Singularly perturbed control systems using noncommutative computer algebra

J.W. Helton* F. Dell Kronewitter* W.M. McEneaney† Mark Stankus‡

Abstract

Most algebraic calculations which one sees in linear systems theory, for example in IEEE TAC, involve block matrices and so are highly noncommutative. Thus conventional commutative computer algebra packages, as in Mathematica and Maple, do not address them.

Here we investigate the usefulness of **noncommutative** computer algebra in a particular area of control theory–singularly perturbed dynamic systems–where working with the noncommutative polynomials involved is especially tedious. Our conclusion is that they have considerable potential for helping practitioners with such computations.

Commutative Gröbner basis algorithms are powerful and make up the engines in symbolic algebra packages' `Solve` commands. Noncommutative Gröbner basis algorithms are more recent, but we shall see that they are useful in manipulating the messy sets of noncommutative polynomial equations which arise in singular perturbation calculations. We use the noncommutative algebra package NCAlgebra and the noncommutative Gröbner basis package NCGB which runs under it on two different problems. We illustrate the method on the classical state feedback optimal control problem, see [KKO86], where we obtain one more (very long) term than was done previously. Then we use it to derive singular perturbation expansions for the relatively new (linear) information state equation.

1 George Zames

All of the authors of this paper remember George Zames either through his work or through first hand experience. George had vast influence on the subject of control and of his accomplishments there are so many accounts that it is pointless to list them here. Not so obvious to an engineering audience is that his invention, H^∞ control, led to a fever of activity in an area of mathematics called operator theory. Ultimately this led to a body of beautiful mathematics.

Some personal remarks by one of the authors, Bill Helton, might be appropriate. Always when I think of George I think of how much fun he was. He had a great sense of adventure, curiosity, and even mischievousness. While most people are very cautious in what they say George would speculate about every sort of thing and ask incessant questions out of curiosity and out of an eagerness to get a freewheeling discussion going.

I might add that George in my experience was a "math buff". Besides many illustrations in private conversations of his interest in new mathematical ideas a very public example was his hosting the second meeting of Mathematical Theory of Networks and Systems in 1975. This was not the 25th meeting of the MTNS, but was the first tentative feelers (since Norbert Wiener's generation) going out between the engineering community and the operator theory community. Ultimately these interactions led to H^∞ control.

The pattern I consistently saw in George was a great sense of adventure in selecting his topic, a relentless focus on the few physical or conceptual issues he thought were central to the topic, and the relentless drive to set these into correspondence with elegant mathematical constructions. I might emphasize this last part, since in modern times young engineers think of George Zames as they would a statue on the engineering school lawn. There is a tendency to think that his achievements came from a very linear process of going from physics to a mathematical treatment. In fact, there was little that was linear about George's thinking and H^∞ control at least came from trying relentlessly to set a physical picture and a mathematical picture in correspondence.

*University of California, San Diego, Math Dept., 9500 Gilman Dr., San Diego, CA 92093-0112, helton@math.ucsd.edu, dell@ieee.org, Partially supported by the AFOSR and the NSF

†Dept. of Mathematics, North Carolina State University, Raleigh, NC 27695–8205, USA, wmm@math.ncsu.edu.

‡Department of Mathematics, California State University. San Luis Obispo,CA 93407, mstankus@calpoly.edu

2 Introduction

2.1 Noncommutative algebra on the computer

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, 5 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 preMATLAB 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.

Finally, within the last few years there are pieces of the great software mosaic becoming available. Publicly available programs which are currently maintained and which include a noncommutative Gröbner Basis finder (which in our opinion is like getting to first base) are listed at CAIN, `http://www.can.nl`. It is not clear which are being supported currently. However, we have had recent contact with the author of OPAL, [KG98], and Anick, `joeb@matematik.su.se`. In this paper we use NCGB [HS97] running under NCAlgebra [HMS96].

Our effort, which includes many features, now consists of 1.68 Megabytes of Mathematica code and 2.14 Megabytes of C++ linked to it. Over the last 5 years we have done many experiments on linear systems type of calculations. In this article we report on what appears to be a very successful application of our methods.

2.2 Singular perturbation vs computer algebra

Singular perturbation is a commonly used technique in the analysis of systems whose dynamics consist of two pieces. One piece might be slow, the other fast, or one might be known where the other is somewhat uncertain. Extensive analysis has been done of this type of plant for the LQR and H^∞ control problems, for example [KKO86], [PB93], [PB94].

Typically one has an equation where some coefficients depend on a parameter $\frac{1}{\varepsilon}$. To solve this equation, one postulates an expansion in ε for the solutions x_ε to the equation, then

(a) substitutes x_ε into the equation,

(b) sorts the equation according to powers of ε, and

(c) finds simple equations for successive terms in the expansion of x_ε.

The sorting in (b) can be tedious and the business of solving (c) can be very involved.

This article concerns methods we are developing for doing these steps automatically. The software runs under NCAlgebra, [HMS96], the most widely distributed Mathematica package for general noncommuting computations. As we shall illustrate, NCAlgebra constructions and commands easily handle steps (a) and (b), thereby producing the (long) list of equations which must be solved. This is straightforward and saves considerable calculation time for those engaged in singular perturbation calculations. Step (c) involves solving complicated systems of equations and this is always tricky business. Thus there is no way of knowing in advance if noncommutative Gröbner basis methods will be effective for (reducing to a simple form) the equations found in large classes of singular perturbation problems. This is the focus of experiments (using the package NCGB which runs under NCAlgebra) we have been conducting and on which we report here.

Most of the paper shows how one can treat the most classic of all singular perturbation problems using computer algebra. Ultimately, we see that Mora's Gröbner basis algorithms are very effective on the equations which result. Indeed our method carries out the expansion one step further than has previously been done, see Section 4.3. Then we sketch another newer H^∞ estimation problem called the "cheap sensor" problem (see [HJM98]). On this our computer techniques proved effective.

2.3 The idea behind Gröbner computer algebra

We shall describe some algorithms we use for handling polynomials many of which make use of Gröbner bases.

2.3.1 Gröbner bases

We say that a set of equations $\{q_j = 0 : 1 \leq j \leq k\}$ eliminates x_k if one of the polynomials has the form $q_j = x_k - r(x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n)$ (so that r is a polynomial which does not depend on x_k).

The noncommutative Gröbner basis algorithm (GBA), due to F. Mora [Mor86], can be used to systematically eliminate variables from a collection (e.g., $\{p_j(x_1, \ldots, x_n) = 0 : 1 \leq j \leq k_1\}$) of polynomial equations so as to put it in triangular form. One specifies an order on the variables ($x_1 < x_2 < x_3 < \ldots < x_n$) [1] which corresponds to your priorities in eliminating them. Here a GBA will try hardest to eliminate x_n and try the least to eliminate x_1. The output from it is a list of equations in a "canonical form" which is triangular:[2]

$$
\begin{aligned}
q_1(x_1) &= 0 && (1)\\
q_2(x_1, x_2) &= 0 && (2)\\
q_3(x_1, x_2) &= 0 && (3)\\
q_4(x_1, x_2, x_3) &= 0 && (4)\\
&\vdots \\
q_{k_2}(x_1, \ldots, x_n) &= 0\,. && (5)
\end{aligned}
$$

Here, the set of solutions to the collection of polynomial equations $\{q_j = 0 : 1 \leq j \leq k_2\}$ equals 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.

If the variables commute, then the Gröbner basis is always finite and can be generated by Buchberger's algorithm. The Buchberger algorithm always terminates in a finite amount of time. It could terminate in seconds, days, or centuries. In the noncommutative case, which is the subject of this paper, the Gröbner basis is usually infinite and then the GBA fails to halt given finite computational resources. Nevertheless, the solution set of the output of a terminated (say k iteration) GBA, $\{q_j = 0\}$, is always equivalent to the solution set of the input, $\{p_i = 0\}$, and this *partial* GBA often proves to be useful in computations as will be shown below. 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.

