Computer Algebra Tailored to Matrix Inequalities in Control [*]

M. C. de Oliveira and J. William Helton [†]

February 20, 2006

Abstract

A major advance in linear systems theory over the last decade has been a formalism for converting systems problems to matrix inequalities. In this tutorial paper we describe computer algebra algorithms, methodology, and implementation which allows users to convert many systems problems to Linear Matrix Inequalities (LMIs). We shall focus on computer algebra methodology which can assist with user in producing LMIs for control design. We provide a step-by-step computer derivation of LMI formulas for the design of linear time-invariant dynamic controllers that achieve a prespecified performance measured by the H_∞ norm of a certain closed loop transfer function.

1 Introduction

Matrix inequalities are lists of requirements that matrix valued polynomials or rational functions of matrix variables be positive semidefinite [BEFB94, SIG98]. If the problem does not contain a direct reference to any system dimension, formulas must hold for matrix variables of any size. Since matrix multiplication is not commutative, one sees much effort going into calculations (by hand) on noncommutative rational functions. Indeed, the textbook classics of linear control, such as the H_∞ control problem [Fra87], do not explicitly mention the dimension of the systems being controlled. The solution is expressed as two Riccati inequalities [ZDG96]. These are, in fact, inequalities on noncommutative expressions. For instance, in the text [ZDG96], which is a thorough summary of linear control as of 1990, somewhere around 90% of the formulas and manipulations are noncommutative equalities. The book [SIG98], which describes systematically the matrix inequality movement of the 1990's, contains nearly all noncommutative formulas. Thus, for symbolic

[*]This work has been partially supported by the NSF and the Ford Motor Company.

[†]University of California, San Diego, 9500 Gilman Drive, La Jolla, CA, 92093-0112, USA.

computing to have a full ranging impact on linear systems research, one needs a program which will do noncommuting operations.

A major symbolic noncommutative algebra package, `NCAlgebra`, has been developed by one of the authors and collaborators. This software runs under `Mathematica`, being platform independent [HSM00]. The package `NCGB` implements and sorts the output of a Non Commutative Gröbner Basis algorithm. Gröbner Basis Algorithms *eliminate unknowns from collections of equations*, and is the heavy machinery behind the algorithms we implement. Eliminating unknowns often gives "nice forms" for the equations and helps one discover practical formulas. `NCGB` links `Mathematica` and `NCAlgebra` to `C++`, and has some platform dependence: we support Solaris, Windows and various versions of Linux. More details about these software packages are available on our webpage `http://www.math.ucsd.edu/~ncalg`. The goal in this paper is to handle *noncommutative inequalities* in a way which can be implemented in this type of package.

While commutative computer algebra has seen heavy development and use, since the `Macsyma` project in the 1960's, general noncommutative computer algebra has only recently come to the beginning stages of experimentation. Still the field is uncharted and at the stages of high adventure. For perspective, six years ago there was little noncommutative algebra software publically available. Unfortunately, to bring noncommutative computer algebra to nearly its potential requires a creation of a small world of algorithms and software. A crude analogy with the pre-`Matlab` days of engineering comes to mind. Suppose only a few reliable algorithms were known, for example, a (slow) matrix inversion and a (slow) eigensolver is known; there are no Riccati solvers or other utilities. The field, to get started, faces the task of programming what is known, of doing many experiments to find a collection of successful applications and of developing algorithms to fill major application gaps. That is much like the starting situation with noncommuting computer algebra.

2 Objective and scope of this paper

We will describe computer algebra algorithms, methodology, and implementation which allows users to convert many systems and control problems to LMIs (Linear Matrix Inequalities). We shall focus on computer algebra methodology which can assist the user with the two most popular techniques for converting control problems into LMIs, the *elimination of variables* method, as described in

Skelton, Iwasaki and Grigoriadis [SIG98], and the *change of variables* method, as described in Scherer, Gahinet and Chilali [SGC97].

We discuss some generic algorithms and ideas that can be implemented using `NCAlgebra` and `NCGB` which might be able to help users "discover" solutions to control problems in an automated fashion, or at least, simplify much of the work that would otherwise be done by hand. We will show how to use commands in `NCAlgebra` and `NCGB` to implement some of these ideas. For instance, we will discuss a strategy for "motivating unknowns" in noncommutative expressions. This strategy works by bringing the original expression into a form that is likely to be simpler to solve under some appropriate change of variables. In particular, we show that the method of [SGC97] can be seen as a special instance of that generic idea. A specific linear control problem, namely the problem of desining a dynamic output feedback linear time-invariant controller satisfying a certain performance requirement on the H_∞ norm of a closed loop transfer function, will be formulated and solved using symbolic noncommutative algebra tools.

The code used in this tutorial will run in `NCAlgebra`, `NCGB` and `Mathematica`. We will provide a brief introduction to `NCAlgebra` and `Mathematica` notation to familiarize the readers with the computer code. The tutorial code, as well as the packages `NCAlgebra` and `NCGB` are available on the web for download [HSM00]. The readers might also be interested in some of other related results we have previously reported in [HS99, dH03b, dH03a, CHSY03]. The authors are currently developing numerical code that would allow the complete manipulation of matrix inequalities using `Mathematica` and `NCAlgebra`, from production to its numerical solution. Results on this numerical tool will be reported soon.

3 Matrix inequalities in systems and control

Matrix equations and inequalities appear naturally in several systems and control problems. Take, for instance, the problem of checking whether the continuous-time linear time-invariant system

$$\dot{x}(t) = Ax(t), \quad x(0) = x_0, \tag{1}$$

given here in state-space form, is asymptotically stable for any choice of x_0. Let us mention two methods which one could use to verify such property: a) compute the eigenvalues of the square matrix A and check whether they lie in the open left half of the complex plane; b) verify whether the set of matrix inequalities

$$P > 0, \quad A^T P + PA < 0, \tag{2}$$

have a feasible solution, where P is a symmetric real matrix with the same dimension as A, and the relation $X > 0$ ($X < 0$) stands for the symmetric matrix X to belong in the interior of the convex cone of positive (negative) semidefinite matrices. This positive (negative) cone is comprised of all symmetric matrices having only positive (negative) eigenvalues.

It is a fact that in most cases one can check the location of the eigenvalues of A, method a), much more reliably and faster than establishing the existence of a solution to the Lyapunov inequalities, method b). However, the latter test possesses an extremely valuable property that is not associated with method a). This property is *convexity*. Indeed, the cone of symmetric positive semidefinite matrices is a convex cone, therefore positive or negative linear (and affine) mappings on the space of symmetric matrix variables also define convex sets. In other words, and using a terminology that became popular in the recent systems and control literature, the method b) amounts at solving a set of Linear Matrix Inequalities (LMIs). Convexity allows the derivation of many important extensions of this result.

For instance, one can generalize this concept of stability to uncertain systems. We have already mentioned the convexity properties of the inequalities (2) with respect to the variable P. Yet, although (2) is not jointly convex on P and A, it is also convex on A, that is, for a given P. This means that if A is now considered to be a convex function of some parameter ξ defined on a bounded polyhedral convex set Ω with a finite number of vertices $\bar{\xi}_i$, $i = 1, \ldots, N$, i.e., $\Omega = \text{hull}(\xi_1, \ldots, \xi_N)$, then the following implication follows from convexity

$$A(\xi)^T P + PA(\xi) \leq A(\bar{\xi}_i)^T P + PA(\bar{\xi}_i), \quad \forall\, \xi \in \Omega = \text{hull}(\xi_1, \ldots, \xi_N). \tag{3}$$

Therefore, stability of the uncertain linear time-invariant[1] system in the form (1) where $A(\xi)$ is now

[1]This result is indeed true for linear time-varying systems, since P can be used to build a constant quadratic Lyapunov function that proves the asymptotic stability of the associated linear-time varying system.

a convex function of the uncertain parameter $\xi \in \Omega$ can be established by verifying the existence of a solution to the finite number of LMI

$$P > 0, \quad A(\bar{\xi}_i)^T P + P A(\bar{\xi}_i) < 0, \quad \forall\, i = 1, \dots, N. \tag{4}$$

Numerous useful results can be obtained in the context of design of filters and controllers, the subject for which the tools described in this paper have been developed. Two methodologies yield virtually all results in this area, namely, *elimination of variables* [SIG98] or *change of variables* [SGC97]. In the first methodology the controller or filter parameters are eliminated from the original inequalities, and a set of equivalent inequalities is obtained which may be convex in some important problems. In the second, new variables are introduced in the problem to replace the original control parameters through a one-to-one mapping. Again, the inequalities in the new variables may be convex in some problems.

The noncommutative symbolic algebra tools to be described in the following sections can assist users to obtain these transformed inequalities, which many times will have better properties than the original ones. We describe, in particular, how to obtain matrix inequalities for dynamic output feedback control, a problem which requires intensive use of matrix operations involving noncommutative products. But first, we will briefly revisit the main steps involved in the production of these design inequalities on a simple problem, namely stabilizability by state feedback.

3.1 Stabilizability by state feedback

Consider the linear time-invariant system with one control input $u(t)$ in the form

$$\dot{x}(t) = Ax(t) + Bu(t), \quad x(0) = x_0, \tag{5}$$

connected in feedback with the *state feedback* controller

$$u(t) = Kx(t), \tag{6}$$

where the matrices A, B, and K are assumed to have compatible dimensions. This arrangement produces the following closed-loop system

$$\dot{x}(t) = (A + BK)x(t), \quad x(0) = x_0, \tag{7}$$

which can be tested for asymptotic stability using the Lyapunov inequalities (2) for any given controller K. Unfortunately, one can easily show that the inequalities

$$P > 0, \quad (A + BK)^T P + P(A + BK) < 0, \tag{8}$$

are not jointly convex on P and K. In the next paragraphs we will apply transformations to (8) so as to obtain inequalities with more favorable properties, such as convexity.

3.2 Elimination of variables

The main tool in the technique known as *elimination of variables* is the following lemma. Proofs of the following lemma can be found, for instance, in the books [BEFB94] or [SIG98].

Lemma 1 (Elimination) *The following statements are equivalent:*

i) There exists G *such that* $E + F^T G H + H^T G^T F < 0$.

ii) $(FF^T > 0 \quad \textit{or} \quad (F^\perp)^T E F^\perp < 0) \quad \textit{and} \quad (HH^T > 0 \quad \textit{or} \quad (H^\perp)^T E H^\perp < 0)$.

Here the symbol $X^\perp$ denotes a matrix satisfying $XX^\perp = 0$ and $X^T X + (X^\perp)^T X^\perp > 0$, that is, $X^\perp$ denotes a basis for the right null space of matrix X.

The strategy is to rewrite a given inequality in the form of item *i*) with G representing the unknowns, or part of the unknowns, so as to produce the potentially nicer inequalities in item *ii*). For instance, the second inequality of the Lyapunov inequalities (8) can be rewritten in the form *i*) by defining

$$E = A^T P + PA, \qquad F = B^T P, \qquad G = K, \qquad H = I. \tag{9}$$

Recalling that $P > 0$ and that B is a known matrix we can compute $F^{\perp} = P^{-1}(B^T)^{\perp}$ and write the inequalities of item *ii*)

$$(F^{\perp})^T E F^{\perp} = [(B^T)^{\perp}]^T (P^{-1}A^T + AP^{-1})(B^T)^{\perp} < 0, \qquad HH^T = I > 0. \tag{10}$$

Then defining $X := P^{-1} > 0$ we can write the complete necessary and sufficient conditions for the existence of a stabilizing controller K as the existence of a symmetric matrix X satisfying

$$X > 0, \quad [(B^T)^{\perp}]^T (XA^T + AX)(B^T)^{\perp} < 0. \tag{11}$$

These are LMIs on the unknown variable X. So they are convex in X.

Rewriting inequalities in the particular form of Lemma 1 can be significantly more involved, specially when the terms E, F, G and H are matrices, possibly with matrix entries. We have developed symbolic algorithms to produce such factorizations in these complicated cases. The use of such algorithms will be discussed in Section 5. Theoretical and implementation issues are briefly discussed in Section 6.

3.3 Change of variables

The task of introducing a suitable change of variable is much less systematic and *ad-hoc* than that of eliminating variables. Indeed, the particular form of factorization required by the Elimination Lemma provides guidance to the user in this process. In this sense, it is harder to code general purpose algorithms for introducing changes of variables. Some issues regarding this topic are further discussed in Section 6.2.

One might try to obtain clues, when possible, by trying to reproduce the operations performed while eliminating variables. For instance, in the case of the Lyapunov inequalities (2), the fact that the nice inequalities previously obtained are functions of $X := P^{-1}$, and not P, suggests that we might start by rewriting the inequalities in terms of X. As $X > 0$, this can be done by multiplying both inequalities in (2) by X on the left and on the right so as to get the equivalent conditions

$$X > 0, \quad X(A + BK)^T + (A + BK)X < 0, \tag{12}$$

We then manipulate the second inequality by expanding the products

$$AX + XA^T + BKX + XK^TB^T < 0. \tag{13}$$

In more complicated inequalities, as the ones arising in output feedback problems, we may also need to *collect on the knowns* [2] to motivate the introduction of new unknowns.

