Computing Dynamic Output Feedback Laws

Jan Verschelde and Yusong Wang

***Abstract*— The pole placement problem asks to find laws to feed the output of a plant governed by a linear system of differential equations back to the input of the plant so that the resulting closed-loop system has a desired set of eigenvalues. Converting this problem into a question of enumerative geometry, efficient numerical homotopy algorithms to solve this problem for general Multi-Input-Multi-Output (MIMO) systems have been proposed recently. Despite the wider application range of dynamic feedback laws, the realization of the output of the numerical homotopies as a machine to control the plant in the time domain has not been addressed before. In this paper we present symbolic-numeric algorithms to turn the solution to the question of enumerative geometry into a useful control feedback machine. We report on numerical experiments with our publicly available software PHCpack and illustrate its application on various control problems from the literature.**

2000 Mathematics Subject Classification. Primary 93B55. Secondary 14Q99, 65H10, 68W30, 93B27.

***Index Terms*— Approximate GCD, control of linear systems, dynamic output feedback, Multi-Input-Multi-Output (MIMO) systems, numerical homotopy algorithms, numerical Schubert calculus, pole placement, Smith normal form, symbolic-numeric computations.**

I. Introduction

Given a linear system of differential equations and a list of eigenvalues, the pole placement problem asks to find laws to feed the output back to the input so that the resulting closed-loop system has the same eigenvalues as the given list. While executing a static feedback law corresponds to a simple matrix-vector multiplication, a dynamic compensator has several internal states.

The theoretical solution of this problem was found in the Schubert calculus, see [3], [4] for the static, and [25], [26], [27] for dynamic compensators. Because of its importance to practical applications, the development of algorithms for this problem was stated as an open problem [29] (see also [5]). See [18] for the relation with inverse eigenvalue and matrix extension problems. The first homotopy algorithms defining a numerical Schubert calculus were proposed in [14]. The Pieri homotopy algorithms of [14] were improved and generalized to dynamic feedback in [15]. In [20], the numerical performance of these homotopies for static feedback was improved.

In [36] we applied static output feedback to use pole placement to keep a satellite in orbit. In this sequel to [36], we consider the application of dynamic feedback laws to this and other applications. As expected, with the more general dynamic feedback laws we can cover a wider range of applications, and turn overdetermined problems into fully determined or underdetermined problems. We define this conversion in the next section.

The motivation for this paper is the realization of dynamic compensators, computed in the frequency domain. By realization we mean the description of the controllers in the time domain. Our data is approximate, i.e.: known only with limited accuracy and subject to roundoff. While realization algorithms are standard (e.g., [1] and [16]), reports on numerical implementations of algorithms which manipulate polynomial matrices are scarce. We found only one numerical study [12] on a commercial implementation, which leaves a numerical Smith normal form as an open question. Nevertheless, the Smith normal form is important for a minimal realization of the transfer function of Multi-Input-Multi-Output (MIMO) systems.

In section three we describe how to compute the greatest common divisor (GCD) of two polynomials with approximate coefficients, as this is important in a numerical Smith normal form. As we experienced, the naive application of the Euclidean algorithm can fail. That many algorithms to solve problems with exact data are numerically unstable is a growing concern in computer algebra, and has led to hybrid symbolic-numeric computation [6]. The approximate GCD was studied in [2], [8], [13], [17], [24], [38] and [39].

We consider the input polynomials not symbolically as polynomials with approximate coefficients, but geometrically as polynomials defined by approximate complex roots. The idea to computing the GCD by matching common approximate roots (within a certain tolerance) can be found in [24]. Compared to alternative methods for approximate GCDs, this approach works well for input polynomials with multiple zeros. We apply the method of Weierstrass (also called the method of Durand-Kerner, see [23]) to find roots of polynomials. To solve the extended GCD problem, we apply Newton interpolation.

In section four we describe an extension to the publicly available software package PHCpack [35] before illustrating its application.

II. Problem Statement

The algorithmic framework is defined by the transitions between the time and the frequency domain. In the time domain, the control of a plant with a dynamic feedback law is shown in Figure 1.

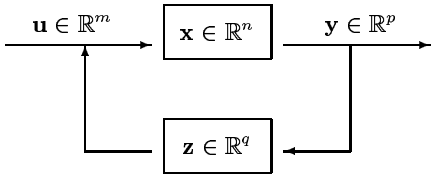

Fig. 1. Control of an m-input and p-output plant by a qth order dynamic compensator in the time domain.

The machine we want to control is given by three matrices (A, B, C), and defines a system of first-order differential equations in the time domain. The input of the Pieri homotopies is sampled in the frequency domain. So to apply these homotopies, we need to translate the input from the time to the frequency domain and to realize the output as a tuple of matrices. Table I defines the three stages in the data flow, visualized in Figure 2

(1)	With plain linear algebra we compute the input for the Pieri homotopies, sampling points from the plant we wish to control.
(2)	Pieri homotopies compute solution maps of degree q, which are transformed into m-by-p polynomial matrices.
(3)	Given the transfer function of the compensator, we realize the compensator by a tuple of four matrices (F, G, H, K).

TABLE I
Transitions in applying the Pieri homotopies.

