Doing Systems and Control with NCAlgebra, a Symbolic Noncommutative Algebra Toolbox

M. C. de Oliveira and J. W. Helton

Abstract— **This paper will show how a symbolic noncommutative algebra package can be of help in the discipline of systems and control. A major advance in linear systems theory over the last decade has been a formalism for converting systems problems to matrix inequalities. Here we describe a notebook which allows users to convert many systems problems to LMI's (Linear Matrix Inequalities).**

I. Prologue

A major symbolic noncommutative algebra package, `NCAlgebra`, runs under Mathematica. One can obtain the software from:

```
www.math.ucsd/~ncalg
```

The material in this paper runs under `NCAlgebra` and `Mathematica`, being platform independent. We assume that the reader is familiar with `Mathematica`.

All text that discuss details of the implementation in `NCAlgebra` and `Mathematica` will be displayed in a section formated like this one. Output from `Mathematica` front end will be displayed inside boxed figures.

II. NCAlgebra

If one reads a typical article on systems and control, specially the ones based on state space A, B, C, D system representation, one finds that most of the algebra involved is noncommutative rather than commutative. Thus, for symbolic computing to have full impact on linear systems research, one needs a program which will do noncommuting operations. `Mathematica`, Macsyma, and Maple (the 3 M's) do not. For example, in standard `Mathematica`, the most basic command

```
Expand[A*(B+C)]
```

gives '`A*B+A*C`' if A, B, C commute but

```
Expand[A**(B+C)]
```

does *not* give '`A**B+A**C`'. Here, '`**`' denotes noncommutative multiply.

We developed a noncommutative algebra package, `NCAlgebra`, which runs under `Mathematica` and does basic operations, block matrix manipulations, and other things. The package might be seen as a competitor to a yellow pad. Like `Mathematica`, the emphasis is on interaction with the program and flexibility.

School of Electrical and Computer Engineering, University of Campinas, Av. Albert Einstein 400, Campinas, SP, Brazil, 13083-970. `mauricio@dt.fee.unicamp.br`.

Department of Mathematics, University of California, San Diego, 9500 Gilman Drive, Mailcode 0112, La Jolla, CA 92093-0112, USA. `helton@ucsd.edu`.

A major goal is to *help a systems and controls engineer discover key formulas* underlying his problem.

An amusing note is that a favorite command in `NCAlgebra` is

```
CommuteEverything[expr]
```

which does just what you think. It converts noncommuting expressions to commuting. Then one manipulates the commuting expression with `Mathematica`'s full arsenal and, if successful, one tries whatever ideas one gets back at the fully noncommutative level.

III. Matrix Inequalities in Control

The main advance in linear systems theory over the last decade has been a formalism for converting systems problems to matrix inequalities. Even a complicated systems problem quickly converts to a list $\mathcal{M}$ of matrix inequalities and the difficulty comes in writing down a list $\mathcal{L}$ of nice (for example, convex or linear) matrix inequalities whose solution set is the same as that of $\mathcal{M}$. Fortunately it is possible to reduce the classic control problems plus others to *Linear Matrix Inequalities*. Thus a modern challenge is to develop methods which say when this possible, do it when possible, or at least help with doing it.

IV. Linear Matrix Inequalities

The main objective in this paper is to illustrate how `NCAlgebra` can be of help with LMI production in a typical control design problem: we show how to find formulas for the design of an output feedback controller such that the closed loop system satisfies an H_∞ performance constraint. It is based on our algorithms for implementing the unified approach of [1].

Other control packages and notebooks of interest to symbolic manipulation of matrix inequalities (noncommutative inequalities) for `NCAlgebra` are listed below.

A. LMI's via Change of Variables

A notebook (that also requires `NCGB`) which produces LMI's by symbolically implementing the method of [2].

`NCGB` is an implementation of a noncommutative Gröebner basis algorithm which links Mathematica and NCAlgebra to C++ and has some platform dependence: we support Solaris, Windows and various versions of Linux.

B. Convexity Checker

One expects that convexity is a substantially more lenient condition than linearity and "convex matrix inequalities" (CMI's) should behave well numerically. Thus it would be a big advance to replace LMI's with CMI's. As a

step in this direction we have an (algebraic) algorithm for determining the region on which a formal matrix inequality is convex.

