Computer algebra in the control of singularly perturbed dynamical systems

J.W. Helton* F. Dell Kronewitter †

March 3, 1999

Abstract

Commutative Groebner basis methods have proven to be a useful tool in the manipulation of sets of polynomial equations. The algorithms which fall in this category make up the engines in symbolic algebra packages' `Solve[ ]` commands. Most algebraic calculations which one sees in linear systems theory, for example IEEE TAC, involve block matrices so are highly noncommutative. Thus the more conventional commutative computer algebra does not address them.

The noncommutative Groebner theory is more recent, [Mor86], but many of the symbolic computational algorithms have been implemented, [HWS98], [KFG98]. The noncommutative versions, though not nearly as well understood as their commutative counterparts, have also been shown to be useful in the manipulation of some noncommutative sets of polynomial equations arising in control theory, [HS97a] and [HWS98].

Here we investigate the usefulness of noncommutative computer algebra in a particular area of control theory, singularly perturbed dynamic systems, where the noncommutative polynomials involved are 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 out one step further than has been done previously. This represents the first step in the production of noncommutative computer algebra tools to assist with singular perturbation calculations.

Contents

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

†University of California, San Diego, Math Dept., 9500 Gilman Dr., San Diego, CA 92093-0112, E-mail:dell@ieee.org, Partially supported by the AFOSR and the NSF

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], [PB93], [PB94].

Typically one has an equation with some coefficients depending on a parameter $\frac{1}{\varepsilon}$. One postulates an expansion for the solutions x_ε to the equation,

(a) substitutes x_ε into the equation,

(b) sorts by powers of ε and

(c) solves for successive terms of the expansion x_ε.

The sorting in (b) can be tedious and the business of solving (c) can be very involved. Indeed our method carries out the expansion one step further than has previously been done, see Section 3.3. This article concerns methods we are developing for doing both of 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 can save one engaged in singular perturbation calculations considerable time. Solving complicated systems of equations is always tricky business; thus there is no way of knowing in advance if noncommutative Groebner basis methods will be effective for (reducing to simple form) equations found in large classes of singular perturbation problems. This is the focus of experiments we have been conducting and on which we report in this paper.

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. Then we sketch another newer H^∞ estimation problem called the "cheap sensor" problem (see [HJM98]). On this our computer techniques proved effective and a longer will report these results.

1.1 The idea behind Groebner computer algebra

The Groebner basis algorithm (GBA), due to F. Mora [Mor86], can be used to systematically eliminate variables from a collection (e.g., $\{p_j(x_1,\ldots,x_n)=0 : 1\le j\le k_1\}$) 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 ones 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 $\{q_j=0 : 1\le j\le k_2\}$ equals the set of solutions to the collection of polynomial equations $\{p_j=0 : 1\le j\le k_1\}$. This canonical form greatly

[1] From this ordering on variables one induces an order on monomials in those variables which is referred to as *lexicographic order*.

[2] There need not be $\le$ n equations in this list and there need not be any equation in just one variable.

simplifies the task of solving the collection of polynomial equations by facilitating back-solving for x_j in terms of $x_1, \dots, x_{j-1}$. 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, the Groebner basis is always finite and can be generated by Buchberger's algorithm if one waits long enough. In the noncommutative case, which is the subject of this paper, the Groebner basis is usually infinite and the GBA could fail to halt given infinite computational resources. Nevertheless, the solution set of the output of a terminated (say k iteration) GBA, $\{q_j\}$, is always equivalent to the solution set of the input, $\{p_i\}$, 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, [HS97b], and OPAL, [KFG98].

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

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 performance objective.

2.1 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 $y \in \mathbb{R}^q$.

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 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

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

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

If K is the solution to this optimal state feedback control problem 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 as presented involves solving equation (10) which is a $n+m$ dimensional Riccati. Since we have noticed the partitioned structures present in the system (6) it seems likely that we might decompose the problem and significantly reduce the complexity of the linear algebra 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 Riccatis, one n dimensional, the other m dimensional, and then using 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.*

The noncommutative Groebner computer algebra which is the subject of our investigations is well suited for manipulating polynomials into a desired form. In the next two subsections we review this decomposition. This is mostly a matter of setting our notation, which in fact is the same notation as [KKO86].

