Proceedings of the 38th
Conference on Decision & Control
Phoenix, Arizona USA • December 1999

Noncommutative computer algebra in the control of singularly perturbed dynamical systems

J.W. Helton[1] F. Dell Kronewitter[1] Mark Stankus[2]

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. For example, the methods introduced here take the most standard textbook singular perturbation calculation, [KKO86], one step further than had been done previously.

Commutative Groebner basis algorithms are powerful and make up the engines in symbolic algebra packages' `Solve` commands. Noncommutative Groebner 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 Groebner basis package NCGB which runs under it.

1 Introduction

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] and the works of Pan and Basar.

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

[2]Department of Mathematics, California State University. San Luis Obispo,CA 93407

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 Groebner 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 Groebner basis algorithms are very effective on the equations which result. Indeed our method carries out the expansion one step further than had previously been done, see Section 4.2. We also mention another newer H^∞ estimation problem called the "cheap sensor" problem (see [HJM98]). On this our computer techniques proved effective and a longer paper will report these results.

U.S. Government work not protected by U.S. copyright.

1.1 The idea behind Groebner computer algebra

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 Groebner 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.

For noncommutative computation which is the subject of this paper, the Groebner 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. Groebner basis computer runs can be (notoriously) memory and time consuming. Thus their effectiveness on any class of problems can only be determined by experiment. Implementations of the GBA include the NCGB component of NCAlgebra, [HS97], and OPAL, [KG98].

Our computer computations were performed with NCGB on a Sun Ultra I with one 166 Mhz processor and 192MB of RAM.

[1] From this ordering on variables, an order on monomials in those variables is induced which which is referred to as *noncommutative 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.

2 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 output so that the controlled system will satisfy some given performance objective.

2.1 The System

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

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

$$y = \begin{bmatrix} M_1 & M_2 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \quad (7)$$

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, and A_{11}, and A_{12}, A_{21}, A_{22}, B_1, B_2, M_{11}, and M_{22} are appropriately sized matrices.

2.2 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 cost

$$J = \int_0^\infty (y^T y + u^T R u) dt \quad (8)$$

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

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

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

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

with

$$A = \begin{bmatrix} A_{11} & A_{12} \\ \frac{A_{21}}{\epsilon} & \frac{A_{22}}{\epsilon} \end{bmatrix}, B = \begin{bmatrix} B_1 \\ \frac{B_2}{\epsilon} \end{bmatrix}, \quad (11)$$

$$\text{and } M = \begin{bmatrix} M_1 & M_2 \end{bmatrix}$$

$$K(\epsilon) = K_0 + \epsilon K_1 + \epsilon^2 K_2 + \ldots \quad (12)$$

Notice that K solves equation (10) which is a Riccati equation of size $n + m$ by $n + m$. Since the structures present in the system (11) 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. 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 Groebner computer algebra which is the subject of our investigations is well suited for manipulating polynomials into a triangular or even decoupled form.

2.3 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.

2.3.1 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

$$\begin{aligned} \frac{dx_s}{dt} &= A_0 x_s(t) + B_0 u_s(t),\ x_s(t_0) = x^0 \quad (13) \\ z_s(t) &= -A_{22}^{-1}(A_{21}x_s(t) + B_2 u_s(t)) \quad (14) \end{aligned}$$

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 (13)-(14) are the slow parts of the vectors in (6).

2.3.2 The Fast System : The fast system has dynamics

$$\frac{dz_f}{dt} = A_{22}z_f(t) + B_2 u_f(t), z_f(t_0) = z^0 - z_s(t_0) \quad (15)$$

where $$z_f = z - z_s \text{ and } u_f = u - u_s. \quad (16)$$

Here the subscript f indicates that the vectors in equations (15)-(16) are the fast parts of the vectors in (6).

2.4 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 $y_s = M_0 x_s + N_0 u_s$ and $y_f = M_2 z_f$.

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

3 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

$$K_N(\epsilon) = \sum_{i=0}^{N} \epsilon^i \begin{bmatrix} k_{(1,i)} & \epsilon k_{(2,i)} \\ \epsilon k_{(2,i)}^T & \epsilon k_{(3,i)} \end{bmatrix}. \quad (17)$$

where $k_{(i,j)}$ are appropriately sized matrices (see (6)). We shall use k_{ij} synonymously with $k_{(i,j)}$, 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 $i \geq 0$.

3.1 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 (17) 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 3.1.1, we will show how this may be done with the assistance of a computer.

3.1.1 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, `[[i,j]]`, extracts the element in the i-th row and the j-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}};                    (18)
```

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]                                     (19)
```

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 (21), (22), and (23).

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

The output

This input creates three polynomials, the third of which is

```
k30**A22+tp[A22]**k30+tp[M2]**M2
-k30**B2**Inv[R]**tp[B2]**k30.                          (20)
```

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

$$\begin{aligned} 0 &= k_{10}A_{11} + A_{11}^T k_{10} + k_{20}A_{21} + A_{21}^T k_{20}^T \\ &+ M_1^T M_1 - k_{10}B_1 R^{-1} B_1^T k_{10} - k_{20}B_2 R^{-1} B_1^T k_{10} \\ &- k_{10}B_1 R^{-1} B_2^T k_{20}^T + k_{20}B_2 R^{-1} B_2^T k_{20}^T \qquad (21) \\ 0 &= k_{20}A_{22} - k_{20}B_2 R^{-1} B_2^T k_{30} + k_{10}A_{12} + A_{21}^T k_{30} \\ &+ M_1^T M_2 - k_{10}B_1 R^{-1} B_2^T k_{30} \qquad (22) \\ 0 &= k_{30}A_{22} + A_{22}^T k_{30} + M_2^T M_2 - k_{30}B_2 R^{-1} B_2^T k_{30} \qquad (23) \end{aligned}$$

Notice that (23), the TEX form of (20), 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, $A_{22} - B_2 R^{-1} B_2^T k_{30}$ has all eigenvalues in the strict left half plane and is therefore invertible. For computer algebra the key property is invertibility of $A_{22} - B_2 R^{-1} B_2^T k_{30}$. This invertibility assumption could also be motivated by (longer) purely algebraic arguments. Our next objective is to find decoupled equations which determine the unknown matrices.