2.3.2 Removing redundant relations

There is another part of our algorithms which is so important it deserves some discussion. A Gröbner basis or even a partial Gröbner basis contains many polynomials, a few of which are important since they contain few unknowns. However, they also contain many, many polynomials which are long and uninteresting.

We have several algorithms for removing redundant relations the specifics of which are described in [HS99] and [HS97]. The one which we have found to be remarkably effective for singular perturbation computations is called *RemoveRedundant*. Briefly, it records the history of the production of the Gröbner basis as a tree and then throws away all polynomials which correspond to nodes which are not seminal nodes.

2.3.3 NCProcess

NCProcess can either be viewed as a Mathematica command or a 5 year research project. The general goal of NCProcess is to take a set of noncommutative polynomials and return an equivalent set of noncommutative

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

[2] 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.

polynomials which is as "useful" as possible. The most important tasks NCProcess performs are creating a Gröbner basis, removing redundant polynomials, and categorizing the output.

2.3.4 Hardware

Computer computations for Section 4 were performed with NCGB on a Sun Ultra I with one 166 MHz processor and 192MB of RAM. The computations done in Section 5 were performed with NCGB on a Sun Ultra II with two 166MHz processors and 1Gb of RAM. The Sun Ultra II was a departmental machine and therefore equivalent computations on a dedicated computer might take less than half of the times reported here.

3 The Standard State Feedback Singular Perturbation Problem

The standard singularly perturbed linear time-invariant model consists of a differential state equation; which depends on some perturbation parameter ϵ; and an output equation. The general control problem is to design some feedback law which specifies the input as a function of the state so that the controlled system will satisfy some given performance objective.

3.1 The system

Here we study the two time scale dynamic system previously analyzed in [KKO86]:

$$\begin{bmatrix} \frac{dx}{dt} \\ \epsilon \frac{dz}{dt} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u \tag{6}$$

$$y = \begin{bmatrix} M_1 & M_2 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \tag{7}$$

where $x \in \mathbb{R}^n$, $z \in \mathbb{R}^m$, $u \in \mathbb{R}^p$, and $y \in \mathbb{R}^q$. Here, m, n, p and q are integers, A_{11} is an $n \times n$ matrix, A_{12} is a $n \times m$ matrix, A_{21} is a $m \times n$ matrix, A_{22} is a $m \times m$ matrix, B_1 is a $n \times p$ matrix, B_2 is a $m \times p$ matrix, M_{11} is a $q \times n$ matrix, and M_{22} is a $q \times m$ matrix.

3.2 The near-optimal LQR problem

The infinite-time optimal linear regulator problem is to find a control, $u(t), t \in [0, \infty]$ which minimizes the quadratic cost

$$J = \int_0^\infty (y^T y + u^T R u)\, dt \tag{8}$$

where R is a positive definite weighting matrix. It is well known that the solution to this problem is of the form

$$u^* = -R^{-1} B^T K(\epsilon) \begin{bmatrix} x \\ z \end{bmatrix} = G(\epsilon) \begin{bmatrix} x \\ z \end{bmatrix} \tag{9}$$

where $K(\epsilon)$ is a solution to the Algebraic Riccati Equation (ARE)

$$KA + A^T K - KBR^{-1}B^T K + M^T M = 0 \tag{10}$$

with

$$A = \begin{bmatrix} A_{11} & A_{12} \\ \frac{A_{21}}{\epsilon} & \frac{A_{22}}{\epsilon} \end{bmatrix}, B = \begin{bmatrix} B_1 \\ \frac{B_2}{\epsilon} \end{bmatrix}, \text{ and } M = \begin{bmatrix} M_1 & M_2 \end{bmatrix} \tag{11}$$

$$K(\epsilon) = K_0 + \epsilon K_1 + \epsilon^2 K_2 + \ldots \tag{12}$$

If K is the solution to this optimal state feedback control problem, then it also may be used to express the optimal cost as a function of the initial state of the system as

$$J^* = \begin{bmatrix} x^T(0) & z^T(0) \end{bmatrix} K \begin{bmatrix} x(0) \\ z(0) \end{bmatrix}. \tag{13}$$

Notice that the solution J^* as presented involves solving equation (10) which is a Riccati equation of size $n+m$ by $n+m$. Since the structures present in the system (6) are partitioned, it seems likely that we might decompose the problem and significantly reduce the sizes of the matrices while deriving an only slightly sub-optimal controller. Indeed, it is standard to divide the problem of solving the $n+m$ dimensional Riccati, (10), into solving two smaller decoupled Riccatis, one which is n dimensional, the other which is m dimensional, and then use these solutions to obtain a control law which gives a performance J nearly equal to the optimal performance J^*. This regulator is known as the *near-optimal regulator.*

In the next two subsections we review this decomposition of the state space into slow and fast parts. This is mostly a matter of setting our notation, which in fact is the same notation as [KKO86]. The noncommutative Gröbner computer algebra which is the subject of our investigations is well suited for manipulating polynomials into a triangular or even decoupled form.

3.3 Decomposing the Problem

Here, we decompose our two time scale system into it's fast parts and slow parts. We will assume throughout that A_{22} is invertible.

3.3.1 The slow system

The dynamics of the slow system can be found by setting ϵ to zero in equation (6) obtaining what is often called the *quasi-steady state* of the system. This transforms the equation involving ϵ in system (6) into an algebraic equation rather than a differential equation,

$$z_s(t) = -A_{22}^{-1}(A_{21}x_s(t) + B_2u_s(t)), \tag{14}$$

and then substitution of this z_s into the top equation in (6) yields

$$\frac{dx_s}{dt} = A_0x_s(t) + B_0u_s(t), \ \ x_s(t_0) = x^0 \tag{15}$$

where

$$A_0 \triangleq A_{11} - A_{12}A_{22}^{-1}A_{21}, \qquad B_0 \triangleq B_1 - A_{12}A_{22}^{-1}B_2.$$

Here the subscript s indicates that the vectors in equations (15)-(14) are the slow parts of the vectors in (6).

3.3.2 The fast system

The fast system has dynamics

$$\frac{dz_f}{dt} = A_{22}z_f(t) + B_2u_f(t), \qquad z_f(t_0) = z^0 - z_s(t_0) \tag{16}$$

where

$$z_f = z - z_s \text{ and } u_f = u - u_s. \tag{17}$$

Here the subscript f indicates that the vectors in equations (16)-(17) are the fast parts of the vectors in (6).

3.4 Decomposing the measurement

We may then also decompose (7) into it's slow and fast parts

$$\begin{aligned} y &= M_1 x + M_2 z \\ &= M_1[x_s + O(\epsilon)] + M_2[-A_{22}^{-1}(A_{21}x_s + B_2 u_s) + z_f + O(\epsilon)] \\ &= y_s(t) + y_f(t) + O(\epsilon) \end{aligned}$$

where

$$y_s = M_0 x_s + N_0 u_s \text{ and } y_f = M_2 z_f$$

and

$$M_0 \triangleq M_1 - M_2 A_{22}^{-1} A_{21} \text{ and } N_0 \triangleq -M_2 A_{22}^{-1} B_2.$$

4 Computer algebra vs. the standard singular perturbation problem

The matrix $K(\epsilon)$ in (12) must be partitioned compatibly with the states, x and z, and is the limit of the power series which is conventionally written

$$K_N(\epsilon) = \sum_{i=0}^{N} \epsilon^i \begin{bmatrix} k_{(1,i)} & \epsilon k_{(2,i)} \\ \epsilon k_{(2,i)}^T & \epsilon k_{(3,i)} \end{bmatrix} \tag{18}$$

where $k_{(j,i)}$ are appropriately sized matrices (see (6)). We shall use k_{ji} synonymously with $k_{(j,i)}$, since this saves space and actually corresponds to the TEX output of NCAlgebra.