2.3 Decomposing the Problem

Here, we decompose our two time scale system into it's fast parts and slow parts by first setting the fast subsystem to it's steady state. In doing so, we will introduce standard notation which will help make the formulas we derive a little more readable.

2.3.1 The Slow System

We will begin our analysis of this perturbed system by considering this system in it's slow state. The dynamics of the slow system can be found by setting ϵ to zero, often referred to as 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

$$\begin{aligned} \frac{dx_s}{dt} &= A_0 x_s(t) + B_0 u_s(t), \qquad x_s(t_0) = x^0 &(14)\\ z_s(t) &= -A_{22}^{-1}(A_{21}x_s(t) + B_2 u_s(t)) &(15) \end{aligned}$$

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

2.3.2 The Fast System

The fast system has dynamics

$$\frac{dz_f}{dt} = A_{22}z_f(t) + B_2 u_f(t), \qquad z_f(t_0) = z^0 - z_s(t_0) \tag{16}$$

where

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

2.4 Decomposing the Output

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

$$\begin{aligned} y &= M_1 x + M_2 z \\ &= M_1[x_s + O(\epsilon)] + M_2[-A_{22}^{-1}(A_{21}x_s + B_2 u_s) + z_f + O(\epsilon)] \\ &= y_s(t) + y_f(t) + O(\epsilon) &(17) \end{aligned}$$

defining

$$y_s = M_0 x_s + N_0 u_s \text{ and } y_f = M_2 z_f \tag{18}$$

where

$$M_0 \triangleq M_1 - M_2 A_{22}^{-1} A_{21} \text{ and } N_0 \triangleq -M_2 A_{22}^{-1} B_2. \tag{19}$$

3 Computer algebra vs. the standard singular perturbation problem

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

$$K_N(\epsilon) = \sum_{i=0}^{N} \epsilon^i \begin{bmatrix} k_{(1,i)} & \epsilon k_{(2,i)} \\ \epsilon k_{(2,i)}^T & \epsilon k_{(3,i)} \end{bmatrix}. \tag{20}$$

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 (coefficients of ϵ^0)

We begin our investigations of the perturbed control problem by searching for the first term of the series (20) consisting of matrices, $k_{(1,0)}$, $k_{(2,0)}$, and $k_{(3,0)}$. We can find an order of ϵ approximation to the optimal K, (20), by taking only the zero-th order terms in ϵ of the equations given in (10). This is problem (b) mentioned in the introduction. In the next section 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.

3.1.1 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, $3x^2yz - yz - 4z^2$. Our computer calls will take either relations or equations. We will slip back and forth between the two notations and our meaning should be clear from the context.

3.1.2 Computer algebra finds the basic equation

Groebner basis theory suggests that 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 occurances of k_{10} we should create a Groebner basis for *all polynomial relations known to hold* under the following order.

$$N_0 < M_0 < R_0 < A_0 < B_0 \ll k_{10} \ll \text{ other variables} \tag{21}$$

Now we will list the input to our computer algebra program which will generate *all polynomial relations known to hold.*

First, we define the block matrices in (11). (What follows is the NCAlgebra notation for the matrices described in (11) and (20). The matrix, $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, is represented as `{{a,b},{c,d}}`. The suffix, `[[`i,j`]]`, extracts the i,j-th element from a matrix. `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}};                                     (22)
```

We also define a K_0.