Our problem is thus to process the output of the Pieri homotopy algorithms and to apply realization algorithms to the numerical output. In [15], the following equivalence relation was derived:

$$\det\left(s\begin{bmatrix} I_n & 0 \\ 0 & I_q \end{bmatrix} - \begin{bmatrix} A+BKC & BH \\ GC & F \end{bmatrix}\right) = 0 \qquad (1)$$

$$\Leftrightarrow \quad \det\begin{bmatrix} I_p & C(sI_n - A)^{-1}B \\ H(sI_q - F)^{-1}G + K & I_m \end{bmatrix} = 0 \qquad (2)$$

This material is based upon work supported by the National Science Foundation under Grant No. 0105739 and Grant No. 0134611. The authors address is Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA. *Email:* jan@math.uic.edu, ywang25@uic.edu. *URL:* http://www.math.uic.edu/~jan.

Equation (1) is the characteristic equation of the closed-loop system. Via elementary row and column operations, this equation can be rewritten into (2), which separates the given data (A, B, C) from the unknown (F, G, H, K). Equation (2) shows the geometric problem: we are looking for curves which produce p-planes in $\mathbb{C}^{m+p}$ which meet given m-planes sampled at prescribed values for s. In Figure 2 we show the transition between the time and the frequency domain.

$$\begin{array}{ccc} \left\{ \begin{array}{l} \dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \\ \mathbf{y} = C\mathbf{x} \end{array} \right. & & \left\{ \begin{array}{l} \dot{\mathbf{z}} = F\mathbf{z} + G\mathbf{y} \\ \mathbf{u} = H\mathbf{z} + K\mathbf{y} \end{array} \right. \\ \downarrow (1) & & \uparrow (3) \\ \left[\begin{array}{c} C(sI_n - A)^{-1}B \\ I_m \end{array} \right] & \xrightarrow{(2)} & \left[\begin{array}{c} I_p \\ H(sI_q - F)^{-1}G + K \end{array} \right] \end{array}$$

Fig. 2. Transitions between time and frequency domain, as defined in Table I.

When controlling a machine with n internal states with a controller using q internal states, we can place $n + q$ poles. But the dimension of the geometric problem is $mp + q(m + p)$. For generic input data, the degree of the solution set to this problem depends solely on m, p, and q and is denoted by $d(m, p, q)$, see [27]. Depending on values for n, m, p, and q, we distinguish three cases:

1) $n + q < mp + q(m + p)$ underdetermined: For a generic machine, there is a set of feedback laws. The set has dimension $mp + q(m + p) - n - q$, and has degree $d(m, p, q)$, i.e: for a generic choice of the parameters, we have $d(m, p, q)$ complex feedback laws.

2) $n + q = mp + q(m + p)$ dimension zero: For a generic machine, there are exactly $d(m, p, q)$ **complex** feedback laws. Every feedback law places all $n + q$ poles at the desired locations. It may be that no feedback law has all its coefficients **real**, see [9], [10].

3) $n + q > mp + q(m + p)$ overdetermined: For a generic machine, there are no feedback laws which place all $n + q$ poles at the desired locations.

The numbers m, p, n are fixed, given on input. We can choose q to arrive always in a favorable condition and get feedback laws, as $q(m + p)$ grows faster than q. For example, a static feedback law may not exists when $n > mp$, but we can find a large enough q and compute dynamic feedback laws.

Presently, we resolve the undetermined case by choosing additional input planes to the geometric problem. Recent advances with homotopies (see e.g. [30], [31], [32], [33]) allow to treat positive dimensional solution sets.

III. Symbolic-Numeric Calculations

To execute stage (2) in Figure 2, we need to calculate the Smith normal form to compute the inverse of a matrix with polynomial entries. More precisely, the output of the homotopies is an $(m + p)$-by-p matrix of polynomials in s:

$$\left[\begin{array}{c} U(s) \\ V(s) \end{array} \right], \tag{3}$$

where $U(s)$ is a p-by-p matrix and $V(s)$ is an m-by-p matrix of polynomials in s, satisfies

$$\det \left[\begin{array}{cc} U(s) & C(sI_n - A)^{-1}B \\ V(s) & I_m \end{array} \right] = 0, \tag{4}$$

for the given poles. We can right multiply (4) by

$$\left[\begin{array}{cc} U^{-1}(s) & 0 \\ 0 & I_m \end{array} \right]. \tag{5}$$

The result of this multiplication is

$$\det \left[\begin{array}{cc} I_p & C(sI_n - A)^{-1}B \\ V(s)U^{-1}(s) & I_m \end{array} \right] = 0. \tag{6}$$

Since the multiplier matrix (5) is of full rank, its determinant is nonzero and the original intersection condition remains. This multiplication does not affect the input conditions, which are at the right part of (2). By comparing (2) with (6), we can apply the realization algorithms to extract (F, G, H, K) from the matrix $V(s)U^{-1}(s)$.

In the next subsection we show how the calculation of a Smith normal form requires the calculation of greatest common divisor.

