP for size 64. In fact, if we did not have a theory for decreasing the Sylvester index, our approach using Kronecker products would be intractable for matrices of large dimensions. If one could solve the linear system of equations for δ_X in its original structured form $\mathbb{H}(\delta_X) = \mathbb{Q}$, without applying Kronecker products and keeping the dimension of the linear system low, a huge saving on the numerical linear solver would probably be attained. This is an open area which we hope members of the community will pursue.

8.4. Some more experiments using MinimumSylvesterIndex. Another interesting experiment is to analyze how the Sylvester index behaves by applying our MinimumSylvesterIndex command to a variety of matrix inequalities which appear in control design. The example just presented, taken from section 9, has shown a great improvement since the Sylvester index reduced from $N = 1014$ to $N = 26$. Now, we present two more examples.

Example 8.8. For the standard Riccati inequality

$$AX + XA^T - XRX + S > 0$$

the Hessian map $\mathcal{H}(\delta_x)$ for the untreated case has a Sylvester index of $N = 20$, while our MinimumSylvesterIndex algorithm applied to it provides a Sylvester index of $N = 4$.

Example 8.9. Now, a more realistic example is used: a mixed H_2/H_∞ control problem

$$\text{(P3)} \qquad \begin{array}{l} \inf \operatorname{Tr}\{W\} \\ \text{subject to} \\ \qquad\qquad 0 < X, \\ \qquad 0 < W - (C_2X + D_{2u}F)X^{-1}(C_2X + D_{2u}F)^T, \\ 0 > AX + XA^T + B_uF + F^TB_u^T + B_wB_w^T \\ \qquad + \left[XC_1^T + F^TD_{1u}^T + B_wD_{1w}^T\right] R^{-1} \left[XC_1^T + F^TD_{1u}^T + B_wD_{1w}^T\right]^T \end{array}$$

with $R = \eta^2 I - D_{1w}D_{1w}^T > 0$.

For the above control problem, there are three unknowns denoted by $W = W^T$, $X = X^T$, and F (not symmetric). Thus, the linear subproblem to be solved will have dimension 3×3, and consequently each entry on this system will contain a Sylvester operator. For instance, the (1,1) entry will be an expression of the form

$$\sum_i^{N_{11}} \mathfrak{a}_i^{11} \delta_w \mathfrak{b}_i^{11} + \sum_i^{\hat{N}_{11}} \hat{\mathfrak{a}}_i^{11} {\delta_w}^T \hat{\mathfrak{b}}_i^{11}.$$

The (1,2) entry will have the form $\sum_i^{N_{12}} \mathfrak{a}_i^{12} \delta_x \mathfrak{b}_i^{12} + \sum_i^{\hat{N}_{12}} \hat{\mathfrak{a}}_i^{12} \delta_x^T \hat{\mathfrak{b}}_i^{12}$. The (1,3) entry will have the form $\sum_i^{N_{13}} \mathfrak{a}_i^{13} \delta_f \mathfrak{b}_i^{13} + \sum_i^{\hat{N}_{13}} \hat{\mathfrak{a}}_i^{13} \delta_f^T \hat{\mathfrak{b}}_i^{13}$. The (2,1) entry is the adjoint case

TABLE 8.2
Sylvester index N and $\hat{N}$ for the MSI Hessian $\mathcal{H}(\delta_w, \delta_x, \delta_f)$.

	Sylv. index N_{ij}			Sylv. index $\hat{N}_{ij}$		
i,j	1	2	3	1	2	3
1	1	2	1	0	0	1
2	2	10	4	0	0	4
3	1	4	2	1	4	4

TABLE 8.3
Sylvester index N and $\hat{N}$ for the UNT Hessian $\mathcal{H}(\delta_w, \delta_x, \delta_f)$.