```
 K0 = {{k10,ep**k20},{ep**tp[k20],ep**k30}};                               (23)
```

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]                       (24)
```

The NCAlgebra command `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 of degree given by the indices. For example, the call

```
 NCTermsOfDegree[ A**B + B**C**C + B**A**C + C**D, {C}, {1} ]              (25)
```

returns

```
 B**A**C + C**D                                                             (26)
```

The following Mathematica commands will extract the 0-th order terms in ϵ, creating the polynomials in (29), (30), and (31).

```
 Ep10 = NCTermsOfDegree[ Riccati[K0][[1,1]] , {ep},{0}]
Ep20 = NCTermsOfDegree[ Riccati[K0][[1,2]] , {ep},{0}]
Ep30 = NCTermsOfDegree[ Riccati[K0][[2,2]] , {ep},{0}]                      (27)
```

3.1.3 The output and it's analysis

Input (27) creates three polynomials, a typical one of which is

```
 k30**A22+tp[A22]**k30+tp[M2]**M2-k30**B2**Inv[R]**tp[B2]**k30.             (28)
```

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

$$\begin{aligned} 0 &= k_{10}\,A_{11} + A_{11}^T\,k_{10} + k_{20}\,A_{21} + A_{21}^T\,k_{20}^T + M_1^T\,M_1 - k_{10}\,B_1 R^{-1}\,B_1^T\,k_{10} - k_{20}\,B_2\,R^{-1}\,B_1^T\,k_{10} \\ &- k_{10}\,B_1 R^{-1}\,B_2^T\,k_{20}^T + k_{20}\,B_2\,R^{-1}\,B_2^T\,k_{20}^T \qquad (29) \\ 0 &= k_{20}\,A_{22} - k_{20}\,B_2\,R^{-1}\,B_2^T\,k_{30} + k_{10}\,A_{12} + A_{21}^T\,k_{30} + M_1^T\,M_2 - k_{10}\,B_1 R^{-1}\,B_2^T\,k_{30} \qquad (30) \\ 0 &= k_{30}\,A_{22} + A_{22}^T\,k_{30} + M_2^T\,M_2 - k_{30}\,B_2\,R^{-1}\,B_2^T\,k_{30} \qquad (31) \end{aligned}$$

Notice that (31), the beautified form of (28), contains only one unknown, k_{30}, and has the Riccati form. Thus one k_{30} is determined by this equation. Equation (29) contains two unknowns, k_{10} and k_{20}, and equation (30) contains all three unknowns, k_{10}, k_{20}, and k_{30}. To solve for k_{20} we use equation (30) 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                                       (32)
```

Upon examination of (32) it is immediate that we may give k_{20} explicitly in terms of k_{10} and k_{30} by assuming the invertibility of the parenthetic expression in the above relation, $A_{22} - B_2 R^{-1} B_2^T k_{30}$. We have

$$k_{20} = \left[-k_{10}A_{12} + k_{10}B_1 R^{-1} B_2^T k_{30} - A_{21}^T k_{30} - M_1^T M_2\right] (A_{22} - B_2 R^{-1} B_2^T k_{30})^{-1}. \qquad (33)$$

Use this definition of k_{20} to change (29) into an equation involving k_{10} and k_{30}. The unknown matrices, k_{i0}, could then be found by first using (31) to solve for k_{30}, then using our transformed (29) to solve for k_{10}, and finally using (33) to obtain k_{20}.

We will show in the next section how (29) may be changed into an equation involving only one unknown, k_{10}, so that k_{30} and k_{10} may be computed concurrently using two independent Riccati equations.

3.1.4 Other algebraic identities which are known to hold

In light of the slow system terminology introduced above in Sections 2.3.1 and 2.4 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 }                                   (34)
```

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]] }     (35)
```

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 $A_{22} - B_2 R^{-1} B_2^T k_{30}$ has already been assumed to be invertible to facilitate the definition of k_{20} in (33). 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) }],                      (36)
```

and combine all of our relations with

```
 Relations = Union[Ep10,Ep20,Ep30,Abbreviations,SelfAdjoints,Inverses].     (37)
```

We must also include the transposes of each of these relations:

```
 AllRelations = NCAddTp[Relations]                        (38)
```

The order mentioned in (21) is specified next.

```
 NCSmartOrder[ {{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 ]                              (39)
```

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

```
 NCProcess[AllRelations,4,"FindK10" ]                     (40)
```