A. Numerical Smith Normal Form

For any n-by-m matrix $A(s)$ whose entries are polynomials in s, there exist a unimodular matrices $P(s)$ and $Q(s)$ so that

$$P(s)A(s)Q(s) = D(s), \tag{7}$$

where $D(s)$ is an n-by-m matrix which has only nonzero polynomials in s on its diagonal. Furthermore, denoting the ith element on the diagonal of $D(s)$ by D_i, we have that D_i divides D_{i+1}. The matrix $D(s)$ is called the Smith normal form of $A(s)$. Since unimodular matrices are invertible, we can rewrite $A(s)$ as

$$A(s) = P^{-1}(s)D(s)Q^{-1}(s), \tag{8}$$

which reveals the following expression for the inverse of $A(s)$:

$$A^{-1}(s) = Q(s)D^{-1}(s)P(s), \tag{9}$$

which of course only exists if $D(s)$ has full rank. We get the inverse $D^{-1}(s)$ by inverting every entry on the diagonal of $D(s)$.

The Smith normal form can be computed by solving the extended GCD problem. In particular, we wish to find $k(s)$ and $l(s)$ satisfying

$$d(s) = \mathrm{GCD}(a(s), b(s)) = k(s)a(s) + l(s)b(s). \tag{10}$$

The calculation of the GCD is used to reduce columns

$$\left[\begin{array}{cc} k(s) & l(s) \\ -\frac{b(s)}{d(s)} & \frac{a(s)}{d(s)} \end{array} \right] \left[\begin{array}{c} a(s) \\ b(s) \end{array} \right] = \left[\begin{array}{c} d(s) \\ 0 \end{array} \right] \tag{11}$$

or to reduce rows

$$\left[\begin{array}{cc} a(s) & b(s) \end{array} \right] \left[\begin{array}{cc} k(s) & -\frac{b(s)}{d(s)} \\ l(s) & \frac{a(s)}{d(s)} \end{array} \right] = \left[\begin{array}{cc} d(s) & 0 \end{array} \right]. \tag{12}$$

The matrices used in the reductions are unimodular.

Collecting the column reductions in $P(s)$ and the row reductions in $Q(s)$, we reduce $A(s)$ to a diagonal form $D(s)$.

B. Numerical Greatest Common Divisor

On input are two polynomials $a(s)$ and $b(s)$ in s with approximate complex coefficients. Let $d(s) = \mathrm{GCD}(a(s), b(s))$, $\deg(d(s)) = r$.

When applying the algorithm taught in elementary school to compute the GCD of two natural numbers, we repeatedly divide. This repetitive division is numerically unstable for polynomials as the subtraction of polynomials with coefficients of equal magnitude may lead to a dramatic loss of accuracy. Our approach hinges on two operations: root finding and interpolation, for which numerically stable algorithms are well known.

For a given tolerance $\epsilon > 0$, we define the numerical GCD of $a(s)$ and $b(s)$ as the monic polynomial whose roots are common to $a(s)$ and $b(s)$ within the given tolerance ϵ. More precisely, if $a(\alpha_i) = 0$, for $i = 1, 2, \ldots, \deg(a(s))$ and $b(\beta_i) = 0$, for $i = 1, 2, \ldots, \deg(b(s))$, we can rearrange the indices of the roots so that the r common roots appear first. Then we can write:

$$a(s) = \prod_{i=1}^{r}(s - \alpha_i) \prod_{i=r+1}^{\deg(a(s))}(s - \alpha_i) = d_1(s) \prod_{i=r+1}^{\deg(a(s))}(s - \alpha_i) \tag{13}$$

and

$$b(s) = \prod_{i=1}^{r}(s-\beta_i) \prod_{i=r+1}^{\deg(b(s))}(s-\beta_i) = d_2(s) \prod_{i=r+1}^{\deg(b(s))}(s-\beta_i) \quad (14)$$

where $|\alpha_i - \beta_i| \leq \epsilon$, for $i = 1, 2, \ldots, r$, and $|\alpha_i - \beta_j| > \epsilon$, for all i and j with index higher than r. The polynomials $d_1(s)$ and $d_2(s)$ are numerical approximations for $d(s) = \mathrm{GCD}(a(s), b(s))$.

Recall we want to find $k(s)$ and $l(s)$ defined in (10). We determine $k(s)$ by interpolation at those roots of $b(s)$ not shared by $a(s)$ replacing in (10) s by β_i, for $i = r+1, \ldots, \deg(b(s))$:

$$d(\beta_i) = k(\beta_i)a(\beta_i) \quad \text{or} \quad k(\beta_i) = \frac{d(\beta_i)}{a(\beta_i)}, \quad \text{for } i > r. \qquad (15)$$

Note that as $i > r$: $a(\beta_i) \neq 0$. The interpolation conditions in (15) determine $k(s)$ uniquely as a polynomial of degree $\deg(b(s)) - r - 1$. We determine $l(s)$ by interpolation at those roots of $a(s)$ not shared by $b(s)$ replacing in (10) s by α_i, for $i = r+1, \ldots, \deg(a(s))$:

$$d(\alpha_i) = l(\alpha_i)b(\alpha_i) \quad \text{or} \quad l(\alpha_i) = \frac{d(\alpha_i)}{b(\alpha_i)}, \quad \text{for } i > r. \qquad (16)$$

Note that as $i > r$: $b(\alpha_i) \neq 0$. The interpolation conditions in (16) determine $l(s)$ uniquely as a polynomial of degree $\deg(a(s)) - r - 1$.