The remainder of this section will be devoted to finding formulas for the $k_{(j,i)}$ for $j \in \{1, 2, 3\}$ and $i \geq 0$.

4.1 The zero-th order term of the Riccati equation (constant (i.e., ϵ^0) coefficients)

We begin our investigations of the perturbed control problem by searching for the first term of the series (18) consisting of matrices, $k_{(1,0)}$, $k_{(2,0)}$, and $k_{(3,0)}$. We substitute $K_0(\epsilon)$ into (10) and take only the zero-th order terms in ϵ of the resulting equations. This is problem (b) mentioned in the introduction. In the next section, Section 4.1.1, we will show how this may be done with the assistance of a computer.

This finally brings us to the subject of our investigations, the manipulation of matrix polynomials with computer algebra methods.

4.1.1 Computer algebra finds the basic equations

We begin by using computer algebra to assist in finding the zero-th order terms in ϵ of the equations given in (10).

First, using NCAlgebra, we define the block matrices in (11). The matrix, $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, is represented in Mathematica by `{{a,b},{c,d}}`. The suffix, `[[`j, i`]]`, extracts the element in the j-th row and the i-th column from a given matrix. In NCAlgebra, `MatMult` performs the matrix multiplication operation, `tpMat` performs the symbolic transpose operation, and `**` indicates noncommutative multiplication.

```
A = {{A11,A12},{1/ep A21,1/ep A22}};
B = {{B1},{1/ep B2}}; M = {{M1,M2}};                          (19)
```

We also define a K_0.

```
K0 = {{k10,ep k20}, {ep tp[k20],ep k30}};
```

The following Mathematica *function* takes as an argument a matrix K and generates the Riccati (10).

```
Riccati[K_] := MatMult[K,A] + MatMult[tpMat[A],K] -
```

```
MatMult[K,B,Inv[R],tpMat[B],K] + MatMult[tpMat[M],M]                                    (21)
```

We next use the NCAlgebra command `NCTermsOfDegree`[3]. The following Mathematica commands will extract the 0-th order terms in ϵ, creating the polynomials in (24), (25), and (26).

```
Ep10 = NCTermsOfDegree[ Riccati[K0][[1,1]] , {ep},{0}]
Ep20 = NCTermsOfDegree[ Riccati[K0][[1,2]] , {ep},{0}]
Ep30 = NCTermsOfDegree[ Riccati[K0][[2,2]] , {ep},{0}]                                  (22)
```

The output
Input (22) creates three polynomials, the third of which is

```
k30**A22+tp[A22]**k30+tp[M2]**M2 -k30**B2**Inv[R]**tp[B2]**k30.                         (23)
```

When all three are output in TeX, which is done easily by NCAlgebra, we get that `Riccati[K0] = 0` corresponds to the equations

$$\begin{aligned} 0 &= k_{10}\,A_{11} + A_{11}^T\,k_{10} + k_{20}\,A_{21} + A_{21}^T\,k_{20}^T + M_1^T\,M_1 - k_{10}\,B_1 R^{-1}\,B_1^T\,k_{10} - k_{20}\,B_2\,R^{-1}\,B_1^T\,k_{10} \\ &- k_{10}\,B_1 R^{-1}\,B_2^T\,k_{20}^T + k_{20}\,B_2\,R^{-1}\,B_2^T\,k_{20}^T \qquad (24) \\ 0 &= k_{20}\,A_{22} - k_{20}\,B_2\,R^{-1}\,B_2^T\,k_{30} + k_{10}\,A_{12} + A_{21}^T\,k_{30} + M_1^T\,M_2 - k_{10}\,B_1 R^{-1}\,B_2^T\,k_{30} \qquad (25) \\ 0 &= k_{30}\,A_{22} + A_{22}^T\,k_{30} + M_2^T\,M_2 - k_{30}\,B_2\,R^{-1}\,B_2^T\,k_{30} \qquad (26) \end{aligned}$$

Simple analysis of the basic equations
Notice that (26), the TeX form of (23), contains only one unknown k_{30} and has the form of a Riccati equation. Thus k_{30} is uniquely determined by this equation if we assume it is the "stabilizing solution". That is $A_{22}-B_2R^{-1}B_2^Tk_{30}$ has all eigenvalues in the strict left half plane and is therefore invertible. We have found in our computer experiments that it is best to make heavy invertibility assumptions, especially at the outset. For computer algebra the key property here is invertibility of $A_{22} - B_2R^{-1}B_2^Tk_{30}$.

We may also motivate the invertibility of $A_{22} - B_2R^{-1}B_2^Tk_{30}$ by purely algebraic arguments as follows. Equation (24) contains two unknowns, k_{10} and k_{20}, and equation (25) contains all three unknowns, k_{10}, k_{20}, and k_{30}. To solve for k_{20} we use equation (25) and call `NCCollect[ Ep20, k20 ]` to get the following relation

```
k20**(A22-B2**Inv[R]**tp[B2]**k30)+tp[A21]**k30 + tp[M1]**M2 - k10**B1**Inv[R]**tp[B2]**k30
+ k10**A12                                                                              (27)
```

Upon examination of (27) it is immediate that we may give k_{20} explicitly in terms of k_{10} and k_{30} by assuming the invertibility of the parenthesized expression in the above relation, $A_{22} - B_2R^{-1}B_2^Tk_{30}$. We have

$$k_{20} = [-k_{10}A_{12} + k_{10}B_1R^{-1}B_2^Tk_{30} - A_{21}^Tk_{30} - M_1^TM_2]\cdot(A_{22} - B_2R^{-1}B_2^Tk_{30})^{-1}. \qquad (28)$$

We use this expression for k_{20} to change (24) into an equation involving k_{10} and k_{30}. The unknown matrices k_{i0} could then be found by first using (26) to solve for k_{30}, then using our transformed (24) to solve for k_{10}, and finally using (28) to obtain k_{20}. A better situation would be to have some decoupling of the unknown matrices so that certain unknown matrices could be computed concurrently rather than sequentially. Our next objective is to find decoupled equations which determine the unknown matrices.

4.1.2 Heavy analysis of the basic equations

We will show how (24) which involves two unknowns may be replaced by an equation involving only one unknown, k_{10}, so that k_{30} and k_{10} may be computed using two independent Riccati equations. We will use the bulk of our Gröbner basis machinery here.
Computer algebra jargon
There are several terms we will use which, though simple conceptually, may not be familiar to the control engineer. A product of variables, $x_1^{\alpha_1}\cdot x_2^{\alpha_2}\cdots x_n^{\alpha_n}$ where $\alpha_k \in \mathbb{N}$, is called a *monomial*. A *polynomial* f is a finite $\mathbb{C}$-linear

[3] `NCTermsOfDegree` takes 3 arguments: a (noncommutative) polynomial, a list of variables, and a set of indices. The command returns an expression such that each term is homogeneous with degree given by the indices. For example, the call `NCTermsOfDegree[ A**B + B**C**C + B**A**C + C**D, {C}, {1} ]` returns `B**A**C + C**D`.

combination of monomials,

$$\sum_{j=1}^{m} a_j x_1^{\alpha_1^j} \cdot x_2^{\alpha_2^j} \cdots x_n^{\alpha_n^j}$$

where $a_j \in \mathbb{C}$. We call a_k the *coefficient* of the *term* $a_k x_1^{\alpha_1^k} \cdot x_2^{\alpha_2^k} \cdots x_n^{\alpha_n^k}$. A *relation* is a polynomial which is assumed to be 0. That is, we may write the equation, $3x^2yz = yz + 4z^2$, as a relation, $3x^2yz - yz - 4z^2$. Many NCAlgebra functions will accept either relations or equations. We will slip back and forth between the two notations and the meaning should be clear from the context.