As the objective is to produce inequalities which depend only linearly or affinely on the unknowns, we introduce the new unknown

$$L := KX \tag{14}$$

We are particularly interested in the introduction of new unknowns through changes of variables that can be used to eliminate existing unknowns from the original problem. In this example, the control gain K can be *eliminated*, or, in other words, K can be explicitly expressed in terms of other unknowns, by solving the change of variable equation (14) for the unknown K. This produces

$$K = LX^{-1}. \tag{15}$$

This process typically requires the use of some assumptions, in this case the invertibility of X. The relation (15) can be used to rewrite the necessary and sufficient conditions for stabilizability (12) as

$$X > 0, \quad AX + XA^T + BL + L^TB^T < 0, \tag{16}$$

which is an LMI on the unknowns X and L.

We can automate many of the above steps using powerful algorithms implemented in `NCAlgebra`. In particular, the products that will be replaced by transformed variables are typically polynomials, and "solving" for the original variables often can be done symbolically using `NCGB`. In the following sections, we will use `NCAlgebra`, `NCGB` and `Mathematica` to generate all of our results, which will be displayed in figures. The complete derivation is available for download in the form of a `Mathematica` notebook from our website [3], along with all other packages used in this paper.

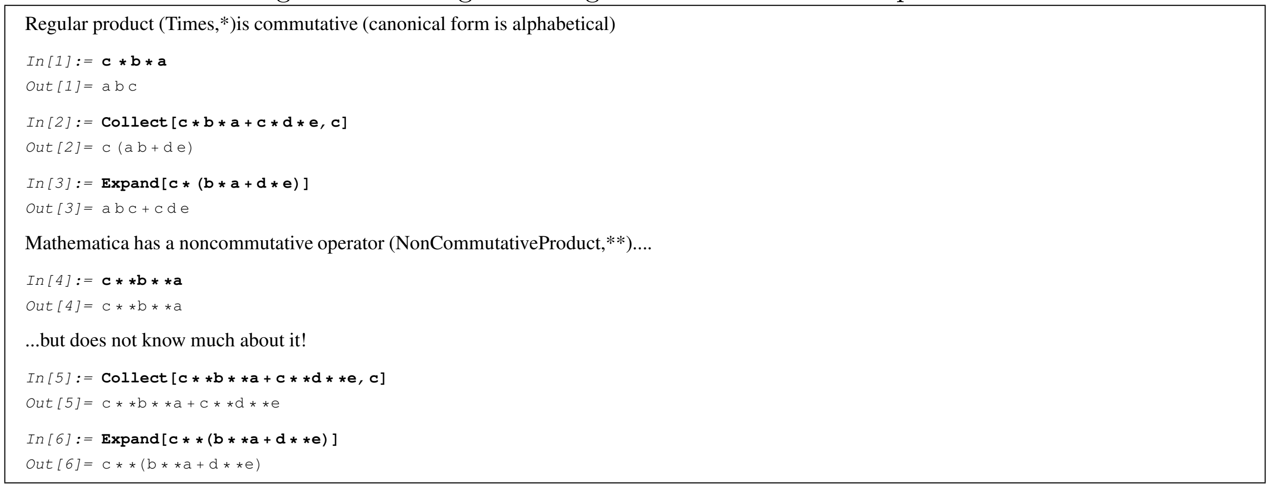
[2]See Section 6.2 for details.
[3]`http://www.math.ucsd.edu/∼ncalg`.

4 What goes wrong with current symbolic tools?

Virtually all commercial software for symbolic calculations, including `Mathematica` and `Maple`, offer little or no support for expressions involving noncommutative products. Let us take, for example, `Mathematica`, which is probably the most popular software in this category.

`Mathematica` has a a `Dot` operator (`.`), which is used to provide elementary functionality for matrix and vector operations, and a built in noncommutative product operator (`**`). However, `Mathematica` provides little support for manipulating expressions with such operators. That is, it does not know how to collect, expand, factor or simplify noncommutative products, as shown in Figure 1.

Figure 1: What goes wrong with noncommutative products?



Regular product (Times,*)is commutative (canonical form is alphabetical)

```
In[1]:= c*b*a
Out[1]= a b c

In[2]:= Collect[c*b*a+c*d*e, c]
Out[2]= c (a b + d e)

In[3]:= Expand[c*(b*a+d*e)]
Out[3]= a b c + c d e
```

Mathematica has a noncommutative operator (NonCommutativeProduct,**)....

```
In[4]:= c**b**a
Out[4]= c**b**a
```

...but does not know much about it!

```
In[5]:= Collect[c**b**a+c**d**e, c]
Out[5]= c**b**a + c**d**e

In[6]:= Expand[c**(b**a+d**e)]
Out[6]= c**(b**a + d**e)
```

`NCAlgebra` is a package written by the authors entirely in `Mathematica` that provides such functionality. The commands `SetNonCommutative` and `SetCommutative` associate commutative properties with symbols, and a large library of functions, including `NCExpand` and `NCCollect`, brings life to expressions with noncommutative products, as shown in Figure 2.

Another tricky task is to handle symbolic matrices. `Mathematica` treats matrices simply as lists of lists. This approach, however, does not handle block matrices, that is, matrices whose entries are matrices, as shown in Figure 3. We overcome this limitation with our package `Matrix`, which enhances `Mathematica`'s ability to properly handle matrices, including features such as automatically handling partitioning and flattening of partitioned matrices. This package is fully integrated with `NCAlgebra`, and knows how to handle matrices with noncommutative entries, as shown in Figure 4.

Figure 2: The `NCAlgebra` solution.

```
In[14]:= << SetNCPath.m;
         << NCAlgebra.m;
```

In NCAlgebra one associates commutative properties with symbols

```
In[15]:= SetCommutative[b];
         SetNonCommutative[a, c, d, e, f];
```

NCAlgebra uses the native noncommutative product operator...

```
In[16]:= c ** b ** a
Out[16]= b c ** a
```

and provides it with functionality.

```
In[17]:= NCCollect[c ** b ** a + c ** d ** e, c]
Out[17]= c ** (a b + d ** e)
```

```
In[18]:= NCExpand[c ** (b ** a + d ** e)]
Out[18]= b c ** a + c ** d ** e
```

Figure 3: What goes wrong with matrices?

Matrices in *Mathematica* are lists of lists

```
In[7]:= m1 = {{c, d}, {e, f}}
Out[7]= {{c, d}, {e, f}}
```

```
In[8]:= MatrixQ[m1]
Out[8]= True
```

But replacing entries of a matrix by another matrix produces a list with depth four...

```
In[9]:= m2 = ReplaceAll[{{a, b}}, {a → m1, b → m1}]
Out[9]= {{{{c, d}, {e, f}}, {{c, d}, {e, f}}}}
```

...which is not a matrix!

```
In[10]:= MatrixQ[m2]
Out[10]= False
```

Products of matrices are fine with Dot...

```
In[11]:= m3 = {{a, b}, {c, d}}
Out[11]= {{a, b}, {c, d}}
```

```
In[12]:= m4 = m1.m3
Out[12]= {{a c + c d, b c + d^2}, {a e + c f, b e + d f}}
```

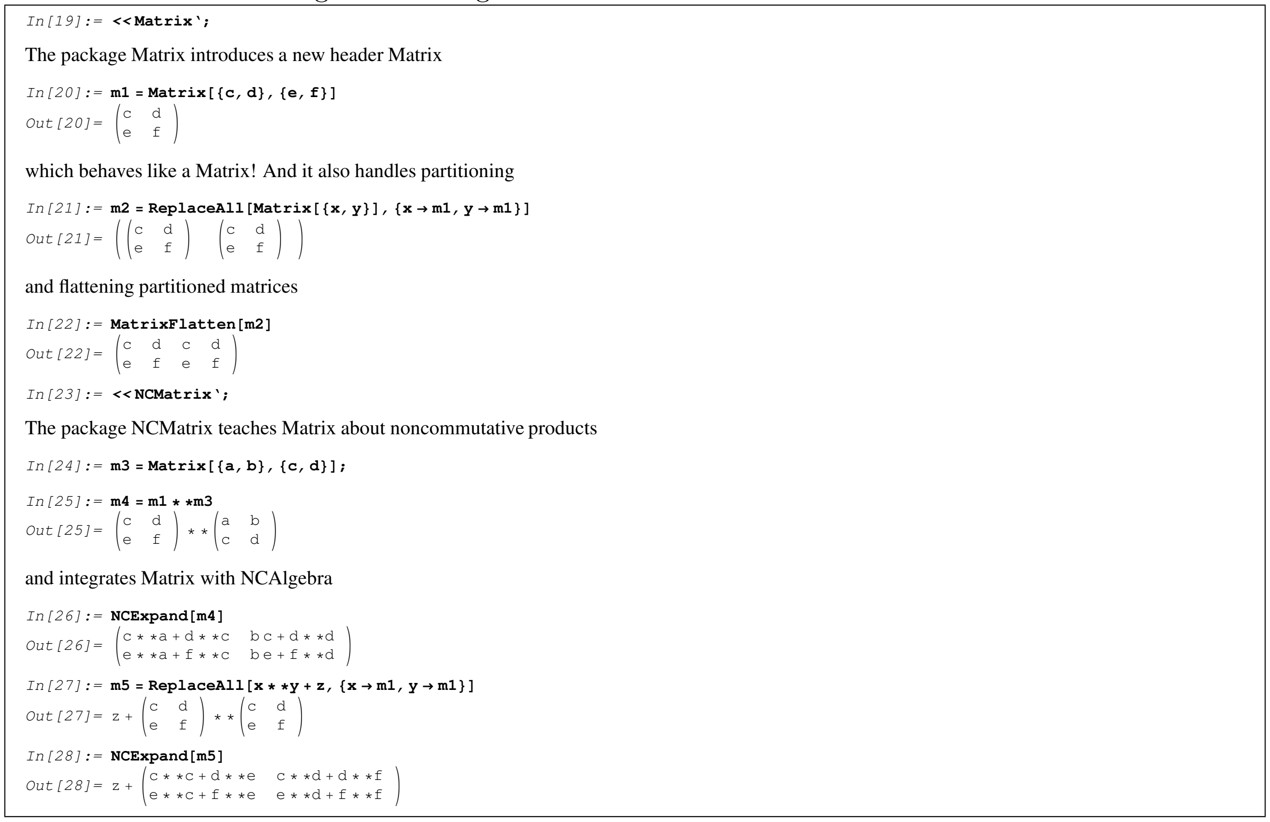
...but Dot assumes that all entries commute!

Plus adds anything that is not a list to a list entrywise...

```
In[13]:= m5 = ReplaceAll[a.b + g, {a → m1, b → m1}]
Out[13]= {{c^2 + d e + g, c d + d f + g}, {c e + e f + g, d e + f^2 + g}}
```

...what if c is a matrix we do not know the entries yet?

Figure 4: Doing matrices with the declaration `Matrix`.



```
In[19]:= <<Matrix`;
```

The package Matrix introduces a new header Matrix

```
In[20]:= m1 = Matrix[{c, d}, {e, f}]
```

$$Out[20]= \begin{pmatrix} c & d \\ e & f \end{pmatrix}$$

which behaves like a Matrix! And it also handles partitioning

```
In[21]:= m2 = ReplaceAll[Matrix[{x, y}], {x → m1, y → m1}]
```

$$Out[21]= \begin{pmatrix} \begin{pmatrix} c & d \\ e & f \end{pmatrix} & \begin{pmatrix} c & d \\ e & f \end{pmatrix} \end{pmatrix}$$

and flattening partitioned matrices

```
In[22]:= MatrixFlatten[m2]
```

$$Out[22]= \begin{pmatrix} c & d & c & d \\ e & f & e & f \end{pmatrix}$$

```
In[23]:= <<NCMatrix`;
```

The package NCMatrix teaches Matrix about noncommutative products

```
In[24]:= m3 = Matrix[{a, b}, {c, d}];

In[25]:= m4 = m1 * *m3
```

$$Out[25]= \begin{pmatrix} c & d \\ e & f \end{pmatrix} ** \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

and integrates Matrix with NCAlgebra

```
In[26]:= NCExpand[m4]
```

$$Out[26]= \begin{pmatrix} c**a+d**c & bc+d**d \\ e**a+f**c & be+f**d \end{pmatrix}$$

```
In[27]:= m5 = ReplaceAll[x * *y + z, {x → m1, y → m1}]
```

$$Out[27]= z+\begin{pmatrix} c & d \\ e & f \end{pmatrix} ** \begin{pmatrix} c & d \\ e & f \end{pmatrix}$$

```
In[28]:= NCExpand[m5]
```

$$Out[28]= z+\begin{pmatrix} c**c+d**e & c**d+d**f \\ e**c+f**e & e**d+f**f \end{pmatrix}$$

Finally, a major tool in symbolic computation is an equation solver. In this paper we use `NCGB`, a platform dependent implementation of the non commutative Gröbner basis algorithm of F. Mora [Mor86, GHK97]. This algorithm is the tool behind the command `NCEliminate`, which will be used in the following section to eliminate unknowns from a given systems of polynomial equations involving noncommutative products. See Section 6.1 for more details.

5 Automated LMI production

In this section we show how `NCAlgebra` can be of help with LMI production in a typical control design problem: the design of an output feedback controller such that the closed loop system satisfies an H_∞ performance constraint. To simplify the notation, we omit the dependence of all signals on the time t.

