Noncommutative computer algebra in the control of singularly perturbed dynamical systems

J.W. Helton1  F. Dell Kronewitter1  Mark Stankus2

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, [KK086], 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 $H_{\infty}$ control problems, for example [KKO86] and the works of Pan and Basar.

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

(a) substitutes $x_\varepsilon$ into the equation,
(b) sorts the equation according to powers of $\varepsilon$, and
(c) finds simple equations for successive terms in the expansion of $x_\varepsilon$.

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 $H_{\infty}$ estimation problem called the “cheap sensor” problem (see [HJM98]). On this our computer techniques proved effective and a longer paper will report these results.
1.1 The idea behind Groebner computer algebra

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

The noncommutative Groebner basis algorithm (GBA), due to F. Mora [Mor86], can be used to systematically eliminate variables from a collection (e.g., \( \{p_i(x_1, \ldots, x_n) = 0 : 1 \leq i \leq k_1 \} \)) of polynomial equations so as to put it in triangular form. One specifies an order on the variables (\( z_1 < z_2 < z_3 < \ldots < z_n \)) \(^1\) which corresponds to your priorities in eliminating them. Here a GBA will try hardest to eliminate \( z_n \) and try the least to eliminate \( z_1 \). The output from it is a list of equations in a “canonical form” which is triangular: \(^2\)

\[
\begin{align*}
q_1(x_1) &= 0 \\
q_2(x_1, x_2) &= 0 \\
q_3(x_1, x_2) &= 0 \\
& \vdots \\
q_k(x_1, \ldots, x_n) &= 0.
\end{align*}
\]

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 \( \{p_i = 0 : 1 \leq i \leq k_1 \} \). This canonical form greatly simplifies the task of solving the collection of polynomial equations by facilitating back-solving for \( x_k \) in terms of \( x_1, \ldots, x_{k-1} \). 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 NCAIgebra, [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.

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 \( \varepsilon \); 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]:

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 \\
\vdots \\
\dot{x}_n
\end{bmatrix} = \begin{bmatrix}
A_{11} & A_{12} & 0 & 0 \\
A_{21} & A_{22} & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix} + \begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_n
\end{bmatrix} u
\]

(6)

\[
y = \begin{bmatrix}
M_1 \\
M_2 \\
\vdots \\
M_n
\end{bmatrix} \frac{\varepsilon}{\varepsilon} \begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix}
\]

(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 \( A_{11}, A_{12}, A_{21}, A_{22}, B_1, B_2, M_1, M_2 \) 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

\[
J = \int_0^\infty (y^T y + u^T R u) dt
\]

(8)

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

\[
u^* = -R^{-1} B^T K(\varepsilon) \begin{bmatrix}
x \\
z
\end{bmatrix} = G(\varepsilon) \begin{bmatrix}
x \\
z
\end{bmatrix}
\]

(9)

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

\[
KA + A^T K - KBR^{-1} B^T K + M^T M = 0
\]

(10)

with

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
\varepsilon A_{11} & \varepsilon A_{12}
\end{bmatrix},
B = \begin{bmatrix}
B_1 \\
B_2
\end{bmatrix}
\]

and

\[
M = \begin{bmatrix}
M_1 \\
M_2
\end{bmatrix}
\]

\[
K(\varepsilon) = K_0 + \varepsilon K_1 + \varepsilon^2 K_2 + \ldots
\]

(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 its 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 to zero in equation (6) obtaining what is often called the quasi-steady state of the system. This transforms the equation involving in system (6) into an algebraic equation rather than a differential equation

$$\frac{dz_s}{dt} = A_0 z_s(t) + B_0 u_s(t), \quad z_s(t) = \varepsilon^0$$

where

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

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

$$\frac{dz_f}{dt} = A_{22} z_f(t) + B_{2f} u_f(t), \quad z_f(t) = \varepsilon^0 - z_s(t)$$

where $z_f = z - z_s$ and $u_f = u - u_s$.