All algebraic identities which hold

As described in Section 2.3, the GBA takes as input a set of polynomials and an order on the variables involved. It outputs a (generally) more desirable set of equations. It is not necessary to know *which* polynomials are needed to derive a certain relation. It is only necessary that all needed polynomials are present in the input. For this reason one generally gives as input to the GBA *all polynomial relations known to hold.* There is no harm in having superfluous (but true) relations in the input. Now we will list the input to our computer algebra program which will generate *all polynomial relations known to hold.*

First the basic relations we wish to study were produced in the previous section, `Ep10`, `Ep20` and `Ep30`; equations (24), (25), and (26).

In light of the slow system terminology introduced above in Sections 3.3.1 and 3.4 we make the following abbreviations.

```
Abbreviations = { N0 == - M2**Inv[A22]**B2, M0 == M1 - M2**Inv[A22]**A21,
A0 == A11 - A12 ** Inv[A22]**A21, B0 == B1 - A12**Inv[A22]**B2, R0 == R + tp[N0]**N0 }   (29)
```

We add the abbreviation R_0 for convenience as done in [KKO86], although it is not essential.
`Inv[R]` is the NCAlgebra representation of R^{-1}. Since `=` denotes assignment, Mathematica uses `==` to denote equality (for equations).

Several of the matrices or matrix polynomials in our problem are assumed to be invertible. It is common to take the matrices, A_{ii}, to be of full rank, since otherwise a transformation could be applied to the original system to reduce the size of the state. The matrix $A_{22} - B_2R^{-1}B_2^T k_{30}$ has already been assumed to be invertible to facilitate the definition of k_{20} in (28). The matrices R and R_0 are positive definite and so must be invertible.

We generate the relations which result from these observations with the following command,

```
Inverses = NCMakeRelations[{Inv, R, R0, A0, A11, A22, (A22 - B2**Inv[R]**tp[B2]**k30) }]  (30)
```

Several of the matrices are known to be self adjoint, and therefore the following relations must hold:

```
SelfAdjoints = { k10 == tp[k10], k30 == tp[k30],
R == tp[R], R0 == tp[R0], Inv[R] == tp[Inv[R]], Inv[R0] == tp[Inv[R0]] }           (31)
```

We combine all of our relations with

```
Relations = Union[Ep10, Ep20, Ep30, Abbreviations, SelfAdjoints, Inverses]           (32)
```

If $p == 0$ is a true equation, then $tp[p] == 0$ is also. We add these "transposed" equations:

```
AllRelations = NCAddTranspose[Relations]                                              (33)
```

Orders

In order to find a polynomial in k_{10}, A_0, B_0, M_0, N_0, R_0, and other variables with a minimal number of occurrences of k_{10}, we should create a Gröbner basis for *all polynomial relations known to hold* under the following order.

$$N_0 < M_0 < R_0 < A_0 < B_0 \ll k_{10} \ll \text{ other variables} \ll k_{20} \tag{34}$$

Experimenting with other orders is easy, but this one works. The order mentioned in (34) is specified using the NCAlgebra command

```
NCAutomaticOrder[ {{N0,M0,R0,A0,B0},{k10}, {B1,B2,M1,M2,R,A11,A12,A21,A22,
Inv[A22- B2**Inv[R]**tp[B2]**k30], tp[Inv[A22-B2**Inv[R]**tp[B2]**k30]]}
{k30},{k20}}, AllRelations ]                                                    (35)
```

This command scans `AllRelations` for unassigned letters and places them in the order compatibly with the order given in the first argument.

Finally, the call to make the Gröbner basis is made. This call will create a four iteration partial Gröbner basis from the polynomials included in `AllRelations` and the output will be stored in the file, "FindK10".

```
NCProcess[AllRelations,4,"FindK10",SBByCat->False ]                             (36)
```

The "option" `SBByCat`, which removes large numbers of "redundant" equations, can be ignored by the reader since in fact we have turned it off to save time.

4.1.3 The output

The output of the command (36) is a set of polynomials which make up the partial Gröbner basis to which *RemoveRedundant* has been applied created from the polynomials in `AllRelations` under the order specified in (35). The software we use actually does more than just create a partial Gröbner basis. `NCProcess` removes redundant relations and categorizes the output depending on how many unknowns lie in each relation. Then it automatically sets them in TEX, TEX's the file, and opens a window displaying them using 'xdvi'. In this case a category was found which consisted of a single relation in the one unknown k_{10} which we will refer to as `k10rel`. NCProcess automatically performs `NCCollect[k10rel,k10]` and displays

---

The expressions with unknown variables $\{k_{10}\}$
and knowns $\{A_0,\, B_0,\, M_0,\, N_0,\, A_0^T,\, B_0^T,\, M_0^T,\, N_0^T,\, R_0^{-1}\}$

$$k_{10}\,\left(A_0 - B_0\,R_0^{-1}\,N_0^T\,M_0\right) + \left(A_0^T\; - M_0^T\,N_0\,R_0^{-1}\,B_0^T\,\right)k_{10} + M_0^T\,M_0 - k_{10}\,B_0\,R_0^{-1}\,B_0^T\,k_{10} - M_0^T\,N_0\,R_0^{-1}\,N_0^T\,M_0 \tag{37}$$

This gives us a desirable decoupled form for the unknown variables k_{10} and k_{30}. There exist independent Riccati equations for k_{10} (see (37)), and k_{30} (see (26)) and it is easy to see from the full output that no equations coupling k_{10} and k_{30} exist. After solving these two equations for k_{10} and k_{20}, we may find k_{20} by substituting k_{10} and k_{30} into (28).

The calculation which computes (37) is no easy feat by hand, as the substitutions and non-standard notation on pages 116-117 of [KKO86] will attest. The answer we found with Gröbner computer algebra is the same as derived there by hand. After the commands were typed into a file, this computation took less than 3 minutes.

4.1.4 The zero-th order term of the controller

The optimal controller (9) has first term in ϵ equal to

$$G = -R^{-1}\left[\begin{array}{cc} B_1^T & \frac{B_2^T}{\epsilon} \end{array}\right]\left[\begin{array}{cc} k_{(1,0)} & \epsilon k_{(2,0)} \\ \epsilon k_{(2,0)}^T & \epsilon k_{(3,0)} \end{array}\right]. \tag{38}$$

and the previous section tells how to compute the k_{j0}.

Note that in many cases it may be advantageous to have an ϵ independent controller. Such a goal may be achieved by setting the upper right entry of K to zero. This gives us

$$\begin{aligned} G\left[\begin{array}{c} x \\ z \end{array}\right] &= \left[\begin{array}{cc} G_{10} & G_{20} \end{array}\right]\left[\begin{array}{c} x \\ z \end{array}\right] \\ &= -R^{-1}(B_1^T k_{(1,0)}x + B_2^T k_{(2,0))}^T x + B_2^T k_{(3,0)}z) \end{aligned} \tag{39}$$

where $k_{(i,0)}$ is defined by equations (37), (28), and (26) for i equal to 1,2, and 3 respectively.

4.2 The order ϵ term of the Riccati equation

In Section 4.1.4, a controller was presented which does not depend on the parameter ϵ. This is especially appropriate if ϵ represents some very small unknown parameter. In fact, there are many circumstances when the parameter ϵ, while small, is known. In such a case, even though the optimal controller is an infinite power series in ϵ one can make an n^{th} order approximation to $G(\epsilon)$ in (9) and arrive at a controller with enhanced performance.

A major obstruction to such an improved approach is the tedious computation required to generate formulas for the coefficients of higher powers of ϵ. We did not find references where anyone generated formulas for coefficients of ϵ higher than 1. The methods in this paper do, see Section 4.3.