	Sylv. index N_{ij}			Sylv. index $\hat{N}_{ij}$		
i,j	1	2	3	1	2	3
1	1	2	2	0	0	2
2	2	73	38	0	0	38
3	2	38	42	2	38	38

of the (1,2) entry. The (2,2) entry will have the form $\sum_i^{N_{22}} \mathfrak{a}_i^{22} \delta_x \mathfrak{b}_i^{22} + \sum_i^{\hat{N}_{22}} \hat{\mathfrak{a}}_i^{22} \delta_x^T \hat{\mathfrak{b}}_i^{22}$ and so forth. It should be noticed that the Sylvester index $\hat{N}_{11}$, $\hat{N}_{12}$, $\hat{N}_{21}$, and $\hat{N}_{22}$ are zero, since the corresponding variables w and x are symmetric.

For the matrix inequalities given in problem (P3), the set of Sylvester indexes N and $\hat{N}$ for the Hessian map $\mathcal{H}(\delta_w, \delta_x, \delta_f)$ simplified by MinimumSylvesterIndex (MSI case) and for the untreated Hessian (UNT case) are respectively presented in Table 8.2 and Table 8.3.

In Tables 8.2 and 8.3, the variables x and f are associated with the entries

$$\text{(i,j)} \in \{(2,2) \quad (2,3) \quad (3,2) \quad (3,3)\}$$

for each one of the subtables. If we only pay attention to the Sylvester index N, we see that the submatrix associated with x and f for the

UNT case $\begin{array}{|c|c|}\hline 73 & 38 \\ \hline 38 & 42 \\ \hline \end{array}$ reduces to only $\begin{array}{|c|c|}\hline 10 & 4 \\ \hline 4 & 2 \\ \hline \end{array}$ in the MSI case.

Similarly, a large reduction is also obtained for the Sylvester index $\hat{N}$. Thus, for the variables x and f, we found that a large reduction on the Sylvester index N and $\hat{N}$ are obtained after applying our MinimumSylvesterIndex command. Naturally, this will represent a considerable saving on the evaluation time for the numerical linear solver.

It is also true that the process of simplifying rational functions, at the symbolic level of Mathematica, can consume a considerable amount of time. However, this computation is performed only once at the beginning of the run. This is in contrast with the numerical part, where solving the linear system to provide the update direction takes place at each inner iteration (which occurs several times). Therefore, the ability to collect factors in an expression (decreasing the Sylvester index) plays a very important role.

9. Numerical experiment: Timing of NCSDP. In this section, our NCSDP code is numerically compared to some available semidefinite programming solvers.

TABLE 9.1
Total CPU time in seconds.

	Matrix Size			
	8	16	32	64
SDPT3	2.59	12.94	163.77	3132.68
LMI-Lab	0.43	2.58	66.03	2124.54
SeDuMi	0.79	2.20	33.63	1254.30
NCSDP	7.73	12.57	81.40	1224.49

TABLE 9.2
CPU time per iteration in seconds.

	Matrix Size					
	16		32		64	
	CPI	IT	CPI	IT	CPI	IT
SDPT3	1.62	8	18.20	9	348.08	9
SeDuMi	0.20	11	2.59	13	89.59	14
NCSDP	0.60	21	4.28	19	72.03	17

For this purpose, the optimization problem to be used is the following eigenvalue minimization problem (stated earlier in section 8.3.1):

$$\text{(P2)} \qquad \begin{array}{l} \inf \lambda_{\max}(CXC^T) \\ \text{subject to} \\ \qquad 10^{-1}I < X, \\ \qquad 0 < G(X) := A_3X + XA_3^T - XR_3^{-1}X + S_3, \\ 0 < F(X) := A_1X + XA_1^T - XR_1^{-1}X + S_1 - (A_2^TX + XA_2)G(X)^{-1}(A_2^TX + XA_2). \end{array}$$

The matrices C, A_1, A_2, and A_3 belong to $\mathbb{R}^{n\times n}$, the invertible matrices R_1, R_3 belong to $\mathbb{S}^n_{++}$ and the matrices S_1, S_3, and X belong to $\mathbb{S}^n$. We do not present the numerical values of those matrices since it would take considerable space. Note that by Schur complement techniques problem (P2) can be equivalently restated as an LMI problem.