3.1.5 The output

The output of this command is a set of polynomials which make up the the Groebner basis created from the polynomials in `AllRelations` under the order specified in (39). The software we use actually does more than just create a Groebner basis. `NCProcess[ ]` categorizes the output depending on how many unknowns lie in each relation. Then it automatically sets them in TeX and opens a xdvi window displaying them. In this case a category was found for k_{10} which consisted of a single relation, `k10rel`. After calling `NCCollect[ k10rel, k10 ]` this polynomial takes the form

---

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

$$k_{10}\,(A_0 - B_0\,R_0^{-1}\,N_0^T\,M_0) + (A_0^T - M_0^T\,N_0\,R_0^{-1}\,B_0^T\,)\,k_{10} + M_0^T\,M_0 - k_{10}\,B_0\,R_0^{-1}\,B_0^T\,k_{10} - M_0^T\,N_0\,R_0^{-1}\,N_0^T\,M_0 \qquad (41)$$

This calculation is no easy feat by hand, as the substitutions and non-standard notation on pages 116-7 of [KKO86] will attest. The answer we found with the Groebner computer algebra is the same as derived there by hand. This computation took less than 3 minutes.

3.1.6 The zero-th order term of the controller

Closer analysis of the first term of the optimal controller discovered in the previous section, Section 3.1,

$$G = -R^{-1} \begin{bmatrix} B_1^T & \frac{B_2^T}{\epsilon} \end{bmatrix} \begin{bmatrix} k_{(1,0)} & \epsilon k_{(2,0)} \\ \epsilon k_{(2,0)}^T & \epsilon k_{(3,0)} \end{bmatrix}, \tag{42}$$

reveals that an ϵ independent controller can be obtained by setting the upper right entry of K to zero. This gives us

$$G \begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} G_{10} & G_{20} \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = -R^{-1}(B_1^T k_{(1,0)} x + B_2^T k_{(2,0))}^T x + B_2^T k_{(3,0)} z) \tag{43}$$

where $k_{(i,0)}$ is defined by equations (41), (33), and (31) for i equal to 1,2, and 3 respectively.

3.2 The order ϵ term of the Riccati equation

In Section 3.1.6, a controller was presented which does not depend on the parameter ϵ. This is especially appropriate if ϵ represents some small unknown parameter. In fact, there are many circumstances when the parameter, ϵ, 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 3.3.

As done in [KKO86] we will now obtain formulas for the matrices $k_{(1,1)}$, $k_{(2,1)}$, and $k_{(3,1)}$ described in (20). Our approach will require considerably less work.

Instead of truncating the series (20) to only one term as done in Section 3.1.6, input (22), 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 }}                    (44)
```

We also append the following abbreviations for the controller discussed above in Section 3.1.6 and defined in equation (43). These formulas are standard, [KKO86].

```
Abbreviations = Union[ Abbreviations, {
G10 == -Inv[R]**(tp[B1]**k10 + tp[B2]**tp[k20]),
G20 == -Inv[R]**tp[B2]**k30 } ]                                        (45)
```

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

```
Inverses = Union[ Inverses, NCMakeRelations[{Inv, (A22 + B2**G20)}] ].    (46)
```

3.2.1 Extracting the coefficients of ϵ^1

The Riccati expression in K_1, `Riccati[K1]`, generates equations quadratic in ϵ. As done in the last section the approach here is to equate coefficients of ϵ. Of course, the coefficients of ϵ^2 cannot be equated since the actual power series (20) would have $k_{(i,2)}$ which have not been introduced in computer input (44). We can extract the coefficients of ϵ in equation (10) with the following commands.

```
Ep11 = NCTermsOfDegree[ Riccati[K1][[1,1]] , {ep},{1}]                  (47)
```

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]          (48)
```

and

```
Ep21 = NCTermsOfDegree[ Riccati[K1][[1,2]] , {ep},{1}]          (49)
```

```
Ep31 = NCTermsOfDegree[ Riccati[K1][[2,2]] , {ep},{1}]          (50)
```

give similar looking formulas.

