COMPUTING DYNAMIC OUTPUT FEEDBACK LAWS WITH PIERI HOMOTOPIES ON A PARALLEL COMPUTER

BY

YUSONG WANG
B.E., University of Science and Technology Beijing, 1997
M.S., University of Illinois at Chicago, 2002

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mathematics
in the Graduate College of the
University of Illinois at Chicago, 2005

Chicago, Illinois

This thesis is dedicated to my parents Baozhong Wang and Junping Li,

my wife Guoqing Yan and my son Andrew Wang.

ACKNOWLEDGMENTS

I want to thank my thesis committee: Dr. Floyd Hanson, Dr. Charles Tier, Dr. Jan Verschelde, Dr. Stephen Yau, and Dr. Zhonggang Zeng for their providing essential knowledge and guidance to help me accomplish my research goal.

In particular, I would like to thank Professor F. Hanson. I gained a lot of valuable knowledge from his teaching, from basic numerical analysis to advanced scientific software and high performance computing. I wish to express my appreciation to Professor C. Tier for his teaching and seminar in computational finance, which stimulated my interest. I also want to express my gratitude to Professor S. Yau for his introducing parallel computing to me at the early stage of my graduate study. I want to thank Professor Z. Zeng for his selflessly providing me his software package for my research and study.

An extraordinary gratitude is owed to my advisor, Professor J. Verschelde, who gave me tremendous guide on research. Without his inspiration and support, I wouldn't have my achievement today. During the academic year 2001-2002, I was supported as research assistant by a grant of the UIC Campus Research Board, and that since summer 2002, I have enjoyed research assistantships thanks to NSF Award 0105739 and NSF CAREER Award 0134611. These grants provided me with equipment and gave me travel support.

I also want to give my thanks to the Department of Mathematics, Statistics, and Computer Science at UIC for its support and enjoyable environment for my study and research.

ACKNOWLEDGMENTS (Continued)

This material is based upon work supported by the National Science Foundation under Grant No. 0105739 and Grant No. 0134611. We also thank the National Center for Supercomputing Applications (NCSA) for the use of the Platinum IA32 Cluster (Project: NFA; Principal Investigator: F. Hanson).

Finally, I want to express acknowledgement to my parents for their encouragement, to my wife, Guoqing Yan, for her support, understanding and patience.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF MATHEMATICS SYMBOLS

The symbols used in the thesis with their explanations are listed below. Different meanings for the same symbol occur when there is no confusion from context.

n	Number of internal states
m	Input dimension of a linear system
p	Output dimension of a linear system
q	Order of dynamic feedback compensator
N	The dimension of the geometric problem: $mp + q(m + p)$
$d_{m,p}$	Number of solutions for static feedback laws
$d_{m,p,q}$	Number of solutions for dynamic feedback laws
(A, B, C)	Given matrices to define linear systems
(F, G, H, K)	Desired matrices for output feedback laws
$\mathbf{u}$	Control input

SUMMARY

The output feedback 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. In 1981, R.W. Brocket and C.I. Byrnes published the theoretical relations between the static output feedback pole placement problem and enumerative geometry problem. In 1996, M.S. Ravi, J. Rosenthal, and X. Wang generalized the theory to solve the dynamic output feedback pole placement problem for general Multi-Input-Multi-Output (MIMO) systems. The output feedback pole placement problem was still stated as an open problem at this time, since there was no algorithm to solve it. In 1998, B. Huber, F. Sottile, and B. Sturmfels proposed "numerical Schubert calculus", which led to an efficient homotopy algorithm to solve the enumerative geometry problem. In this thesis, we present symbolic-numeric algorithms to turn the solutions of the homotopies to the output feedback laws of the pole placement problem.

Despite the wider application range of the dynamic output 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. Since the numeric calculation of the Smith normal form is essential for minimal realization of the output feedback laws and the inverse of a polynomial matrix, an original algorithm through computing the extended GCD of two polynomials with the root matching method is implemented. To efficiently solve high dimensional problems with a very large number of solutions, a parallel Pieri homotopy algorithm is designed and implemented.

SUMMARY (Continued)

We successfully applied our numerical approach for solving the output feedback pole placement problem to several applications from the literature. Both the realization algorithm and parallel Pieri homotopies are implemented in PHCpack (ACM TOMS 795), a publicly available software package for solving polynomial system with homotopy methods.

CHAPTER 1

INTRODUCTION

1.1 Feedback Control

Feedback is a fundamental mechanism arising in natural and mechanical processes. A common example of an automatic feedback control system is the cruise control system in an automobile, which maintains the speed of the automobile at a certain desired value with acceptable tolerances. It is essential in automatic control of dynamic processes with uncertainties in their model descriptions. A control law decision process is based not only on predictions about the system behavior from a process model (as in open-loop control), but also on information about the actual behavior (closed-loop feedback control) [2]. As we can only approximate a physical phenomenon with a mathematical model by a certain degree accuracy, in most case it suffices to use linearized ordinary differential equations with constant coefficients. This thesis is restricted to linear systems design, because linear systems usually provide an adequate approximation and are much easier to work with compared with nonlinear differential equations.

Since the behavior of the closed-loop system is governed by the eigenvalues of the closed-loop matrix, we can control a system by placing its eigenvalues or poles to achieve desired performance. The eigenvalue assignment can be applied in both state feedback and output feedback control. The state-space methods, introduced by modern control theory, have provided a deep insight into the structure of dynamical systems and have helped the analysis of systems.

On the other hand, state-space synthesis techniques have faced serious problems in the design of practical control problems. One of the main problems is the requirement of availability or reconstruction of all the state-variables of a system for control purposes. In recent years, there has been a move towards bringing the modern theory closer to practice. This is being achieved through the development of practically-oriented techniques for the feedback design of dynamic systems in which emphasis is placed on the available outputs of the system rather than on the system state-variables. Thanks to the recent development in both theory [5] [41] and algorithms [25] [26] [33] of algebraic geometry and homotopy methods for solving the output feedback pole placement problem, it becomes possible to apply these results to practical problems. In this section, both the state feedback and output feedback pole placement methods will be introduced with examples from control literature.

1.1.1 State Feedback Pole Placement Problem

The purpose of controlling a system is to achieve certain goals which are meaningful for us. Suppose a rocket constrained by a mechanical arm on a movable platform may be slightly misaligned from the vertical position after the mechanical arm is removed (see [38]). The misalignment is desired to be corrected just before the rocket is launched by applying a control force to the platform. With this idealized physical model, we can simplify it to the classical inverted pendulum model with its mass concentrated at the tip [37]. With Newton's laws of motion, we can derive a mathematical state-space model consisting of a system of nonlinear ordinary differential equations and then linearize these equations for small angular deviations of the rocket from vertical to arrive a linear system of ordinary differential equations. The

response of the linear system is governed by the eigenvalues of the system matrix. For the state feedback control (see Figure 1), the feedback can be represented as a linear combination of state variables.

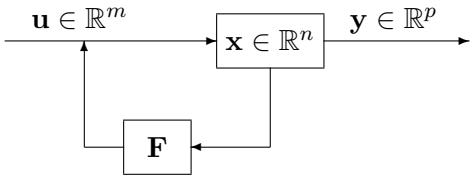

Figure 1. Control of a plant by full-state feedback.

Figure 2 shows the configuration of the rocket and platform system with the inverted pendulum length l and mass m concentrated at the end. The movable platform has mass M. Let $\dot{s}$ and $\dot{\theta}$ denote the derivative of s and θ with respect to time. Then the states of the system can be expressed by a vector defined by $\mathbf{x} = [s, \dot{s}, \theta, \dot{\theta}]^T$. The linearized model can be described as a system of ordinary differential equations:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}, \tag{1.1}$$

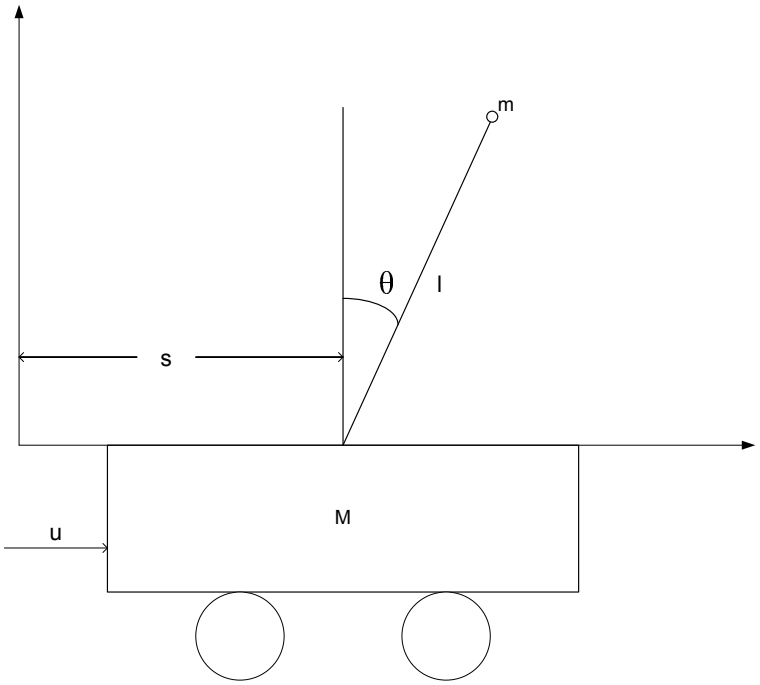

Figure 2. Rocket and platform configuration

where

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)g}{Ml} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{mg}{M} & 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ -\frac{1}{Ml} \\ 0 \\ \frac{1}{M} \end{pmatrix}, \tag{1.2}$$

and M, m, g, l are constants. To achieve the desired behavior, a popular method is to use state feedback:

$$\mathbf{u} = F\mathbf{x}, \tag{1.3}$$

where F is called the gain matrix and $\mathbf{u}$ is the control input. Substituting Equation 1.3 to Equation 1.1 gives the following closed-loop system:

$$\dot{\mathbf{x}} = (A + BF)\mathbf{x}. \tag{1.4}$$

The goal of the state feedback pole placement is finding the gain matrix F to make the closed-loop system have the same eigenvalues as given list. This problem can be solved with the algorithm of Kautsky, Nichols, and Van Dooren (KNvD) [30]. While the algorithm deals only with the nondefective case. It attempts to compute a set of n linearly independent eigenvectors for $A + BF$ before actually computing F. A simulation result (see [37]) of the rocket angle θ and the platform position s of the rocket and platform problem is given in Figure 3.