C. Numerical Experiments

The algorithms described above have been implemented in C. We did practical comparisons between our new algorithm and the elementary approach, for random and specific input data.

When the tolerance of 10^{-8} is used to decide whether two numbers are equal, the elementary school algorithm runs much faster than the new one, but the new algorithm is numerically stable when the degrees of the input polynomials are less than 30 and the degree of the GCD is less than 15. Running 1000 tests on polynomials of degree 30 with a common divisor of degree 15 with random coefficients, we see that the elementary school algorithm reports only in 88% of the cases a correct answer, whereas our new algorithm never fails.

It is easy to find polynomials for which the elementary method fails completely. This happens when the higher degree coefficients of the two input polynomials are very near to each other, say within a distance of 10^{-5} from each other. In such case, our new approach shows the same numerical stability as in the random case.

Concerning the speed, the time needed of the realization is negligible compared to calculating the feedback laws with homotopies.

IV. Software

The dynamic feedback laws were calculated with the aid of PHCpack [35]. We developed an interface around the Ada code of PHCpack to allow the homotopies in PHCpack to be called by routines written in C. Also the realization is done with C code, publicly available since release 2.2 of PHCpack. MATLAB is used to to verify and to analyze the conditioning of the eigenvalues.

The organization of the software is outlined in Figure 3, the arrows indicate the order of function calls in the computation and realization of the dynamic feedback laws.

V. Applications

Input and output data comes with the software and is also at `http://www.math.uic.edu/~jan/feedback_data.htm`.

To verify the results, we calculate the condition number (see [11, page 323]) for the eigenvalue λ_i by

$$\frac{1}{|\mathbf{y}_i^H \mathbf{x}_i|}, \quad i = 1, 2, \ldots, n+q, \qquad (17)$$

where vectors $\mathbf{x}_i$ and $\mathbf{y}_i$ denote the unit right and left eigenvectors of the closed-loop system.

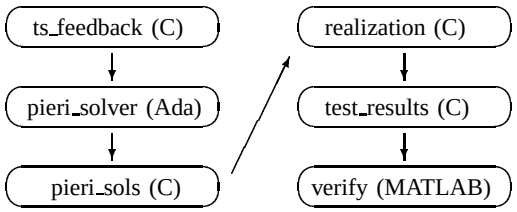

Fig. 3. The software consists of a sequence of calls, from C to Ada, and from Ada to C functions, followed by realization and verification.

A. Satellite Trajectory Control

We want output feedback laws to keep a satellite in orbit (see [7], [16]). In [36], we found two real static output feedback laws. To test our method, we compute dynamic feedback laws.

For this satellite example, we have $n = 4$, $m = p = 2$, which is dimension zero for the static case. When $q = 1$, the system become underdetermined since $n + q = 5 < mp + q(m+p) = 8$ and there are three degrees of freedom. We choose the eigenvalues as $(\frac{-2+i}{\sqrt{5}}, \frac{-2-i}{\sqrt{5}}, -5, -7, -3.0, -0.1068, -0.7834, -0.9582)$. The last three of eigenvalues are randomly selected. We find two real feedback laws and six complex feedback laws.

Substituting the result into the closed-loop system, the relative difference between the computed and the given eigenvalues is bounded by 10^{-11} and the order of condition numbers is at most 10^3, calculated with (17). The total CPU time is 2.32 seconds on a 2.4GHz workstation running Linux.

B. Numerical Examples

We report on two numerical examples in [28] [37].

Numeric Example A: The example in [28] illustrates the following situation: when a system is overdetermined for static output feedback ($n > mp$, $q = 0$), for which no feedback laws can be found at the desired poles, we can choose a q to convert it into a case ($n + q \leq mp + q(m+p)$), for which we can find feedback laws.

Here, $m = p = 2$, $n = 6$, therefore $n > mp$ which is overdetermined for static case. We will choose a q to make the system underdetermined. For $q = 1$, $n + q = 7 < mp + q(m + p) = 8$, there is one degree of freedom. We can easily get some dynamic compensators of degree $q = 1$ to control the system. For a choice $(-0.1, -1.5, -0.9, -0.7, -6.0, -3.5, -8.0)$ of 7 eigenvalues and one additional pole -0.1053 generated at random, the relative difference between the computed and the given eigenvalues of the closed-loop system is bounded by 10^{-9}. We find 8 solutions, of which 4 are real (sometimes 6 are real, depending on the additional input plane which are randomly generated). The order of the condition number computed with (17) is no more than 10^3 for all of the given eigenvalues. The total CPU time is around 3 seconds on the workstation mentioned above.

Numeric Example B: The second example can be found in [37, Example 3.7], with $n = 8$ and $m = p = 3$.

This system is underdetermined for static output feedback ($n < mp, q = 0$). When the given poles are $(-0.8090 + 0.5878i, -0.9511 - 0.3090i, -0.3090 - 0.9511i, -0.3090 + 0.9511i, -0.9511 + 0.3090i, -0.8090 - 0.5878i, -0.5878 + 0.8090i, -0.5878 - 0.8090i, -0.1883)$, in which the first 8 are picked on the unit circle and the last one is a random number, we find 42 feedback laws and 4 of them are real. The relative difference between the computed and the given eigenvalues is 10^{-9}. The order of the condition number (17) is bounded by 10^5. The total CPU time is around 50 seconds to find all 42 feedback laws. It takes 1.18 seconds if only one feedback law is needed.