As done in [KKO86] we will now obtain decoupled formulas for the matrices $k_{(1,1)}$, $k_{(2,1)}$, and $k_{(3,1)}$ described in (18). Our approach will require considerably less work than doing it by hand.

Instead of truncating the series (18) to only one term as done in Section 4.1.4 (input (19)) here we define symbolic entries for the second term of K as well.

```
K1 = {{ k10, ep k20 }, {ep tp[k20], ep k30 }}
+ ep {{ k11, ep k21 }, {ep tp[k21], ep k31 }}                      (40)
```

We also append the following abbreviations for the controller discussed above in Section 4.1.4 and defined in equation (39). These formulas are standard, [KKO86].

```
Abbreviations = Union[ Abbreviations, { G10 == -Inv[R]**(tp[B1]**k10 + tp[B2]**tp[k20]),
G20 == -Inv[R]**tp[B2]**k30 } ].                                   (41)
```

Since $A_{22} + B_2 G_{20} = A_{22} - B_2 R^{-1} B_2^T k_{30}$ was previously assumed to be invertible, we also add the invertibility relation,

```
Inverses = Union[ Inverses, NCMakeRelations[{Inv, (A22 + B2**G20)}] ].   (42)
```

4.2.1 Extracting the coefficients of ϵ^1

The Riccati expression in K_1, `Riccati[K1]`, is an equation with linear and quadratic terms in ϵ. As done in the last section, the approach here is to equate coefficients of ϵ. Of course, equating coefficients of ϵ^2 is bogus, since the actual power series (18) would have $k_{(i,2)}$ which have not been introduced in computer input (40). We can extract the coefficients of ϵ in equation (10) with the following commands.

```
Ep11 = NCTermsOfDegree[ Riccati[K1][[1,1]] , {ep},{1}]             (43)
```

creates the following polynomial

```
ep k11**A11+ep k21**A21+ep tp[A11]**k11+ep tp[A21]**tp[k21]-
ep k10**B1**Inv[R]**tp[B1]**k11-ep k10**B1**Inv[R]**tp[B2]**tp[k21]-
ep k20**B2**Inv[R]**tp[B1]**k11-ep k20**B2**Inv[R]**tp[B2]**tp[k21]-
ep k11**B1**Inv[R]**tp[B1]**k10-ep k11**B1**Inv[R]**tp[B2]**tp[k20]-
ep k21**B2**Inv[R]**tp[B1]**k10-ep k21**B2**Inv[R]**tp[B2]**tp[k20]   (44)
```

and

```
Ep21 = NCTermsOfDegree[ Riccati[K1][[1,2]] , {ep},{1}]             (45)
Ep31 = NCTermsOfDegree[ Riccati[K1][[2,2]] , {ep},{1}]             (46)
```

give similar looking formulas.

4.2.2 Solving for the unknowns

These valid relations can now be added to *all relations known to hold*, (19), (21), (29), (31), and (30), with the following command. Since the output of the `NCTermsOfDegree[ ]` command includes the variable `ep` and we want just the coefficients of `ep`, we set the unwanted variable, `ep`, to 1. We do this by appending the Mathematica suffix `/.ep->1` to expressions involving `ep`.

```
AllRelations = Union[Ep11/.ep->1, Ep21/.ep->1,Ep31/.ep->1, Ep10,Ep20,Ep30,
Abbreviations,SelfAdjoints,Inverses]                                   (47)
```

Considering the analysis done in Section 4.1.1, $k_{(1,0)}$, $k_{(2,0)}$, and $k_{(3,0)}$ (`k10, k20, k30`) can be regarded as known and we are now looking at the second term of the series (18), which is made up of these and other knowns and unknowns `k11, k21`, and `k31` introduced above in (40). We wish to find simple formulas which determine the unknowns.

With this distinction between known variables and unknown variables, the following order on the variables is appropriate

$$
\begin{aligned}
N_0 < M_0 < R < A_0 < B_0 < B_1 < B_2 < M_1 < M_2 < \\
R_0 < A_{11} < A_{12} < A_{21} < A_{22} < G_{10} < G_{20} < \\
k_{30} < k_{20} < k_{10} \ll k_{11} \ll \text{ other variables}
\end{aligned}
\tag{48}
$$

This order is imposed with the command

```
NCAutomaticOrder[ {{N0,M0,R,A0,B0, A11,A12,A21,A22,B1,B2,M1,M2,R0,G10,G20,
k30,k20,k10},{k11},{k31},{
Inv[A22- B2**Inv[R]**tp[B2]**k30],tp[Inv[A22- B2**Inv[R]**tp[B2]**k30]]
},{k21}}, AllRelations ];                                              (49)
```

We next call `NCProcess` with an iteration count of 3. The computer input is similar to (36). With this input, `NCProcess` took less than 7 minutes. The output of this command contains a single relation with 24 terms involving the single unknown matrix $k_{(1,1)}$, `k11rel`. Collecting around the `k11` with the command `NCCollect[ k11rel, k11 ]` gives us the following relation

$$
\begin{aligned}
&-1\, k_{11}\,(A_0 - B_0\, R_0^{-1}\, B_0^T\, k_{01} - B_0\, R_0^{-1}\, N_0^T\, M_0) + (k_{01}\, B_0\, R_0^{-1}\, B_0^T + M_0^T\, N_0\, R_0^{-1}\, B_0^T - A_0^T)\, k_{11} + A_0^T\, k_{02}\, A_{22}^{-1}\, A_{21} + \\
&A_{21}^T\, A_{22}^{T-1}\, k_{02}^T\, A_0 - k_{01}\, B_0\, R_0^{-1}\, B_0^T\, k_{02}\, A_{22}^{-1}\, A_{21} - k_{01}\, B_0\, R_0^{-1}\, B_2^T\, A_{22}^{T-1}\, k_{02}^T\, A_0 - A_0^T\, k_{02}\, A_{22}^{-1}\, B_2\, R_0^{-1}\, B_0^T\, k_{01} - \\
&A_0^T\, k_{02}\, A_{22}^{-1}\, B_2\, R_0^{-1}\, N_0^T\, M_0 - A_{21}^T\, A_{22}^{T-1}\, k_{02}^T\, B_0\, R_0^{-1}\, B_0^T\, k_{01} - A_{21}^T\, A_{22}^{T-1}\, k_{02}^T\, B_0\, R_0^{-1}\, N_0^T\, M_0 - \\
&M_0^T\, N_0\, R_0^{-1}\, B_0^T\, k_{02}\, A_{22}^{-1}\, A_{21} - M_0^T\, N_0\, R_0^{-1}\, B_2^T\, A_{22}^{T-1}\, k_{02}^T\, A_0 + k_{01}\, B_0\, R_0^{-1}\, B_0^T\, k_{02}\, A_{22}^{-1}\, B_2\, R_0^{-1}\, B_0^T\, k_{01} + \\
&k_{01}\, B_0\, R_0^{-1}\, B_0^T\, k_{02}\, A_{22}^{-1}\, B_2\, R_0^{-1}\, N_0^T\, M_0 + k_{01}\, B_0\, R_0^{-1}\, B_2^T\, A_{22}^{T-1}\, k_{02}^T\, B_0\, R_0^{-1}\, B_0^T\, k_{01} + \\
&k_{01}\, B_0\, R_0^{-1}\, B_2^T\, A_{22}^{T-1}\, k_{02}^T\, B_0\, R_0^{-1}\, N_0^T\, M_0 + M_0^T\, N_0\, R_0^{-1}\, B_0^T\, k_{02}\, A_{22}^{-1}\, B_2\, R_0^{-1}\, B_0^T\, k_{01} + \\
&M_0^T\, N_0\, R_0^{-1}\, B_0^T\, k_{02}\, A_{22}^{-1}\, B_2\, R_0^{-1}\, N_0^T\, M_0 + M_0^T\, N_0\, R_0^{-1}\, B_2^T\, A_{22}^{T-1}\, k_{02}^T\, B_0\, R_0^{-1}\, B_0^T\, k_{01} + \\
&M_0^T\, N_0\, R_0^{-1}\, B_2^T\, A_{22}^{T-1}\, k_{02}^T\, B_0\, R_0^{-1}\, N_0^T\, M_0
\end{aligned}
\tag{50}
$$