This is implemented in our command

```
NCConvexityRegion[fun, list-of-variables]
```

where `fun` is a rational function in the noncommutative variables given in `list-of-variables`. The output is a list of formal inequalities with the property that if a set $\mathcal{R}$ of matrices satisfy them, then the function `fun` is "matrix convex" with respect to the variables in `list-of-variables` on $\mathcal{R}$.

C. Mathematica's Control Toolbox

`Mathematica` has a control toolbox which operates at the commuting level. Thus it will not deal with block systems. We have a small package under `NCAlgebra` which gives `Mathematica`'s package a reasonable amount of noncommuting capability. This requires no change to `Mathematica`'s Control Toolbox; just load our package in. Then one can manipulate the A, B, C, D of systems theory much as a human would do.

D. Singular Perturbations

`NCAlgebra` and `NCGB` seem very effective at facilitating some of the grubby and very tedious calculations commonly found in control of singularly perturbed linear systems. A notebook which does such calculations done with Del Kronewitter does this.

V. Automated LMI Production

We now show how `NCAlgebra` can be of help with LMI production in a typical control design problem: we show how to find formulas for the design of an output feedback controller such that the closed loop system satisfies an H_∞ performance constraint.

The first step required to work with `NCAlgebra` is to load packages. We set some options that facilitate the manipulation of systems and control problems. This is done in `Mathematica` as in Figure 1.

In the first line we load `NCAlgebra`.

The second instruction sets a parameter that makes the output look prettier in the `Mathematica` front end.

The third instruction alters the internal behavior of `NCAlgebra` so as to make the noncommutative multiplication operator thread over matrices, which is an essential feature in systems and control applications.

The fourth line makes the operator `inv` not distributive. This is important so as to prevent `inv[a**b]` from evaluating into `inv[b]**inv[a]`, which would be incorrect, for instance, for a valid product of nonsquare `a` and `b` matrices.

In the fifth line we load `MatFlatten.m`, a package that introduces the command `MatFlatten`. This commands concatenates matrices from lists of matrices, a feature that `Mathematica` does not provide. For instance,

```
MatFlatten[{{a,b},{c,d}}]
```

Load NCAlgebra

```
In[89]:= Get["NCAlgebra.m"];
         SetOutput[all → True];
         NCGuts[NCSetNC → True, NCStrongProduct2 → True];
         ExpandQ[inv] := False;
         << MatFlatten.m;
```

Fig. 1. Loading NCAlgebra

Open loop (A,B,C,D) matrices

```
In[94]:= oLoopMats =
           {{A, Bw, Bu}, {Cz, Dzw, Dzu}, {Cy, Dyw, Dyu}};
         MatrixForm[%]

Out[95]//MatrixForm=
         A   Bw   Bu
         Cz  Dzw  Dzu
         Cy  Dyw  Dyu
```

Closed loop (A,B,C,D) matrices

```
In[96]:= cA = {{A, Bw}, {Cz, Dzw}};
         cB = {{Bu}, {Dzu}};
         cC = {{Cy, Dyw}};
         cLoopMats = cA + NCExpand[cB ** K ** cC];
         MatrixForm[%]

Out[100]//MatrixForm=
         A + Bu.K.Cy     Bw + Bu.K.Dyw
         Cz + Dzu.K.Cy   Dzw + Dzu.K.Dyw
```

Fig. 2. Open and closed loop matrices

will produce a matrix if `a`, `b`, `c` and `d` are matrices of compatible size.

A. Problem Statement

Given linear models for the system to be controlled and the controller

$$\begin{pmatrix}\dot{x}\\ z\\ y\end{pmatrix} = \begin{bmatrix}A & B_w & B_u\\ C_z & D_{zw} & D_{zu}\\ C_y & D_{yw} & 0\end{bmatrix}\begin{pmatrix}x\\ w\\ u\end{pmatrix}, \quad \begin{pmatrix}u\\ \dot{x}_c\end{pmatrix} = K\begin{pmatrix}y\\ x_c\end{pmatrix}$$

the objective is to compute a matrix of controller parameters K such that $\|H_{wz}(s)\|_\infty < \mu$, where $\mu > 0$ is a given performance level.