5.1 Problem statement

Given the linear time-invariant model of the system to be controlled

$$\begin{pmatrix} \dot{x} \\ z \\ y \end{pmatrix} = \begin{bmatrix} A_p & B_w & B_u \\ C_z & D_{zw} & D_{zu} \\ C_y & D_{yw} & 0 \end{bmatrix} \begin{pmatrix} x \\ w \\ u \end{pmatrix}, \quad x(0) = 0, \tag{17}$$

and the linear time-invariant dynamic output-feedback controller

$$\begin{pmatrix} \dot{x}_c \\ u \end{pmatrix} = \begin{bmatrix} A_c & B_c \\ C_c & D_c \end{bmatrix} \begin{pmatrix} x_c \\ y \end{pmatrix}, \quad x_c(0) = 0, \tag{18}$$

we will show how to compute controller parameters (A_c, B_c, C_c, D_c) such that the closed loop system is asymptotically stable and $\|H_{wz}(s)\|_\infty < \mu$, where $H_{wz}(s)$ is the closed loop transfer function from the input w to the output z and $\mu > 0$ is a prespecified performance level. For simplicity we assume that the controller has the same order as the plant (that is, it is a *full-order* controller). Such an assumption is essential, as will be seen later, to ensure that this control design problem can be recast as the problem of finding a feasible solution to sets of LMIs.

The closed loop connection of the plant and controller is a linear system in the form

$$\begin{pmatrix} \dot{\tilde{x}} \\ z \end{pmatrix} = \begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{bmatrix} \begin{pmatrix} \tilde{x} \\ w \end{pmatrix}, \qquad \tilde{x} := \begin{pmatrix} x \\ x_c \end{pmatrix}. \tag{19}$$

We seek to compute the matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. This closed loop system is asymptotically stable and the H_∞ norm of $H_{wz}(s)$ is less than μ if, and only if, the following inequalities, from the well known Bounded Real Lemma [SIG98],

$$\mathcal{P} > 0, \quad \begin{bmatrix} \mathcal{A}^T\mathcal{P} + \mathcal{P}\mathcal{A} & \mathcal{C}^T & \mathcal{P}\mathcal{B} \\ \mathcal{C} & -\mu & \mathcal{D} \\ \mathcal{B}^T\mathcal{P} & \mathcal{D}^T & -\mu \end{bmatrix} < 0, \tag{20}$$

have some feasible solution for some symmetric matrix $\mathcal{P}$ with the same dimension as matrix $\mathcal{A}$.

Figure 5: Setting up the closed loop signal equations.

Define plant matrix and signals:

```
In[1]:= Plant = Matrix[{Ap, Bw, Bu}, {Cz, Dzw, Dzu}, {Cy, Dyw, 0}];
        xdotzy = Matrix[{xdot}, {z}, {y}];
        xwu = Matrix[{x}, {w}, {u}];
```

Plant equation:

```
In[2]:= xdotzy - Plant * *xwu
        (xdot )   (Ap   Bw    Bu )     (x)
Out[2]= (  z  ) - (Cz   Dzw   Dzu) * * (w)
        (  y  )   (Cy   Dyw    0 )     (u)
```

Define controller matrix and signals

```
In[3]:= Controller = Matrix[{Ac, Bc}, {Cc, Dc}];
        xcdotu = Matrix[{xcdot}, {u}];
        xcy = Matrix[{xc}, {y}];
```

Controller equation

```
In[4]:= xcdotu - Controller * *xcy
        (xcdot )   (Ac   Bc)     (xc )
Out[4]= (  u   ) - (Cc   Dc) * * ( y )
```

Combined equations:

```
In[5]:= eqns = Matrix[{xdotzy - Plant * *xwu}, {xcdotu - Controller * *xcy}];
        eqns = MatrixFlatten[NCExpand[eqns]]
         xdot - Ap * *x - Bu * *u - Bw * *w
        ( z - Cz * *x - Dzu * *u - Dzw * *w )
Out[5]= (      y - Cy * *x - Dyw * *w       )
        (    xcdot - Ac * *xc - Bc * *y     )
              u - Cc * *xc - Dc * *y
```

Notice that the Lyapunov inequalities are embedded in the above inequalities: the second inequality in (2) appears as the first diagonal block in the second inequality above. Indeed, this is the reason why these inequalities ensure asymptotic internal stability to the closed loop system.

5.2 Computing the closed loop signals

The first problem we address using our symbolic tools is the computation of the closed loop matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. This will be done by solving for the closed loop signals appearing in the plant and controller equations. This illustrates the use of `NCEliminate` on solving systems of equations involving noncommutative variables. This problem could also be solved by programming constructive and simpler "system connection" rules. Indeed, `NCAlgebra` can be set up to work together with the `Mathematica` toolbox *Control System Professional* so as to allow high level manipulation of systems with noncommutative entries.

We look at (17) and (18) as systems of signal equations. We want to close the loop by eliminating the internal signals u and y from these equations. The left hand side of these equations are defined in Figure 5.

We then set u and y as unknowns in Figure 6 and call `NCEliminate`. This routine takes three

Figure 6: Eliminating u and y.

```
Define the unknwons

In[6]:= unknowns = {u, y}
Out[6]= {u, y}

Call NCEliminate to solve equations:

In[7]:= solution = NCEliminate[MatrixToList[eqns], unknowns, 1];
        ColumnForm[solution]
Out[7]= y → Cy * *x + Dyw * *w
        Bc * *Dyw * *w → xcdot - Ac * *xc - Bc * *Cy * *x
        u → Cc * *xc + Dc * *Cy * *x + Dc * *Dyw * *w
        Dzu * *Dc * *Dyw * *w → z - Cz * *x - Dzw * *w - Dzu * *Cc * *xc - Dzu * *Dc * *Cy * *x
        Bu * *Dc * *Dyw * *w → xdot - Ap * *x - Bw * *w - Bu * *Cc * *xc - Bu * *Dc * *Cy * *x
```

Figure 7: Computing the closed loop matrices.

```
Substitute the solution:

In[8]:= ClosedLoopSignals = NCExpand[eqns/.solution]
          xdot - Ap * *x - Bw * *w - Bu * *Cc * *xc - Bu * *Dc * *Cy * *x - Bu * *Dc * *Dyw * *w
         ⎛z - Cz * *x - Dzw * *w - Dzu * *Cc * *xc - Dzu * *Dc * *Cy * *x - Dzu * *Dc * *Dyw * *w⎞
Out[8]=  ⎜                                        0                                          ⎟
         ⎝                    xcdot - Ac * *xc - Bc * *Cy * *x - Bc * *Dyw * *w                ⎠
                                                  0

Factor the signals to obtain the closed loop matrices

In[9]:= ClosedLoopRHS = MatrixAffineFactor[ClosedLoopSignals, {x, xc, w}]
          xdot        Bw + Bu * *Dc * *Dyw    Bu * *Cc    Ap + Bu * *Dc * *Cy
         ⎛  z  ⎞   ⎛Dzw + Dzu * *Dc * *Dyw   Dzu * *Cc   Cz + Dzu * *Dc * *Cy⎞     ⎛ w ⎞
Out[9]=  ⎜  0  ⎟ - ⎜           0                 0                0          ⎟ * * ⎜xc ⎟
         ⎝xcdot⎠   ⎝       Bc * *Dyw             Ac           Bc * *Cy       ⎠     ⎝ x ⎠
            0                  0                 0                0

Extract the closed loop matrices

In[10]:= ClosedLoop = ClosedLoopRHS/.a_ - b_ * *c_ → b;
         ClosedLoop = ClosedLoop[[{1, 4, 2}, {3, 2, 1}]];
         Acl = ClosedLoop[[{1, 2}, {1, 2}]];
         Bcl = ClosedLoop[[{1, 2}, {3}]];
         Ccl = ClosedLoop[[{3}, {1, 2}]];
         Dcl = ClosedLoop[[{3}, {3}]];

In[11]:= Matrix[{Acl, Bcl}, {Ccl, Dcl}]
          ⎛ ⎛Ap + Bu * *Dc * *Cy   Bu * *Cc ⎞   ⎛Bw + Bu * *Dc * *Dyw⎞ ⎞
Out[11]=  ⎜ ⎝     Bc * *Cy           Ac     ⎠   ⎝     Bc * *Dyw      ⎠ ⎟
          ⎝(Cz + Dzu * *Dc * *Cy   Dzu * *Cc )  (Dzw + Dzu * *Dc * *Dyw )⎠
```

arguments: a list of expressions for which all entries must equal zero, a list of unknowns, and the number of iterations of the underlying `NCGB` algorithm.

In Figure 7, the solution found is substituted back into the original equations to obtain the closed loop signals. The closed loop matrices are then obtained from by factoring the signal equations in the form of sums of matrix products which are affine on the variables x, x_c and w. This is done by the command[4] `MatrixAffineFactor`. This function will be also used later to factorize expressions in a form that suits Lemma 1. Finally, the closed loop matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ are extracted from the factorized closed loop signal equations using `Mathematica`'s pattern match capabilities. In the `Mathematica` sessions, symbols that end in '`cl`' denote matrices associated with closed loop analysis, such as $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. For examples, `Bcl` is used for $\mathcal{B}$.

[4]See Section 6.3 for details.

Figure 8: Obtaining the Bounded Real Lemma.

Set pretty output mode

```
In[1]:= NCSetOutput[All → True]
```

Define the bounded real lemma matrix

```
In[2]:= SetCommutative[μ];
        BRLol = Matrix[{tp[Aol] ** Pol + Pol ** Aol, tp[Col], Pol ** Bol}, {Col, -μ, Dol}, {tp[Bol] ** Pol, tp[Dol], -μ}]
         ⎛Pol.Aol + Aol^T.Pol  Col^T  Pol.Bol⎞
Out[2]=  ⎜       Col            -μ      Dol  ⎟
         ⎝    Bol^T.Pol        Dol^T    -μ   ⎠
```

Impose a particular structure on Pcl

```
In[3]:= Pcl = Matrix[{X, -Z}, {-Z, Z}]
         ⎛ X   -Z ⎞
Out[3]=  ⎝-Z    Z ⎠
```

Substitute the closed loop matrices

```
In[4]:= BRLcl = MatrixFlatten[NCExpand[BRLol/.{Pol → Pcl, Aol → Acl, Bol → Bcl, Col → Ccl, Dol → Dcl}]];
        ColumnForm[SymmetricMatrixToVector[BRLcl]]
Out[4]= X.Ap + Ap^T.X - Z.Bc.Cy - Cy^T.Bc^T.Z + X.Bu.Dc.Cy + Cy^T.Dc^T.Bu^T.X
        -Z.Ap - Ac^T.Z + Z.Bc.Cy + Cc^T.Bu^T.X - Z.Bu.Dc.Cy
        Z.Ac + Ac^T.Z - Z.Bu.Cc - Cc^T.Bu^T.Z
        Cz + Dzu.Dc.Cy
        Dzu.Cc
        -μ
        Bw^T.X - Dyw^T.Bc^T.Z + Dyw^T.Dc^T.Bu^T.X
        -Bw^T.Z + Dyw^T.Bc^T.Z - Dyw^T.Dc^T.Bu^T.Z
        Dzw^T + Dyw^T.Dc^T.Dzu^T
        -μ
```

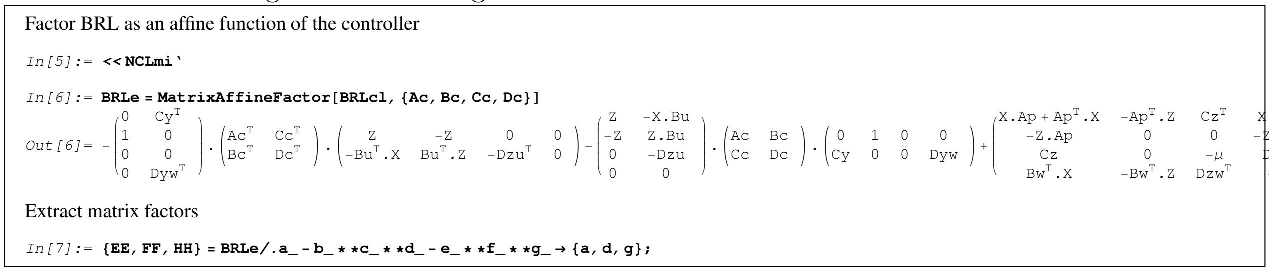
5.3 Elimination of the controller variables

We now apply our tools to the derivation of LMIs for the H_∞ control design problem. The first method is the elimination of the controller variables [SIG98]. As explained in Section 3.2, the major algebraic manipulation is the reformulation of the second inequality of (20) in the form of item *i*) in Lemma 1.

In Figure 8, we first define the matrix coefficient of the second inequality from the bounded real lemma (20). The symbols ending in '`ol`' denote matrices associated with open loop analysis. We then go from open loop to closed loop analysis by simply substituting for the previously computed closed loop matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. The obtained expression in expanded form is too large to be displayed in one line. Instead, we use the function `SymmetricMatrixToVector` to convert this symmetric matrix into a list, which leaves out redundant entries. This list is displayed vertically using `Mathematica`'s formatting operator `ColumnForm`. This practice will be repeatedly used whenever the obtained matrices do not fit the page width.

As anticipated in the previous section, in Figure 9, we invoke the command `MatrixAffineFactor` to factor the BRL as a sum of products of matrices which is affine on the controller variables. This expression is in the form of the inequality in item *i*) of Lemma 1. The factors E, F and H are extracted from the factored expression by pattern matching.

Figure 9: Factoring in a form which is affine on the controller.