The results of this experiment (shown in Table 9.1) show the overall CPU time spent by the solvers SDPT3, LMI-Lab, SeDuMi, and NCSDP to solve the above optimization problem (P2) within the required accuracy of 10^{-4} for the objective value. The LMI-Lab toolbox (Version 1.0.8) is based on the projective method of [8]. The SeDuMi solver (Version 1.02) from [25] implements the self-dual embedding technique for optimization over self-dual homogeneous cones. The SDPT3 solver (Version 3.0) from [28] implements an infeasible path-following algorithm that employs a predictor-corrector method. The starting feasible points were the same for all the solvers.

From Table 9.1, one sees that for matrices of size 64, the solvers SeDuMi and NCSDP were the fastest code for the eigenvalue minimization problem (P2). The CPU times per iteration (CPI) and number of (outer) iterations (IT) are presented in Table 9.2. We believe that NCSDP might be significantly faster than SeDuMi for matrices of dimensions larger than 64×64. However, we did not run this experiment

TABLE 9.3
Numerical behavior of NCSDP.

SIZE	IT/NeNe	FE	KP	LS	g	$\lambda_{\min}(H)$	$\lambda_{\max}(H)$
8	25/94	2.32	0.07	0.14	$1.14E+04$	$6.74E+03$	$2.59E+10$
16	21/75	3.05	2.88	2.16	$8.94E+03$	$1.62E+03$	$2.41E+10$
32	19/69	8.24	41.9	26.7	$7.38E+03$	$1.14E+03$	$2.24E+10$
64	17/61	43.4	595	580	$5.84E+03$	$9.81E+02$	$2.76E+10$

since the overall elapsed time would be extremely long and because of the requirement of large RAM memory availability. The computer used for our experiments was an Intel Celeron at 2800 MHz CPU clock, 512MB of RAM, 1GB of swap, running Linux (kernel 2.4.20-31.9), MATLAB Version 6.5.0 R13, Mathematica 4.0, and NCAlgebra Version 3.7.

We believe our NCSDP code, even in its raw stage, has been competitive mainly because it allows nonlinear matrix inequalities, so it avoids the increase in dimensions when converting to LMIs using Schur complements. Also, we think that the techniques in the paper allow the numerical Newton equations to be derived more efficiently. However, we did not take advantage of the special structure of the linear subproblem when solving it.

For these experiments, we installed the codes listed above using their default installer. However, since these codes are for general SDP problems, where the input data should be expressed in a "standard" SDP form, which is not the standard LMI form (like the input from LMILab), we make use of the package LMILab Translator (LMITrans) that translates from LMILab form to the SeDuMi and SDPT3 form. The timing presented in Table 9.1 did not incorporate the (modest amount of) time consumed by this interface.

We do not know if LMITrans does nearly the "optimal" conversion for each solver; this adds uncertainty to the experiments. It might be the case that there exists an optimal conversion for a particular solver, in this case, the solver would perform better than for the default options used in LMITrans. However, we also used YALMIP as a front-end for SeDuMi and SDPT3 at the suggestion of a referee after the paper was complete. In a few tests, we found that it did not affect the timings significantly: the timing was approximately the same as the timing obtained using LMITrans.

9.1. Numerical behavior of NCSDP. We now provide the numerical details of the results from NCSDP presented in Table 9.1. The trade-off between the number of inner iterations, NeNe, and the number of outer iterations, Iter (as seen from Table 9.3), is a characteristic of Barrier methods, in particular, the method of centers [2], and depends mainly on the centralization parameter θ, given in (6.1) from section 6.2. In practice, a smaller θ induces a higher number of inner iterations, NeNe. For all the experiments, θ was set to $\theta = 0.2$.

For matrices of small size, the most expensive part is the time spent on evaluating the Sylvester terms $\mathfrak{a}_i$, $\mathfrak{b}_i$, and $\mathcal{Q}$, presented in column FE. However, when the size of the matrices increases above 8, the time spent on Kronecker products, column KP, and the time spent on solving the linear system, column LS, begins to dominate.

In Table 9.3, the column g stands for the gradient, and columns $\lambda_{\min}(H)$ and $\lambda_{\max}(H)$ stand, respectively, for the minimum and the maximum eigenvalues of the

Hessian matrix H at the optimum. Those values show that the condition number of the Hessian is large at the optimal solution. This ill-conditioning in the Hessian is a well-known fact for classical barrier methods [30, 18], where it has been shown that this behavior is highly influenced by the set of constraints that are or are not active (binding) at the solution. However, it is not an immediate task to determine the set of active constraints in the semidefinite programming framework.