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 its slow and fast parts

$$y = M_1 x + M_2 z$$

$$= M_1 [x_s + O(\varepsilon)] + M_2 [-A_{21}^t (A_{21} z_s + B_2 u_s) + z_f + O(\varepsilon)]$$

$$= y_s(t) + y_f(t) + O(\varepsilon)$$

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

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

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 NCAIgebra.

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

3.1 The zero-th order term of the Riccati equation (constant (i.e., $\varepsilon^0$) 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 $K_0(\varepsilon)$ into (10) and take only the zero-th order terms in $\varepsilon$ 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 $\varepsilon$ of the equations given in (10).

First, using NCAIgebra, 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 NCAIgebra, MatMult performs the matrix multiplication operation, tpMat performs the symbolic transpose operation, and ** indicates noncommutative multiplication.
We also define a $K_0$.

$$K_0 = \{ (k_{10}, ep \cdot k_{20}), (ep \cdot tp[k_{20}], ep \cdot k_{30}) \}$$

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

$$\text{Riccati}[K] := \text{MatMult}[X, A] + \text{MatMult}[\text{tpMat}[A], X]$$

$$\text{MatMult}[X, B, \text{Inv}[A] \cdot \text{tpHatC}] = K_1$$

We next use the NCAlgebra command $\text{NCTermsOfDegree}$\(^3\). The following Mathematica commands will extract the 0-th order terms in $\epsilon$, creating the polynomials in (21), (22), and (23). \(^4\)

$$\text{Ep0} = \text{NCTermsOfDegree}[\text{Riccati}[K_0][1,1], \{ep\}, \{0\}]$$

$$\text{Ep20} = \text{NCTermsOfDegree}[\text{Riccati}[K_0][2,2], \{ep\}, \{0\}]$$

$$\text{Ep30} = \text{NCTermsOfDegree}[\text{Riccati}[K_0][2,2], \{ep\}, \{0\}]$$

The output

This input creates three polynomials, the third of which is

$$k_{30} A_{22} + tp[A_{22}] + k_{30} + tp[M_1] + k_{30} + k_{20} A_{21} + A_{21}^2 k_{30}$$

When all three output are in $\LaTeX$, which is done easily by NCAlgebra, we get that $\text{Riccati}[K_0] = 0$ corresponds to the equations

$$0 = k_{10} A_{11} + A_{11}^2 k_{10} + k_{20} A_{21} + A_{21}^2 k_{20} + M_1^2 k_{10} - k_{10} B_{11} R^{-1} B_1^2 k_{10} - k_{20} B_2 R^{-1} B_2^2 k_{10} - k_{10} B_{11} R^{-1} B_1^2 k_{10} + k_{30} A_{22} + k_{30} B_2 R^{-1} k_{30} + k_{10} A_{21} + A_{21}^2 k_{30} + M_2^2 M_2 - k_{10} B_{11} R^{-1} B_1^2 k_{30}$$

Notice that (23), the $\LaTeX$ 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, $A_{22} - B_2 R^{-1} B_2^2 k_{30}$ has all eigenvalues in the strict left half plane and is therefore invertible. For computer algebra the key property is invertibility of $A_{22} - B_2 R^{-1} B_2^2 k_{30}$. 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.

3.1.2 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, $x_1^{a_1} \cdot x_2^{a_2} \cdot \ldots \cdot x_n^{a_n}$, where $a_k \in \mathbb{N}$, is called a monomial. A polynomial, $f$, is a finite $\mathbb{C}$-linear combination of monomials,

$$\sum_{j=1}^m a_j x_1^{a_1} \cdot x_2^{a_2} \cdot \ldots \cdot x_n^{a_n}$$

where $a_j \in \mathbb{C}$. We call $a_k$ the coefficient of the term $x_1^{a_1} \cdot x_2^{a_2} \cdot \ldots \cdot x_n^{a_n}$. A relation is a polynomial which is assumed to be 0. That is, we may write the equation, $3z^2yz = yz + 4z^2$, as a relation, $3z^2yz - yz - 4z^2$. 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 relations in the input. Now we will list the 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, $\text{Ep10}, \text{Ep20}$ and $\text{Ep30}$; 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.