Factor BRL as an affine function of the controller

```
In[5]:= <<NCLmi`

In[6]:= BRLe = MatrixAffineFactor[BRLcl, {Ac, Bc, Cc, Dc}]
```

$$\text{Out[6]}= -\begin{pmatrix} 0 & Cy^T \\ 1 & 0 \\ 0 & 0 \\ 0 & Dyw^T \end{pmatrix} . \begin{pmatrix} Ac^T & Cc^T \\ Bc^T & Dc^T \end{pmatrix} . \begin{pmatrix} Z & -Z & 0 & 0 \\ -Bu^T.X & Bu^T.Z & -Dzu^T & 0 \end{pmatrix} - \begin{pmatrix} Z & -X.Bu \\ -Z & Z.Bu \\ 0 & -Dzu \\ 0 & 0 \end{pmatrix} . \begin{pmatrix} Ac & Bc \\ Cc & Dc \end{pmatrix} . \begin{pmatrix} 0 & 1 & 0 & 0 \\ Cy & 0 & 0 & Dyw \end{pmatrix} + \begin{pmatrix} X.Ap + Ap^T.X & -Ap^T.Z & Cz^T & X.Bw \\ -Z.Ap & 0 & 0 & -Z.Bw \\ Cz & 0 & -\mu & Dzw \\ Bw^T.X & -Bw^T.Z & Dzw^T & -\mu \end{pmatrix}$$

Extract matrix factors

```
In[7]:= {EE, FF, HH} = BRLe/.a_ - b_**c_**d_ - e_**f_**g_ → {a, d, g};
```

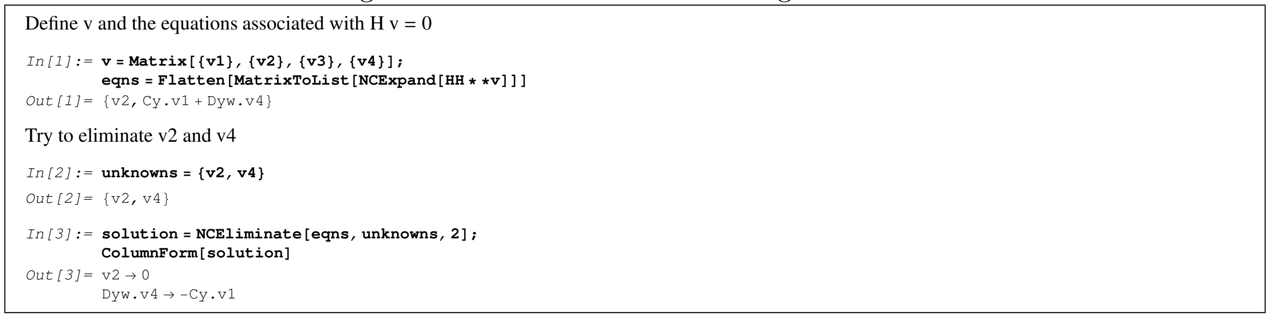
Notice that we have assumed a very particular partitioning structure in the definition of $\mathcal{P}$, which depends upon only two sub-blocks, X and Z, instead of three. However, in the H_∞ control problem, such structure can be imposed without loss of generality whenever a full order controller is to be computed. Since a similarity transformation can always be applied on the controller state space variables so as to obtain such particular structure on $\mathcal{P}$ without modifying the controller transfer function. We have also used the function `NCSetOutput[All->True]` to "beautify" the output in the `Mathematica` notebook, so that noncommutative products are now displayed using the pleasant '`.`' instead of '`**`'. This function also display transpose as $(\cdot)^T$ and inverse using $(\cdot)^{-1}$. These transformations are only cosmetic, and the internal representation still contains the original functions `**`, `tp` and `inv`.

5.3.1 Computing null spaces

The next step in generating the inequalities of item *ii*) in Lemma 1 is computing the null spaces of matrices F and H. Computing $X^\perp$ (the null space of a matrix X) means finding all solutions to the linear algebra problem $Xv = 0$, where v is a vector of compatible dimension. If X is a matrix with scalar entries, then one can use a decomposition algorithms to reliably compute the null space. Even when such entries are symbolic, one can carry on the computations loosely assuming invertibility of certain symbols and expressions generated during the iterations. Here, as we are interested in computing the null space of matrices whose entries are matrices, *a priori* knowledge of invertibility of certain symbols become mandatory, if one wants to obtain explicit solutions. In most systems and control problems such hypothesis can be imposed without much difficulty.

We start by computing $H^\perp$, first without assuming invertibility of any symbols. By partitioning v according to H we can set up and attempt to solve this linear problem using `NCEliminate`.

Figure 10: First iteration on solving of $Hv = 0$.



Define v and the equations associated with H v = 0

```
In[1]:= v = Matrix[{v1}, {v2}, {v3}, {v4}];
        eqns = Flatten[MatrixToList[NCExpand[HH ** v]]]
Out[1]= {v2, Cy.v1 + Dyw.v4}
```

Try to eliminate v2 and v4

```
In[2]:= unknowns = {v2, v4}
Out[2]= {v2, v4}

In[3]:= solution = NCEliminate[eqns, unknowns, 2];
        ColumnForm[solution]
Out[3]= v2 → 0
        Dyw.v4 → -Cy.v1
```

This is done in Figure 10. The product Hv defines two equations. As v has four variables, generically speaking, only two variables will be independent. Here we picked v_1 and v_3 to be the independent ones and v_2 and v_4 to be the dependent ones, that is, the ones to be eliminated. The set of solutions produced by `NCEliminate` is shown at the bottom of Figure 10.

Notice that it was not possible to eliminate v_2 and v_4 completely, as v_4 is multiplied on the left by D_{yw}. The algorithm can only proceed further if we explicitly state invertibility of D_{yw}. In the context of the H_∞ control problem, one can assume without further ado that D_{yw} is right invertible, or, in other words, that D_{yw} has full row rank. For the purpose of computing a basis of the null space, we can therefore compute only the solutions of v_4 spanned by D_{yw}^T, that is $v_4 = D_{yw}^T \tilde{v}_4$. This is done in Figure 11. Notice that the right invertibility of D_{yw} is informed to `NCEliminate` by adding the directive `NCInvertible` to the list of equations. Indeed, right invertibility of D_{yw} is stated as the invertibility of $D_{yw} D_{yw}^T$. With this assumption, the variables v_2 and v_4 can now be completely eliminated, as shown in Figure 11 (if we had chosen v_1 as an independent variable, an invertibility assumption on C_y would be needed).

The last step is to generate $H^\perp$, whose columns are a basis for the null space of H, done in Figure 12. The matrix $H^\perp$ will have as columns a complete set of linearly independent vectors v, generated by assigning particular values for the independent variables v_1 and v_3. Typically the most esthetic choice suffices, in this case $(v_1, v_3) = (I, 0)$ and $(v_1, v_3) = (0, I)$. In some cases, such as in the computation of $F^\perp$ to be addressed next, we want to make choices that depend on nonsingular matrices that can be later chosen to our advantage. For $H^\perp$ the most parsimonious choice is enough.

The same basic procedure is used to solve the equation $Fv = 0$, and is shown in Figure 13. We

Figure 11: Complete solution to $Hv = 0$ using invertibility of D_{yw}.

```
Chose particular solution for v4 in the range of tp[Dyw]

In[4]:= hypothesis = {v4 → tp[Dyw] ** v4t}
Out[4]= {v4 → Dyw^T.v4t}

Compute modified equations

In[5]:= eqnst = eqns/.hypothesis
Out[5]= {v2, Cy.v1 + Dyw.Dyw^T.v4t}

State right invertibility of Dyw

In[6]:= invs = {NCInvertible[Dyw ** tp[Dyw]]}
Out[6]= {NCInvertible[Dyw.Dyw^T]}

Solve again for v2 and v4t

In[7]:= unknowns = {v2, v4t}
Out[7]= {v2, v4t}

In[8]:= solution = NCEliminate[Join[eqnst, invs], unknowns, 2];
        ColumnForm[solution]
Out[8]= v2 → 0
        v4t → -(Dyw.Dyw^T)^-1.Cy.v1
```

Figure 12: Computing $H^{\perp}$.

```
Compute general solution

In[9]:= vt = v/.hypothesis/.solution;

Generate basis

In[10]:= Hperp = MatrixFlatten[Matrix[vt/.{{v1 → 1, v3 → 0}, {v1 → 0, v3 → 1}}]]
```

$$\text{Out[10]=} \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ -\text{Dyw}^T.(\text{Dyw}.\text{Dyw}^T)^{-1}.\text{Cy} & 0 \end{pmatrix}$$

chose v_1 and v_4 as the independent variables and v_2 and v_3 as the variables to be eliminated. This time the assumption is the left invertibility of D_{zu}, which is informed to `NCEliminate` by using the directive `NCInvertible` for $D_{zu}^T D_{zu}$. An additional complication in this case is the fact that the matrix F involves the variables X and Z. As these matrices are blocks of $\mathcal{P} > 0$ we have that $X > 0$, $Z > 0$ and $X - Z > 0$, which implies that X, Z and $X - Z$ must be invertible. Only one of these three relations is essential to compute the solution of $Fv = 0$. This is the invertibility of Z, which is added via `NCInvertible`. Matrix $F^{\perp}$ is generated in Figure 14 by choosing of $(v_1, v_4) = (U, 0)$ and $(v_1, v_4) = (0, I)$, where U is a nonsingular otherwise arbitrary matrix to be determined later. The need for such extra freedom will be justified in the next section.

5.3.2 Computing the LMIs

With matrices $F^{\perp}$ and $H^{\perp}$ in hand, the calculation of the inequalities in item *ii*) of Lemma 1 is done in Figure 15. We have one more step to get the desired LMIs. The matrices $(F^{\perp})^T E F^{\perp}$ and $(H^{\perp})^T E H^{\perp}$ are too complicated to be displayed in this tutorial paper. For this reason, we will

Figure 13: Solving $Fv = 0$.

Define v and the equations associated with F v = 0

```
In[1]:= v = Matrix[{v1}, {v2}, {v3}, {v4}];
        eqns = Flatten[MatrixToList[NCExpand[FF ** v]]];
```

Chose particular solution for v3 in the range of Dzu

```
In[2]:= hypothesis = {v3 → Dzu ** v3t}
Out[2]= {v3 → Dzu.v3t}
```

Compute modified equations

```
In[3]:= eqnst = eqns/.hypothesis
Out[3]= {Z.v1 - Z.v2, -Bu^T.X.v1 + Bu^T.Z.v2 - Dzu^T.Dzu.v3t}
```

State left invertibility of Dzu and invertibility of Z

```
In[4]:= invs = {NCInvertible[tp[Dzu] ** Dzu], NCInvertible[Z]}
Out[4]= {NCInvertible[Dzu^T.Dzu], NCInvertible[Z]}
```

Solve again for v2 and v3t

```
In[5]:= unknowns = {v2, v3t}
Out[5]= {v2, v3t}

In[6]:= solution = NCEliminate[Join[eqnst, invs], unknowns, 10];
        ColumnForm[solution]
Out[6]= v2 → v1
        v3t → -(Dzu^T.Dzu)^-1.Bu^T.X.v1 + (Dzu^T.Dzu)^-1.Bu^T.Z.v1
```

Figure 14: Computing $F^{\perp}$.

Compute general solution

```
In[7]:= vt = NCExpand[v/.hypothesis/.solution];
        vt = NCCollect[vt, tp[Bu]]
         (                    v1                    )
         (                    v1                    )
Out[7]=  ( -Dzu.(Dzu^T.Dzu)^-1.Bu^T.(X.v1 - Z.v1)   )
         (                    v4                    )
```

Generate basis

```
In[8]:= Fperp = MatrixFlatten[Matrix[vt/.{{v1 → Y, v4 → 0}, {v1 → 0, v4 → 1}}]]
         (                    Y                   0 )
         (                    Y                   0 )
Out[8]=  ( -Dzu.(Dzu^T.Dzu)^-1.Bu^T.(X.Y - Z.Y)   0 )
         (                    0                   1 )
```

Figure 15: Computing $(F^{\perp})^T E F^{\perp}$ and $(H^{\perp})^T E H^{\perp}$.

Compute the expressions in item ii) of the Elimination Lemma

```
In[1]:= FpEF1 = NCExpand[tp[Fperp] ** EE ** Fperp];
        HpEH1 = NCExpand[tp[Hperp] ** EE ** Hperp];
```

Figure 16: Simplifying $(F^{\perp})^T E F^{\perp}$ and $(H^{\perp})^T E H^{\perp}$ using orthogonality.