9.2. Implementation speed-ups. At this stage, the numeric part of our NCSDP code is "completely" implemented using MATLAB functions (not compiled). The only compiled part of our code is the Kronecker product, since the MATLAB function *kron.m* was extremely slow for our needs. On the other hand, most of the other solvers have their core subroutines written in either Fortran or C, which significantly improve their overall performance.

To make the experiment transparent, the stopping criteria for the inner loop in NCSDP is kept constant throughout all the iterations. We stop as soon as $\sigma = 1$ (see section 6.3). Changing dynamically this stopping criteria might also improve the timing of the solver.

Although in this paper we have focused on convex optimization problems over matrix inequalities, the extension of our numerical ideas to finding local solutions in nonconvex situations is immediate; however, reliability has not been tested [4].

We reiterate that fast methods for solving numerical linear equations of Sylvester form have not been investigated and are the main open question motivated by this paper. A big advance here would translate directly into a big reduction in run times.

Appendix A. Illustrating our methodology by an example. In this section we explain the ideas behind Theorem 7.1 and Corollary 7.2 through a simple optimization problem. The extrapolation to a more general case is straightforward, but it gives messy formulas. Let us consider the optimization problem

$$\min \{\mathrm{Tr}\,\{X\} \,:\, X \in \mathrm{closure}(\mathcal{G})\} \tag{A.1}$$

with $F(X) := AX + XA^T - XRX + Q$, and the domain $\mathcal{G}$ given by

$$\mathcal{G} = \{X \in \mathbb{S}^n \,:\, F(X) > 0\}\,.$$

We assume (1) all matrices have dimension $n \times n$; (2) the matrices X, R, Q are symmetric; (3) the closure of the set $\mathcal{G}$ is compact.

A.1. Describing the central path. Let us define the unconstrained auxiliary potential function $\phi_\gamma(X)$ as described in Theorem 7.1 as

$$\phi_\gamma(X) = \zeta \log\Big(1/(\gamma - \mathrm{Tr}\,\{X\})\Big) - \log\det F(X). \tag{A.2}$$

The analytic center for the potential $\phi_\gamma(X)$ is the path given by

$$X^*(\gamma^k) = \mathrm{argmin}\,\phi_{\gamma^k}(X). \tag{A.3}$$

A.2. Solving for the analytic center. The above optimization problem (A.1) has now been replaced by a sequence of unconstrained minimization problems in the form (A.3). In this way, we are interested in finding update directions which lead toward the central path of (A.3). To find those directions, Newton's method is applied to minimize the second-order Taylor series expansion of the potential function $\phi(x)$.

These procedures are summarized in section 7.1. Here, we go through each step precisely. For clarity of notation, we omit the subscript γ in $\phi_\gamma(x)$. To compute the quadratic approximation of $\phi(x)$, we take δ_x to be the update direction for x. Thus, assuming $x^* = x + \delta_x$, the series expansion of $\phi(x)$ up to the second term is given by

$$\tilde{\phi}(\delta_x) := \phi(x^*) - \phi(x) = D\phi(x)\,[\delta_x] + \frac{1}{2}D^2\phi(x)\,[\delta_x, \delta_x]\,. \tag{A.4}$$

A.3. Directional eerivatives of $F(x)$. In order to compute the derivatives in (A.4), we need to have at hand the first and second directional derivatives of $F(x)$. Recalling that x is symmetric, and therefore so is the update direction δ_x, the first directional derivative of $F(x)$ in the direction δ_x is given by

$$DF(x)\,[\delta_x] = (a - xr)\delta_x + \delta_x(a^T - rx) = \operatorname{sym}\{(a - xr)\delta_x\}$$