Defining the matrices

$$\begin{bmatrix}\mathbb{A} & \mathbb{B}\\ \mathbb{C} & 0\end{bmatrix} := \left[\begin{array}{cc|c} \mathbf{A} & \mathbf{B}_w & \mathbf{B}_u\\ \mathbf{C}_z & \mathbf{D}_{zw} & \mathbf{D}_{zu}\\ \hline \mathbf{C}_y & \mathbf{D}_{yw} & 0\end{array}\right] := \left[\begin{array}{cc|c|cc} A & 0 & B_w & B_u & 0\\ 0 & 0 & 0 & 0 & I\\ \hline C_z & 0 & D_{zw} & D_{zu} & 0\\ \hline C_y & 0 & D_{yw} & 0 & 0\\ 0 & I & 0 & 0 & 0\end{array}\right],$$

which have been obtained from the system to be controlled data, it is possible to describe the closed loop connection of the plant and controller in the form

$$\begin{pmatrix}\dot{\tilde{x}}\\ z\end{pmatrix} = (\mathbb{A} + \mathbb{B}K\mathbb{C})\begin{pmatrix}\tilde{x}\\ w\end{pmatrix},$$

where $\tilde{x} := \begin{pmatrix}x^T & x_c^T\end{pmatrix}^T$.

In Figure 2 we define a matrix that represent the open loop system and compute the closed loop matrices as a function of a generic feedback matrix K.

Augmented open loop matrices

```
In[101]:= bA = {{A, 0}, {0, 0}};
          bBw = {{Bw}, {0}};
          bBu = {{Bu, 0}, {0, 1}};
          bCz = {{Cz, 0}};
          bDzw = {{Dzw}};
          bDzu = {{Dzu, 0}};
          bCy = {{Cy, 0}, {0, 1}};
          bDyw = {{Dyw}, {0}};
          bDyu = {{0, 0}, {0, 0}};

In[110]:= rulesAugmentedSystem =
            { A → bA, Bw → bBw, Bu → bBu,
              Cz → bCz, Dzw → bDzw, Dzu → bDzu,
              Cy → bCy, Dyw → bDyw, Dyu → bDyu };

In[111]:= MatFlatten[oLoopMats /. rulesAugmentedSystem];
          MatrixForm[%]

Out[112]//MatrixForm=
          ⎛ A   0   Bw   Bu   0 ⎞
          ⎜ 0   0   0    0    1 ⎟
          ⎜ Cz  0   Dzw  Dzu  0 ⎟
          ⎜ Cy  0   Dyw  0    0 ⎟
          ⎝ 0   1   0    0    0 ⎠

In[113]:= rulesController = {K → {{Dc, Cc}, {Bc, Ac}}};
          MatrixForm[K /. rulesController]

Out[114]//MatrixForm=
          ⎛ Dc  Cc ⎞
          ⎝ Bc  Ac ⎠
```

Fig. 3. Open and closed loop augmented matrices

The data corresponding to the augmented plant representation and the controller matrices is entered in NCAlgebra as in Figure 3.

B. Open Loop Analysis

Given a linear system in the form

$$\begin{pmatrix} \dot{x} \\ z \end{pmatrix} = \begin{bmatrix} A & B_w \\ C_z & D_{zw} \end{bmatrix} \begin{pmatrix} x \\ w \end{pmatrix},$$

the H_∞ norm of the transfer function from w to z is less than μ if, and only if, the following LMI, known as BRL (*Bounded Real Lemma*),

$$\Psi + \Delta^T \Sigma \Lambda + \Lambda^T \Sigma^T \Delta < 0, \tag{1}$$

where

$$\Psi := \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\mu & 0 \\ 0 & 0 & -\mu \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} A & B_w \\ C_z & D_{zw} \end{bmatrix},$$
$$\Lambda := \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \end{bmatrix}, \qquad \Delta := \begin{bmatrix} X & 0 & 0 \\ 0 & 0 & I \end{bmatrix},$$

is feasible for some symmetric and positive definite Lyapunov matrix X.

The matrices appearing in the BRL are defined in Figure 4. In the same figure, the left hand side of this inequality is evaluated for the open loop matrices.