Define symmetry and orthogonality assumptions

```
In[2]:= symmetry = {tp[X] → X, tp[Z] → Z}
Out[2]= {Xᵀ → X, Zᵀ → Z}

In[3]:= ortho = {tp[Cz] * *Dzu → 0, tp[Dzu] * *Cz → 0, tp[Dzu] * *Dzu → 1, Dyw * *tp[Bw] → 0, Bw * *tp[Dyw] → 0, Dyw * *tp[Dyw] → 1}
Out[3]= {Czᵀ.Dzu → 0, Dzuᵀ.Cz → 0, Dzuᵀ.Dzu → 1, Dyw.Bwᵀ → 0, Bw.Dywᵀ → 0, Dyw.Dywᵀ → 1}
```

Apply the assumptions

```
In[4]:= FpEF2 = Substitute[FpEF1, Join[ortho, symmetry]];
        ColumnForm[SymmetricMatrixToVector[FpEF2]]
Out[4]= Yᵀ.X.Ap.Y - Yᵀ.Z.Ap.Y + Yᵀ.Apᵀ.X.Y - Yᵀ.Apᵀ.Z.Y - μ (Yᵀ.X.Bu.Buᵀ.X.Y - Yᵀ.X.Bu.Buᵀ.Z.Y - Yᵀ.Z.Bu.Buᵀ.X.Y + Yᵀ.Z.Bu.Buᵀ.Z.Y)
        Bwᵀ.X.Y - Bwᵀ.Z.Y - Dzwᵀ.Dzu.Buᵀ.X.Y + Dzwᵀ.Dzu.Buᵀ.Z.Y
        -μ

In[5]:= HpEH2 = Substitute[HpEH1, Join[ortho, symmetry]]
Out[5]= ( X.Ap + Apᵀ.X - μ Cyᵀ.Cy   Czᵀ - Cyᵀ.Dyw.Dzwᵀ )
        (    Cz - Dzw.Dywᵀ.Cy              -μ          )
```

compute a particular case of these inequalities in the presence of the orthogonality assumptions

$$C_z^T D_{zu} = 0, \qquad D_{zu}^T D_{zu} = I, \qquad B_w D_{yw}^T = 0, \qquad D_{yw} D_{yw}^T = I. \tag{21}$$

It is interesting to notice that these assumptions can be enforced without loss of generality, in the sense that a procedure is available to transform a general H_∞ control problem in one for which these relations are true (see [ZDG96], for instance). Anyway, the practical result here is that significantly simpler expressions can be obtained in Figure 16.

From these expressions we note that $(H^{\perp})^T E H^{\perp}$ is already affine on the variable X, while $(F^{\perp})^T E F^{\perp}$ is not. Indeed, $(F^{\perp})^T E F^{\perp}$ is still too complicated to be displayed in one single line and we use the function `SymmetricMatrixToVector` to convert this symmetric matrix into a list, which leaves out redundant entries. This list is displayed vertically using `Mathematica`'s formatting operator `ColumnForm`.

We use the strategy discussed in Section 6.2 to try to *motivate unknowns* that could simplify the matrix $(F^{\perp})^T E F^{\perp}$. In figure 17, we first *collect on knowns*, that is, we use the command `NCCollectOnVariables` to apply collect transformations on $(H^{\perp})^T E H^{\perp}$ involving all known matrices from the plant. In the transformed expression we note the presence of the term $(XY - ZY)$. This motivates the introduction of a new unknown defined by the relation

$$U := (X - Z), \tag{22}$$

which is substituted in the expression in order to eliminate the variable Z. The resulting expression

Figure 17: Making $(F^{\perp})^T E F^{\perp}$ affine.

Collect on all known plant parameters

```
In[6]:= FpEF3 = NCCollectOnVariables[FpEF2, {Ap, tp[Ap], Bu, tp[Bu], Bw, tp[Bw], Dzu, tp[Dzu], Dzw, tp[Dzw]}];
        ColumnForm[SymmetricMatrixToVector[FpEF3]]
Out[6]= Y^T.Ap^T.(X.Y - Z.Y) + (Y^T.X - Y^T.Z).Ap.Y - μ (Y^T.X - Y^T.Z).Bu.Bu^T.(X.Y - Z.Y)
        Bw^T.(X.Y - Z.Y) - Dzw^T.Dzu.Bu^T.(X.Y - Z.Y)
        -μ
```

Introduce U:=X - Z

```
In[7]:= FpEF4 = Substitute[FpEF3, {Z → X - U}];
        FpEF5 = NCSimplifyRational[FpEF4]
Out[7]= (Y^T.U.Ap.Y + Y^T.Ap^T.U.Y - μ Y^T.U.Bu.Bu^T.U.Y   Y^T.U.Bw - Y^T.U.Bu.Dzu^T.Dzw )
        (        Bw^T.U.Y - Dzw^T.Dzu.Bu^T.U.Y                          -μ               )
```

Make it affine by chosing U = Y^{-1}

```
In[8]:= FpEF6 = FpEF5/.{U → inv[Y], tp[Y] → Y}
Out[8]= (Ap.Y - μ Bu.Bu^T + Y.Ap^T   Bw - Bu.Dzu^T.Dzw )
        (  Bw^T - Dzw^T.Dzu.Bu^T             -μ         )
```

can be made linear in the variable Y by choosing the product UY to be constant, say the identity matrix. That is, by choosing $U = Y^{-1}$. The resulting affine expression is given in the bottom of Figure 17. It is interesting to note that setting U to be the the inverse of Y we are implicitly setting $Y = (X - Z)^{-1} > 0$. Indeed, we would obtain the same results if we had chosen Y to assume this value in the computation of $F^{\perp}$. This explains why Y was not made equal to the identity matrix at that point!

We have converted the expressions $(F^{\perp})^T E F^{\perp} < 0$ and $(H^{\perp})^T E H^{\perp} < 0$ into LMIs on the variables X and Y. However, we have ignored the first inequality $\mathcal{P} > 0$ in (20). Recalling that $Y = (X - Z)^{-1}$ we have that

$$Z = X - Y^{-1} \tag{23}$$

so that

$$\mathcal{P} = \begin{bmatrix} X & Y^{-1} - X \\ Y^{-1} - X & X - Y^{-1} \end{bmatrix} > 0. \tag{24}$$

Using the Schur Complement Lemma the above inequality can be rewritten as

$$X > 0, \quad Y^{-1} > 0, \quad X - Y^{-1} > 0, \tag{25}$$

which, using Schur Complement Lemma one more time, can be simply restated as the LMI

$$\begin{bmatrix} X & I \\ I & Y \end{bmatrix} > 0. \tag{26}$$

This implies that set of feasible H_∞ dynamic output feedback controllers can be equivalently represented as a set of LMIs.

5.4 Changing the controller variables

We now explore the second methodology for obtaining LMIs, which consists in changing the controller variables [SGC97]. Before we start, we introduce the nonsingular matrices Y and

$$\mathcal{T} := \begin{bmatrix} I & Y \\ 0 & Y \end{bmatrix}.$$

The closed loop system state $\tilde{x}$, given in (19), is then transformed as

$$\bar{x} = \mathcal{T}^{-1}\tilde{x} = \begin{bmatrix} I & -I \\ 0 & Y^{-1} \end{bmatrix} \begin{pmatrix} x \\ x_c \end{pmatrix} = \begin{pmatrix} x - x_c \\ Y^{-1}x_c \end{pmatrix}.$$

If x_c is interpreted as an estimate of the plant state, the above transformation results in a closed loop realization where the state $\bar{x}$ is the plant state estimation error $(x - x_c)$, and the transformed controller state $Y^{-1}x_c$. There are two motivations for applying this change of coordinates. The first is to introduce the extra variable Y, whose role is to parametrize a change of coordinates in the controller state. The second, as will be shown later, is to make the controller parameter A_c appear in only one term of the closed loop bounded real lemma. These facts will be important when computing changes of variables.

We start the symbolic computation by applying the above similarity transformation on the closed loop system, i.e., we redefine the closed loop matrices as

$$(\mathcal{P}, \mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}) \longleftarrow \left(\mathcal{T}^T\mathcal{P}\mathcal{T}\,,\, \mathcal{T}^{-1}\mathcal{A}\mathcal{T}\,,\, \mathcal{T}^{-1}\mathcal{B}\,,\, \mathcal{C}\mathcal{T}\,,\, \mathcal{D}\right).$$

The transformations are computed in Figure 18 and used to produce the matrix coefficient of the second inequality of (20) for the transformed closed loop system. We now use our previously discussed strategy of *collecting on the knowns* to motivate new unknowns. The matrix expressions obtained in this section are mostly long formulas, and we make extensive use of the command

Figure 18: Changing the closed loop coordinates.

Define similarity transformation matrix

```
In[1]:= T = Matrix[{1, Y}, {0, Y}]
Out[1]= (1  Y )
        (0  Y )

In[2]:= inv[T]
Out[2]= (1  -1  )
        (0  Y^-1)
```

Apply transformation on the closed loop matrices

```
In[3]:= {PclT, AclT, BclT, CclT, DclT} = NCExpand[{tp[T] ** Pcl ** T, inv[T] ** Acl ** T, inv[T] ** Bcl, Ccl ** T, Dcl}];
```

Substitute the closed loop matrices into the BRL

```
In[4]:= BRLclT0 = MatrixFlatten[NCExpand[BRLol/.{Pol → PclT, Aol → AclT, Bol → BclT, Col → CclT, Dol → DclT}]];
```

Figure 19: Collecting on the knowns.

Collect on knowns Cy, Dyw, Bw and Ap

```
In[5]:= BRLclT1 = NCCollectOnVariables[BRLclT0, {Cy, tp[Cy], Dyw, tp[Dyw], Bw, tp[Bw], Ap, tp[Ap]}];
        ColumnForm[SymmetricMatrixToVector[BRLclT1]]
Out[5]= X.Ap + Ap^T.X - Cy^T.(Bc^T.Z - Dc^T.Bu^T.X) - (Z.Bc - X.Bu.Dc).Cy
        (Y^T.X - Y^T.Z).Ap + (Y^T.X.Bu.Dc - Y^T.Z.Bu.Dc).Cy - Y^T.Ac^T.Z + Y^T.Ap^T.X - Y^T.Cy^T.(Bc^T.Z - Dc^T.Bu^T.X) + Y^T.Cc^T.Bu^T.X
        Y^T.Ap^T.(X.Y - Z.Y) + Y^T.Cy^T.(Dc^T.Bu^T.X.Y - Dc^T.Bu^T.Z.Y) + (Y^T.X - Y^T.Z).Ap.Y + (Y^T.X.Bu.Dc - Y^T.Z.Bu.Dc).Cy.Y + Y^T.X.Bu.Cc.Y - Y^T.Z.Bu.Cc.Y + Y^T.Cc^T.Bu^T.X.Y
        Cz + Dzu.Dc.Cy
        Cz.Y + Dzu.Cc.Y + Dzu.Dc.Cy.Y
        -μ
        Bw^T.X - Dyw^T.(Bc^T.Z - Dc^T.Bu^T.X)
        Bw^T.(X.Y - Z.Y) + Dyw^T.(Dc^T.Bu^T.X.Y - Dc^T.Bu^T.Z.Y)
        Dzw^T + Dyw^T.Dc^T.Dzu^T
        -μ
```

Collect on known Bu

```
In[6]:= BRLclT2 = NCCollectSymmetric[BRLclT1, Bu];
        BRLclT3 = NCCollectSymmetric[BRLclT2, Bu];
        ColumnForm[SymmetricMatrixToVector[BRLclT3]]
Out[6]= X.Ap + Ap^T.X - Cy^T.(Bc^T.Z - Dc^T.Bu^T.X) - (Z.Bc - X.Bu.Dc).Cy
        (Y^T.X - Y^T.Z).Ap - Y^T.Ac^T.Z + Y^T.Ap^T.X - Y^T.Cy^T.(Bc^T.Z - Dc^T.Bu^T.X) + Y^T.Cc^T.Bu^T.X + (Y^T.X - Y^T.Z).Bu.Dc.Cy
        Y^T.Ap^T.(X.Y - Z.Y) + (Y^T.X - Y^T.Z).Ap.Y + (Y^T.X - Y^T.Z).Bu.(Cc.Y + Dc.Cy.Y) + (Y^T.Cc^T + Y^T.Cy^T.Dc^T).Bu^T.(X.Y - Z.Y)
        Cz + Dzu.Dc.Cy
        Cz.Y + Dzu.Cc.Y + Dzu.Dc.Cy.Y
        -μ
        Bw^T.X - Dyw^T.(Bc^T.Z - Dc^T.Bu^T.X)
        Bw^T.(X.Y - Z.Y) + Dyw^T.Dc^T.Bu^T.(X.Y - Z.Y)
        Dzw^T + Dyw^T.Dc^T.Dzu^T
        -μ
```

`SymmetricMatrixToVector` to convert the lower diagonal part of symmetric matrices into lists which can be better visualized using `Mathematica`'s formatting operator `ColumnForm`.