and the second directional derivative is

$$D^2F(x)\,[\delta_x, \delta_x] = -\operatorname{sym}\{\delta_x r \delta_x\}$$

A.4. Connection with Theorem 7.1. Comparing the above derivatives of $F(x)$ with the formulas (4.2) and (4.3), one readily verifies that $k = 1$, $a_1 = (a - xr)$, and $b_1 = 1$, for the first directional derivative, and that $w_1 = 1, w_2 = 0, w_3 = 0$, $m_1 = 1$, $n_1 = -r$, and $t_1 = 1$, for the second directional derivative. With this notation, we can directly apply Theorem 7.1 to obtain the algebraic linear system of equations. For the gradient term $\mathcal{Q}$ we have

$$\begin{aligned}\mathcal{Q} &= \sum_{i=1}^{k} a_i^T F(x)^{-1} b_i^T - \frac{1}{2}\zeta\,(\gamma - \operatorname{Tr}\{x\})^{-1}\,\mathfrak{I}_d \\ &= (a - xr)^T F(x)^{-1} - \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-1}\mathfrak{I}_d.\end{aligned}$$

For the Hessian $\mathcal{H}(\delta_x)$ we calculate

1. $\sum_{i=1}^{k}\sum_{j=1}^{k} a_i^T F(x)^{-1} a_j \delta_x b_j F(x)^{-1} b_i^T = (a - xr)^T F(x)^{-1}(a - xr)\delta_x F(x)^{-1}$;
2. $\sum_{i=1}^{k}\sum_{j=1}^{k} a_i^T F(x)^{-1} b_j^T \delta_x^T a_j^T F(x)^{-1} b_i^T = (a - xr)^T F(x)^{-1}\delta_x(a - xr)F(x)^{-1}$;
3. $-\sum_{j=1}^{w_1} n_j^T \delta_x^T m_j^T F(x)^{-1} t_j^T + m_j^T F(x)^{-1} t_j^T \delta_x^T n_j^T = r\delta_x F(x)^{-1} + F(x)^{-1}\delta_x r$.

Thus $\mathcal{H}(\delta_x)$ becomes

$$\begin{aligned}\mathcal{H}(\delta_x) = {} & \frac{1}{2}F(x)^{-1}\delta_x r + \frac{1}{2}r\delta_x F(x)^{-1} + (a - xr)^T F(x)^{-1}(a - xr)\delta_x F(x)^{-1} \\ & + (a - xr)^T F(x)^{-1}\delta_x(a - xr)^T F(x)^{-1} + \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-2}\operatorname{Tr}\{\delta_x\}\,\mathfrak{I}_d.\end{aligned}$$

Consequently, the algebraic linear system of equations is described by

$$\operatorname{Tr}\left\{\delta_V(\mathcal{H}(\delta_x) - \mathcal{Q})^T + (\mathcal{H}(\delta_x) - \mathcal{Q})\delta_V\right\} = 0 \tag{A.5}$$

with $\mathcal{H}(\delta_X)$ and $\mathcal{Q}$ as given above

These are the steps someone would need in order to apply Theorem 7.1 directly. However, we shall go through the details of the manipulation that leads to this main result.

A.5. Directional derivatives of the barrier function. Having the above directional derivatives of $F(x)$ available, we are ready to take the directional derivatives needed in (A.4). However, to clarify the exposition, we split the potential function into two parts:

$$\phi_1(x) = -\log\det F(x) \quad \text{and} \quad \phi_2(x) = \zeta \log\Big(1/(\gamma - \operatorname{Tr}\{x\})\Big).$$

A.6. Symbolic directional derivatives of $\phi_1(x) = -\log\det F(x)$. The first and second directional derivative of $\phi_1(x)$ in the direction δ_x are given by

$$\begin{aligned} D\phi_1(x)[\delta_x] &= -\operatorname{Tr}\left\{F(x)^{-1} DF(x)[\delta_x]\right\} \\ &= -\operatorname{Tr}\left\{F(x)^{-1}\operatorname{sym}\{(a - xr)\delta_x\}\right\}, \end{aligned}$$

$$\begin{aligned} D^2\phi_1(x)[\delta_x,\delta_x] &= \operatorname{Tr}\left\{\left(F(x)^{-1}DF(x)[\delta_x]\right)^2\right\} - \operatorname{Tr}\left\{F(x)^{-1}D^2F(x)[\delta_x,\delta_x]\right\} \\ &= \operatorname{Tr}\left\{\left(F(x)^{-1}\operatorname{sym}\{(a-xr)\delta_x\}\right)^2\right\} \\ &\quad + \operatorname{Tr}\left\{F(x)^{-1}\operatorname{sym}\{\delta_x r \delta_x\}\right\}. \end{aligned}$$