The coefficients of $k_{(1,1)}$ in equation (50), the polynomials in parentheses, suggest that we make the following abbreviation[4]

```
F0 == A0-B0**Inv[R0]**(tp[N0]**M0 + tp[B0]**k01)                       (51)
```

With computer input (51) appended to *all relations known to hold*, `AllRelations`, we can put F_0 low in the order and run `NCProcess` again, creating a new (partial) Gröbner basis. The output of this command contains the aesthetically pleasing relation defining $k_{(1,1)}$

$$
\begin{aligned}
&-k_{11}\, F_0 - 1\, F_0^T\, k_{11} + A_{21}^T\, (A_{22} + B_2\, G_{20})^{-T}\, k_{20}^T\, F_0 + F_0^T\, k_{20}\, (A_{22} + B_2\, G_{20})^{-1}\, A_{21} + \\
&F_0^T\, k_{20}\, (A_{22} + B_2\, G_{20})^{-1}\, B_2\, G_{10} + G_{10}^T\, B_2^T\, (A_{22} + B_2\, G_{20})^{-T}\, k_{20}^T\, F_0.
\end{aligned}
\tag{52}
$$

This is a simple Lyapunov equation whose solution, $k_{(1,1)}$, is unique as long as `F0`, is Hurwitz. We will therefore regard $k_{(1,1)}$ as *known* from this point forward.

Similar to the zero-th order case the equation defining $k_{(3,1)}$ is an immediate consequence of (53) and takes the collected form

[4] More analytic methods can be used, [KS72], to show that this expression (51) is of the form $A_0 + B_0 G_0$ where G_0 is the optimal control for the slow part of the LQR optimal problem, (15). The focus here is on the computer algebra and the expression, F_0, is discovered purely algebraically.

$k_{31}\,(A_{22} - B_2\,R^{-1}\,B_2^T\,k_{30}) \;+\; (A_{22}^T - k_{30}\,B_2\,R^{-1}\,B_2^T\,)\,k_{31} \;+\; A_{12}^T\,k_{20} \;+\; k_{02}^T\,A_{12} \;-\; k_{30}\,B_2\,R^{-1}\,B_1^T\,k_{20} \;-$
$k_{20}^T\,B_1\,R^{-1}\,B_2^T\,k_{30}$ (53)

Also similar to the zero-th order case we have an explicit formula for $k_{(2,1)}$ in terms of $k_{(1,1)}$ and $k_{(3,1)}$. The following relatively simple formula was also in the output of the `NCProcess` command which generated (52).

$k_{21} \rightarrow -1\,k_{11}\,A_{12}\,A_{22}^{-1} - A_{11}^T\,k_{20}\,(A_{22} + B_2\,G_{20})^{-1} - A_{21}^T\,k_{31}\,(A_{22} + B_2\,G_{20})^{-1} - G_{10}^T\,B_1^T\,k_{20}\,(A_{22} + B_2\,G_{20})^{-1} -$
$G_{10}^T\,B_2^T\,k_{31}\,(A_{22} + B_2\,G_{20})^{-1} + k_{11}\,B_0\,R_0^{-1}\,(N_0^T\,M_2\,A_{22}^{-1} - B_2^T\,A_{22}^{T-1}\,k_{30})$ (54)

Here, and in the rest of the paper, an arrow can be interpreted to mean equal sign. Expressions equivalent to (52), (54), and (53) can be found in [KKO86].

Notice that a similar procedure could be done as described in Section 4.1.4 to derive an order ϵ controller.

4.3 The order ϵ^2 term of the Riccati equation

At this point the tale is growing long and the weary reader can most likely guess what will be done in this section from the title. For the sake of presenting a formula which has not appeared before we create a three term approximation to $K(\epsilon)$,

```
K2 = {{ k10, ep k20 }, {ep tp[k20], ep k30 }}
+ ep {{ k11, ep k21 }, {ep tp[k21], ep k31 }}
+ ep^2 {{ k21, ep k22 }, {ep tp[k22], ep k32 }},                    (55)
```

For this problem a three iteration partial Gröbner basis was created and we arrived at formulas defining $k_{(1,2)}$, $k_{(2,2)}$, and $k_{(3,2)}$. Even without running `NCProcess` one sees that $k_{(3,2)}$ satisfies a Riccati equation. This is analogous to the lower order cases.

We found one equation which expresses $k_{(2,2)}$ in terms of $k_{(1,2)}$ and $k_{(3,2)}$. We found a Lyapunov equation in the unknown $k_{(1,2)}$ consisting of 150 lines in Mathematica notation. There were also several equations in the unknowns $k_{(1,2)}$ and $k_{(3,2)}$. In analogy with the lower order cases we expect that these "coupled" equations are redundant and provided no additional information or contraints. Our algebraic software in principle if given enough time can determine this but these algorithms are more computer intensive and did not finish when run on this problem.

We used a version of `NCProcess` which was specialized to display only equations involving the unknowns; $k_{(1,2)}$ and $k_{(2,2)}$. This substantially speeds up run times. Still, our rather formidable conclusion took 21.5 minutes. The formulas can be found at
`http://math.ucsd.edu/~ncalg/SingularPerturbation.`

It is gratifying that our Gröbner equation processing techniques prevailed on such a large problem. It leads us to think that many singular perturbation problems are well within the scope of our computer algebra techniques.

5 Perturbing singular solutions of the Information State Equation

We would also like to mention that the techniques illustrated on the previous problem apply to other problems. In particular we mention a singular perturbation analysis of an important entity in the output feedback H^∞ control problem, the information state. It corresponds not to fast and slow time scales but to sensors becoming increasingly accurate, for details see [HJM98].

5.1 The general problem

Consider the system

$$\frac{dx}{dt} = A(x) + B(x)v \tag{56}$$

$$out = C(x) + D(x). \tag{57}$$

An equation useful in estimation associated to this (details in [HJM98]) is the *information state equation*, **ISE**:

$$
\begin{aligned}
-\frac{dp}{dt} &= (A(x)+B(x)\cdot v(t))^T \nabla_x p_t(x) - (\nabla_x p_t(x))^T Q(x) \nabla_x p_t(x) \\
&- [C(x)-Dv(t)]^T J [C(x)-Dv(t)] \\
&+ \frac{1}{\epsilon^2}(Lv(t)-x)^T R (Lv(t)-x)
\end{aligned} \tag{58}
$$

where J is self adjoint.

5.2 The linear case

Assuming that the associated vector field is linear and ϵ is fixed, it is known that a solution exists of the form

$$p_t(x) = \frac{1}{2}(x-x_\epsilon)^T P_\epsilon (x-x_\epsilon) + \phi^\epsilon(t), \tag{59}$$

where P_ϵ *is a symmetric matrix which does not depend on* t*, but* x *and* x_ϵ *do depend on* t. Then

$$\nabla_x p = P_\epsilon (x - x_\epsilon).$$

Noting that $(Ax+Bv(t))^T \nabla p$ is scalar, we can symmetrize equation (58), the information state equation, ISE, and arrive at

$$
\begin{aligned}
-\frac{dp}{dt} &= [Ax+Bv(t)]^T P_\epsilon (x-x_\epsilon) + (x-x_\epsilon)^T P_\epsilon [Ax+Bv(t)] \\
&- (x-x_\epsilon)^T P_\epsilon Q P_\epsilon (x-x_\epsilon) - [Cx-Dv(t)]^T J [Cx-Dv(t)] \\
&+ \frac{1}{\epsilon^2}(Lv(t)-x)^T R (Lv(t)-x).
\end{aligned} \tag{60}
$$

where R is positive semi-definite.