C. Aircraft Control

It may be that for a given selection of poles, all static feedback laws have coefficients with nonzero imaginary parts. In this case, real dynamic feedback laws may still be found, using the given poles. The jet model from MathWorks [21] illustrates this.

For this example, $n = 4$, $m = p = 2$, so $n = mp$ when $q = 0$, which is the dimension zero case. When the chosen poles are $(-0.234, -1+3.2i, -1-3.2i, -3.0)$, we find two **complex** feedback laws. The total CPU time is 140 milliseconds, and the realization is trivial. Computing the eigenvalues of the closed-loop system, the difference between the computed and the given eigenvalues is just 10^{-14} and the condition number is around 10.

Suppose we need **real** feedback laws. For $q = 1$, $n + q = 5 < mp + q(m + p) = 8$, so we have three degrees of freedom. We find 8 solutions, including 2 or 4, or 6 real solutions, depending on different additional eigenvalues. Taking the first four poles the same as before, adding $(-7.0, -0.944, -0.995, -0.904)$, the relative difference between the computed and the given eigenvalues is bounded by 10^{-10}. The total CPU time is 2.68 seconds. For this choice of poles, we found four real feedback laws. For most of the 8 solutions, the condition numbers of the closed-loop system are less than 10^2, and few of the solutions have the condition number 10^4.

VI. Conclusions

In this paper we showed the practical feasibility of computing dynamic feedback laws using numerical homotopy algorithms, and we applied our software to examples from the control literature. Our publicly available implementation of the numerical greatest common divisor and numerical Smith normal form is of independent interest.

References

[1] P.J. Antsaklis and A.N. Michel. *Linear Systems*. McGraw-Hill, 1997.

[2] B. Beckermann and G. Labahn. Numeric and Symbolic Computation of problems defined by Structured Linear Systems. *Reliable Computing* 6:365-390, 2000.

[3] R.W. Brockett and C.I. Byrnes. Multivariate Nyquist criteria, root loci, and pole placement: a geometric viewpoint. *IEEE Trans. Automat. Contr.*, 26:271–284, 1981.

[4] C.I. Byrnes. Pole assignment by output feedback. In *Three Decades of Mathematical Systems Theory*, edited by H. Nijmacher and J.M. Schumacher, pages 13–78. Springer–Verlag, Berlin, 1989.

[5] E.K. Chu. Optimization and pole assignment in control system design. *International Journal of Applied Mathematics and Computer Science* 11(5):1035–1053, 2001.

[6] R.M. Corless, E. Kaltofen, and S.M. Watt. Hybrid methods. In *Computer Algebra Handbook*, edited by J. Grabmeier, E. Kaltofen, and V. Weispfenning, pages 112-125, Springer-Verlag, 2002.

[7] D.C. Dorf and R.H. Bishop. *Modern Control Systems*. Addison-Wesley, 1998.

[8] I.Z. Emiris, A. Galligo, and H. Lombardi. Certified approximate univariate GCDs. *J. Pure Applied Algebra* 117&118: 229-251, 1997.

[9] A. Eremenko and A. Gabrielov. Counterexamples to pole placement by static output feedback. *Linear Algebra and Appl.* 351–352: 211–218, 2002.

[10] A. Eremenko and A. Gabrielov. Pole placement by static output feedback for generic linear systems. *SIAM J. Control Optim.* 41(1):303–313, 2002.

[11] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Third Edition. The Johns Hopkins University Press, 1996.

[12] D. Henrion and M. Šebek. Reliable numerical methods for polynomial matrix triangulation. *IEEE Trans. Automat. Contr.* 44(3):497–508, 1999.

[13] M.A. Hitz, E. Kaltofen, and Y.N. Lakshman. Efficient algorithms for computing the nearest polynomial with a real root and related problems. In *ISSAC 99 Proc. 1999 Internat. Symp. Symbolic Algebraic Comput.*, edited by S. Dooley, pages 205-212, ACM, New York, 1999.

[14] B. Huber, F. Sottile, and B. Sturmfels. Numerical Schubert calculus. *J. Symbolic Computation* 26(6):767–788, 1998.

[15] B. Huber and J. Verschelde. Pieri homotopies for problems in enumerative geometry applied to pole placement in linear systems control. *SIAM J. Control Optim.* 38(4):1265–1287, 2000.

[16] T. Kailath. *Linear Systems*. Prentice-Hall, 1980.

[17] N. Karmarkar and Y.N. Lakshman. Approximate polynomial greatest common divisors and nearest singular polynomials. In *ISSAC 96 Proc. 1996 Internat. Symp. Symbolic Algebraic Comput.*, edited by Y.N. Lakshman, pages 35-42, ACM, New York, 1996.

[18] M. Kim, J. Rosenthal, and X. Wang. Pole Placement and matrix extension problems: A common point of view. *SIAM J. Control. Optim.* 42(6):2078–2093, 2004.

[19] T.Y. Li, T. Sauer, and J.A. Yorke. The cheater's homotopy: an efficient procedure for solving systems of polynomial equations. *SIAM J. Numer. Anal.* 26(5):1241–1251, 1989.