A.7. Symbolic directional derivatives of $\phi_2(x) = \zeta\log(1/(\gamma - \operatorname{Tr}\{x\}))$. The first derivative is given by

$$D\phi_2(x)[\delta_x] = \zeta\left(\gamma - \operatorname{Tr}\{x\}\right)^{-1}\operatorname{Tr}\{\delta_x\}$$

and the second by

$$D^2\phi_2(x)[\delta_x,\delta_x] = \zeta\left(\left(\gamma - \operatorname{Tr}\{x\}\right)^{-1}\operatorname{Tr}\{\delta_x\}\right)^2.$$

A.8. Optimality conditions. Now we are ready to write the optimality conditions which will provide the update direction. These conditions are the first-order necessary optimality conditions for problem (A.3), obtained by taking directional derivatives of the Taylor expansion $\phi(x+\delta_x)$, given by (A.4), as a function of δ_x in the direction δ_V. To accomplish this step, we should compute $D\tilde{\phi}(\delta_x)[\delta_V]$ or equivalently

$$D\left(D\phi(x)[\delta_x] + \frac{1}{2}D^2\phi(x)[\delta_x,\delta_x]\right)[\delta_V] = 0. \tag{A.6}$$

Using the directional derivatives just computed in the previous sections, the expression for the second-order approximation $\tilde{\phi}(\delta_x)$ is given

$$\begin{aligned} \tilde{\phi}(\delta_x) &= -\operatorname{Tr}\left\{F(x)^{-1}\operatorname{sym}\{(a-xr)\delta_x\}\right\} + \frac{1}{2}\operatorname{Tr}\left\{\left(F(x)^{-1}\operatorname{sym}\{(a-xr)\delta_x\}\right)^2\right\} \\ &\quad + \frac{1}{2}\operatorname{Tr}\left\{F(x)^{-1}\operatorname{sym}\{\delta_x r\delta_x\}\right\} + \zeta\left(\gamma - \operatorname{Tr}\{x\}\right)^{-1}\operatorname{Tr}\{\delta_x\} \\ &\quad + \frac{1}{2}\zeta\left(\left(\gamma - \operatorname{Tr}\{x\}\right)^{-1}\operatorname{Tr}\{\delta_x\}\right)^2. \end{aligned} \tag{A.7}$$

To proceed, we now set to zero the directional derivative of $\tilde{\phi}(\delta_x)$ as a function of δ_x in the direction δ_V. After a few manipulations, the term $D\tilde{\phi}(\delta_x)\,[\delta_V]$ is given by

$$\begin{aligned}
D\tilde{\phi}(\delta_x)\,[\delta_V] = \operatorname{Tr}\Big\{ \delta_V \Big(& F(x)^{-1}\delta_x(a-xr)^T F(x)^{-1}(a-xr) - F(x)^{-1}(a-xr) \\
& + \frac{1}{2} r\delta_x F(x)^{-1} + \frac{1}{2} F(x)^{-1}\delta_x r \\
& + F(x)^{-1}(a-xr)\delta_x F(x)^{-1}(a-xr) \\
& + \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-1}\mathfrak{I}_d + \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-2}\operatorname{Tr}\{\delta_x\}\,\mathfrak{I}_d \Big) \\
+ \Big(& (a-xr)^T F(x)^{-1}(a-xr)\delta_x F(x)^{-1} - (a-xr)^T F(x)^{-1} \\
& + \frac{1}{2} F(x)^{-1}\delta_x r + \frac{1}{2} r\delta_x F(x)^{-1} \\
& + (a-xr)^T F(x)^{-1}\delta_x (a-xr)^T F(x)^{-1} \\
& + \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-1}\mathfrak{I}_d + \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-2}\operatorname{Tr}\{\delta_x\}\,\mathfrak{I}_d \Big)\delta_V \Big\}.
\end{aligned}$$