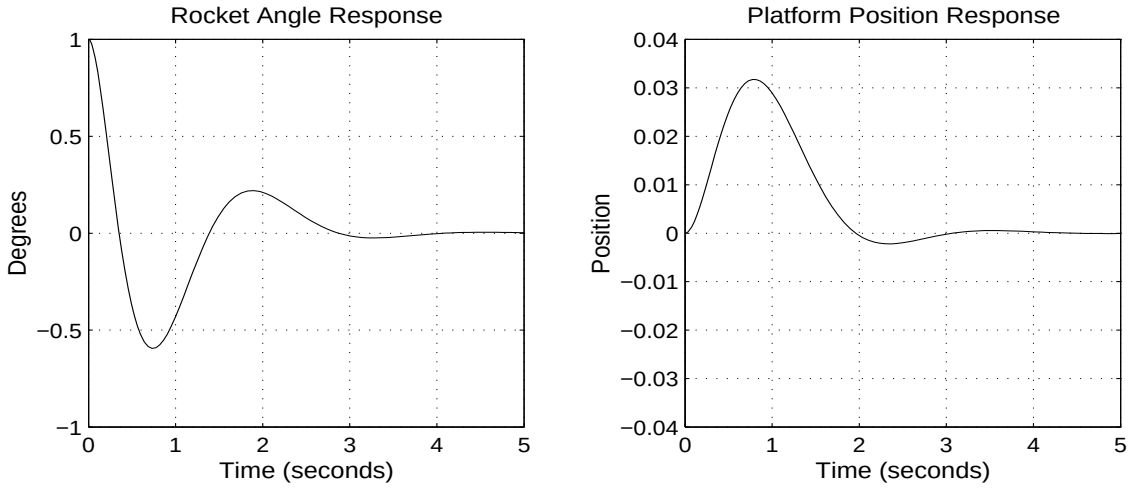

Figure 3. Rocket and platform simulation

Solving the control problem using state feedback pole placement can be divided into two parts. First, choose eigenvalues, closed under complex conjugation, to make the closed-loop has the desired response. Second determine a gain matrix F to make the closed-loop matrix $A+BF$ will have these eigenvalues. The second part is called the *eigenvalue assignment problem* by mathematicians and the *pole assignment problem* by engineers. Engineers use the term *pole* because the eigenvalues are the poles of the system transfer function.

1.1.2 Output Feedback Pole Placement Problem

For the linear state feedback control problem, it is implicitly assumed that all the states are available for measurement. However, in many practical problems, it may be either impossible or impractical to measure the states directly. The output feedback provides a feasible way of designing stabilizing controllers which are based only on a portion of the states, or measured output [10].

Output feedback pole placement: Given a linear system of differential equations and a list of eigenvalues, the output feedback 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.

Because of its importance to practical applications, the development of numerically stable algorithms for this problem was stated as an open problem in [46]: "*the output feedback pole assignment problem (OPAP) is essentially unsolved*" (see also [9]).

The control of an m-input and p-output plant with output feedback control is shown in Figure 4.

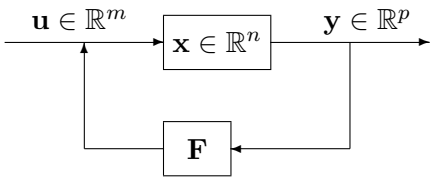

Figure 4. Control of a plant by output feedback.

The satellite trajectory control problem occurred frequently in textbooks, such as [13] and [29]. It is proper to illustrate the output feedback pole placement problem with this example. As in [29], polar coordinates are used for the satellite being in a circular equatorial orbit. The goal of the feedback is to keep the satellite in the same orbit when disturbances such as aerodynamic drag [13] cause it to deviate. The state vector is $\mathbf{x} = [r\ \dot{r}\ \theta\ \dot{\theta}]^T$, and the input is $\mathbf{u} = [u_r\ u_t]^T$, with u_r and u_t respectively the radial and tangential thrusters. The linearized state-space equations are defined by the matrices

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 3\omega_0^2 & 0 & 0 & 2\omega_0 r_0 \\ 0 & 0 & 0 & 1 \\ 0 & -2\omega_0/r_0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & 0 \\ 1/m & 0 \\ 0 & 0 \\ 0 & 1/mr_0 \end{pmatrix}, \tag{1.5}$$

where the radius r_0 and angular velocity ω_0 are such that $r_0^3\omega_0^2$ equals a constant. The mass of the satellite is m. As we now have a system with two inputs, we can also have two outputs to

control the system. We can define $C \in \mathbb{R}^{2\times 4}$ as some random matrix – which can be interpreted as a random projection of the states onto a plane. We can also choose C as

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.6}$$

to find feedback laws. This special choice of C is in agreement with the demonstration in [13, pages 659–660] that the satellite is completely controllable with the tangential thruster u_t only, thus without u_r.

Without loss of generality, assume we are given a linear system with m inputs $\mathbf{u} \in \mathbb{R}^m$, and p outputs $\mathbf{y} \in \mathbb{R}^p$ by three matrices: $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $C \in \mathbb{R}^{p\times n}$, where n equals the number of internal states stored by the vector $\mathbf{x} \in \mathbb{R}^n$. These three matrices define the system of linear first order differential equations :

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}, \tag{1.7}$$

with output

$$\mathbf{y} = C\mathbf{x}. \tag{1.8}$$

We assume the system is completely observable and controllable. The most simple output feedback law (also called the static feedback law) can be denoted by a single matrix $F \in \mathbb{C}^{m\times p}$, and then we have $\mathbf{u} = F\mathbf{y}$. Elimination of $\mathbf{u}$ and $\mathbf{y}$ leads to the following *closed-loop system*:

$$\dot{\mathbf{x}}(t) = (A + BFC)\mathbf{x}(t), \quad \dot{\mathbf{x}}(t) = \frac{d}{dt}\mathbf{x}(t), \tag{1.9}$$

whose behavior is determined by the eigenvalues of $A + BFC$. Our problem is then to find feedback laws F, for given eigenvalues $\lambda_i = 1, 2, \ldots, n$. Thus F is defined by the equations

$$\det(\lambda_i I_n - (A + BFC)) = 0, \quad i = 1, 2, \ldots, n, \tag{1.10}$$

where I_n is the identity matrix of size n.

1.2 A Geometric View of the Output Feedback Control

The theoretical relation between enumerative geometry and output feedback control problem was first made in [5] for $q = 0$, where q is the order of the compensator. Then it was generalized to dynamic case ($q > 0$) in [41] [42] [43]. In this section, we will give the derivation from the static ($q = 0$) output feedback pole placement problem to an enumerative geometry problem, and then show how to solve the geometry problem with homotopy continuation in the next section. A more general case of the dynamic ($q > 0$) output feedback will be discussed in the next Chapter.

To derive the geometric model, we consider the equation equivalent to Equation 1.10:

$$\det \begin{pmatrix} \lambda_i I_n - (A + BFC) & BF & -B \\ 0 & I_p & 0 \\ 0 & 0 & I_m \end{pmatrix} = 0, \quad i = 1, 2, \ldots, n. \tag{1.11}$$

The matrices BF and $-B$ will serve us well in the reduction of $\lambda I_n - (A + BFC)$ to I_n. First we eliminate the BFC: we right multiply the second column of the matrix in Equation 1.11 by C and add the result to the first column. We remove BF by right multiplication of the third column by F and addition of the result to the second column. So, we obtain

$$\det \begin{pmatrix} \lambda_i I_n - A & 0 & -B \\ C & I_p & 0 \\ 0 & F & I_m \end{pmatrix} = 0, \quad i = 1, 2, \ldots, n. \tag{1.12}$$

We multiply the first row of the matrix in Equation 1.12 by $(\lambda_i I_n - A)^{-1}$ (inverse exists if λ_i is not an eigenvalue of A) and then we eliminate C by subtracting C times the first row from the second row. Thus,

$$\det \begin{pmatrix} I_n & 0 & -(\lambda_i I_n - A)^{-1} B \\ 0 & I_p & C(\lambda_i I_n - A)^{-1} B \\ 0 & F & I_m \end{pmatrix} = 0, \quad i = 1, 2, \ldots, n, \tag{1.13}$$

or equivalently,

$$\det \begin{pmatrix} I_p & C(\lambda_i I_n - A)^{-1} B \\ F & I_m \end{pmatrix} = 0, \quad i = 1, 2, \ldots, n. \tag{1.14}$$

In this equivalent formulation of our original problem, the given data: (A, B, C) and the desired feedback laws F are in separated columns.