[20] T.Y. Li, X. Wang, and M. Wu. Numerical Schubert calculus by the Pieri homotopy algorithm. *SIAM J. Numer. Anal.* 20(2):578–600, 2002.

[21] The MathWorks. Control System Toolbox. For Use with MATLAB. Getting Started. Version 5. June 2001.

[22] A.P. Morgan and A.J. Sommese. Coefficient-parameter polynomial continuation. *Appl. Math. Comput.*, 29(2):123–160, 1989. Errata: *Appl. Math. Comput.* 51:207(1992).

[23] V. Pan. Solving a polynomial equation: some history and recent progress. *SIAM Review* 39(2):187–220, 1997.

[24] V. Pan. Computation of approximate polynomial GCDs and an extension. *Information and Computation* 167:71–85, 2001.

[25] M.S. Ravi, J. Rosenthal, and X. Wang. Dynamic pole placement assignment and Schubert calculus. *SIAM J. Control Optim.* 34(3):813–832, 1996.

[26] M.S. Ravi, J. Rosenthal, and X. Wang. Degree of the generalized Plücker embedding of a quot scheme and quatum cohomology. *Math. Ann.*, 311:11–26, 1998.

[27] J. Rosenthal. On dynamic feedback compensation and compactifications of systems. *SIAM J. Control and Optimization*, 32(1):279–296, 1994.

[28] J. Rosenthal and X.A. Wang. Output Feedback Pole Placement with Dynamic Compensators. *IEEE Trans. Automat. Contr.* 41(6):830–843, 1996.

[29] J. Rosenthal and J.C. Willems. Open problems in the area of pole placement. In *Open Problems in Mathematical Systems and Control Theory*, edited by V.D. Blondel, E.D. Sontag, M. Vidyasagar, and J.C. Willems, pages 181–191. Springer–Verlag, 1998.

[30] A.J. Sommese and J. Verschelde. Numerical homotopies to compute generic points on positive dimensional algebraic sets. *Journal of Complexity* 16(3):572–602, 2000.

[31] A.J. Sommese, J. Verschelde and C.W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. *SIAM J. Numer. Anal.* 38(6):2022–2046, 2001.

[32] A.J. Sommese, J. Verschelde and C.W. Wampler. Symmetric functions applied to decomposing solution sets of polynomial systems. *SIAM J. Numer. Anal.* 40(6):2026–2046, 2002.

[33] A.J. Sommese and C.W. Wampler. Numerical algebraic geometry. In *The Mathematics of Numerical Analysis*, volume 32 of *Lectures in Applied Mathematics*, edited by J. Renegar, M. Shub, and S. Smale, pages 749–763, 1996. Proceedings of the AMS-SIAM Summer Seminar in Applied Mathematics, Park City, Utah, July 17-August 11, 1995, Park City, Utah.

[34] F. Sottile and B. Sturmfels. A sagbi basis for the quantum Grassmannian. *J. Pure and Appl. Algebra* 158(2-3): 347-366, 2001.

[35] J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software* 25(2):251–276, 1999. Software available at `http://www.math.uic.edu/~jan/download.html`.

[36] J. Verschelde and Y. Wang. Numerical Homotopy Algorithms for Satellite Trajectory Control by Pole Placement. Proceedings of MTNS 2002, Mathematical Theory of Networks and Systems (CDROM), Notre Dame, August 12-16, 2002.

[37] X.A. Wang. Grassmannian, central projection, and output feedback pole assignment of linear systems. *IEEE Trans. Automat. Contr.* 41(6):786-794, 1996.

[38] L.H. Zhi and M.-T. Noda. Approximate GCD of Multivariate Polynomials. In *Proceedings of the Asian Symposium on Computer Mathematics*, pages 9-18, Chiangmai, Thailand, 2000.

[39] L.H. Zhi and W.D. Wu. Nearest singular polynomial. *J. Symbolic Comput.* 26(6):667-675, 1998. Special issue on Symbolic Numeric Algebra for Polynomials, edited by S.M. Watt and H.J. Stetter.

APPENDIX A
REALIZATION OF MULTI-INPUT MULTI-OUTPUT SYSTEMS

We gave the derivation of the transfer function of the dynamic compensator as the output of the homotopies. We will give a modified algorithm based on [1, pages 389–416] to obtain minimal or irreducible realizations, which realize a system with the least number of dynamic elements. These modifications were made to fit the output format of the software used to compute the feedback laws. The necessity of the modifications will be discussed at the end of this section. We will show how to obtain realizations $\{F_c, G_c, H_c, K_c\}$ of the transfer function $T(s)$ in controller form first. Then, we will use the property of the output of homotopies to show the realizations are irreducible, so they are also observable.

From (6), the transfer function can be written as

$$T(s) = V(s)U^{-1}(s). \tag{18}$$

In accordance with convention, we would replace $V(s)$ by $N(s)$ which stands for numerator, replace $U(s)$ by $D(s)$ which stands for denominator.