Therefore, the algebraic linear system of equations is described by

$$\operatorname{Tr}\left\{\delta_V(\mathcal{H}(\delta_x) - \mathcal{Q})^T + (\mathcal{H}(\delta_x) - \mathcal{Q})\delta_V\right\} = 0 \tag{A.8}$$

with $\mathcal{H}(\delta_X)$ and $\mathcal{Q}$ respectively given by

$$\mathcal{Q} = (a-xr)^T F(x)^{-1} - \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-1}\mathfrak{I}_d,$$

$$\begin{aligned}
\mathcal{H}(\delta_x) = & \frac{1}{2} F(x)^{-1}\delta_x r + \frac{1}{2} r\delta_x F(x)^{-1} + (a-xr)^T F(x)^{-1}(a-xr)\delta_x F(x)^{-1} \\
& + (a-xr)^T F(x)^{-1}\delta_x (a-xr)^T F(x)^{-1} + \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-2}\operatorname{Tr}\{\delta_x\}\,\mathfrak{I}_d.
\end{aligned}$$

A.9. Connection with Theorem 7.1. We shall emphasize that this illustrative example gives a reasonable idea of how the proof of Theorem 7.1 was constructed, since it follows very similar steps.

A.10. The algebraic linear system of equations. Since the unknown x is restricted to being symmetric (so is δ_x) the subspace $\mathcal{V}$ equals the space of symmetric matrices. Consequently, its orthogonal complement $\mathcal{V}^\perp$ is the set of all skew symmetric matrices. Therefore, we obtain the following linear system in δ_x:

$$\mathcal{H}(\delta_x) + \mathcal{H}(\delta_x)^T = \mathcal{Q} + \mathcal{Q}^T.$$

We can rewrite this equation using a suitable choice of variables $\mathfrak{a}_i$ and $\mathfrak{b}_i$ as follows:

$$\operatorname{sym}\left\{\sum_{i=1}^{4}\left(\mathfrak{a}_i\delta_x\mathfrak{b}_i\right)\right\} + \varrho\operatorname{Tr}\{\delta_x\} = \mathcal{Q} + \mathcal{Q}^T \tag{A.9}$$

with

$$\begin{aligned}
\mathfrak{a}_1 &= F(x)^{-1}(a - xr), & \mathfrak{b}_1 &= F(x)^{-1}(a - xr),\\
\mathfrak{a}_2 &= F(x)^{-1}, & \mathfrak{b}_2 &= (a - xr)^T F(x)^{-1}(a - xr),\\
\mathfrak{a}_3 &= \frac{1}{2}F(x)^{-1}, & \mathfrak{b}_3 &= r,\\
\mathfrak{a}_4 &= r, & \mathfrak{b}_4 &= \frac{1}{2}F(x)^{-1},
\end{aligned}$$

and

$$\mathcal{Q} = (a - xr)^T F(x)^{-1} - \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-1}\mathfrak{I}_d, \qquad \varrho = \frac{1}{2}\zeta(\gamma - \operatorname{Tr}\{x\})^{-2}\mathfrak{I}_d.$$

A.11. Connection with Corollary 7.2. These are the $\mathfrak{a}_i$ and $\mathfrak{b}_i$ described in the corollary for the specific case where the subspace $\mathcal{V}$ equals the space of symmetric matrices $\mathbb{S}^n$. The proof of Corollary 7.2 is illustrated by our example, since the proof mainly consists in determining the orthogonal complement of the subspace $\mathcal{V}$.

Acknowledgments. Thanks are due to M. C. de Oliveira for many ideas and suggestions. We also thank the referees for their very valuable comments and suggestions.

REFERENCES

[1] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton, *Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results*, SIAM J. Optim., 8 (1998), pp. 746–768.

[2] S. Boyd and L. El Ghaoui, *Method of centers for minimizing generalized eigenvalues*, Linear Algebra Appl., 188 (1993), pp. 63–111.

[3] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, New York, 2004.

[4] J. F. Camino, *Optimization over Convex Matrix Inequalities*, Ph.D. thesis, University of California, San Diego, 2003.