In seeking an asymptotic expansion one can try various possible forms for a solution. We have found our computer algebra methods very effective in going from a postulated form of an expansion to explicit formulas for their coefficients. Let us illustrate this with the following postulates for P_ϵ and x_ϵ. P_ϵ is a function of ϵ which has a series form

$$P_\epsilon = \frac{1}{\epsilon}P_{-1} + P_0 + \epsilon P_1 + \epsilon^2 P_2 + \ldots \tag{61}$$

and we expand x_ϵ as

$$x_\epsilon(t) = x_{\epsilon,0}(t) + \epsilon x_{\epsilon,1}(t) + \epsilon^2 x_{\epsilon,2}(t) + \ldots \tag{62}$$

The treatment of ϕ^ϵ is less critical as we shall see.

5.3 Finding the expansions of unknowns using computer algebra

We now apply NCAlgebra methods very similar to the ones demonstrated in Section 4 to the ISE singular perturbation problem. We do not give as much detail since now the idea should be clear. However, we state our expansions precisely, since this must be done to set notation.

5.3.1 Setting the expansions of unknowns

We will begin by creating a symbolic entry for P_ϵ. This is done with the following computer input

```
Pe = (1/ep) Pm1 + P0 + ep P1 + ep^2 P2 + ep^3 P3 ;                          (63)
```

where `ep`, the symbolic entry for ϵ, is declared to be commutative and `Pm1, P0, P1, P2`, and `P3`, symbolic entries for P_{-1}, P_0, P_1, P_2, and P_3 respectively, are declared to be noncommutative.

We do not need a formula for p_t itself, since what we use is `GP` $= \nabla_x p_t(x)$. It is

```
GP = Pe ** (x-xe)                                              (64)
```

The x_ϵ described in (62) is defined next

```
xe = xe0 + ep xe1 + ep^2 xe2 + ep^3 xe3                        (65)
```

We also need the derivative $\frac{dx_\epsilon}{dt}$ and it has the expansion

```
dxe = dxe0 + ep dxe1 + ep^2 dxe2 + ep^3 dxe3.                  (66)
```

Since $\phi^\epsilon(t)$ introduced in equation (59) does not depend on x, its derivative in t appears and that is on the left side of (60). This means that whatever we find as an approximation to P_ϵ and x_ϵ can be put in the right side of the equation

$$-\frac{d\phi^\epsilon(t)}{dt} = -(Bv)^T P_\epsilon x_\epsilon - x_\epsilon^T P_\epsilon Bv + x_\epsilon^T P_\epsilon Q P_\epsilon x_\epsilon + (Dv)^T JDv + \frac{1}{\epsilon^2}(Lv)^T RLv \tag{67}$$

which can be integrated to produce $\phi^\epsilon(t)$. Also we can expand $\frac{d\phi^\epsilon(t)}{dt}$ out a few terms in ϵ and obtain formulas for these terms in the series.

5.3.2 The equations

The ISE (58) is implemented next, and the command `NCTermsOfDegree` is used to sort this by powers of ϵ and x. This produced an equation from each coefficient of $\frac{1}{\epsilon^2}$, $\frac{1}{\epsilon}$, ϵ^0, and x, x^T and x and x^T, to produce a set SE of equations. We did not include terms without x and so no ϕ's will appear in SE. Details are strictly analogous to what we just did in Section 4 so they will be omitted.

We will assume the matrix A is invertible, by defining its inverse:

```
inverses = {Inv[A]**A - 1 == 0, A**Inv[A] - 1 == 0 };         (68)
```

Many of our matrices are known to be self adjoint, so the equations establishing this are also included with SE and `inverses` to obtain the full set of equations we put into our Gröbner basis algorithm.

5.3.3 Applying the Gröbner basis algorithm

We need to select a monomial order. $A, B, C, D, J, Q, R, v,$ and L are known and we certainly do not wish to solve for them, so we set them at the bottom of the order. Since the P_ϵ and x_ϵ, are unknown and must be solved for, we set the symbolic unknowns in their expansion above the knowns in the monomial order. The order we choose on the unknowns puts P_ϵ at the bottom of the unknowns, but still above the knowns which means that once they are solved for they can be used in in formulas for x_ϵ. To implement this and automatically impose a corresponding order on transposes and inverses of variables we use the command `NCAutomaticOrder` and produce the following order

$$\begin{gathered} A < A^{-1} < A^T < B < B^T < Q < J < C < C^T < R < D < D^T < v < v^T < L < L^T \ll \\ Pm_1 < P_0 < P_1 < P_2 < P_3 \ll xe_0 < {xe_0}^T \ll xe_1 < {xe_1}^T \ll xe_2 < {xe_2}^T \ll xe_3 < {xe_3}^T \ll \\ dxe_0 < {dxe_0}^T \ll dxe_1 < {dxe_1}^T \ll dxe_2 < {dxe_2}^T \ll dxe_3 < {dxe_3}^T \end{gathered}$$

Finally a partial Gröbner basis is created and "redundant equations" are removed using the `NCProcess[ ]` command

```
NCProcess[AllRelations,3,"Answer"] ;                           (69)
```

5.4 The answer

The output of this command includes the following relations which will give us formulas and differential equations which have been derived purely algebraically. The computations took 3 minutes and 7 seconds.

5.4.1 Relations defining P_ϵ

The relations involving P_ϵ in the output of NCProcess are exactly

---

The expressions with unknown variables $\{Pm_1\}$
and knowns $\{Q, R\}$

$$Pm_1\, Q\, Pm_1 \rightarrow R \tag{70}$$

The expressions with unknown variables $\{P_0, Pm_1\}$
and knowns $\{A, Q, A^T\}$

$$P_0\, Q\, Pm_1 \rightarrow \tfrac{1}{2}\, Pm_1\, A + \tfrac{1}{2}\, A^T\, Pm_1 - Pm_1\, Q\, P_0 \tag{71}$$

The expressions with unknown variables $\{P_1, P_0, Pm_1\}$
and knowns $\{A, C, J, Q, A^T, C^T\}$

$$P_1\, Q\, Pm_1 \rightarrow \tfrac{1}{2}\, P_0\, A + \tfrac{1}{2}\, A^T\, P_0 - P_0\, Q\, P_0 - Pm_1\, Q\, P_1 - C^T\, J\, C \tag{72}$$

The expressions with unknown variables $\{P_2, P_1, P_0, Pm_1\}$
and knowns $\{A, Q, A^T\}$

$$P_2\, Q\, Pm_1 \rightarrow \tfrac{1}{2}\, P_1\, A + \tfrac{1}{2}\, A^T\, P_1 - P_0\, Q\, P_1 - P_1\, Q\, P_0 - Pm_1\, Q\, P_2 \tag{73}$$

Indeed this TeX display is exactly what one sees on the screen. In view of the above equations we have the following recursive formula for the matrices P_i. We begin with the defining relation for P_{-1}.

$$P_{-1}QP_{-1} = R$$

Then we must also have that P_0 satisfies the algebraic Lyapunov equation

$$P_{-1}A + A^T P_{-1} = 2\left(\begin{bmatrix} P_0Q & P_{-1}Q \end{bmatrix}\begin{bmatrix} P_{-1} \\ P_0 \end{bmatrix}\right) \tag{74}$$

and P_1 satisfies

$$P_0A + A^T P_0 = 2\left(\begin{bmatrix} P_1Q & P_0Q & P_{-1}Q \end{bmatrix}\begin{bmatrix} P_{-1} \\ P_0 \\ P_1 \end{bmatrix} + C^T JC\right) \tag{75}$$

Inspection strongly suggests that the coefficient matrices of higher powers of ϵ are given by the recursive formula

$$P_{i-1}A + A^T P_{i-1} = 2\left(\begin{bmatrix} P_{-1}Q & P_0Q & \cdots & P_iQ \end{bmatrix}\begin{bmatrix} P_i \\ \vdots \\ P_0 \\ P_{-1} \end{bmatrix}\right) \tag{76}$$

which alternatively may be written as

$$P_iQP_{-1} + P_{-1}QP_i = \frac{1}{2}(P_{i-1}A + A^T P_{i-1}) - P_{i-1}QP_0 - P_{i-2}QP_1 - \ldots - P_1QP_{i-2} - P_0QP_{i-1} \tag{77}$$

In many cases we will have $Q = BB^T$ where B is an input matrix corresponding to a dynamical system. Since we expect B to be a tall, skinny matrix we also expect $BB^T P_{-1}$ to have a significant number of eigenvalues equal to zero and hence have a high degree of non uniqueness in the P_i solving equation (77).