According to Theorem 3.3 in [1, page 391], realizations exist if and only if $T(s)$ is a matrix of rational functions and satisfies

$$\lim_{s\to\infty} T(s) < \infty, \tag{19}$$

i.e., if and only if $T(s)$ is a proper rational matrix. Given the transfer function matrix $T(s) = N(s)D^{-1}(s)$ as a $(m \times p)$ proper rational matrix. Let d_j = the highest degree of jth column in the $D(s)$ ($d_j \geq 0, j = 1, 2, \ldots, p$). Define

$$\Lambda(s) = \text{diag}(s^{d_1}, \ldots, s^{d_p}), \tag{20}$$

and

$$S(s) = \text{block diag}\left(\begin{bmatrix} 1 \\ s \\ \vdots \\ s^{d_j - 1} \end{bmatrix} j = 1, \ldots, p \right). \tag{21}$$

If $d_j = 0$, just skip that column and continue to fill the next column of the $S(s)$ matrix. Note that $S(s)$ is an $q(= \sum_{j=1}^{p} d_j) \times p$ polynomial matrix. Write

$$D(s) = D_h \Lambda(s) + D_l S(s) \tag{22}$$

D_h is the highest column degree coefficient matrix of $D(s)$. For example, if $D(s) = \begin{bmatrix} 3s^2+1 & 2s \\ 2s & s \end{bmatrix}$, then the highest column degree coefficient matrix $D_h = \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix}$, and $D_l S(s)$ given in (22) accounts for the remaining lower column degree terms $D(s)$, with D_l being a matrix of coefficients.

In general, $|D_h| \neq 0$, and define $p \times p$ and $p \times q$ matrices

$$G_p = D_h^{-1}, \qquad F_p = -D_h^{-1} D_l, \tag{23}$$

respectively. Then F_c, G_c can be determined from

$$F_c = \overline{F}_c + \overline{G}_c F_p, \qquad G_c = \overline{G}_c G_p, \tag{24}$$

where $\overline{F}_c = \text{block diag}[F_1, F_2 \ldots, F_p]$ with

$$F_j = \begin{bmatrix} 0 & & & \\ \vdots & & I_{d_{j-1}} & \\ 0 & 0 & \cdots & 0 \end{bmatrix} \in R^{d_j \times d_j}, \tag{25}$$

$$\overline{G}_c = \text{block diag}\left(\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \in R^{d_j}, j = 1, \ldots, p \right). \tag{26}$$

When $d_j = 0$, we just skip the corresponding F_j and continue to fill the $\overline{F}_c$ matrix with the F_{j+1} matrix; we also need to add a zero column at the jth column of the $\overline{G}_c$ matrix.

Then we can determine H_c and K_c such that

$$N(s) = H_c S(s) + K_c D(s), \tag{27}$$

and note that

$$K_c = \lim_{s\to\infty} T(s). \tag{28}$$

Therefore, H_c can be determined from (27).

An qth-order realization of $T(s)$ in controller form is now given by the equations

$$\dot{\mathbf{z}}_c = F_c \mathbf{z}_c + G_c \mathbf{y}, \quad \mathbf{u} = H_c \mathbf{z}_c + K_c \mathbf{y}. \tag{29}$$

According to the format of the output of the software, $q = \sum_{j=1}^{p} d_j$ is equal to the minimal order of the dynamic compensator. Therefore, this algorithm gives us a minimal realization of the transfer function matrix $T(s)$ and the result is also observable.

The main difference between the modified algorithm and the original algorithm given in [1] is that the original algorithm limits $d_j \geq 1$, while our modified algorithm works for $d_j \geq 0$, where d_j is the highest column degree of jth column in the $D(s)$. Some d_j must be equal to zero when the number of output is larger than the order of the dynamic compensator. In this case the modified algorithm become necessary. The correctness of the modified algorithm is verified with experiments.

APPENDIX B
DETAILED DESCRIPTION OF OUR SOFTWARE

The dynamic feedback laws were calculated with the aid of PHCpack [35]. While the second public release of PHCpack implemented the dynamic pole placement problem in its geometric form, additional software had to be written, concerning:

0. a limit on the number of feedback laws;
1. an interface between Ada and C; and
2. a collection of C routines for the realization.

The limit on the number of feedback laws was imposed as a matter of convenience, to control the practical complexity. We elaborate the other two items in the following subsections.

A. A C interface to PHCpack

PHCpack is written in Ada, while the programs to process the feedback laws are in the lower level language C.

We can build a portable interface to the Ada routines in PHCpack with C functions because the language Ada has the `pragma Import` construction to call routines from other languages such as C and it supports conversions for C integers, doubles, and arrays of these C types. Furthermore, the gnu-ada compiler provides a mechanism to call Ada routines from a C main program and to call C functions from Ada. As the gnu-ada compiler is integrated in the gcc compilation system, our interface is portable. In particular, we ran our implementation successfully on SUN workstations running Solaris and on PCs running Linux and Windows.

To exchange data efficiently, programs in Ada or C should define exchange protocols of structured data types into basic data types for which automatic conversions are supported. More precisely, we represent structured data types into arrays of doubles and arrays of integers. The language C is restricted in returning dynamically allocated variables. Therefore, data allocated in a C function is passed by the C function calling an Ada function for further processing of the data.