3.2.2 Solving for the unknowns

These valid relations can now be added to *all relations known to hold*, (22), (24), (34), (35), and (36), with the following command. Since the output of the `NCTermsOfDegree[ ]` command includes the variable, `ep`, which we took the coefficients of we must set the unwanted variable, `ep`, to 1.[3] We do this by appending the Mathematica *rule* `/.ep->1` to expressions involving `ep`.

```
AllRelations = Union[Ep11/.ep->1,Ep21/.ep->1,Ep31/.ep->1,
Ep10,Ep20,Ep30, Abbreviations,SelfAdjoints,Inverses]          (51)
```

Considering the analysis done in Section 3.1.2, $k_{(1,0)}$, $k_{(2,0)}$, and $k_{(3,0)}$ (`k10, k20, k30`) can be regarded as known and we are now looking for formulas which describe the second term of the series (20), made up of symbols `k11, k21`, and `k31` introduced above, (44).

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

$$\begin{aligned} &N_0 < M_0 < R < A_0 < B_0 < B_1 < B_2 < M_1 < M_2 < R_0 < \\ &A_{11} < A_{12} < A_{21} < A_{22} < G_{10} < G_{20} < k_{30} < k_{20} < k_{10} \ll k_{11} \ll \text{ other variables} \end{aligned} \tag{52}$$

This order is imposed with the command

```
NCSmartOrder[ {{N0,M0,R,A0,B0,
A11,A12,A21,A22,B1,B2,M1,M2,R0,G10,G20,k30,k20,k10},
{k11},{k31, k21},{
Inv[A22- B2**Inv[R]**tp[B2]**k30],tp[Inv[A22- B2**Inv[R]**tp[B2]**k30]]
}}, AllRelations ];          (53)
```

A three iteration Groebner basis is created with the `NCProcess[ ]` command similar to (40). On 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 ]          (54)
```

gives us the following relation

$$\begin{aligned} &-1\, k_{11}\,(A_0 - B_0\,R_0^{-1}\,B_0^T\,k_{01} - B_0\,R_0^{-1}\,N_0^T\,M_0) + (k_{01}\,B_0\,R_0^{-1}\,B_0^T + M_0^T\,N_0\,R_0^{-1}\,B_0^T - A_0^T)\,k_{11} + A_0^T\,k_{02}\,A_{22}^{-1}\,A_{21} + \\ &A_{21}^T\,A_{22}^{T-1}\,k_{02}^T\,A_0 - k_{01}\,B_0\,R_0^{-1}\,B_0^T\,k_{02}\,A_{22}^{-1}\,A_{21} - k_{01}\,B_0\,R_0^{-1}\,B_2^T\,A_{22}^{T-1}\,k_{02}^T\,A_0 - A_0^T\,k_{02}\,A_{22}^{-1}\,B_2\,R_0^{-1}\,B_0^T\,k_{01} - \\ &A_0^T\,k_{02}\,A_{22}^{-1}\,B_2\,R_0^{-1}\,N_0^T\,M_0 - A_{21}^T\,A_{22}^{T-1}\,k_{02}^T\,B_0\,R_0^{-1}\,B_0^T\,k_{01} - A_{21}^T\,A_{22}^{T-1}\,k_{02}^T\,B_0\,R_0^{-1}\,N_0^T\,M_0 - \\ &M_0^T\,N_0\,R_0^{-1}\,B_0^T\,k_{02}\,A_{22}^{-1}\,A_{21} - M_0^T\,N_0\,R_0^{-1}\,B_2^T\,A_{22}^{T-1}\,k_{02}^T\,A_0 + k_{01}\,B_0\,R_0^{-1}\,B_0^T\,k_{02}\,A_{22}^{-1}\,B_2\,R_0^{-1}\,B_0^T\,k_{01} + \\ &k_{01}\,B_0\,R_0^{-1}\,B_0^T\,k_{02}\,A_{22}^{-1}\,B_2\,R_0^{-1}\,N_0^T\,M_0 + k_{01}\,B_0\,R_0^{-1}\,B_2^T\,A_{22}^{T-1}\,k_{02}^T\,B_0\,R_0^{-1}\,B_0^T\,k_{01} + \\ &k_{01}\,B_0\,R_0^{-1}\,B_2^T\,A_{22}^{T-1}\,k_{02}^T\,B_0\,R_0^{-1}\,N_0^T\,M_0 + M_0^T\,N_0\,R_0^{-1}\,B_0^T\,k_{02}\,A_{22}^{-1}\,B_2\,R_0^{-1}\,B_0^T\,k_{01} + \\ &M_0^T\,N_0\,R_0^{-1}\,B_0^T\,k_{02}\,A_{22}^{-1}\,B_2\,R_0^{-1}\,N_0^T\,M_0 + M_0^T\,N_0\,R_0^{-1}\,B_2^T\,A_{22}^{T-1}\,k_{02}^T\,B_0\,R_0^{-1}\,B_0^T\,k_{01} + \\ &M_0^T\,N_0\,R_0^{-1}\,B_2^T\,A_{22}^{T-1}\,k_{02}^T\,B_0\,R_0^{-1}\,N_0^T\,M_0 \end{aligned} \tag{55}$$

The coefficients of $k_{(1,1)}$ in equation (55), the relations in parentheses, suggest that we make the following abbreviation

[3] Notice that Mathematica output (26) contains the variable C.