5.4.2 Relations defining $x_\epsilon(t)$

The entire output of NCProcess which involves x_ϵ is exactly the following plus their transposes.

The expressions with unknown variables $\{xe_0\}$
and knowns $\{L,\ R,\ v\}$

$R\,(xe_0 - L\,v) == 0$ (78)

The expressions with unknown variables $\{dxe_0,\ xe_1,\ xe_0,\ Pm_1\}$
and knowns $\{A,\ B,\ R,\ v\}$

$Pm_1\,dxe_0 \rightarrow 2\,R\,xe_1 + Pm_1\,A\,xe_0 + Pm_1\,B\,v$ (79)

The expressions with unknown variables $\{dxe_1,\ dxe_0,\ xe_2,\ xe_1,\ xe_0,\ P_0,\ Pm_1\}$
and knowns $\{A,\ B,\ C,\ D,\ J,\ R,\ v,\ C^T\}$

$Pm_1\,dxe_1 \rightarrow -1\,P_0\,dxe_0 + 2\,R\,xe_2 + P_0\,A\,xe_0 + P_0\,B\,v + Pm_1\,A\,xe_1 - 2\,C^T\,J\,C\,xe_0 + 2\,C^T\,J\,D\,v$ (80)

The expressions with unknown variables $\{dxe_2,\ dxe_1,\ dxe_0,\ xe_3,\ xe_2,\ xe_1,\ xe_0,\ P_1,\ P_0,\ Pm_1\}$
and knowns $\{A,\ B,\ C,\ J,\ R,\ v,\ C^T\}$

$Pm_1\,dxe_2 \rightarrow -1\,P_0\,dxe_1 - P_1\,dxe_0 + 2\,R\,xe_3 + P_0\,A\,xe_1 + P_1\,A\,xe_0 + P_1\,B\,v + Pm_1\,A\,xe_2 - 2\,C^T\,J\,C\,xe_1$ (81)

These relations may be written quite succinctly in the following way.

We have the following relations governing the behavior of the terms of the expansion of x_ϵ introduced in equation (62).

$$x_{\epsilon,0} = Lv \text{ on the subspace orthogonal to the kernel of } R \tag{82}$$

or in other words

$$R(x_{\epsilon,0} - Lv) = 0. \tag{83}$$

The terms $x_{\epsilon,0}$, $x_{\epsilon,1}$, and $x_{\epsilon,2}$ are governed by the pair of differential equations

$$P_{-1}\frac{dx_{\epsilon,0}}{dt} = P_{-1}Ax_{\epsilon,0} + 2Rx_{\epsilon,1} \tag{84}$$

$$P_{-1}\frac{dx_{\epsilon,1}}{dt} + P_0\frac{dx_{\epsilon,0}}{dt} = \begin{bmatrix} P_{-1}A & P_0A \end{bmatrix}\begin{bmatrix} x_{\epsilon,1} \\ x_{\epsilon,0} \end{bmatrix} + 2Rx_{\epsilon,2} - 2C^TJCx_{\epsilon,0} + 2\left(C^TJD + P_0B\right)v \tag{85}$$

and then the higher terms, $x_{\epsilon,k}$ for $k \geq 2$, satisfy

$$\begin{aligned} \begin{bmatrix} P_{-1} & P_0 & \cdots & P_{k-1} \end{bmatrix}\begin{bmatrix} \frac{dx_{\epsilon,k}}{dt} \\ \vdots \\ \frac{dx_{\epsilon,1}}{dt} \\ \frac{dx_{\epsilon,0}}{dt} \end{bmatrix} &= \begin{bmatrix} P_{-1}A & P_0A & \cdots & P_{k-1}A \end{bmatrix}\begin{bmatrix} x_{\epsilon,k} \\ \vdots \\ x_{\epsilon,1} \\ x_{\epsilon,0} \end{bmatrix} \\ &- \quad 2C^TJCx_{\epsilon,k-1} + 2Rx_{\epsilon,k+1} + P_{k-1}Bv \end{aligned} \tag{86}$$

5.4.3 Relations defining $\phi^\epsilon(t)$

There are two ways to proceed for computing $\phi^\epsilon(t)$. An approximation to $\phi^\epsilon(t)$ may be computed using the formula (67) along with the expansions just obtained.

The longer method which we do not reproduce here is to give full expansions for $\phi^\epsilon(t)$ in terms of $\phi_l^\epsilon(t)$ where $l \geq -2$. We did this with our computer and found rather long formulas for the first few terms. This was easy, but there is no point in listing them here.

References

[CLS92] D. Cox, J. Little, and D. O' Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Springer-Verlag, Undergraduate Texts in Mathematics, 1992.

[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)*, LNCS# 1232, pages 331–334. Springer-Verlag, 1997.

[HJM98] J. W. Helton, M. R. James, and W. M. McEneaney. Nonlinear control: The joys of having an extra sensor. *Proceedings of the 37th IEEE CDC,Tampa, Florida, USA, Dec 16-18*, 4:3609–13, 1998.

[HMS96] J.W. Helton, R.L. Miller, and M. Stankus. *NCAlgebra: A Mathematica Package for doing noncommuting Algebra.* available from http://math.ucsd.edu/~ncalg, 1996. Beware that to perform the computations in this paper you need NCGB. (See [HS97]).

[HS97] J.W. Helton and M. Stankus. *NCGB: Noncommutative Groebner bases.* available from http://math.ucsd.edu/~ncalg, 1997. As of April'99 this requires Mathematica and the Solaris operating system. A Windows version is coming soon.

[HS99] J.W. Helton and M. Stankus. Computer assistance for discovering formulas in system engineering and operator theory. *Journal of Functional Analysis*, 161:289–368, 1999.

[KG98] B. Keller and E. Green. *The OPAL System.* available from http://hal.cs.vt.edu/opal, 1998.

[KKO86] Petar V. Kokotovic, Hassan K. Khalil, and John O'Reilly. *Singular Perturbation Methods in Control: Analysis and Design.* Academic Press, 1986.

[KS72] Huibert Kwakernaak and Raphael Sivan. *Linear optimal control systems.* Wiley Interscience., 1972.

[Mor86] F. Mora. Groebner bases for non-commutative polynomial rings. *Lecture Notes in Computer Science*, 229:353–362, 1986.

[PB93] Z.G. Pan and T. Basar. H_∞ optimal control for singularly perturbed systems 1. perfect state measurements. *Automatica*, 29:401–423, 1993.

[PB94] Z.G. Pan and T. Basar. H_∞ optimal control for singularly perturbed systems 2. imperfect state measurements. *IEEE Transactions on Automatic Control*, 39:280–299, 1994.

Not For Publication

Contents