For each eigenvalue λ_i, we create an $(m+p)$ by m matrix $[C(\lambda_i I_n - A)^{-1} B \quad I_m]^T$, which is the last m columns of the matrix in Equation 1.14. We interpret this matrix as the matrix whose columns contain the generators of an m-plane. Then the feedback laws, which are contained in the first p columns of the matrix in Equation 1.14, correspond to p-planes in $\mathbb{C}^{m+p}$. From a geometric point of view, satisfying Equation 1.14 means the desired p-planes are required to intersect with all the given m-planes. For a complete intersection problem: $n = mp$, because we have mp unknown coefficients in $F \in \mathbb{C}^{m \times p}$.

The control of a machine with pole assignment by a compensator of q internal states corresponds to a problem from enumerative geometry for which the so-called Pieri homotopies were derived. The geometry of the Pieri deformations is illustrated with a simple example of determining the two lines meeting four given lines in projective 3-space in Figure 5, where $m = 2$, and $p = 2$. A 2-plane in projective space, spanned by two generators, is a line in affine space. For visualization purposes, the positive orthant of projective real space is mapped into a tetrahedron. For general case, we are looking for curves which produce p-planes in $\mathbb{C}^{m+p}$ and meet given m-planes sampled at prescribed interpolation points, where m and p correspond to the input and output dimension of the system in Figure 4.

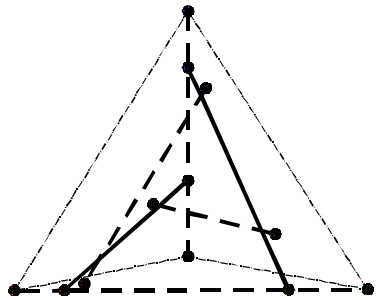

Figure 5. Two (solid) lines meet four given (dashed) lines.

1.3 Numerical Homotopy Algorithms

To find the output feedback laws of the pole placement problem by solving the corresponding enumerative geometry problem, we choose to use homotopy continuation method. See [31] for a survey on recent methods to construct efficient homotopies. In this section we outline the three different stages of our method, using the following homotopies, which have been implemented in PHCpack [56].

A. Pieri homotopies to solve one generic instance in complex space.

Geometric problems are often solved by bringing the input configuration from a general to a special position where the solution can be determined by inspection, or by induction to a lower dimensional problem. For the intersection problem in the previous section, we move one of the four given lines to special position, i.e.: one given line touches two other given

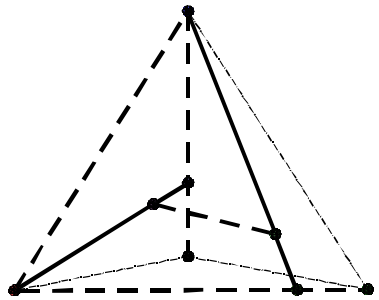

Figure 6. Two (solid) lines meet four given (dashed) lines in special position.

lines (see Figure 6). It is easy to give the solution lines with this special configuration. As the special line at the left moves to the general position showed in Figure 5, we continuously adjust the two solution lines so that they keep meeting the four given lines. If the solution paths defined by this deformation stay finite and free of singularities – except possibly for some algebraic set – then the principle of "conservation of number" applies. This principle is used in enumerative geometry to *count* the number of solutions, but leads very naturally to an efficient numerical homotopy algorithm to *compute* all solutions.

The first homotopy algorithms defining a numerical Schubert calculus were proposed in [25]. Then the algorithms were improved and generalized to dynamic feedback in [26]. In [33], the numerical performance of these homotopies for static feedback was improved.

B. Cheater's homotopy to solve one specific real instance.

With "real" we do not mean that the numbers need to be real, because eigenvalues with nonzero imaginary parts lead to complex input planes, but that the input data have a physical meaning. For the cheater's homotopy to work, we assume we have already solved a problem with similar structure, but with different coefficients. As it is in general not obvious how to obtain the solutions to such a similar problem, there is some cheating going on. For our problem, Pieri homotopies deliver the solutions to a generic problem instance.

We can define the transition from the generic to the specific as a convex linear combination between two systems of equations, defined in Equation 1.14. In a nonlinear cheater's homotopy, we put the continuation parameter inside the determinant. In both cases, singularities can only occur at the end of the paths.

A more general version of cheater's homotopy [32] is coefficient-parameter polynomial continuation [36], which can also be applied to the next type of homotopies.

C. Natural parameter homotopies for design and sensitivity analysis.

The natural parameters are the eigenvalues of the closed-loop system. As we wish to move those eigenvalues, the corresponding feedback laws are varying continuously as well. As with many design problems, the outcome of the design process can only be evaluated through simulation. For fine tuning and sensitivity analysis of the feedback laws it is important that we do not apply the Pieri homotopies or the Cheater's homotopy from a

random complex instance to a real case. Note that eigenvalues are generally chosen to minimize the sensitivity of the closed-loop system to errors [37].

1.4 Contributions of this Thesis

The contributions of the thesis are listed below:

1) realization algorithm and its implementation for the dynamic output feedback laws;

2) algorithm and software for Smith normal form;

3) parallel Pieri homotopies;

4) applications to a test suite of examples.

Despite the wider application range of dynamic output 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 the thesis, the symbolic-numeric algorithms are developed to turn the solutions of enumerative geometry to the feedback control machine. As the output of the geometry problem can be converted to the feedback machine in the frequency domain with some matrix manipulations, a modified realization algorithm based on [2, pages 389–416] is applied to turn the results to the time domain. The Smith normal form is important for a minimal realization of the transfer function of Multiple-Input-Multiple-Output (MIMO) systems, while the related reports on numerical implementations are scare, a commercial implementation [24] leaves numerical Smith normal form as an open question. In order to arrive these results, a new algorithm of computing the greatest common divisor (GCD) of two polynomials by

matching common approximate roots within a certain tolerance (see [40]) is applied, which is more numerically stable than the Euclidean algorithm and works well for multiple roots.

The number of feedback laws of a linear system grows exponentially with its dimension. For a linear system with m inputs and p outputs, Brockett and Byrnes showed the number of complex static output feedback laws, connected to the classic Schubert calculus, as the degree of the Grassmann manifold in [5]:

$$d_{m,p} = \deg Grass(m, m+p) = \frac{1!2!3!\cdots(p-2)!(p-1)!\cdot(mp)!}{m!(m+1)!(m+2)!\cdots(m+p-1)!}. \tag{1.15}$$

While for a more general situation, i.e.: the order of the feedback compensator $q > 0$, the number of complex dynamic output feedback laws was given in [41]:

$$d_{m,p,q} = (-1)^{q(m+1)}(mp+q(m+p))! \sum_{n_1+\cdots+n_m=q} \frac{\prod_{k<j}(j-k+(n_j-n_k)(m+p))}{\prod_{j=1}^{m}(p+j+n_j(m+p)-1)!}. \tag{1.16}$$

So the need for parallel computation is very real. Homotopy methods to solve polynomial systems are well suited for parallel computing because the solution paths defined by the homotopy can be tracked independently. The efficiency of the algorithms for solving systems of nonlinear equations using probability-one homotopy methods in parallel is discussed in [1] [8] [23]. Tracking all paths defined by **one** homotopy is "embarassingly parallel" [18], as the tasks no longer communicate with each other once they are created. While Pieri homotopies are **not** "embarassingly parallel" any more, as the target solution of one homotopy may served as the start solution of another homotopy. This pattern is determined by the poset of localization

patterns used in a combinatorial root count. To implement the Pieri homotopy in parallel efficiently, a new algorithm is designed by mapping the poset to a tree structure.

Both the realization algorithm and the parallel Pieri homotopies are implemented in C using MPI in the version 2 of PHCpack [56], which is written in Ada using gcc and publicly available. With all these efforts, we successfully applied our method to several concrete examples in the literature and solved some higher dimensional dynamic output feedback problems with a parallel computer.

The realization algorithms and the application to some examples are presented in Chapter 2. The parallel Pieri homotopies are described in Chapter 3. The software implementation of the numerical Smith normal form computation and its application are introduced in Chapter 4

CHAPTER 2

NUMERIC REALIZATION OF DYNAMIC FEEDBACK

In the previous Chapter, the static ($q = 0$) output feedback pole placement problem is illustrated with the example of satellite control. In this Chapter, the dynamic ($q > 0$) output feedback problem will be discussed as a more general situation in Section 2.1. Since the calculation of numeric Smith normal form plays an important role for the realization algorithm, Section 2.2 will describe how to compute it through the extended GCD. In Section 2.3, the realization algorithm of Multi-Input Multi-Output Systems will be given to turn the solutions of a polynomial system into the output feedback laws of the control problems. Several examples from control literature will be used to illustrate the usefulness of our approach in Section 2.5. As expected, the dynamic output feedback can be applied to a wider range of applications, such as turning overdetermined problems into fully determined or underdetermined problems and finding real feedback laws which sometimes are not available for the static case. The implementation of the algorithm which involves the interface between C and Ada, the organization and availability of the software will be depicted in Section 2.4.