A typical sequence of calls goes as follows. First a C function gathers problem data and prepares the input to an Ada routine of PHCpack. The Ada routine, called from C, uses path tracking to solve the problem, and then calls a C function to process the results obtained with PHCpack. So the C programmer who uses PHCpack should thus provide two C functions: one to prepare the input and one to process the output. This "hand-in-glove" interface is appropriate for a C programmer collaborating with an Ada programmer (which is the case of the authors), who only have to agree on the prototypes of the routines.

B. The Organization of the Software

In Figure 3, the arrows indicate the order of function calls in the computation and realization of the dynamic feedback laws. In this section we give a short description for each of the procedures, some written in C, others in Ada.

ts_feedback(C): ts_feedback reads all the input information from a file, including the number of the internal states n, the input dimension m, the output dimension p, the number of the internal states for the dynamic compensator q and the number of output feedback. Also the user should give the A, B, C matrices (or let the matrices be generated randomly) of the given plant and $n+q$ eigenvalues. ts_feedback.c computes $C(sI_n - A)^{-1}B$ at the interpolation points as the input planes. With C to Ada interface, we pass the arrays of the input planes and the interpolation points to the pieri_solver (an Ada procedure in PHCpack).

pieri_solver(Ada): The pieri_solver calculates the corresponding dynamic output feedbacks and passes them to the C program pieri_sols.c.

pieri_sols(C): With Ada to C interface, the arrays in Ada form are converted to the form in C. Then pieri_sols.c calls realization.c and tests the results.

realization(C): We use modified realization algorithm based on [1] to get the realization of the dynamic output feedback, organized as follows:

(a) Get the transfer function $T(s) = N(s)D(s)^{-1}$ from the output of the Ada program. The inverse of a polynomial matrix is a rational polynomial matrix and it is mainly done by Poly_Smith.
(b) The realization function implements the modified realization algorithm to get a minimal realization of the dynamic compensator.
(c) Evaluate the transfer function $T(s)$ at some random point and compare it with the result after realization ($H(sI_q - F)^{-1}G + K$) at the same point. If the values are the same, the realization is correct.

test_results(C): We can evaluate equation (2) at the given poles and calculate the determinant with the previous result. If the determinant is zero, the pole is the eigenvalue of the closed-loop system. As mentioned above, equation (2) is algebraically equal to equation (1), which is the characteristic equation of the closed-loop system.

verify(MATLAB): Finally, a MATLAB script verifies the results by comparing the computed poles with the given poles and finding the condition number for each given pole.

C. Availability of the Software

The C routines for the numerical realization algorithms written by Yusong Wang are available at `http://www.math.uic.edu/~jan` in the distribution of release 2.2 of the source code of PHCpack (see [35], the first version of this package is archived by Netlib). In particular, the collection of routines can be found in the directory "Feedback" of the source code. Also the input data files and output for the applications discussed in the next section are available in this directory.

APPENDIX C
(A,B,C) APPLICATION DATA

In this appendix, we give the actual values for the matrices A, B, and C which define the linear system $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$, and $\mathbf{y} = C\mathbf{x}$, where $\mathbf{x}, \mathbf{u}$, and $\mathbf{y}$ are vectors of internal states, input, and output respectively.

A. Satellite Trajectory Control

We treated this problem in [36]. This model is described in [7], [16]. C is some random matrix, which can be interpreted as a random projection of the four state variables onto the two output variables.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.3578 & 0 & 0 & 0.8525 \\ 0 & 0 & 0 & 1 \\ 0 & -0.5596 & 0 & 0 \end{bmatrix};$$
$$B = \begin{bmatrix} 0 & 0 \\ 1.3411 & 0 \\ 0 & 0 \\ 0 & 1.0867 \end{bmatrix}. \tag{30}$$

B. Numerical Examples

In this section we report on two numerical examples in the literature [28] [37].

Numeric Example A: (from [28])

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \end{bmatrix};$$
$$B = \begin{bmatrix} -1 & -3 \\ 0 & 0 \\ 0 & 1 \\ 0 & -1 \\ 0 & -1 \\ 0 & 0 \end{bmatrix}; \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{31}$$

Numeric Example B: (from [37])

$$A = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 0 & 0 & 1 & 0 & 0 & -2 \\ 0 & -1 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -7 & 0 & 0 & -2 \\ 0 & -1 & 0 & 1 & 4 & 0 & 0 & 2 \\ 0 & -2 & 0 & 0 & 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & -2 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & -1 \end{bmatrix};$$

$$B = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ -1 & -1 & -3 \\ 1 & 0 & 1 \\ 0 & 2 & 4 \\ 2 & 1 & 5 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \end{bmatrix}; \tag{32}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

C. Aircraft Control

The jet model during cruise flight at MACH=0.8 and H=40,000ft. is taken from Mathworks [21].

$$A = \begin{bmatrix} -0.0558 & -0.9968 & 0.0802 & 0.0415 \\ 0.5980 & -0.1150 & -0.0318 & 0 \\ -3.0500 & 0.3880 & -0.4650 & 0 \\ 0 & 0.0850 & 1.0000 & 0 \end{bmatrix};$$

$$B = \begin{bmatrix} 0.0073 & 0 \\ -0.4750 & 0.0077 \\ 0.1530 & 0.1430 \\ 0 & 0 \end{bmatrix}; \quad C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{33}$$