[5] J. F. Camino, J. W. Helton, R. E. Skelton, and J. Ye, *Matrix inequalities: A symbolic procedure to determine convexity automatically*, Integral Equations Operator Theory, 46 (2003), pp. 399–454.

[6] L. El Ghaoui and S. Niculescu, *Advances in Linear Matrix Inequality Methods in Control*, Adv. Des. Control, SIAM, Philadelphia, 1999.

[7] A. V. Fiacco and G. P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, Classics Appl. Math., SIAM, Philadelphia, 1990.

[8] P. Gahinet, A. Nemirovskii, A. J. Laub, and M. Chilali, *LMI Control Toolbox*, The Math Works, Natick, MA, 1995.

[9] G. Golub and C. V. Loan, *Matrix Computation*, Johns Hopkins University Press, Baltimore, 1983.

[10] E. L. Green, *Multiplicative bases, Gröbner bases, and right Gröbner bases*, J. Symbolic Comput., 29 (2000), pp. 601–623.

[11] J. W. Helton and J. J. Wavrik, *Rules for computer simplification of the formulas in operator model theory and linear systems*, in Nonselfadjoint Operators and Related Topics Oper. Theory Adv. Appl. 73, Birkhäuser, Basel, 1994, pp. 325–354.

[12] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, New York, 1999.

[13] K. Jbilou and A. Messaoudi, *Matrix recursive interpolation algorithm for block linear systems direct methods*, Linear Algebra Appl., (1999), pp. 137–154.

[14] M. Konstantinov, V. Mehrmann, and P. Petkov, *On properties of Sylvester and Lyapunov operators*, Linear Algebra Appl., 312 (2000), pp. 35–71.

[15] F. Leibfritz and E. M. Mostafa, *An interior point constrained trust region method for a special class of nonlinear semidefinite programming problems*, SIAM J. Optim., 12 (2002), pp. 1048–1074.

[16] A. S. Lewis and M. L. Overton, *Eigenvalue optimization*, Acta Numer., 5 (1996), pp. 149–190.
[17] P. A. Linnell, *Noncommutative Localization in Group Rings*, arXiv: Math. RA/0311071.
[18] W. Murray, *Analytic expression for the eigenvalues and eigenvectors of the Hessian matrices of barrier and penalty functions*, J. Optim. Theory Appl., 7 (1971), pp. 189–196.
[19] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*, Stud. Appl. Math., SIAM, Philadelphia, 1994.
[20] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Classics in Appl. Math., SIAM, Philadelphia, 2000.
[21] M. L. Overton, *On minimizing the maximum eigenvalue of a symmetric matrix*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 256–268.
[22] P. A. Parrilo, *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*, Ph.D. thesis, California Institute of Technology, 2000.
[23] R. E. Skelton, T. Iwasaki, and K. M. Grigoriadis, *A Unified Algebraic Approach to Linear Control Design*, Taylor & Francis, London, 1998.
[24] M. Stankus, J. W. Helton, and J. Wavrik, *Computer simplification of formulas in linear systems theory*, IEEE Trans. Automat. Control, 43 (1998), pp. 302–314.
[25] J. F. Sturm, *Using SeDuMi* 1.02*, a MATLAB toolbox for optimization over symmetric cones*, Optim. Methods Softw., 11/12 (1999), pp. 625–653.
[26] J. F. Sturm, *Implementation of interior point methods for mixed semidefinite and second order cone optimization problems*, Optim. Methods Softw., 17 (2002), pp. 1105–1154.
[27] M. J. Todd, *Semidefinite optimization*, Acta Numer., 10 (2001), pp. 515–560.
[28] K. C. Toh, M. J. Todd, and R. H. Tütüncü, *SDPT*3*—A MATLAB software package for semidefinite programming, version* 2.1, Optim. Methods Softw., 11 (1999), pp. 545–581.
[29] H. Wolkowicz, R. Saigal, and L. Vandenberghe, eds., *Handbook of Semidefinite Programming*, Internat. Ser. Oper. Res. Management Sci. 27, Kluwer Academic Publishers, Boston, 2000.
[30] M. H. Wright, *Interior methods for constrained optimization*, Acta Numer., 1 (1992), pp. 341–407.