2.1 Solving Output Feedback Pole Placement Problem

As in Chapter 1, we considered a machine with m inputs and p outputs, whose evolution in time is governed by a system of linear differential equations. The derivation from the output feedback pole placement to the corresponding geometry problem for the static case [5] was given

in the Section 1.2. A more general and useful dynamic case (see [41] [43] [45]) will be discussed in this section: i.e.: instead of controlling the linear system with a single matrix, which can be solved with matrix manipulation, we want to find feedback laws with several internal states to achieve our desired behavior (see Figure 7 for illustration).

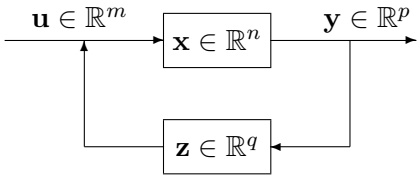

Figure 7. Control of a plant by dynamic output feedback.

Suppose we want to control a plant with n internal states $\mathbf{x} \in \mathbb{R}^n$ that takes m-inputs $\mathbf{u} \in \mathbb{R}^m$ and produces p-outputs $\mathbf{y} \in \mathbb{R}^p$, the dynamic compensator that has q internal states $\mathbf{z} \in \mathbb{R}^q$ can be described by the following ordinary differential equations:

$$\begin{cases} \dot{\mathbf{z}} = F\mathbf{z} + G\mathbf{y} \\ \mathbf{u} = H\mathbf{z} + K\mathbf{y} \end{cases} \quad \text{with} \quad \mathbf{z} \in \mathbb{R}^q \quad \text{and} \quad \begin{array}{l} F \in \mathbb{R}^{q\times q}, G \in \mathbb{R}^{q\times p}, \\ H \in \mathbb{R}^{m\times q}, K \in \mathbb{R}^{m\times p}. \end{array} \tag{2.1}$$

Together with the given first-order differential equation system:

$$\begin{cases} \dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} & \text{with } \mathbf{x} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^p, \\ \mathbf{y} = C\mathbf{x} & \text{and } A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}, C \in \mathbb{R}^{p\times n}, \end{cases} \tag{2.2}$$

we can get the following closed-loop system by eliminating $\mathbf{u}$, $\mathbf{y}$ and concatenating the expressions of $\dot{\mathbf{x}}$ and $\dot{\mathbf{z}}$:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}} \end{bmatrix} = \begin{bmatrix} A + BKC & BH \\ GC & F \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} \tag{2.3}$$

So, the characteristic equation of Equation 2.3 is:

$$\det\left(s \begin{bmatrix} I_n & 0 \\ 0 & I_q \end{bmatrix} - \begin{bmatrix} A + BKC & BH \\ GC & F \end{bmatrix} \right) = 0 \tag{2.4}$$

where s is the eigenvalue or pole of the closed-loop system. Equation 2.4 has its equivalent format :

$$\det \begin{bmatrix} sI_n - A - BKC & -BH \\ -GC & sI_q - F \end{bmatrix} = 0 \tag{2.5}$$

By adding two identity matrices I_p, I_m in the diagonal and some auxiliary matrices, we can rewrite the Equation 2.5 with Equation 2.6. The purpose of introducing these new matrices is to separate the output from input information with the condition in Equation 2.4 satisfied (the

property of the determinant of a matrix) and interpret the problem in a geometric way. The derivation procedure (see [26]) by elementary row and column operations is given below:

$$\det \begin{bmatrix} sI_n - A - BKC & -BH & BK & -B \\ -GC & sI_q - F & G & 0 \\ 0 & 0 & I_p & 0 \\ 0 & 0 & 0 & I_m \end{bmatrix} = 0 \tag{2.6}$$

$$\Leftrightarrow \det \begin{bmatrix} sI_n - A & 0 & 0 & -B \\ 0 & sI_q - F & G & 0 \\ C & 0 & I_p & 0 \\ 0 & -H & K & I_m \end{bmatrix} = 0 \tag{2.7}$$

$$\Leftrightarrow \det \begin{bmatrix} I_n & 0 & 0 & -(sI_n - A)^{-1}B \\ 0 & I_q & (sI_q - F)^{-1}G & 0 \\ C & 0 & I_p & 0 \\ 0 & -H & K & I_m \end{bmatrix} = 0 \tag{2.8}$$

$$\Leftrightarrow \det \begin{bmatrix} I_n & 0 & 0 & -(sI_n - A)^{-1}B \\ 0 & I_q & (sI_q - F)^{-1}G & 0 \\ 0 & 0 & I_p & C(sI_n - A)^{-1}B \\ 0 & 0 & H(sI_q - F)^{-1}G + K & I_m \end{bmatrix} = 0 \tag{2.9}$$

We can write Equation 2.9 in a reduced order form:

$$\det \begin{bmatrix} I_p & C(sI_n - A)^{-1}B \\ H(sI_q - F)^{-1}G + K & I_m \end{bmatrix} = 0 \tag{2.10}$$

The last m columns in the matrix of Equation 2.10 can be thought as a generator of m-planes defined by the given triplet (A, B, C) sampled at prescribed values for s, and the first p columns in the matrix can be thought as a generator of p-planes containing the dynamic feedback laws. Now we can describe the dynamic pole placement problem in a geometric way: we are looking for curves which produce p-planes in $\mathbb{C}^{m+p}$ determined by the unknown quadruple (F, G, H, K) meet the given m-planes. As these p-planes depend on the variable s introduced by the the term $(sI_q - F)^{-1}$, they have maximal minors of degree q and we call them degree q-maps. It is easy to see that for each specification of an eigenvalue s_i, the characteristic polynomial of Equation 2.10 enforces one polynomial condition on the set of degree q-maps.

We have already known the fact that finding the feedback laws of output feedback pole placement problem corresponds to the enumerative geometry problem which can be solved with Pieri homotopies. In [26], the enumerative geometry problem has been solved and implemented in PHCpack. The main task now is to turn the output of this geometry problem in frequency domain to a tuple of matrices in time domain to control the given linear system. The so-called realization will be employed during this transformation. Since the inputs of the Pieri homotopy are some geometry planes, we need transfer the input information of (A, B, C) matrices from time domain to frequency domain by sampling them at the given poles (points). In Figure 8,

we show the transition between the time and the frequency domain. The input and output of the pole placement problem are displayed in the left and right part of Figure 8 respectively. Table I describes the three stages in Figure 8 in detail.

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

Figure 8. Transitions between time and frequency domain.

TABLE I

TRANSITIONS IN APPLYING THE PIERI HOMOTOPIES

(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).

When controlling a machine with n internal states and a controller using q internal states, we can place $n + q$ poles. On the other hand, the dimension of the geometric problem is $N = mp + q(m + p)$. 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 $d(m, p, q)$ (see Equation 1.16), i.e: for a generic choice of the parameters, we have $d(m, p, q)$ complex feedback laws. This degree can easily be computed combinatorially.

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 [15; 16]).

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 respectively number of inputs, number of outputs, and number of internal states which are fixed and given on input. We can choose q to always arrive in a favorable condition and get feedback laws. In particular, if a system is overdetermined for static output feedback ($q = 0$ and $n > mp$), we can always find q to make it convert to underdetermined or dimension zero case to control the system with dynamic output feedback laws. This is because $q(m + p)$ always increases faster than q.

2.2 Symbolic-Numeric Calculations

The stage (2) of Figure 8 requires to transform the solution obtained by the homotopy methods into a description of a machine to execute the dynamic feedback law. We need to calculate the Smith normal form to compute the inverse of a matrix with polynomial entries for this transformation. More precisely, the output of the homotopies is an $(m+p)$-by-p matrix of polynomials in s:

$$\begin{bmatrix} U(s) \\ V(s) \end{bmatrix}, \tag{2.11}$$

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

$$\det \begin{bmatrix} U(s) & C(sI_n - A)^{-1}B \\ V(s) & I_m \end{bmatrix} = 0, \tag{2.12}$$

for the given poles. We can right multiply the matrix argument of Equation 2.12 by

$$\begin{bmatrix} U^{-1}(s) & 0 \\ 0 & I_m \end{bmatrix}. \tag{2.13}$$

The result of this multiplication is

$$\det \begin{bmatrix} I_p & C(sI_n - A)^{-1}B \\ V(s)U^{-1}(s) & I_m \end{bmatrix} = 0. \tag{2.14}$$

Since the multiplier matrix (Equation 2.13) 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 the matrix in Equation 2.10. By comparing Equation 2.10 with Equation 2.14, we can apply the realization algorithms to extract (F, G, H, K) from the matrix $V(s)U^{-1}(s)$.

As we need compute the inverse of a polynomial matrix through its Smith normal form, in the next subsection we show how the calculation of a Smith normal form requires the calculation of greatest common divisor.