C. Closed Loop Analysis

In order to analyze the closed loop H_∞ performance we redefine the factors appearing in the BRL to be compatible with the augmented system matrices and substitute

Open loop analysis

```
In[117]:= SetCommutative[mu];

In[118]:= oLoopPsi = {{0, 0, 0}, {0, -mu, 0}, {0, 0, -mu}};
          MatrixForm[%]

Out[119]//MatrixForm=
          ⎛ 0   0    0  ⎞
          ⎜ 0  -mu   0  ⎟
          ⎝ 0   0   -mu ⎠

In[120]:= oLoopDelta = {{X, 0, 0}, {0, 0, 1}};
          MatrixForm[%]

Out[121]//MatrixForm=
          ⎛ X  0  0 ⎞
          ⎝ 0  0  1 ⎠

In[122]:= oLoopLambda = {{1, 0, 0}, {0, 1, 0}};
          MatrixForm[%]

Out[123]//MatrixForm=
          ⎛ 1  0  0 ⎞
          ⎝ 0  1  0 ⎠

In[124]:= oLoopSigma = {{A, Bw}, {Cz, Dzw}};
          MatrixForm[%]

Out[125]//MatrixForm=
          ⎛ A   Bw  ⎞
          ⎝ Cz  Dzw ⎠

In[126]:= oLoopSymmetryRules = {tp[X] → X};

        oLoopBRL = oLoopPsi + NCExpand[
             tp[oLoopDelta] ** oLoopSigma ** oLoopLambda +
              tp[oLoopLambda] ** tp[oLoopSigma] **
               oLoopDelta] /. oLoopSymmetryRules;
        MatrixForm[%]

Out[128]//MatrixForm=
          ⎛ X.A + Aᵀ.X   X.Bw    Czᵀ   ⎞
          ⎜   Bwᵀ.X      -mu     Dzwᵀ  ⎟
          ⎝    Cz         Dzw    -mu   ⎠
```

Fig. 4. Open loop analysis

the value of Σ by the closed loop matrices obtained in the previous section

$$\Sigma = (\mathbb{A} + \mathbb{B} K \mathbb{C}).$$

If one performs such operations, the BRL can be rewritten, as a function of the controller parameters K in the form

$$F + H^T K G + G^T K^T H < 0, \tag{2}$$

where

$$F := \Psi + \Delta^T \mathbb{A} \Lambda + \Lambda^T \mathbb{A}^T \Delta, \quad G := \mathbb{C} \Lambda, \quad H := \mathbb{B}^T \Delta.$$

We redefine, in Figure 5, the factors of the BRL so they can have dimensions compatible with the augmented system matrices. We also define the expression for the closed loop Σ matrix.

D. LMI Synthesis

The matrix inequality (2) is not jointly convex on X, the Lyapunov matrix, and K, the controller parameters, which represents the main obstacle for its solution. Two techniques can be used to overcome this difficulty: the 'elimination of variables' of [1] and the 'change of variables' of [2]. These techniques produce sets of convex inequalities that are affine on all variables, that is, LMI's.

Closed loop analysis

```
In[129]:= cLoopPsi = {{0, 0, 0, 0},
             {0, 0, 0, 0},
             {0, 0, -mu, 0},
             {0, 0, 0, -mu}};
           MatrixForm[%]

Out[130]//MatrixForm=
          ( 0  0   0    0  )
          ( 0  0   0    0  )
          ( 0  0  -mu   0  )
          ( 0  0   0   -mu )

In[131]:= cLoopDelta = {{X1, X2, 0, 0},
             {tp[X2], X3, 0, 0},
             {0, 0, 0, 1}};
           MatrixForm[%]

Out[132]//MatrixForm=
          (  X1   X2  0  0 )
          ( X2^T  X3  0  0 )
          (  0    0   0  1 )

In[133]:= cLoopLambda = {{1, 0, 0, 0},
             {0, 1, 0, 0},
             {0, 0, 1, 0}};
           MatrixForm[%]

Out[134]//MatrixForm=
          ( 1  0  0  0 )
          ( 0  1  0  0 )
          ( 0  0  1  0 )

In[135]:= cLoopA = MatFlatten[cA /. rulesAugmentedSystem];
          cLoopB = MatFlatten[cB /. rulesAugmentedSystem];
          cLoopC = MatFlatten[cC /. rulesAugmentedSystem];
          cLoopSigma = cLoopA + NCExpand[
              cLoopB ** K ** cLoopC /. rulesController];
          MatrixForm[%]

Out[139]//MatrixForm=
          (      A + Bu.Dc.Cy      Bu.Cc      Bw + Bu.Dc.Dyw )
          (         Bc.Cy            Ac           Bc.Dyw     )
          ( Cz + Dzu.Dc.Cy     Dzu.Cc     Dzw + Dzu.Dc.Dyw )

        cLoopSymmetryRules =
          {tp[X1] → X1, tp[X3] → X3, tp[Y1] → Y1, tp[Y3] → Y3};

        cLoopBRL = cLoopPsi + NCExpand[
             tp[cLoopDelta] ** cLoopSigma ** cLoopLambda +
              tp[cLoopLambda] ** tp[cLoopSigma] **
               cLoopDelta] /. cLoopSymmetryRules;
```