In Figure 19, we collect on some of the plant matrices to motivate unknowns, first using the function `NCCollectOnVariables`. We note the presence of a couple of terms that have been collected (the parenthesized expressions) in `BRLclT1`. We collect further on the known B_u using `NCCollectSymmetric`. In expression `BRLclT3`, we see at least three terms which look like good candidates for changing variables. We introduce one new variable at a time. We start by

$$New_1 := -(ZB_c - XB_uD_c), \tag{27}$$

in Figure 20. We use `NCElminate` to eliminate B_c from this expression using invertibility assump-

Figure 20: Introducing the first new unknown.

```
Introduce variable New1=-(Z**Bc-X**Bu**Dc)

In[7]:= NewVar1eq = {-(Z**Bc - X**Bu**Dc) - New1}
Out[7]= {-New1 - Z.Bc + X.Bu.Dc}

Invertibility hypothesis on X, Y, Z and (X - Z)

In[8]:= invs1assumption = {NCInvertible[X], NCInvertible[Y], NCInvertible[Z], NCInvertible[X - Z]}
Out[8]= {NCInvertible[X], NCInvertible[Y], NCInvertible[Z], NCInvertible[X - Z]}

Implement the change of variables (eliminate Bc)

In[9]:= solution1 = NCEliminate[Join[NewVar1eq, invs1assumption], {Bc}, 2];
        ColumnForm[solution1]
Out[9]= Z.(X - Z)^-1 → -1 + X.(X - Z)^-1
        (X - Z)^-1.Z → -1 + (X - Z)^-1.X
        Z^-1.X.(X - Z)^-1 → (X - Z)^-1 + Z^-1
        (X - Z)^-1.X.Z^-1 → (X - Z)^-1 + Z^-1
        Bc → -Z^-1.New1 + Z^-1.X.Bu.Dc
```

Figure 21: Eliminating B_c in the closed loop Bounded Real Lemma.

```
Symmetric hypothesis on X and Z

In[10]:= syms1assumption = {tp[X] → X, tp[Z] → Z}
Out[10]= {X^T → X, Z^T → Z}

Eliminate Bc from the closed loop BRL

In[11]:= BRLclT4 = NCExpand[BRLclT3/.solution1/.syms1assumption];
         ColumnForm[SymmetricMatrixToVector[BRLclT4]]
Out[11]= New1.Cy + X.Ap + Ap^T.X + Cy^T.New1^T
         Y^T.X.Ap - Y^T.Z.Ap - Y^T.Ac^T.Z + Y^T.Ap^T.X + Y^T.Cy^T.New1^T + Y^T.Cc^T.Bu^T.X + Y^T.X.Bu.Dc.Cy - Y^T.Z.Bu.Dc.Cy
         Y^T.X.Ap.Y - Y^T.Z.Ap.Y + Y^T.Ap^T.X.Y - Y^T.Ap^T.Z.Y + Y^T.X.Bu.Cc.Y - Y^T.Z.Bu.Cc.Y + Y^T.Cc^T.Bu^T.X.Y - Y^T.Cc^T.Bu^T.Z.Y + Y^T.X.Bu.Dc.Cy.Y - Y^T.Z.Bu.Dc.Cy.Y + Y^T.Cy
         Cz + Dzu.Dc.Cy
         Cz.Y + Dzu.Cc.Y + Dzu.Dc.Cy.Y
         -μ
         Bw^T.X + Dyw^T.New1^T
         Bw^T.X.Y - Bw^T.Z.Y + Dyw^T.Dc^T.Bu^T.X.Y - Dyw^T.Dc^T.Bu^T.Z.Y
         Dzw^T + Dyw^T.Dc^T.Dzu^T
         -μ
```

tions on the matrices X, Z and $(X - Z)$ (since a full order controller is being used) and on Y (which parametrizes a nonsingular similarity transformation). The result is used to eliminate B_c from the closed loop Bounded Real Lemma in Figure 21, where we have also used rules to enforce the symmetry of X and Z. This procedure will be repeated until an LMI is obtained.

In Figure 22, we collect on knowns to motivate the new unknown

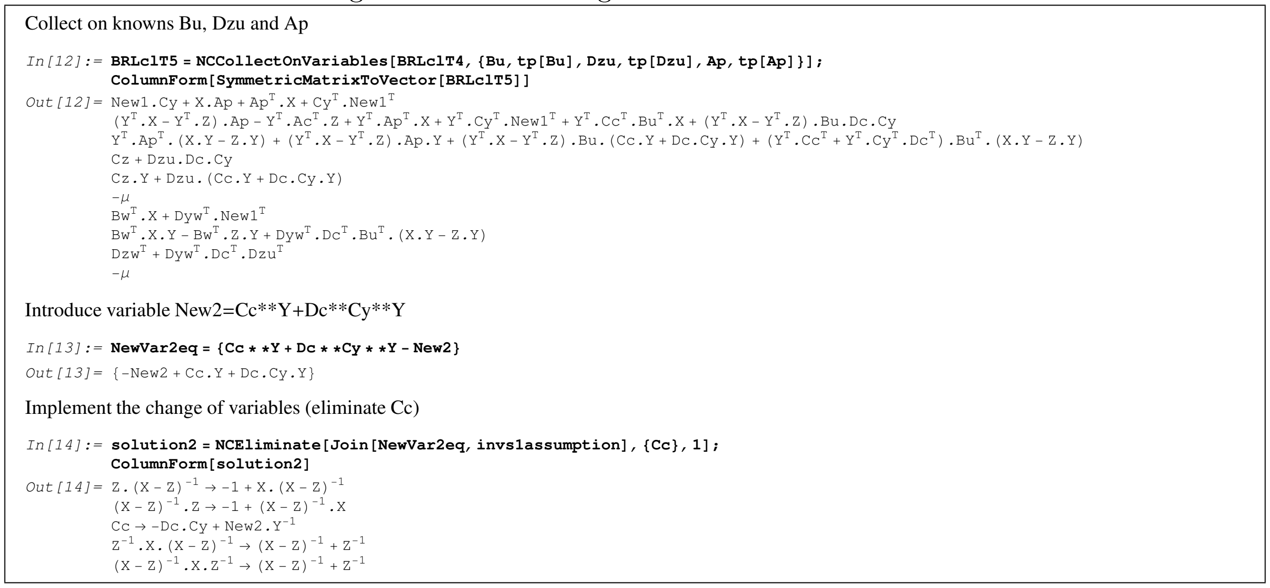
$$New_2 := C_c Y + D_c C_y Y, \tag{28}$$

which is used to eliminate C_c. The variable C_c is then eliminated from the Bounded Real Lemma and the third new unknown

$$U := X - Z, \tag{29}$$

is motivated after using `NCCollectOnVariables` in Figure 23. Note that this motivated unknown have already appeared in Section 5.3.2, as expression (22). For this reason, we use the same

Figure 22: Introducing the second new unknown.



Collect on knowns Bu, Dzu and Ap

```
In[12]:= BRLclT5 = NCCollectOnVariables[BRLclT4, {Bu, tp[Bu], Dzu, tp[Dzu], Ap, tp[Ap]}];
         ColumnForm[SymmetricMatrixToVector[BRLclT5]]
Out[12]= New1.Cy + X.Ap + Ap^T.X + Cy^T.New1^T
         (Y^T.X - Y^T.Z).Ap - Y^T.Ac^T.Z + Y^T.Ap^T.X + Y^T.Cy^T.New1^T + Y^T.Cc^T.Bu^T.X + (Y^T.X - Y^T.Z).Bu.Dc.Cy
         Y^T.Ap^T.(X.Y - Z.Y) + (Y^T.X - Y^T.Z).Ap.Y + (Y^T.X - Y^T.Z).Bu.(Cc.Y + Dc.Cy.Y) + (Y^T.Cc^T + Y^T.Cy^T.Dc^T).Bu^T.(X.Y - Z.Y)
         Cz + Dzu.Dc.Cy
         Cz.Y + Dzu.(Cc.Y + Dc.Cy.Y)
         -μ
         Bw^T.X + Dyw^T.New1^T
         Bw^T.X.Y - Bw^T.Z.Y + Dyw^T.Dc^T.Bu^T.(X.Y - Z.Y)
         Dzw^T + Dyw^T.Dc^T.Dzu^T
         -μ
```

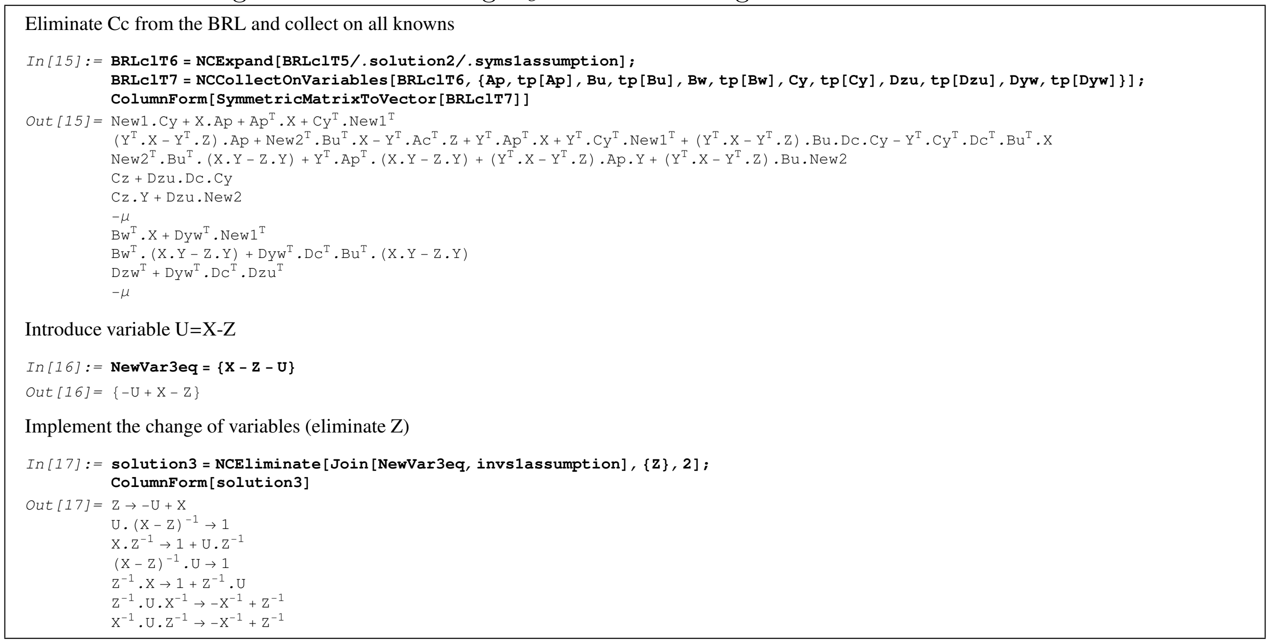
Introduce variable New2=Cc**Y+Dc**Cy**Y

```
In[13]:= NewVar2eq = {Cc ** Y + Dc ** Cy ** Y - New2}
Out[13]= {-New2 + Cc.Y + Dc.Cy.Y}
```

Implement the change of variables (eliminate Cc)

```
In[14]:= solution2 = NCEliminate[Join[NewVar2eq, invs1assumption], {Cc}, 1];
         ColumnForm[solution2]
Out[14]= Z.(X - Z)^-1 → -1 + X.(X - Z)^-1
         (X - Z)^-1.Z → -1 + (X - Z)^-1.X
         Cc → -Dc.Cy + New2.Y^-1
         Z^-1.X.(X - Z)^-1 → (X - Z)^-1 + Z^-1
         (X - Z)^-1.X.Z^-1 → (X - Z)^-1 + Z^-1
```

Figure 23: Eliminating C_c and introducing the third new unknown.



Eliminate Cc from the BRL and collect on all knowns

```
In[15]:= BRLclT6 = NCExpand[BRLclT5/.solution2/.syms1assumption];
         BRLclT7 = NCCollectOnVariables[BRLclT6, {Ap, tp[Ap], Bu, tp[Bu], Bw, tp[Bw], Cy, tp[Cy], Dzu, tp[Dzu], Dyw, tp[Dyw]}];
         ColumnForm[SymmetricMatrixToVector[BRLclT7]]
Out[15]= New1.Cy + X.Ap + Ap^T.X + Cy^T.New1^T
         (Y^T.X - Y^T.Z).Ap + New2^T.Bu^T.X - Y^T.Ac^T.Z + Y^T.Ap^T.X + Y^T.Cy^T.New1^T + (Y^T.X - Y^T.Z).Bu.Dc.Cy - Y^T.Cy^T.Dc^T.Bu^T.X
         New2^T.Bu^T.(X.Y - Z.Y) + Y^T.Ap^T.(X.Y - Z.Y) + (Y^T.X - Y^T.Z).Ap.Y + (Y^T.X - Y^T.Z).Bu.New2
         Cz + Dzu.Dc.Cy
         Cz.Y + Dzu.New2
         -μ
         Bw^T.X + Dyw^T.New1^T
         Bw^T.(X.Y - Z.Y) + Dyw^T.Dc^T.Bu^T.(X.Y - Z.Y)
         Dzw^T + Dyw^T.Dc^T.Dzu^T
         -μ
```