2.2.1 Numerical Smith Normal Form

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

$$P(s)M(s)Q(s) = D(s), \tag{2.15}$$

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 $M(s)$. Since unimodular matrices are invertible, we can rewrite $M(s)$ as

$$M(s) = P^{-1}(s)D(s)Q^{-1}(s), \tag{2.16}$$

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

$$M^{-1}(s) = Q(s)D^{-1}(s)P(s), \tag{2.17}$$

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 greatest common divisor (GCD) problem. In particular, we wish to find polynomials $k(s)$ and $l(s)$ so that the following equality holds:

$$d(s) = \mathrm{GCD}(a(s), b(s)) = k(s)a(s) + l(s)b(s). \tag{2.18}$$

The calculation of the greatest common divisor is used to reduce columns

$$\begin{bmatrix} k(s) & l(s) \\ -\frac{b(s)}{d(s)} & \frac{a(s)}{d(s)} \end{bmatrix} \begin{bmatrix} a(s) \\ b(s) \end{bmatrix} = \begin{bmatrix} d(s) \\ 0 \end{bmatrix} \tag{2.19}$$

or to reduce rows

$$\begin{bmatrix} a(s) & b(s) \end{bmatrix} \begin{bmatrix} k(s) & -\frac{b(s)}{d(s)} \\ l(s) & \frac{a(s)}{d(s)} \end{bmatrix} = \begin{bmatrix} d(s) & 0 \end{bmatrix}. \tag{2.20}$$

The matrices we use in the column and row reductions are called unimodular matrices since they have the absolute value of the determinant equal to one.

Collecting the column reductions in $P(s)$ and the row reductions in $Q(s)$, we can reduce any polynomial matrix $M(s)$ to a diagonal form $D(s)$, and so obtain the Smith normal form of $M(s)$.

2.2.2 Numerical Greatest Common Divisor

Computation of approximate greatest common divisor (GCD) of polynomials has a wide range of applications, such as system control, image restoration and computer-aided geometric design, especially the computation of numerical Smith normal form. On the other hand, it is well known that this problem is numerically ill-posed since a tiny perturbation of a polynomial may cause a dramatic decrease of the degree of the GCD. Our data is approximate, i.e.: known only with limited accuracy and subject to roundoff.

If we apply the algorithm taught in elementary school to compute the greatest common divisor of two natural numbers, then we repeatedly have to divide polynomials. This repetitive division is numerically unstable as it involves the subtraction of polynomials, which may lead to a dramatic loss of significant digits when the polynomials have coefficients of equal magnitude. Recently, two relatively stable algorithms for computing approximate GCD are developed: ε-GCD and δ-GCD [40].

For the ε-GCD, where the GCD is defined based on the perturbation of the polynomial coefficients, we choose the univariate GCD package uvGCD (a Maple package, see [63][64]) as a representative, which has been proved to be more stable and accurate than all the available approximate GCD algorithms in the SNAP package [28] in Maple: QuasiGCD [3], EpsilonGCD [3] and QRGCD [11]. The uvGCD employs a successive Sylvester matrix updating process for iden-

tifying the maximum degree of the approximate GCD along with an initial approximation to the GCD factors. Then the Gauss-Newton iteration is applied to certify the GCD and to refine the polynomial factors via solving a regular quadratic least squares problem. For its counterpart δ-GCD, where the GCD is defined based on the perturbation of polynomial zeros, we can find the GCD by matching common roots method [40] via Weierstrass method (also called the method of Durand-Kerner, see [39]) as the root-finder. After finding the roots of the polynomials, we have two ways to find the GCD: the straight way is by expanding $\prod_{i=1}^{r}(x - c_i)$, where r is the number of common roots (within some tolerance) and c_i is the i-th common root; the other approach is getting GCD by finding the extended GCD factor with interpolating the non-common roots of the polynomials (see [58]) . For different numerical examples, one approach could be better than the other one, as showed in our experiments. While for the computation of the numeric Smith normal form, the second approach is required to find the extended GCD factor for reducing the rows or columns of a polynomial matrix.

We chose δ-GCD to compute the greatest common divisor of polynomials with approximate coefficient with two operations: root finding and interpolation. Any introductory course in numerical analysis describes numerically stable algorithms for these two operations. Since our primitive implementation of Weierstrass method as the root-finder could fail when the degree of polynomial is higher than 50*, we employ MPSolve [4] and Eigensolve [17] as auxiliary root-finders for high degree polynomials.

---

*For the output feedback control problems we have studied by now, we never met the case where the degree of polynomial is higher than 30. So it is stable enough for our study.

In this section, we will first describe the δ-GCD method. Then give some numerical experiment results to compare the GCD algorithm by division with δ-GCD method. Finally, we will compare δ-GCD and ε-GCD with some difficult numerical examples for which the SNAP package in Maple could fail and list the advantages and disadvantages for each method.

A) Computing extended GCD with δ-GCD: Suppose we are given two polynomials in one variable and with complex coefficients. For a given tolerance $\epsilon > 0$, we define the numerical greatest common divisor 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 $i = 1, 2, \ldots, \deg(b(s))$, then we can rearrange the indices of the roots of the two polynomials so that the r common roots appear with the lowest indices. Then we write

$$a(s) = \prod_{i=1}^{r}(s - \alpha_i) \prod_{i=r+1}^{\deg(a(s))} (s - \alpha_i) = d_1(s) \prod_{i=r+1}^{\deg(a(s))} (s - \alpha_i) \tag{2.21}$$

and

$$b(s) = \prod_{i=1}^{r}(s - \beta_i) \prod_{i=r+1}^{\deg(b(s))} (s - \beta_i) = d_2(s) \prod_{i=r+1}^{\deg(b(s))} (s - \beta_i) \tag{2.22}$$

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 the greatest common divisor $d(s)$ of $a(s)$ and $b(s)$.

Now that we have numerical approximations for $d(s)$, we want to find $k(s)$ and $l(s)$ defined in Equation 2.18. We determine $k(s)$ by interpolation at those roots of $b(s)$ not shared by $a(s)$ replacing s in the Equation 2.18 by β_i, for $i = r+1, \ldots, \deg(b(s))$:

$$d(\beta_i) = k(\beta_i)a(\beta_i) \quad \text{or} \quad k(\beta_i) = \frac{d(\beta_i)}{a(\beta_i)}, \quad \text{for } i > r. \tag{2.23}$$

Note that as $i > r$: $a(\beta_i) \neq 0$. The interpolation conditions in Equation 2.23 determine $k(s)$ uniquely as a polynomial of degree $\deg(b(s)) - r - 1$. Analogously, we determine $l(s)$ by interpolation at those roots of $a(s)$ not shared by $b(s)$ replacing s in the Equation 2.18 by α_i, for $i = r+1, \ldots, \deg(a(s))$:

$$d(\alpha_i) = l(\alpha_i)b(\alpha_i) \quad \text{or} \quad l(\alpha_i) = \frac{d(\alpha_i)}{b(\alpha_i)}, \quad \text{for } i > r. \tag{2.24}$$

Note that as $i > r$: $b(\alpha_i) \neq 0$. The interpolation conditions in Equation 2.24 determine $l(s)$ uniquely as a polynomial of degree $\deg(a(s)) - r - 1$.

B) Comparing δ-GCD with repetitive division method: The algorithms described above have been implemented with C language in PHCpack. In this section we list some numerical results, obtained by practical comparisons between our new algorithm and the elementary approach, for random and specific input data.

We call the elementary school algorithm by repetitive division the "naive algorithm" and the algorithm with root finding the "advanced algorithm". For simplicity, we assume two given polynomials have same degree. When the tolerance of 10^{-8} is used to decide

whether two numbers are equal, we find the naive algorithm runs much faster than the other one, while the advanced algorithm is more numerically stable when the polynomial degree is less than 30 and the degree of the GCD is less than 15. Although the advanced algorithm is relatively slow, the time spent for each GCD computation is just trivial, less than 10 milliseconds even for the polynomial of degree 30 and the GCD of degree 15. Table II summarizes this experiment.

TABLE II

COMPARISON OF THE NAIVE ALGORITHM AND ADVANCED ALGORITHM ON RANDOM POLYNOMIALS

degree of a and b	degree of GCD	naive(%)[a]	advanced(%)[a]
5	3	100.0	100.0
10	5	99.2	100.0
15	8	99.5	100.0
20	10	99.5	100.0
25	13	98.2	100.0
30	15	88.2	100.0

[a]The percentage of success for 1000 random tests.

To explore the weakness of the naive algorithm, we also compared the performance of the two algorithms with some specific data.

First, we set the leading coefficient of polynomial b is 10^{-5} times leading coefficient of polynomial a. We found that the naive algorithm fails completely when polynomials degree d_p and GCD degree d_g satisfy the following relation:

$$d_p = \begin{cases} 2d_g - 1 & \text{for } d_g \text{ is odd} \\ 2d_g & \text{for } d_g \text{ is even} \end{cases} \tag{2.25}$$

Other relations between the degrees work fine. While the advanced algorithm remains numerically stable no matter what the relations between the degrees are. See the column with header specific numbers(1) in Table III for the experiment of this case.

Secondly, we test with some other specific numbers, say the higher degree coefficients of two polynomials are very near each other (for example, the difference is 10^{-5}), the two algorithms perform totally different. When the degree of the input polynomials is smaller than 5 and the degree of the GCD is smaller than 3, both algorithms work fine. When the degree of the input polynomials is larger than 10 and the degree of GCD is larger than 5, the naive algorithm fails completely, but the advanced algorithm shows the same numerical stability as on the random number case. See columns with header specific numbers(2) in Table III for the experiment of this case.

These experiments provide us with practical evidence that for both random and specific inputs, the advanced algorithm shows its strong numerical stability. Concerning the speed of the GCD algorithm, the time needed of the realization algorithm is negligible compared to the calculation time of the feedback laws with homotopies.