Fig. 5. Closed loop analysis

Null space of G

```
In[138]:= G = cLoopC ** cLoopLambda;
          MatrixForm[%]

Out[139]//MatrixForm=
          ( Cy  0  Dyw  0 )
          (  0  1   0   0 )

In[140]:= NG = {
             {NG11, NG12, NG13},
             {NG21, NG22, NG23},
             {NG31, NG32, NG33},
             {NG41, NG42, NG43}};
           MatrixForm[%]

Out[141]//MatrixForm=
          ( NG11  NG12  NG13 )
          ( NG21  NG22  NG23 )
          ( NG31  NG32  NG33 )
          ( NG41  NG42  NG43 )

In[142]:= ruleNullG = Flatten[
             Solve[NCExpand[G ** NG] == ZeroMatrix[2, 3]]];
           MatrixForm[%]

Out[143]//MatrixForm=
          (          NG21 → 0          )
          (          NG22 → 0          )
          (          NG23 → 0          )
          ( Cy.NG11 → -Dyw.NG31        )
          ( Cy.NG12 → -Dyw.NG32        )
          ( Cy.NG13 → -Dyw.NG33        )
```

Particular choice of basis

```
        NG = NG //. Flatten[{
             ruleNullG,
             NG11 → 1, NG12 → 0, NG13 → 0,
             NG31 → -tp[Dyw] ** inv[Dyw ** tp[Dyw]] **
                Cy ** NG11, NG32 → 0, NG33 → p[Dyw],
             NG41 → 0, NG42 → 1, NG43 → 0}];
        MatrixForm[%]

Out[145]//MatrixForm=
          (                  1                0    0 )
          (                  0                0    0 )
          ( -Dyw^T.(Dyw.Dyw^T)^-1.Cy  0  p[Dyw] )
          (                  0                1    0 )
```

Fig. 6. Computing the null space of G

In this paper we focus on the procedure of [1], which is based on the following result, known as *Elimination Lemma*.

Lemma 1: Let $F \in \mathbb{S}^n$, $G \in \mathbb{R}^{m\times n}$, and $H \in \mathbb{R}^{p\times n}$ with $\text{rank}(G) < n$ and $\text{rank}(H) < n$. The following statements are equivalent:

i) $G^{\perp^T} F G^{\perp} < 0, \quad H^{\perp^T} F H^{\perp} < 0,$

ii) $\exists\, K \in \mathbb{R}^{p\times m}: \quad F + H^T K G + G^T K^T H < 0.$

In the above lemma, the symbol $X^{\perp}$ represents any basis for the null space of matrix X.

The closed loop BRL, inequality (2), is clearly in the form *ii)* of the above lemma. The strategy to obtain an LMI synthesis problem consists on moving from form *ii)* to *i)*, hence eliminating the controller parameters from the problem. The main computation involved is the calculation of the null space basis of the matrices G and H.

In order to make the synthesis formulas look simpler and more familiar we adopt the following set of standard simplifying assumptions.

$$C_z^T D_{zu} = 0, \qquad D_{zu}^T D_{zu} = I, \tag{3}$$

$$B_w D_{yw}^T = 0, \qquad D_{yw} D_{yw}^T = I. \tag{4}$$

These assumptions can be imposed with no loss of generality (see, for instance, [3]).

Computing a null space basis for G involves only constant matrices, hence it can sometimes be done by using the `Mathematica` linear equation solver combined with some level of user intervention. The user helps the program to pick up a particularly convenient choice of basis, based on the extra assumptions (3). This is done in Figure 6.

The functional notation `p[X]` has been used to denote $X^{\perp}$.