```
F0 == A0-B0**Inv[R0]**(tp[N0]**M0 + tp[B0]**k01)
```
[4] (56)

With computer input (56) appended to *all relations known to hold*, `AllRelations`, we can put F_0 low in the order and run `NCProcess[ ]` again, creating a new Groebner basis. The output of this command contains the aesthetically pleasing relation defining $k_{(1,1)}$

$$-k_{11}\,F_0 \quad - \quad 1\,F_0^T\,k_{11} \quad + \quad A_{21}^T\,(A_{22}+B_2\,G_{20})^{-T}\,k_{20}^T\,F_0 \quad + \quad F_0^T\,k_{20}\,(A_{22}+B_2\,G_{20})^{-1}\,A_{21} \quad + \quad F_0^T\,k_{20}\,(A_{22}+B_2\,G_{20})^{-1}\,B_2\,G_{10}+G_{10}^T\,B_2^T\,(A_{22}+B_2\,G_{20})^{-T}\,k_{20}^T\,F_0. \tag{57}$$

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 (58) and takes the collected form

$$k_{31}\,\left(A_{22}-B_2\,R^{-1}\,B_2^T\,k_{30}\right) \;+\; \left(A_{22}^T-k_{30}\,B_2\,R^{-1}\,B_2^T\,\right)k_{31} \;+\; A_{12}^T\,k_{20} \;+\; k_{02}^T\,A_{12} \;-\; k_{30}\,B_2\,R^{-1}\,B_1^T\,k_{20} \;-\; k_{20}^T\,B_1\,R^{-1}\,B_2^T\,k_{30} \tag{58}$$

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 (57).

$$k_{21}\rightarrow -1\,k_{11}\,A_{12}\,A_{22}^{-1}-A_{11}^T\,k_{20}\,(A_{22}+B_2\,G_{20})^{-1}-A_{21}^T\,k_{31}\,(A_{22}+B_2\,G_{20})^{-1}-G_{10}^T\,B_1^T\,k_{20}\,(A_{22}+B_2\,G_{20})^{-1}-G_{10}^T\,B_2^T\,k_{31}\,(A_{22}+B_2\,G_{20})^{-1}+k_{11}\,B_0\,R_0^{-1}\,\left(N_0^T\,M_2\,A_{22}^{-1}-B_2^T\,A_{22}^{T-1}\,k_{30}\right) \tag{59}$$

Expressions equivalent to (57), (59), and (58) can be found in [KKO86].

Notice that a similar procedure could be done as that described in Section 3.1.6 to derive an order ϵ controller.

3.3 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 }},
```
(60)

For this problem a three iteration Groebner basis was created and we arrived at formulas defining $k_{(1,2)}$, $k_{(2,2)}$, and $k_{(3,2)}$ which in total are 1,478 lines long in Mathematica notation (inefficient).