TABLE III

COMPARISON OF THE NAIVE ALGORITHM AND ADVANCED ALGORITHM ON SPECIFIC POLYNOMIALS

		specific numbers(1)[b]		specific numbers(2)[c]	
degree of a and b	degree of GCD	naive method(%)	advanced method(%)	naive method(%)	advanced method(%)
5	3	0.0	100.0	99.9	100.0
10	5	99.1	100.0	0.1	100.0
15	8	98.8	100.0	0.0	100.0
20	10	0.3	100.0	0.0	100.0

[a]The data in columns three to six shows the percentage of success for 1000 tests.

[b]The leading coefficient of b equals to the leading coefficient of a multiplied by 10^{-5}.

[c]The leading coefficients of a and b are equal and the difference between the second coefficients equals 10^{-5}.

C) Comparing δ-GCD with ε-GCD: All test results of the ε-GCD are obtained from uvGCD, with hardware floating-point computation environment in Maple on a workstation of a 2.4GHz Intel XEON CPU and 1GB memory running Windows. The experiments of the δ-GCD are done with the same workstation running Linux. Both ε-GCD and δ-GCD are tested with precision set to 16 digits to simulate IEEE double floating point precision. The first three examples can be found in [63] and the last example is available in the paper [40].

Example 1 A highly sensitive case:

For an even number d, let $k = d/2$,

$$p_d = u_d v_d \tag{2.26}$$

and

$$q_d = u_d w_d, \tag{2.27}$$

where

$$u_d = \prod_{j=1}^{k} [(x - r_1\alpha_j)^2 + r_1^2\beta_j^2], \tag{2.28}$$

$$v_d = \prod_{j=1}^{k} [(x - r_2\alpha_j)^2 + r_2^2\beta_j^2], \tag{2.29}$$

$$w_d = \prod_{j=k+1}^{d} [(x - r_1\alpha_j)^2 + r_1^2\beta_j^2], \tag{2.30}$$

and $\alpha_j = \cos\frac{j\pi}{d}, \beta_j = \sin\frac{j\pi}{d}, r_1 = 0.5, r_2 = 1.5$.

We summarize this experiment in Table IV. In the experiment, we found choosing proper tolerance is important for both ε-GCD and δ-GCD. If 10^{-15} is chosen for the tolerance of the ε-GCD, the uvGCD will fail when $d \geq 10$. While when 10^{-10} is chosen as the tolerance, the uvGCD works fine. Our δ-GCD will fail for $d \geq 16$ when 10^{-8} is chosen as the tolerance to decide if two roots are equal. Setting the tolerance

TABLE IV

EXPERIMENT ON EXAMPLE 1

degree of u_d, v_d and w_d	condition number	ε-GCD[a]		δ-GCD[b]	
		error	time	error	time
6	5.66×10^2	3.0×10^{-16}	2.66s	5.2×10^{-15}	<1ms
10	7.43×10^5	2.7×10^{-14}	3.88s	3.2×10^{-12}	<1ms
16	3.38×10^{10}	3.6×10^{-9}	15.94s	3.2×10^{-8}	0.11s
18	1.74×10^{12}	8.0×10^{-9}	27.00s	6.9×10^{-7}	0.15s
20	7.14×10^{13}	1.3×10^{-6}	44.12s	4.4×10^{-5}	0.18s

[a]The tolerance is set to 10^{-10} for the uvGCD.

[b]The tolerance is set to 10^{-5} for our δ-GCD to decide if two roots are equal.

to 10^{-5} makes our δ-GCD with the Weierstrass method achieve almost the same accuracy as the uvGCD, but the extended GCD fails because of the instability of interpolation on the non-common roots of these ill-conditioned polynomials. While the δ-GCD based on the MPSolve and Eigensolve both fail when $d \geq 16$ for this high sensitive case. We can conclude that the looser tolerance should be chosen when the condition number is larger for which more digits of accuracy are lost.

An observation from this example is that the "UseHardwareFloats" option in Maple can improve the time up to 30% compared with the default, which turns this option off for 16 digits. The accuracy is also improved with 1 or 2 digits by turning on this option.

Example 2 Approximate GCD of large degree:

Let

$$p_d = u_d v \tag{2.31}$$

and

$$q_d = u_d w, \tag{2.32}$$

where

$$v(x) = \sum_{j=0}^{3} x^j \tag{2.33}$$

and

$$w(x) = \sum_{j=0}^{4} (-x)^j \tag{2.34}$$

are fixed,

$$u_d = \sum_{j=0}^{d} r_j x^j, \tag{2.35}$$

r_j is a random integer coefficient in [-5 5].

As showed in Table V, both ε-GCD and δ-GCD can find the correct GCD within reasonable error, even for a very high degree (i.e.: $d = 2000$). For our δ-GCD with

TABLE V

EXPERIMENT ON EXAMPLE 2

degree of GCD	ε-GCD[a]		δ-GCD[b]	
	coefficient-wise error	time	coefficient-wise error	time
50	1.00×10^{-15}	4.36s	5.2×10^{-15}	0.05s
100	4.34×10^{-16}	12.39s	8.2×10^{-15}	0.22s
500	1.21×10^{-15}	288.72s	3.3×10^{-15}	6.50s
1000	2.00×10^{-15}	1208.25s	4.1×10^{-15}	29.62s
2000	2.00×10^{-15}	2.80h	1.1×10^{-14}	118.27s

[a]The tolerance is set to 10^{-10} for the uvGCD.

[b]The tolerance for the δ-GCD based on MPSolve is set to 10^{-5} to decide if two roots are equal.

primitive implementation of Weierstrass method, it only works for $d \leq 50$ because of the stability of the root-finder. With more sophisticated root-finder, such as MPSolve (Eigensolve is slower for this example), the *matching common roots* method works fine for $d \leq 100$, but expanding a polynomial from its roots is not stable for high degree. While we have accurate roots, we can find the extended GCD factor from the non-common roots of the two polynomials, then we only need interpolate polynomials of degree 3 or 4, so we can have stable GCD results.

We can also find the running time of our δ-GCD implemented in C could be up to 85 times faster than the uvGCD on the top of Maple. Although it is unfair to compare the two GCD-finders implemented on different platforms, we can assure that the δ-GCD can be implemented efficiently with a low computation cost.

Example 3 Approximate GCD with large variation in coefficient magnitudes:

For fixed $v(x)$ and $w(x)$ as in Equation 2.33 and Equation 2.34 of the last example, let

$$u(x) = \sum_{j=0}^{15} c_j 10^{e_j} x^j \tag{2.36}$$

where for every j, c_j and e_j are random integers in $[-5\ 5]$ and $[0, 6]$ respectively. In this example, $u(x)$ is the known exact GCD whose coefficient jumps between 0 and 5×10^6. We approximate the number of the correct digits with $log_{10}\xi$ where ξ is the coefficient-wise relative errors of ε-GCD and δ-GCD respectively. The test is repeated 100 times.

In Figure 9, we can find the average correct digits for uvGCD is 13, while our δ-GCD with the Weierstrass method only returned 8 correct digits, some times even failed. Although MPSolve and Eigensolve can improve 1 or 2 digits accuracy, it can't achieve the same accuracy as uvGCD running on the hardware floating-point environment. The author of uvGCD reported 11 correct digits which are close to our δ-GCD with MPSolve. As a conclusion, the uvGCD can get more accurate result than our implementation of δ-GCD with available root-finders. It also more stable than our primitive implementation of δ-GCD with the Weierstrass method for this particular example.

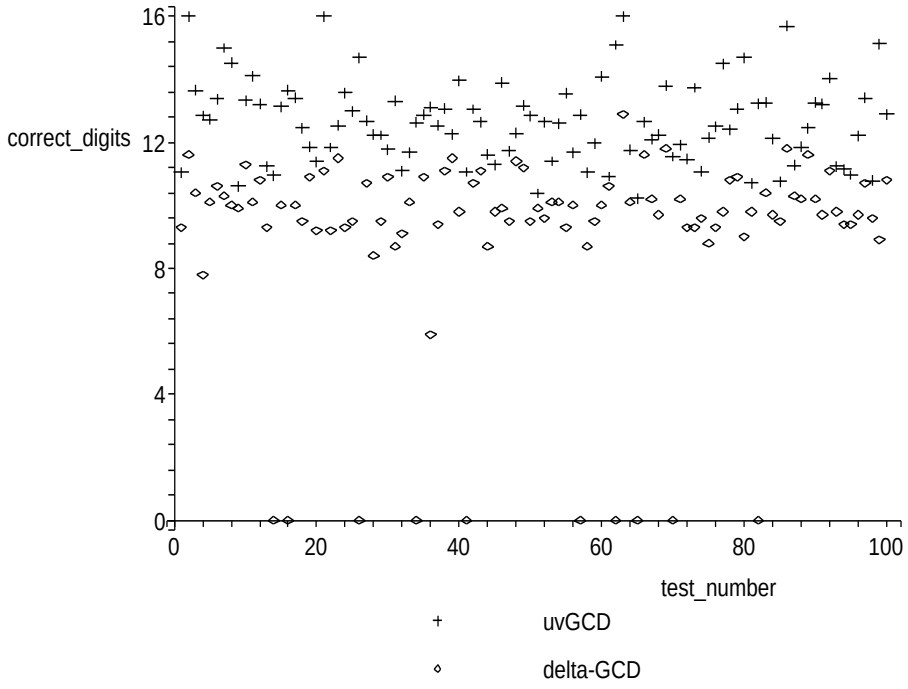

Figure 9. Accuracy comparison on 100 polynomial pairs in Example 3

Example 4 A coefficient-sensitive example:

In [40], the following example (with modification) is used to illustrate how the ε-GCD is sensitive to the small perturbations of the coefficients. Let d be an even and large integer, $\epsilon = 2^{-d}$,

$$u(x) = x^d - 2^{-d}, \tag{2.37}$$

$$v(x) = (x-1)^d. \tag{2.38}$$

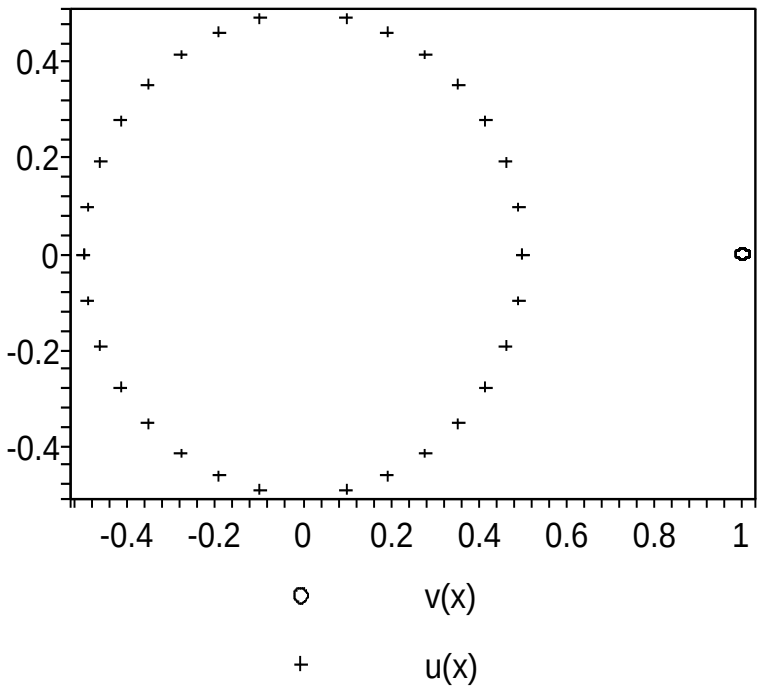

Figure 10. The roots for $u(x)$ and $v(x)$ respectively

Then GCD$(u(x), v(x))$=1, since the minimum distance between zeros of $u(x)(x_k = (1/2)exp(2\pi k\sqrt{-1}/d), k = 0, 1, \ldots, d-1)$ and the only zero of $v(x)(z = 1)$ is large enough to make the approximate GCD to be constant (see Figure 10). While the uvGCD returned a GCD with degree 3 (see Figure 11) when $n = 32$, for which the

tolerance equals 2.33×10^{-10}. The Maple code for this example can be found in appendix A. We can think the polynomial pair

$$u^{\star}(x) = u(x), \tag{2.39}$$

$$v^{\star}(x) = v(x) - 2^{-d}. \tag{2.40}$$

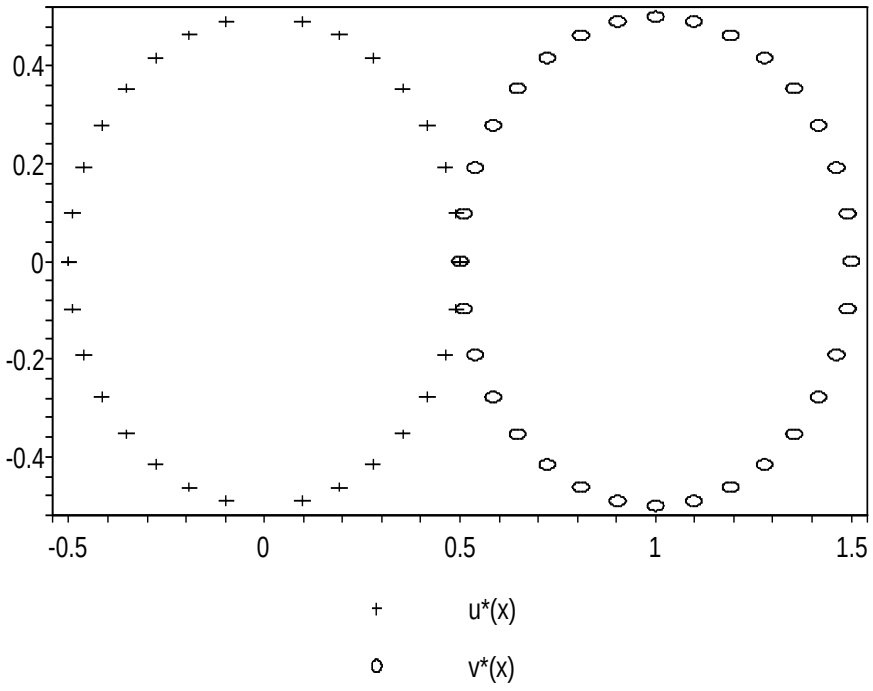

Figure 11. The roots for $u^{\star}(x)$ and $v^{\star}(x)$ respectively

It is obvious that, $x - 0.5$ divides both $u^\star(x)$ and $v^\star(x)$, which can be thought in the same group of polynomials $u(x)$ and $v(x)$ within the given tolerance ϵ, according to the way of defining ε-GCD with coefficient perturbation. While for the δ-GCD method, since we consider the perturbation based on the polynomial zeros instead of coefficients, it won't be a problem. Most sophisticated root-finders can solve the $u(x)$ and $v(x)$, therefore find the corresponding GCD correctly.

From the computation experiments, we can conclude that the uvGCD is more accurate and robust than our δ-GCD when the GCD has large variation in coefficient magnitude. But uvGCD could fail when the input polynomials have multiple or clustered zeros which will not cause problem for the δ-GCD if the correct roots are founded. Both two algorithms can achieve reasonable precision when the condition number or degree of a polynomial is high. As a conclusion, the uvGCD is indeed a robust and stable Maple package for most of the cases, while the δ-GCD implemented in C could be more efficient for large problem if the underlying root-finder is stable.

2.3 Realization of Multi-Input Multi-Output Systems

In the previous section, 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 [2] 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 homotopy methods 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 Equation 2.14, the transfer function can be written as

$$T(s) = V(s)U^{-1}(s). \tag{2.41}$$

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 the theorem in [2], realizations exist if and only if $T(s)$ is a matrix of rational functions and satisfies

$$\lim_{s\to\infty} T(s) < \infty, \tag{2.42}$$

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

$$\Lambda(s) = \text{diag}(s^{d_1}, \ldots, s^{d_p}), \tag{2.43}$$

and

$$S(s) = \text{block diag} \left(\begin{bmatrix} 1 \\ s \\ \vdots \\ s^{d_j-1} \end{bmatrix} j = 1, \ldots, p \right). \tag{2.44}$$

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

$$D(s) = D_h \Lambda(s) + D_l S(s) \tag{2.45}$$

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 Equation 2.45 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

$$G_p = D_h^{-1}, \qquad F_p = -D_h^{-1} D_l, \tag{2.46}$$

respectively. Then $F_c,\ G_c$ can be determined from

$$F_c = \overline{F}_c + \overline{G}_c F_p, \qquad G_c = \overline{G}_c G_p, \tag{2.47}$$

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

$$F_j = \begin{bmatrix} 0 & & & \\ \vdots & & I_{d_{j-1}} & \\ 0 & 0 & \ldots & 0 \end{bmatrix} \in R^{d_j \times d_j}, \tag{2.48}$$

$$\overline{G}_c = \text{block diag} \left(\left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ 1 \end{array} \right] \in R^{d_j}, j = 1, \ldots, p \right). \tag{2.49}$$

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

$$N(s) = H_c S(s) + K_c D(s), \tag{2.50}$$

and note that

$$K_c = \lim_{s \to \infty} T(s). \tag{2.51}$$

Therefore, H_c can be determined from Equation 2.50.

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

$$\dot{z}_c = F_c z_c + G_c y, \qquad u = H_c z_c + K_c y. \tag{2.52}$$

According to the format of the output of the homotopies, $q = \sum_{j=1}^{p} d_j$ 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 [2] 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 becomes necessary. The correctness of the modified algorithm is verified with experiments.

2.4 Software Implementation

The dynamic feedback laws were calculated with the aid of PHCpack [56]. While the second public release of PHCpack implemented the dynamic pole placement problem in its geometric form, additional software had to be written:

0. a limit on the number of feedback laws
1. an interface between Ada and C
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.

2.4.1 An Interface Between C and Ada

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 author), who only have to agree on the prototypes of the routines.

2.4.2 Organization of the Software

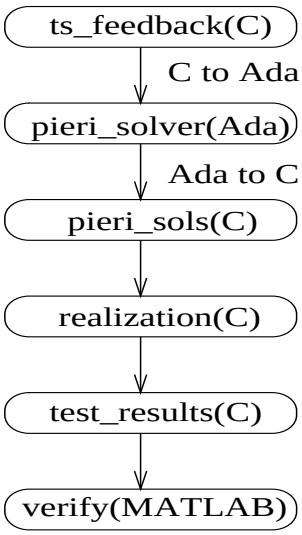

Figure 12. Organization of the software

ts_feedback(C): In ts_feedback.c file, we read 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 user should give the A, B, C matrices (or generate matrices 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 (Ada programs, PHC).

pieri_solver(Ada): Then pieri_solver calculates the corresponding dynamic output feedbacks and pass 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 [2] to get the realization of the dynamic output feedback.