Matrix H, on the contrary of matrix G, is also a function of the Lyapunov matrix X. However, noticing that

$$\Delta = \tilde{\Delta} T,$$

where

$$\tilde{\Delta} := \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & I \end{bmatrix}, \qquad T := \begin{bmatrix} X & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix},$$

and that X, and consequently T are nonsingular matrices, a null space basis for H can be trivially computed from any null space basis of

$$\tilde{H} := \mathbb{B}^T \tilde{\Delta},$$

Null space of HTilde

```
In[146]:= cLoopDeltaTilde = {{1, 0, 0, 0},
            {0, 1, 0, 0},
            {0, 0, 0, 1}};
          MatrixForm[%]

Out[147]//MatrixForm=
          ⎛1  0  0  0⎞
          ⎜0  1  0  0⎟
          ⎝0  0  0  1⎠

In[148]:= HTilde = tp[cLoopB] ** cLoopDeltaTilde;
          MatrixForm[%]

Out[149]//MatrixForm=
          ⎛Bu^T  0  0  Dzu^T⎞
          ⎝  0   1  0    0  ⎠

In[150]:= NHTilde = {
            {NH11, NH12, NH13},
            {NH21, NH22, NH23},
            {NH31, NH32, NH33},
            {NH41, NH42, NH43}};
          MatrixForm[%]

Out[151]//MatrixForm=
          ⎛NH11  NH12  NH13⎞
          ⎜NH21  NH22  NH23⎟
          ⎜NH31  NH32  NH33⎟
          ⎝NH41  NH42  NH43⎠

In[152]:= ruleNullHTilde = Flatten[
            Solve[NCExpand[HTilde ** NHTilde]
              == ZeroMatrix[2, 3]]];
          MatrixForm[%]

Out[153]//MatrixForm=
          ⎛          NH21 → 0          ⎞
          ⎜          NH22 → 0          ⎟
          ⎜          NH23 → 0          ⎟
          ⎜Bu^T.NH11 → -Dzu^T.NH41     ⎟
          ⎜Bu^T.NH12 → -Dzu^T.NH42     ⎟
          ⎝Bu^T.NH13 → -Dzu^T.NH43     ⎠
```

Particular choice of basis

```
NHTilde = NHTilde //. Flatten[{
     ruleNullHTilde,
     NH11 → 1, NH12 → 0, NH13 → 0,
     NH31 → 0, NH32 → 1, NH33 → 0,
     NH41 → -Dzu ** inv[tp[Dzu] ** Dzu] **
        tp[Bu] ** NH11, NH42 → 0,
     NH43 → p[tp[Dzu]]}];
MatrixForm[%]

Out[155]//MatrixForm=
          ⎛                1                0    0⎞
          ⎜                0                0    0⎟
          ⎜                0                1    0⎟
          ⎝-Dzu.(Dzu^T.Dzu)^-1.Bu^T  0  p[Dzu^T]⎠
```

Fig. 7. Computing the null space of $\tilde{H}$

as

$$H^{\perp} = T^{-1}\tilde{H}^{\perp}.$$

As for matrix G, we compute a null space basis for matrix $\tilde{H}$ using the assumptions (4). This is done in Figure 7.

In Figure 8, a null space basis of the matrix H is computed from $\tilde{H}^{\perp}$ as explained above.

Notice that the null space of H requires evaluating the inverse of matrix X. In order to accomplish that we define a matrix Y of compatible block structure and compute the rules derived from the fact that $XY = YX = I$. These identities will be used in the next step.

After computing $G^{\perp}$ and $H^{\perp}$ we are ready to generate the matrix inequalities in item i) of Lemma 1. These inequalities, where the controller parameters have been eliminated, will be LMI's.