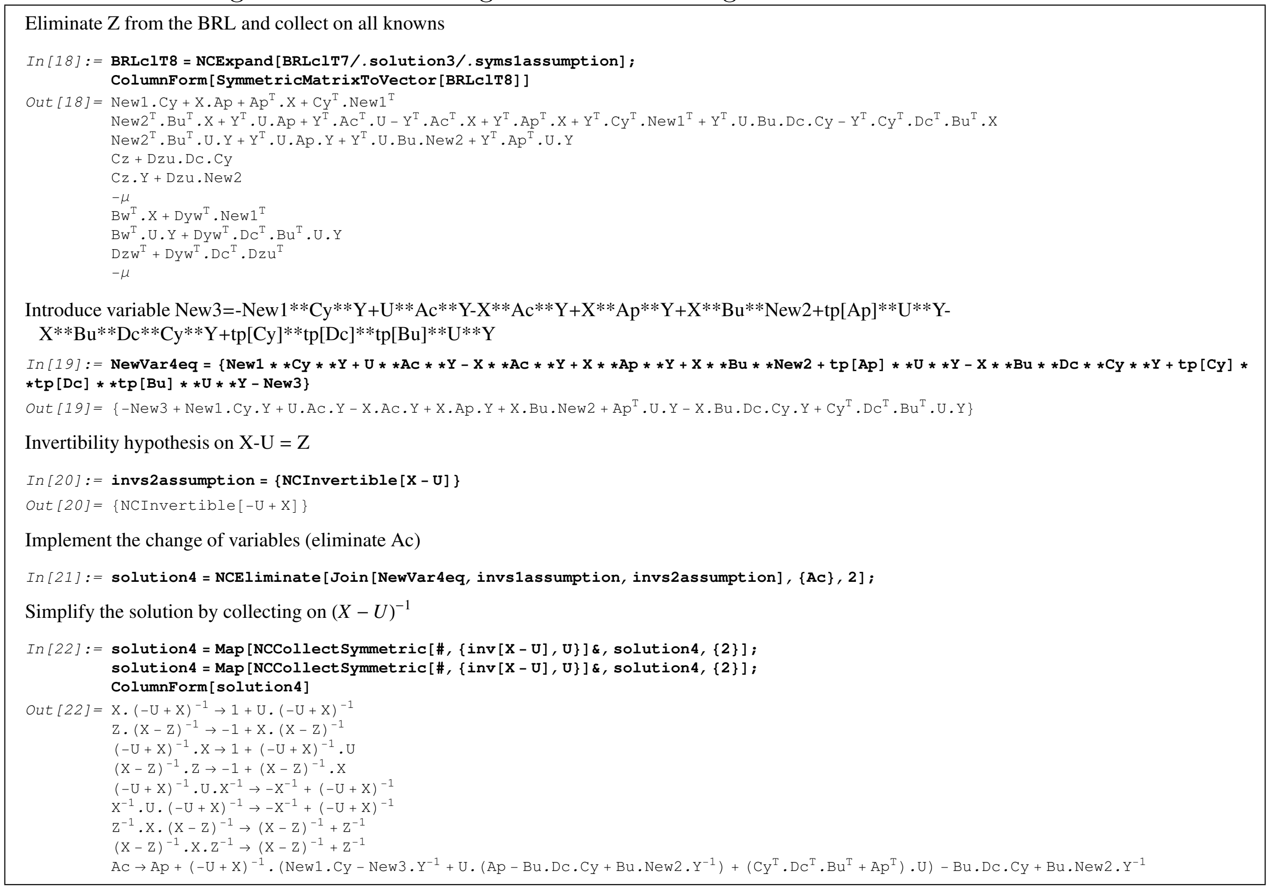
Introduce variable U=X-Z

```
In[16]:= NewVar3eq = {X - Z - U}
Out[16]= {-U + X - Z}
```

Implement the change of variables (eliminate Z)

```
In[17]:= solution3 = NCEliminate[Join[NewVar3eq, invs1assumption], {Z}, 2];
         ColumnForm[solution3]
Out[17]= Z → -U + X
         U.(X - Z)^-1 → 1
         X.Z^-1 → 1 + U.Z^-1
         (X - Z)^-1.U → 1
         Z^-1.X → 1 + Z^-1.U
         Z^-1.U.X^-1 → -X^-1 + Z^-1
         X^-1.U.Z^-1 → -X^-1 + Z^-1
```

Figure 24: Eliminating Z and introducing the fourth new unknown.



Eliminate Z from the BRL and collect on all knowns

```
In[18]:= BRLclT8 = NCExpand[BRLclT7/.solution3/.syms1assumption];
         ColumnForm[SymmetricMatrixToVector[BRLclT8]]
Out[18]= New1.Cy + X.Ap + Ap^T.X + Cy^T.New1^T
         New2^T.Bu^T.X + Y^T.U.Ap + Y^T.Ac^T.U - Y^T.Ac^T.X + Y^T.Ap^T.X + Y^T.Cy^T.New1^T + Y^T.U.Bu.Dc.Cy - Y^T.Cy^T.Dc^T.Bu^T.X
         New2^T.Bu^T.U.Y + Y^T.U.Ap.Y + Y^T.U.Bu.New2 + Y^T.Ap^T.U.Y
         Cz + Dzu.Dc.Cy
         Cz.Y + Dzu.New2
         -μ
         Bw^T.X + Dyw^T.New1^T
         Bw^T.U.Y + Dyw^T.Dc^T.Bu^T.U.Y
         Dzw^T + Dyw^T.Dc^T.Dzu^T
         -μ
```

Introduce variable New3=-New1**Cy**Y+U**Ac**Y-X**Ac**Y+X**Ap**Y+X**Bu**New2+tp[Ap]**U**Y-X**Bu**Dc**Cy**Y+tp[Cy]**tp[Dc]**tp[Bu]**U**Y

```
In[19]:= NewVar4eq = {New1 * *Cy * *Y + U * *Ac * *Y - X * *Ac * *Y + X * *Ap * *Y + X * *Bu * *New2 + tp[Ap] * *U * *Y - X * *Bu * *Dc * *Cy * *Y + tp[Cy] *
*tp[Dc] * *tp[Bu] * *U * *Y - New3}
Out[19]= {-New3 + New1.Cy.Y + U.Ac.Y - X.Ac.Y + X.Ap.Y + X.Bu.New2 + Ap^T.U.Y - X.Bu.Dc.Cy.Y + Cy^T.Dc^T.Bu^T.U.Y}
```

Invertibility hypothesis on X-U = Z

```
In[20]:= invs2assumption = {NCInvertible[X - U]}
Out[20]= {NCInvertible[-U + X]}
```

Implement the change of variables (eliminate Ac)

```
In[21]:= solution4 = NCEliminate[Join[NewVar4eq, invs1assumption, invs2assumption], {Ac}, 2];
```

Simplify the solution by collecting on $(X-U)^{-1}$

```
In[22]:= solution4 = Map[NCCollectSymmetric[#, {inv[X - U], U}]&, solution4, {2}];
         solution4 = Map[NCCollectSymmetric[#, {inv[X - U], U}]&, solution4, {2}];
         ColumnForm[solution4]
Out[22]= X.(-U + X)^-1 → 1 + U.(-U + X)^-1
         Z.(X - Z)^-1 → -1 + X.(X - Z)^-1
         (-U + X)^-1.X → 1 + (-U + X)^-1.U
         (X - Z)^-1.Z → -1 + (X - Z)^-1.X
         (-U + X)^-1.U.X^-1 → -X^-1 + (-U + X)^-1
         X^-1.U.(-U + X)^-1 → -X^-1 + (-U + X)^-1
         Z^-1.X.(X - Z)^-1 → (X - Z)^-1 + Z^-1
         (X - Z)^-1.X.Z^-1 → (X - Z)^-1 + Z^-1
         Ac → Ap + (-U + X)^-1.(New1.Cy - New3.Y^-1 + U.(Ap - Bu.Dc.Cy + Bu.New2.Y^-1) + (Cy^T.Dc^T.Bu^T + Ap^T).U) - Bu.Dc.Cy + Bu.New2.Y^-1
```

symbol U as before, and save New_3 for the next unknown.

In Figure 24, the variable Z is eliminated and the fourth new unknown

$$New_3 := New_1C_yY + UA_cY - XA_cY + XA_pY + XB_uNew_2 + A_p^TUY - XB_uD_cC_yY + C_y^TD_c^TB_u^TUY \quad (30)$$

is motivated. This variable has been motivated not by collecting on the knowns, but after observing that we can take advantage of the fact that variable A_c appears in only one entry of the bounded real lemma coefficient. Recall that this was exactly one of the purposes of introducing a preliminary change of coordinates in the closed loop system. This new unknown is then introduced so as to replace this entire entry. Notice that we have used the fact that $X-U=Z$ is nonsingular. We have also used the function `NCCollectSymmetric` on `solution4` in order to collect terms in $(X-U)^{-1}$. Variable A_c is eliminated from the closed loop Bounded Real Lemma in Figure 25. The result is

Figure 25: Eliminating A_c.

Symmetric hypothesis

```
In[23]:= syms2assumption = {tp[U] → U}
Out[23]= {U^T → U}
```

Eliminate Ac and simplify resulting expression

```
In[24]:= BRLclT9 = BRLclT8/.solution4/.syms1assumption/.syms2assumption;
         BRLclT10 = NCSimplifyRational[BRLclT9];
         ColumnForm[SymmetricMatrixToVector[BRLclT10]]
Out[24]= New1.Cy + X.Ap + Ap^T.X + Cy^T.New1^T
         New3^T
         New2^T.Bu^T.U.Y + Y^T.U.Ap.Y + Y^T.U.Bu.New2 + Y^T.Ap^T.U.Y
         Cz + Dzu.Dc.Cy
         Cz.Y + Dzu.New2
         -μ
         Bw^T.X + Dyw^T.New1^T
         Bw^T.U.Y + Dyw^T.Dc^T.Bu^T.U.Y
         Dzw^T + Dyw^T.Dc^T.Dzu^T
         -μ
```

Figure 26: Changing variables in $\mathcal{P}$.

Collect on Y

```
In[25]:= PclT1 = NCCollectSymmetric[PclT, Y]
Out[25]= (     X              (X - Z).Y   )
         ( Y^T.(X - Z)    Y^T.(X - Z).Y  )
```

Use change of variables from solution3

```
In[26]:= PclT2 = PclT1/.solution3
Out[26]= (  X      U.Y   )
         ( Y^T.U  Y^T.U.Y )
```

a noncommutative rational function, and we invoke the function `NCSimplifyRational` to simplify this expression.

The expression `BRLclT10` is much simpler than the one we originally started, although it is not yet an LMI. We now move our attention to the first inequality in (20), and apply the above changes of variable so as to express $\mathcal{P} > 0$ in terms of the same set of variables. This is done in Figure 26, where we see that the only change of variables that need to be applied is the one that eliminates variable Z. Indeed, collecting on Y motivates the introduction of U as in (29).

The expressions `BRLclT10` and `PclT1` are almost affine on the unknowns X, Y, New_1, New_2, New_3, and D_c, except for the presence of the product UY. Again, as Section 5.3.2, the fact that U appears only in the form of products with Y can be used to make these expressions affine by choosing $U = Y^{-1}$. This is performed in Figure 27, where the final form of the coefficient matrices of the Bounded Real Lemma inequalities are expressed as affine functions of the set of new unknowns.

At this point we use the change of variable to produce an explicit formula for the controller parameters A_c, B_c, C_c and D_c in terms of the set of new unknowns. This is done in Figure 28,

Figure 27: Obtaining the LMI.

Set U -> Y^{-1}

```
In[27]:= linearize = {U → inv[Y], tp[Y] → Y}
```

$$\text{Out[27]= } \{U \to Y^{-1}, Y^T \to Y\}$$

on the Bounded Real Lemma

```
In[28]:= BRLclT11 = BRLclT10/.linearize
```

$$\text{Out[28]= } \begin{pmatrix} New1.Cy + X.Ap + Ap^T.X + Cy^T.New1^T & New3 & Cz^T + Cy^T.Dc^T.Dzu^T & New1.Dyw + X.Bw \\ New3^T & Ap.Y + Bu.New2 + Y.Ap^T + New2^T.Bu^T & Y.Cz^T + New2^T.Dzu^T & Bw + Bu.Dc.Dyw \\ Cz + Dzu.Dc.Cy & Cz.Y + Dzu.New2 & -\mu & Dzw + Dzu.Dc.Dyw \\ Bw^T.X + Dyw^T.New1^T & Bw^T + Dyw^T.Dc^T.Bu^T & Dzw^T + Dyw^T.Dc^T.Dzu^T & -\mu \end{pmatrix}$$

and on Pcl

```
In[29]:= PclT3 = PclT2/.linearize
```

$$\text{Out[29]= } \begin{pmatrix} X & 1 \\ 1 & Y \end{pmatrix}$$

Figure 28: The controller formula.

Compute controller formula

```
In[30]:= K = Matrix[{Ac, Bc}, {Cc, Dc}]
```

$$\text{Out[30]= } \begin{pmatrix} Ac & Bc \\ Cc & Dc \end{pmatrix}$$

by inverting the change of variables

```
In[31]:= K1 = Substitute[K, Join[solution1, solution2, solution3, solution4, linearize]];
         K2 = NCCollectSymmetric[NCSimplifyRational[K1], inv[X - inv[Y]]]
```

$$\text{Out[31]= } \begin{pmatrix} (X - Y^{-1})^{-1}.(New1.Cy - New3.Y^{-1} + X.Ap + Ap^T.Y^{-1} - X.Bu.Dc.Cy + X.Bu.New2.Y^{-1} + Cy^T.Dc^T.Bu^T.Y^{-1}) & -(X - Y^{-1})^{-1}.(New1 - X.Bu.Dc) \\ -Dc.Cy + New2.Y^{-1} & Dc \end{pmatrix}$$

Factor as affine function of the transformed variables

```
In[32]:= K3 = MatrixAffineFactor[K2, {New1, New2, New3, Dc}]
```

$$\text{Out[32]= } \begin{pmatrix} (X - Y^{-1})^{-1}.Cy^T \\ 0 \end{pmatrix}.Dc^T.\begin{pmatrix} Bu^T.Y^{-1} & 0 \end{pmatrix} + \begin{pmatrix} (X - Y^{-1})^{-1} & -(X - Y^{-1})^{-1}.X.Bu \\ 0 & -1 \end{pmatrix}.\begin{pmatrix} New1 & New3 \\ Dc & New2 \end{pmatrix}.\begin{pmatrix} Cy & -1 \\ -Y^{-1} & 0 \end{pmatrix} + \begin{pmatrix} (X - Y^{-1})^{-1}.X.Ap + (X - Y^{-1})^{-1}.Ap^T.Y^{-1} & 0 \\ 0 & 0 \end{pmatrix}$$

where we also factorize the obtained formula as an affine function of the unknowns New_1, New_2, New_3 and D_c. Note that the result states that A_c, B_c and C_c depend on D_c, which is the only controller parameter that has not been transformed.