We used a version of `NCProcess[ ]` which was specialized to display only equations involving the unknowns; $k_{(1,2)}$ and $k_{(2,2)}$. (The formula for $k_{(3,2)}$ was immediate from the order two Riccati as in the lower order cases.) This substantially speeds up run times. Still, our rather formidable conclusion took 21 and a half 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.

[4] More analytic methods can be used, [KS72], to show that this expression 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, (14). The focus here is on the computer algebra and the expression, F_0, is discovered purely algebraically.

4 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].

4.1 The General Problem

Consider the system

$$\begin{aligned} \frac{dx}{dt} &= A^\times(x) + B(x)v & (61)\\ out &= C(x) + D(x). & (62) \end{aligned}$$

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

$$\begin{aligned} -\frac{dp}{dt} &= (A^\times(x) + B(x)\cdot v(t))^T \nabla_x p_t(x) + (\nabla_x p_t(x))^T Q(x) \nabla_x p_t(x) & (63)\\ &+ [C(x) - Dv(t)]^T J[C(x) - Dv(t)] + \frac{1}{\epsilon^2}\|W[v(t) - x]\|^2, \end{aligned}$$

where J is self adjoint. It is written even more concisely as

$$\begin{aligned} -\frac{dp}{dt} &= \alpha(x,t)\nabla_x p_t(x) + (\nabla_x p_t(x))^T Q(x)\nabla_x p_t(x) & (64)\\ &+ J(x,t) + \frac{1}{\epsilon^2}\|W[v(t) - x]\|^2. \end{aligned}$$

4.2 The linear case

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

$$p_t(x) = \frac{1}{2}(x - x_\epsilon)^T P_\epsilon (x - x_\epsilon) + \phi_t^e, \tag{65}$$

where *P_ϵ is a matrix which does not depend on t, but x and x_ϵ do depend on t*, or more simply as

$$p_t(x) = \frac{1}{2}\tilde{x}^T P_\epsilon \tilde{x} + \phi_t^e \text{ with } \tilde{x} = x - x_\epsilon \tag{66}$$

We expand x_ϵ to arrive at

$$x_\epsilon = x_{e0} + \epsilon x_{\epsilon,1} + \epsilon^2 x_{\epsilon,2} + \dots \tag{67}$$

A few examples have led us to believe that P_ϵ is an irrational function of ϵ which has a series form

$$P_\epsilon = \frac{1}{\epsilon}P_{-1} + P_0 + \epsilon P_1 + \epsilon^2 P_2 + \dots \tag{68}$$

Then

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

and noting that $(A^\times x + Bv(t))^T \nabla p$ is scalar, we can symmetrize the above information state equation, ISE, (63), and arrive at

$$\begin{aligned} -\frac{dp}{dt} &= [A^\times x + B\cdot v(t)]^T P_\epsilon (x - x_\epsilon) + (x - x_\epsilon)^T P_\epsilon [A^\times x + B\cdot v(t)] & (69)\\ &+ (x - x_\epsilon)^T P_\epsilon Q P_\epsilon (x - x_\epsilon) + [Cx - Dv(t)]^T J[Cx - Dv(t)] + \frac{1}{\epsilon^2}(v(t) - x)^T R(v(t) - x). \end{aligned}$$

where $R = W^T W$.

4.3 Computer algebra works

We applied NCAlgebra methods very similar to the ones demonstrated in Section 3 to the ISE singular perturbation problem just described and found them highly effective. Results will be reported in the longer paper corresponding to this conference note.

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.

[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.

[HS97a] J.W. Helton and M. Stankus. Computer assistance in discovering formulas and theorems in system engineering. *to appear in Journal of Functional Analysis*, 1997.

[HS97b] 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.

[HWS98] J.W. Helton, J.J. Wavrik, and M. Stankus. Computer simplification of formulas in linear systems theory. *IEEE Transactions on Automatic Control*, 43:302–14, 1998.

[KFG98] B. Keller, C.D. Feustel, and E. Green. *The GRB/OPAL System.* available from http://csgrad.cs.vt.edu/~keller, 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.