Null space of H

```
In[156]:= ruleXY = {X -> {{X1, X2},
                {tp[X2], X3}}, Y -> {{Y1, Y2},
                {tp[Y2], Y3}}};
          MatrixForm[%]

Out[157]//MatrixForm=
          ⎛X → {{X1, X2}, {X2^T, X3}}⎞
          ⎝Y → {{Y1, Y2}, {Y2^T, Y3}}⎠

In[158]:= inverseRules = Flatten[{
              Thread[Flatten[NCExpand[X ** Y /. ruleXY]] →
                Flatten[IdentityMatrix[2]]],
              Thread[Flatten[NCExpand[Y ** X /. ruleXY]] →
                Flatten[IdentityMatrix[2]]]}];
          MatrixForm[%]

Out[159]//MatrixForm=
          ⎛X1.Y1 + X2.Y2^T → 1⎞
          ⎜X1.Y2 + X2.Y3 → 0  ⎟
          ⎜X3.Y2^T + X2^T.Y1 → 0⎟
          ⎜X3.Y3 + X2^T.Y2 → 1⎟
          ⎜Y1.X1 + Y2.X2^T → 1⎟
          ⎜Y1.X2 + Y2.X3 → 0  ⎟
          ⎜Y3.X2^T + Y2^T.X1 → 0⎟
          ⎝Y3.X3 + Y2^T.X2 → 1⎠

In[160]:= Tinv = {{Y1, Y2, 0, 0},
            {tp[Y2], Y3, 0, 0},
            {0, 0, 1, 0},
            {0, 0, 0, 1}};
          MatrixForm[%]

Out[161]//MatrixForm=
          ⎛Y1    Y2  0  0⎞
          ⎜Y2^T  Y3  0  0⎟
          ⎜ 0    0   1  0⎟
          ⎝ 0    0   0  1⎠

In[162]:= NH = NCExpand[Tinv ** NHTilde];
          MatrixForm[%]

Out[163]//MatrixForm=
          ⎛              Y1                0    0⎞
          ⎜              Y2^T              0    0⎟
          ⎜               0                1    0⎟
          ⎝-Dzu.(Dzu^T.Dzu)^-1.Bu^T  0  p[Dzu^T]⎠
```

Fig. 8. Computing the null space of H

In the final step, described in Figure 9, we evaluate the matrix F by setting the controller parameters to zero in the closed loop BRL. The two LMI are then obtained by post and pre multiplying F by $G^{\perp}$ and $H^{\perp}$ and their transposes.

Notice that the final expression has been automatically simplified using the rules derived from assumptions (3) and (4), the rules of inverse derived in Figure 8, and the rules derived from null space definition.

It is worth remarking on the amazing amount of simplification performed automatically by `NCAlgebra` at this last step. An impressive demonstration of the power of this tools is obtained after comparing the final form of the first LMI with the expression obtained right after multiplying ${G^{\perp}}^T F G^{\perp}$ without applying any rules, which is shown, unformatted, in Figure 10.

E. Solving the Problem and Constructing the Controller

Before solving the problem we recall that $X > 0$. Since the two LMI's obtained in Figure 9 involve only X_1 and Y_1, the first blocks of, respectively, X and its inverse Y, positivity of X can be enforced whenever $X_1 > Y_1^{-1}$ (see [1]

Computing the LMI

```
In[168]:= F = cLoopBRL /. {Ac → 0, Bc → 0, Cc → 0, Dc -> 0};
          MatrixForm[%]

Out[169]//MatrixForm=
          ⎛ X1.A + A^T.X1   A^T.X2   X1.Bw   Cz^T  ⎞
          ⎜   X2^T.A          0     X2^T.Bw   0    ⎟
          ⎜  Bw^T.X1        Bw^T.X2   -mu    Dzw^T ⎟
          ⎝    Cz             0       Dzw    -mu   ⎠

In[170]:= simplifyingAssumptions = {
                l___ ** tp[Dzu] ** Dzu ** r___ → l ** r,
                l___ ** tp[Cz] ** Dzu ** r___ → 0,
                l___ ** tp[Dzu] ** Cz ** r___ → 0,
                l___ ** Dyw ** tp[Dyw] ** r___ → l ** r,
                l___ ** Dyw ** tp[Bw] ** r___ → 0,
                l___ ** Bw ** tp[Dyw] ** r___ → 0
            };

In[171]:= perpRules = {
              l___ ** Dyw ** p[Dyw] ** r___ → 0,
              l___ ** tp[Dzu] ** p[tp[Dzu]] ** r___ → 0
            };
          perpRules = Flatten[{perpRules, Flatten[Map[tp, perpRules, {2}]]}];

In[173]:= NCExpand[tp[NG] ** F ** NG] //.
            Flatten[{simplifyingAssumptions, cLoopSymmetryRules, perpRules}];
          NCCollect[%, {p[Dyw], tp[p[Dyw]]}];
          MatrixForm[%]

Out[175]//MatrixForm=
          ⎛ X1.A + A^T.X1 - mu Cy^T.Cy      -Cy^T.Dyw.Dzw^T + Cz^T      X1.Bw.p[Dyw]      ⎞
          ⎜    Cz - Dzw.Dyw^T.Cy                    -mu                  Dzw.p[Dyw]        ⎟
          ⎝ p[Dyw]^T.(Bw^T.X1 + mu Dyw^T.Cy)    p[Dyw]^T.Dzw^T      -mu p[Dyw]^T.p[Dyw]   ⎠

In[176]:= NCExpand[tp[NH] ** F ** NH] //.
            Flatten[{simplifyingAssumptions, cLoopSymmetryRules, perpRules}];
          NCCollect[%, {p[tp[Dzu]], tp[p[tp[Dzu]]], A ** Y1, Y1 ** tp[A], Bw, tp[Bw]}];
          % //. inverseRules;
          MatrixForm[%]

Out[179]//MatrixForm=
          ⎛ A.Y1 - mu Bu.Bu^T + Y1.A^T      Bw - Bu.Dzu^T.Dzw      Y1.Cz^T.p[Dzu^T]      ⎞
          ⎜   -Dzw^T.Dzu.Bu^T + Bw^T                -mu              Dzw^T.p[Dzu^T]       ⎟
          ⎝ p[Dzu^T]^T.(Cz.Y1 + mu Dzu.Bu^T)   p[Dzu^T]^T.Dzw   -mu p[Dzu^T]^T.p[Dzu^T]   ⎠
```