Finally, in this example, all three terms that have been motivated in expression `BRLclT3`, in Figure 19, could have been used to eliminate variables B_c, C_c and Z all at one once. This is done in Figure 29, where we feed `NCEliminate` with three equations defining three new unknowns and simultaneously eliminate B_c, C_c and Z. The solution is used to produced `BRLclT13`, which is equal to `BRLclT8`, in Figure 24. This skips five intermediate expressions.

6 Wrapping it all together: a word about algorithms

In this section we provide the reader with a very brief introduction to the algorithms used behind the `NCAlgebra` commands used in this paper.

Figure 29: Eliminating B_c, C_c and Z simultaneously.

Introduce the new unknowns New1, New2 and U all at once

```
In[33]:= NewVar123eq = Join[NewVar1eq, NewVar2eq, NewVar3eq];
         ColumnForm[NewVar123eq]
Out[33]= -New1 - Z.Bc + X.Bu.Dc
         -New2 + Cc.Y + Dc.Cy.Y
         -U + X - Z
```

Implement the change of variables (eliminate Bc, Cc and Z)

```
In[34]:= solution5 = NCEliminate[Join[NewVar123eq, invs1assumption], {Bc, Cc, Z}, 2];
         ColumnForm[solution5]
Out[34]= Z → -U + X
         U.(X - Z)^-1 → 1
         X.Z^-1 → 1 + U.Z^-1
         (X - Z)^-1.U → 1
         Z^-1.X → 1 + Z^-1.U
         Z^-1.U.X^-1 → -X^-1 + Z^-1
         Cc → -Dc.Cy + New2.Y^-1
         X^-1.U.Z^-1 → -X^-1 + Z^-1
         Bc → Bu.Dc - Z^-1.New1 + Z^-1.U.Bu.Dc
```

Eliminate Bc, Cc and Z from the BRL

```
In[35]:= BRLclT12 = NCExpand[Substitute[BRLclT3, solution5]/.syms1assumption/.syms2assumption];
         BRLclT13 = NCSimplifyRational[BRLclT12];
         ColumnForm[SymmetricMatrixToVector[BRLclT13]]
Out[35]= New1.Cy + X.Ap + Ap^T.X + Cy^T.New1^T
         New2^T.Bu^T.X + Y^T.U.Ap + Y^T.Ac^T.U - Y^T.Ac^T.X + Y^T.Ap^T.X + Y^T.Cy^T.New1^T + Y^T.U.Bu.Dc.Cy - Y^T.Cy^T.Dc^T.Bu^T.X
         New2^T.Bu^T.U.Y + Y^T.U.Ap.Y + Y^T.U.Bu.New2 + Y^T.Ap^T.U.Y
         Cz + Dzu.Dc.Cy
         Cz.Y + Dzu.New2
         -μ
         Bw^T.X + Dyw^T.New1^T
         Bw^T.U.Y + Dyw^T.Dc^T.Bu^T.U.Y
         Dzw^T + Dyw^T.Dc^T.Dzu^T
         -μ
```

6.1 Eliminating variables via Gröbner basis

Commutative Gröbner basis algorithms are powerful and make up the engines in symbolic algebra packages' `Solve` commands. Non commutative Gröbner basis algorithms are more recent but have similar potential.

The Non Commutative Gröbner Basis Algorithm, due to F. Mora [Mor86, GHK97], can be used to systematically eliminate variables from a collection (e.g., $\{p_j(x_1, \ldots, x_n) = 0 : 1 \le j \le k_1\}$) of polynomial equations so as to put it in triangular form.

Our computational implementation of this algorithm is the package `NCGB`. One specifies an order[5] on the variables ($x_1 < x_2 < x_3 < \ldots < x_n$) which corresponds to priorities in eliminating them. Here `NCGB` will try hardest to eliminate x_n and try the least to eliminate x_1. The output is

[5]From this ordering on variables (written $<$), an order on monomials in those variables is induced which is referred to as *non commutative graded lexicographic order.* In this paper we also write $\ll$ to mean a *non commutative pure lexicographic order.*

a list of equations in a "canonical form" which is triangular

$$
\begin{aligned}
q_1(x_1) &= 0,\\
q_2(x_1, x_2) &= 0,\\
q_3(x_1, x_2) &= 0,\\
q_4(x_1, x_2, x_3) &= 0,\\
&\vdots\\
q_{k_2}(x_1, \ldots, x_n) &= 0.
\end{aligned}
\tag{31}
$$

The set of solutions to the collection of polynomial equations $\{q_j = 0 : 1 \leq j \leq k_2\}$ equals[6] the set of solutions to the collection of polynomial equations $\{p_j = 0 : 1 \leq j \leq k_1\}$. This canonical form greatly simplifies the task of solving the collection of polynomial equations by facilitating back-solving for x_j in terms of $x_1, \ldots, x_{j-1}$. The effect of the ordering is to specify that variables high in the order will be eliminated, while variables low in the order will not be eliminated. Thus, in a problem, *knowns* are set below *unknowns* in the order.

If the variables commute, then the Gröbner basis is always finite and can be generated by Buchberger's algorithm [CLO92]. The Buchberger algorithm always terminates in a finite amount of time. It could terminate in seconds, days, or centuries. In the non commutative case, which is used in this paper, the Gröbner basis is usually infinite and then `NCGB` fails to halt given finite computational resources. Nevertheless, the solution set of the output of a terminated (say k iteration) `NCGB`, $\{q_j = 0\}$, is always equivalent to the solution set of the input, $\{p_i = 0\}$, and this *partial* basis often proves to be useful in computations. Gröbner basis computer runs can be (notoriously) memory and time consuming. Thus their effectiveness on any class of problems can only be determined by experiment.

`NCGB` is the tool behind the *user-friendly* command `NCEliminate`, used extensively in this paper. `NCEliminate` simplifies the interface with `NCGB` by generating automatic ordering of knowns, translating directives to assert assumptions on variables, such as `NCInvertible`, and processing the output in the form of rules that can be directly used in `Mathematica`. For an example, see Figure 10.

[6] k_2 may be larger than n (i.e., there need not be $\leq n$ equations in the list) and there need not be any equation in just 1 or 2 variables.

6.2 Changes of variables and motivated unknowns

It is often the case that some variables in the formulation of a problem are not the natural "coordinates" for solution of the problem. Gröbner Basis Algorithms, which lie at the core of our method, are very good at eliminating unknowns, but have no way of finding good changes of variables. The paper [HS99] gives a way of producing changes of variables (CoV) for computer assisted method. The generic idea is briefly discussed below.

6.2.1 Decompose

Suppose that it can be shown algebraically that an expression, such as $x_1x_2 + x_3$, solved a Riccati equation, e.g.,

$$(x_1x_2 + x_3)a_1a_2(x_1x_2 + x_3) + a_3(x_1x_2 + x_3) + a_5a_6 = 0 \tag{32}$$

The left hand side of (32) depends on three unknowns x_1, x_2 and x_3. It is natural, however, to view (32) as an equation in one new unknown $y = x_1x_2 + x_3$ and to rewrite the left hand side of (32) as the composition

$$k(a_1, \dots, a_6, q(a_1, \dots, a_6, x_1, x_2, x_3)) \tag{33}$$

where $y = q(a_1, \dots, a_6, x_1, x_2, x_3) := x_1x_2 + x_3$ and

$$k(a_1, \dots, a_6, y) = ya_1a_2y + a_3y + a_5a_6.$$

The challenge to computer algebra is to start with an expanded version of (32), which is a mess that you would likely see in a computer algebra session, and to automatically motivate the selection of y. Now we describe such a conceptual procedure for producing what we call a *motivated unknown*, y.

Suppose that a polynomial $p(a_1, \dots, a_r, x_1, \dots, x_s)$ appears in a computer session and has the property that there are other monomials $L(a_1, \dots, a_r, x_1, \dots, x_s)$ and $R(a_1, \dots, a_r, x_1, \dots, x_s)$ for which LpR has a decomposition

$$LpR = k(a_1, \dots, a_r, q(a_1, \dots, a_r, x_1, \dots, x_s))$$

where k is a polynomial in one unknown.

Definition 1 *A polynomial* p motivates an unknown y via the equation $y = q(a_1, \ldots, a_r, x_1, \ldots, x_s)$ *if there exist monomials* $L(a_1, \ldots, a_r, x_1, \ldots, x_s)$ *and* $R(a_1, \ldots, a_r, x_1, \ldots, x_s)$ *and there exists a polynomial in one unknown* $k(a_1, \ldots, a_r, y)$ *such that* $LpR = k(a_1, \ldots, a_r, q(a_1, \ldots, a_r, x_1, \ldots, x_s))$.

Of course, from the perspective of finding zeros of collections of polynomials, if p has a zero, then LpR has a zero and so k has a zero. Since k is a polynomial in only one unknown variable, finding the zeros of k is bound to be easier than finding the zeros of p. Weaker than having an unknown which is motivated by a decomposition (33) is one with two

$$k(a, q_1(a, x), q_2(a, x)),$$

or more motivated unknowns.

6.2.2 Implementation

The authors do not know how to fully implement the decompose operation. Finding decompositions by hand can be facilitated with the use of a certain type of *collect* command. In this paper we have used the `NCAlgebra` commands `NCCollectOnVariables`, `NCCollect` and `NCCollectSymmetric` to perform this task. The collect commands both assist the user in performing decompositions of a particular polynomial and help in finding other polynomials in the ideal which would produce motivated unknowns.

In particular, the `NCCollectOnVariables` command is used to collect *knowns* and *products of knowns* out of expressions. For example, suppose that A and B are knowns and X, Y and Z are unknowns. The collected form of

$$XABX + XABY + YABX + YABY + AX + AY \tag{34}$$

is the simpler expression

$$(X + Y)AB(X + Y) + A(X + Y). \tag{35}$$

Clearly this suggests a decomposition of (34) and, indeed, the command `NCCollectOnVariables` helps to find decompositions of much more complicated polynomials, as illustrated in this paper. For an example, see Figure 19.

6.3 Factorization of noncommutative expressions as matrix affine functions

In this paper we have used the command `MatrixAffineFactor` to produce factored forms of expressions which are affine on a given list of variables. The unique feature of this algorithm is the ability to factor noncommutative products of distinct parameters and variables into products of matrices which are affine on matrices built from the given list of variables. A problem that has to be overcome in the symbolic implementation of this algorithm is the fact that the factored form is, in general, not unique. For instance, one expression that can be symbolically factored as the product of three matrices ABC also admits factorizations in the form $(AP_1)(P_1BP_2)(P_2C)$, where P_1 and P_2 are arbitrary permutation matrices. Our algorithm, which worked very effectively in this paper, symbolically "normalizes" products of matrices in order to produce factorizations which are almost always unique. This is enforced by ensuring that all matrices in a product of matrices will have as a first non null entry a positive number or a symbolic expression where the smallest symbol (in alphabetical order) is always multiplied by a positive number. For an example, see Figure 9.

7 Acknoledgements

We would like to thank Robert Skelton and Juan Camino for valuable discussions and comments.

References

[BEFB94] Stephen P. Boyd, Laurent El Ghaoui, Eric Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory.* SIAM, Philadelphia, PA, 1994.

[CHSY03] J. F. Camino, J. W. Helton, R. E. Skelton, and Jieping Ye. Matrix inequalities: a symbolic procedure to determine convexity automatically. *Integral Equation and Operator Theory*, 46(4), 2003.

[CLO92] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, NY, 1992.

[dH03a] Maurício Carvalho de Oliveira and John William Helton. Computer algebra tailored to matrix inequalities in control. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 4973–4978, Maui, Hawaii, 2003.

[dH03b] Maurício Carvalho de Oliveira and John William Helton. Doing systems and control with NCAlgebra, a symbolic noncommutative algebra toolbox. In *Proceedings of the*

11th Mediterranean Conference on Control and Automation, Rhodes, Greece, June 2003. Special session on 'Computational Toolboxes in Control Design'.

[Fra87] Bruce A. Francis. *A Course in H_∞ Control Theory.* Springer-Verlag, New York, NY, 1987.

[GHK97] E. L. Green, L. S. Heath, and B.J. Keller. OPAL: A system for computing noncommutative Gröbner bases. In H. Comon, editor, *Eighth International Conference on Rewriting Techniques and Applications (RTA-97)*, volume 1232 of *Lecture Notes in Computer Science*, pages 331–334. Springer-Verlag, New York, NY, 1997.

[HS99] John William Helton and Mark Stankus. Computer assistance for 'discovering' formulas in system engineering and operator theory. *Journal of Functional Analysis*, 161(2):289–363, 1999.

[HSM00] J. W. Helton, Mark Stankus, and Robert L. Miller. *NCAlgebra.* Mathematics Department, University of California, San Diego, CA, 2000. `http://www.math.ucsd.edu/∼ncalg`.

[Mor86] F. Mora. Gröebner bases for non-commutative polynomial rings. volume 229 of *Lecture Notes in Computer Science*, pages 353–362. Springer-Verlag, New York, NY, 1986.

[SGC97] Carsten W. Scherer, Pascal Gahinet, and Mahmoud Chilali. Multiobjective output-feedback control via LMI optimization. *IEEE Transactions on Automatic Control*, 42(7):896–911, 1997.

[SIG98] Robert E. Skelton, T. Iwasaki, and K. Grigoriadis. *A Unified Algebraic Approach to Control Design.* Taylor & Francis, London, UK, 1998.

[ZDG96] K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control.* Prentice Hall, Inc, Englewood Cliffs, NJ, 1996.

NOT FOR PUBLICATION

Contents