$$\text{Abbreviations} = \{ A_0 = -M_2 \cdot \text{Inv}[A_{22}] \cdot B_2, M_0 = M_1 - M_2 \cdot \text{Inv}[A_{22}] \cdot A_{21}, A_0 = A_{11} - A_{12} \cdot \text{Inv}[A_{22}] \cdot A_{21}, B_0 = B_1 - A_{12} \cdot \text{Inv}[A_{22}] \cdot B_2, R_0 = R + \text{tp}[M_0] \cdot B_2 \}$$

\(^{3}\)\text{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 $\text{NCTermsOfDegree}[A + B \cdot C \cdot D + E \cdot F + G + H, \{C,\{1\}\}]$ returns $B \cdot E \cdot F + C \cdot D$.\(^{4}\)
We add the abbreviation $R_0$ for convenience as done in [KKO86], although it is not essential.

Inv[R] is the NCAIgebra 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

\[
\]

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”.

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”.

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

\[\text{NCProcess}[\text{AllRelations}, 4, \text{"FindK10"}, \text{SBBbyCat} \rightarrow \text{False}]
\]

The optimal controller (9) has first term $i = 0$ in $c$ equal to

\[G_0 = -R^{-1} \begin{bmatrix} B_1^T & B_1^T & \cdots \end{bmatrix} \begin{bmatrix} k_{i,0}^{(1,0)} & k_{i,0}^{(2,0)} & \cdots \end{bmatrix}
\]

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

3.1.3 The output: The output of this command is a set of polynomials which make up 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. NCPProcess categorizes the output depending on how many and which unknowns lie in each relation. Then it automatically sets them in TPX, TPX’s 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

\[k_{20} = -k_{10} A_{12} + k_{10} B_1 R^{-1} B_2^T k_{30} - A_2^T k_{30} - M_4^T M_5 \] (29)

solves for $k_{20}$ in terms of $k_{10}$ and $k_{30}$. Moreover it is easy to see from the full output of NCPProcess 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 $c$ equal to

\[G_0 = -R^{-1} \begin{bmatrix} B_1^T & B_1^T & \cdots \end{bmatrix} \begin{bmatrix} k_{i,0}^{(1,0)} & k_{i,0}^{(2,0)} & \cdots \end{bmatrix}
\] (30)
4 Higher Order Terms in $\varepsilon$

There are many circumstances when the parameter $\varepsilon$, while small, is known. In such a case, even though the optimal controller is an infinite power series in $\varepsilon$ one can make an $n^{th}$ order approximation to $G(\varepsilon)$ 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 $\varepsilon$. We did not find references where anyone generated formulas for coefficients of $\varepsilon$ higher than 1. The methods in this paper do, see Section 4.2. The NCA/NCGB package removes much of the tedium. The next subsection address our computer solution of the standard order $\varepsilon^2$ term, then we turn to our new results on the order $\varepsilon^3$ term.

4.1 The order $\varepsilon$ 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, NCA/NCGB works quite quickly (7 min) to produce an answer equivalent to the standard one found in [KKO86].

4.2 The order $\varepsilon^2$ term of the Riccati equation

For the sake of presenting a formula which has not appeared before we create a three term approximation to $K(\varepsilon)$,

$$K_2 = \{(k_{10},ep\,k_{20}), (ep\,tp[k_{20}], ep\,k_{30}) \} +$$

$$\{ (k_{11}, ep\,k_{21}), (ep\,tp[k_{21}], ep\,k_{31}) \} +$$

$$\{ (k_{21}, ep\,k_{22}), (ep\,tp[k_{22}], ep\,k_{32}) \}$$

(31)

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

We found one equation which expresses $k_{3(2),2}$ in terms of $k_{1(2),2}$ and $k_{3(2),2}$. We found a Lyapunov equation in the unknown $k_{1(2),2}$ consisting of 150 lines in Mathematica notation. There were also several equations in the unknowns $k_{3(2),2}$ and $k_{3(2),2}$. 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 $k_{1(2),2}$ and $k_{3(2),2}$; 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; $k_{1(2),2}$ and $k_{3(2),2}$. 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 $H_\infty$ control problem, the information state.

References