Fig. 9. Computing the LMI

```
In[185]:= NCExpand[tp[NH] ** F ** NH]

Out[185]= {{Y2.X2^T.A.Y1 + Y1^T.A^T.X1.Y1 + Y1^T.A^T.X2.Y2^T + Y1^T.X1^T.A.Y1 - Bu.(Dzu^T.Dzu)^-1.Dzu^T.Cz.Y1 -
             Y1^T.Cz^T.Dzu.(Dzu^T.Dzu)^-1.Bu^T - mu Bu.(Dzu^T.Dzu)^-1.Dzu^T.Dzu.(Dzu^T.Dzu)^-1.Bu^T,
            Y2.X2^T.Bw + Y1^T.X1^T.Bw - Bu.(Dzu^T.Dzu)^-1.Dzu^T.Dzw,
            Y1^T.Cz^T.p[Dzu^T] + mu Bu.(Dzu^T.Dzu)^-1.Dzu^T.p[Dzu^T]},
           {Bw^T.X1.Y1 + Bw^T.X2.Y2^T - Dzw^T.Dzu.(Dzu^T.Dzu)^-1.Bu^T, -mu, Dzw^T.p[Dzu^T]},
           {p[Dzu^T]^T.Cz.Y1 + mu p[Dzu^T]^T.Dzu.(Dzu^T.Dzu)^-1.Bu^T,
            p[Dzu^T]^T.Dzw, -mu p[Dzu^T]^T.p[Dzu^T]}}
```

Fig. 10. LMI before simplification

for details). This inequality can be written as the LMI

$$\begin{bmatrix} X_1 & I \\ I & Y_1 \end{bmatrix} > 0$$

by applying Schur complement. This LMI and the two LMI's obtained in Figure 9 constitute the convex problem to be solved that guarantee the closed loop H_∞ performance level μ.

Once this LMI problem has been solved, the set controller parameters can be computed, for instance, using the formula

$$K = -\rho H J G^T (G J G^T)^{-1},$$

where ρ is an arbitrary scalar that makes

$$J := (\rho H^T H - F)^{-1} > 0.$$

Such a ρ is guaranteed to exist (see [1] for details).

VI. Conclusions

One can considerably facilitate the production of LMI's using the notebook described in this article. Underneath it are symbolic algorithms which do not impinge on the users experience.

References

[1] R. E. Skelton, T. Iwasaki, and K. Grigoriadis, *A Unified Algebraic Approach to Control Design*, Taylor & Francis, London, UK, 1997.

[2] C. W. Scherer, P. Gahinet, and M. Chilali, "Multiobjective output-feedback control via LMI optimization," *IEEE Transactions on Automatic Control*, vol. 42, no. 7, pp. 896–911, 1997.

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