[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`.

3.1.2 Analysis of the basic equations : We will show how (21),(22), and (23) may be replaced by an equation involving only one unknown k_{10} and equation (23) whose only unknown is k_{30} (so that k_{30} and k_{10} may be computed using two independent Riccati equations) and by a single equation solving for k_{20} and k_{30}. We will use our noncommutative Groebner 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 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 1.1, 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 (21), (22), and (23).

In light of the slow system terminology introduced above in Sections 2.3.1 and 2.4 we make the following abbreviations.

```
Abbreviations = { M0 == - 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[M0]**M0 }                                  (24)
```

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 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 $A_{22} - B_2 R^{-1} B_2^T k_{30}$ has already been assumed to be invertible. The matrices R and R_0 are positive definite and so must be invertible. We generate the relations which result from these observations with

```
Inverses = NCMakeRelations[{Inv, R,R0,A0,A11,A22,
(A22 - B2**Inv[R]**tp[B2]**k30) }]                    (25)
```

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]] }                             (26)
```

We combine all of our relations with

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

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

```
AllRelations = NCAddTranspose[Relations]
```

Orders

We shall create a Groebner basis for *all polynomial relations known to hold* under the following order.

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

Experimenting with other orders is easy, but this one works. This order 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 ]                                       (27)
```

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

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

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

3.1.3 The output : The output of this command is a set of polynomials which make up the the partial Groebner basis created from the polynomials in `AllRelations` under the order specified in (27). The software we use actually does more than just create a Groebner basis. `NCProcess` categorizes the output depending on how many and which 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 $\{A_0, B_0, M_0, N_0, A_0^T, B_0^T, M_0^T, N_0^T, R_0^{-1}\}$

$$k_{10}\left(A_0 - B_0 R_0^{-1} N_0^T M_0\right) + \left(A_0^T - M_0^T N_0 R_0^{-1} B_0^T\right) k_{10} + M_0^T M_0 - k_{10} B_0 R_0^{-1} B_0^T k_{10} - M_0^T N_0 R_0^{-1} N_0^T M_0 \qquad (28)$$

This gives us a very desirable decoupled Riccati equation for k_{10} (see (28)) and k_{30} (see (23)). Also an equation in the output

---

$$k_{20} = [-k_{10}A_{12} + k_{10}B_1 R^{-1} B_2^T k_{30} - A_{21}^T k_{30} - M_1^T M_2] \cdot (A_{22} - B_2 R^{-1} B_2^T k_{30})^{-1} \qquad (29)$$

---

solves for k_{20} in terms of k_{10} and k_{30}. Moreover it is easy to see from the full output of NCProcess that no equations coupling k_{10} and k_{30} exist.

The calculation which computes (28) 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 Groebner 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.

3.1.4 The zero-th order term of the controller : The optimal controller (9) has first term $(i = 0)$ in ϵ equal to

$$G_0 = -R^{-1} \begin{bmatrix} B_1^T & \frac{B_2^T}{\epsilon} \end{bmatrix} \begin{bmatrix} k_{(1,0)} & \epsilon k_{(2,0)} \\ \epsilon k_{(2,0)}^T & \epsilon k_{(3,0)} \end{bmatrix} \qquad (30)$$

and the previous section tells how to compute all of the $k_{j,0}$.

4 Higher Order Terms in ε

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 4.2. The NCAlgebra/NCGB package removes much of the tedium. The next subsection address our computer solution of the standard order ε term, then we turn to our new results on the order ε^2 term.

4.1 The order ϵ term of the Riccati equation

Space does not permit a discussion except to say that the pattern of the solution is the same as before. [KKO86] uses clever notation to make his formulas elegant, but our automatic procedures of course do not do this. However, NCAlgebra and NCGB works quite quickly (7 min) to produce an answer equivalent to the standard one found in [KKO86].

4.2 The order ϵ^2 term of the Riccati equation

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 }}     (31)
```

For this problem a three iteration partial Groebner 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. The algorithms we use to check nonredundancy are more computer intensive and did not finish when run on this problem. Note that our (Riccati and Lyapunov) formulas could be used for numerical calculations to compute $k_{(1,2)}$ and $k_{(3,2)}$; then coupling could be checked on a case by case basis.

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 Groebner 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 successfully to other problems. In particular we mention an ongoing singular perturbation analysis of an important entity in the output feedback H_∞ control problem, the information state.

References

[CLS92] D. Cox, J. Little, and D. O' Shea. Springer, Undergraduate Texts in Mathematics, 1992.

[GHK97] E.L. Green, L.S. Heath, and B.J. Keller. Opal: A system for computing noncommutative gröner bases. In H. Comon, editor, *Eigth International Conf. on Rewriting Techniques and Applications (RTA-97)*, LNCS# 1232, pages 331–334. Springer, 1997.

[HJM98] J. W. Helton, M. R. James, and W. M. McEneaney. *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. To perform the computations in this paper you need NCGB,([HS97]).

[HS97] J.W. Helton and M. Stankus. *NCGB: Noncommutative Groebner bases.* available from http://math.ucsd.edu/~ncalg, 1997.

[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.

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