(a) Get 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(C program). See Section 2.2.1 for more detail about how Smith form can work for the inverse of a polynomial matrix.

(b) The realization(C program) implements the modified realization algorithm to get a minimal realization of the dynamic compensator.

(c) Evaluate the transfer function $T(s)$ at a specific point and compare it with the result after realization $(H(sI_q - F)^{-1}G + K)$ at the same point. If they are same, the realization is correct.

test_results(C): We can evaluate Equation 2.10 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 2.10 is algebraically equal to Equation 2.4, 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.

2.4.3 Software Availability and Usage

The software implementation is available with version 2 of PHCpack. The source code and executable files for this release are available at `http://www.math.uic.edu/~jan/download.html`.

After downloading the executable file (or making it from the source code), user can type "phc -k input_file output_file" in the command line to compute the dynamic output feedback laws of the pole placement problem. Where the input_file consists of all the information about the linear system, such as the dimension of input and output, given matrices and poles. A sample input file for the satellite trajectory control problem with its explanation can be found in appendix B. More input files for the examples discussed in the Section 2.5 are available at: `http://www2.math.uic.edu/~ywang25/feedback_data.htm`.

2.5 Applications

In this section we illustrate the usefulness of our approach. Some application examples will show it is necessary to find some dynamic output feedback laws for some specific situations, especially for the cases there is no feedback laws or no real feedback laws at some given poles for the static feedback problem.

We assume that the input data for our applications is given as (A, B, C), i.e.: a triple of three matrices of 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.

2.5.1 Satellite Trajectory Control

This application concerns the design of output feedback laws to keep a satellite in orbit. We treated this problem in Chapter 1 (also in [57]) with static output feedback and wrote ad hoc MATLAB scripts to aid the manipulations. In [57], we already found two real output feedback laws for static case. Now we want to get some feedback laws for dynamic case as an extension of our method.

The input data of the linearized state-space equations for the satellite problem is given below:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.3578 & 0 & 0 & 0.8525 \\ 0 & 0 & 0 & 1 \\ 0 & -0.5596 & 0 & 0 \end{bmatrix};$$

$$B = \begin{bmatrix} 0 & 0 \\ 1.3411 & 0 \\ 0 & 0 \\ 0 & 1.0867 \end{bmatrix}. \tag{2.53}$$

We choose value for each parameter randomly. While this may seem unrealistic, our choice can be justified by an appropriate selection of units. More detail about the model can be found in [13; 29]. We define C as some random matrix – which can be interpreted as a random projection of the states onto a plane.

For this satellite example, we have $n = 4$, $m = p = 2$, which is dimension zero for the static case. When $q = 1$, the system becomes 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 found two real feedback laws and six complex feedback laws. We also use the MATLAB script to verify the results and calculate the condition number for the eigenvalue λ_i by

$$\frac{1}{|\mathbf{y}_i^H \mathbf{x}_i|}, \quad i = 1, 2, \ldots, n + q, \tag{2.54}$$

where vectors $\mathbf{x}_i$ and $\mathbf{y}_i$ denote the unit right and left eigenvectors of the closed-loop system. See [20, page 323] for the derivation of $|\mathbf{y}_i^H \mathbf{x}_i|$ as the reciprocal of the condition number for the eigenvalue λ_i. By substituting the result into the closed-loop system, we find the relative difference of the computed eigenvalues vs. the given eigenvalues is bounded by 10^{-11} and the order of condition numbers are at most 10^3, computed with Equation 2.54. The total CPU time spent on the calculation of the dynamic feedback laws is 2 seconds and 320 milliseconds on a 2.4GHz workstation running Linux System.

2.5.2 Numerical Examples

In this section we report on two numerical examples in the literature [45][62].

Numeric Example A: We will use the example in [45] to illustrate 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 make it convert to underdetermined or dimension zero case ($n + q \leq mp + q(m + p)$), for which we can find feedback laws.

Consider the system, given by

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \end{bmatrix};$$

$$B = \begin{bmatrix} 1 & 3 \\ 0 & 0 \\ 0 & -1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}; \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.55}$$

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. The minimum possible q is 1, then we have $n + q = 7 < mp + q(m + p) = 8$, so there is one degree of freedom. With the calculation of PHCpack and the C interface, we can easily get some dynamic compensators of McMillan degree $q = 1$ to control the system. We verified the result by comparing the given and computed eigenvalues with a MATLAB script. For a choice $(-0.1, -1.5, -0.9, -0.7, -6.0, -3.5, -8.0)$ of 7 eigenvalues and an additional pole -0.1053 which is generated by random, the relative difference of the computed eigenvalues vs. the given eigenvalues of the closed-loop system is bounded by 10^{-9}. In this specific example, we find 8 solutions and 4 of them are real (sometimes 6 are real, depending on the additional input plane which are randomly generated). The order of the condition number computed with Equation 2.54 is no more than 10^3 for all of the given eigenvalues. The total CPU time spent for this numeric example is around 3 seconds with the workstation mentioned above.

Numeric Example B: The second example can be found in [62, Example 3.7], with $n = 8$ and $m = p = 3$.

The system is given as below

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

$$B = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ -1 & -1 & -3 \\ 1 & 0 & 1 \\ 0 & 2 & 4 \\ 2 & 1 & 5 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \end{bmatrix}; \tag{2.56}$$

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

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 at the unit circle and the last one is a negative real number chosen by random, we find 42 feedback laws with our software and 4 of them are real. The relative difference of the computed eigenvalues vs. the given eigenvalues is 10^{-9}. The order of the condition number is bounded by 10^5 if computed with Equation 2.54. The total CPU time spent on this example is around 50 seconds to find all 42 feedback laws. It only takes 1 second and 180 milliseconds if user just needs one feedback law.

2.5.3 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, we will design dynamic output feedback laws, and exploit the additional freedom to place the poles at their originally selected locations. The model from MathWorks' Control System Toolbox manual [34] could be used to illustrate this kind of situation. It also can be found in any standard text in aviation for a more complete discussion of the physics behind aircraft flight. The input data for state-space equations is given below.

The jet model during cruise flight at MACH=0.8 and H=40,000ft. is

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

$$B = \begin{bmatrix} 0.0073 & 0 \\ -0.4750 & 0.0077 \\ 0.1530 & 0.1430 \\ 0 & 0 \end{bmatrix}; \quad C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.57}$$

For this example, $n = 4$, $m = p = 2$, so $n = mp$ when $q = 0$, which is the dimension zero case. We can find some feedback laws with PHCpack. When the chosen poles are $(-0.234, -1+3.2i, -1-3.2i, -3.0)$, we found two complex feedback laws. The total time spent is 140 milliseconds and the realization part just needs trivial time. By computing the eigenvalues of the closed-loop system with the MATLAB script, we find the difference between the computed eigenvalues and the given eigenvalues is just 10^{-14} and the condition number is around 10. Although the result seems quite nice, we still want to find some real feedback laws, which have more practical meaning. We could choose $q = 1$ to find some dynamic feedback laws, therefore $n + q = 5 < mp + q(m + p) = 8$, then we have three degrees of freedom. With the calculation of PHCpack and C interface, we get 8 solutions, including 2 or 4, or 6 real solutions, depending

on different additional eigenvalues. For a choice of $(-0.234, -1 + 3.2i, -1 - 3.2i, -3.0, -7.0)$, in which the first four are the same as in the static case, and three randomly generated poles are $(-0.944, -0.995, -0.904)$, the relative difference of the computed eigenvalues vs. the given eigenvalues is bounded by 10^{-10}. The total CPU time is 2 seconds and 680 milliseconds. 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.

2.6 Future Work

From our application examples, we can observe that finding the real dynamic feedback laws is usually expensive, especially when we have to compute all the complex feedback laws. In [44], [61] and [45], the authors give the conditions where the real feedback laws exist for static and dynamic cases respectively. A natural question is: "do algorithms exist which enable us to find real feedback laws in these cases?" The workload will be significantly decreased if we can develop an effective version of the algorithm to compute the real feedback laws only.

Presently, we resolve the underdetermined case by choosing additional input planes to the geometric problem. For nongeneric machines, the dimension of the set of feedback laws is higher than expected. Recent advances with homotopies (see [47] [48] [49] [50]) allow to treat positive dimensional solution sets, but we defer the application of these recent homotopy methods to a future research.

As pointed in the Section 2.2.2, our implementation of the realization algorithm through numerical Smith normal form is based on the computation of the extended GCD, whose stability

highly depends on the underlying root-finder method. Although we can deal with applications in the Section 2.5 without any trouble, we could meet more practical problems with larger dimensions and high degree of polynomials. For such a situation, more advanced root-finders should be considered. The experiment results in 2.2.2 show that, both MPSolve and Eigensolve are very useful for finding the roots of very high degree polynomial. While uvGCD can be employed as an alternative when there is a large variation in coefficient magnitude.

Simulating the linear system with the output feedback laws computed with our software is another interesting project, where we can place the poles to make the system have desired behavior.

CHAPTER 3

PARALLEL PIERI HOMOTOPIES

Tracking all paths defined by **one** homotopy is "embarassingly parallel", as the tasks no longer communicate with each other once they are created [18]. Pieri homotopies are harder to parallelize because one solution at the end of one path may serve as the start solution for several other pathes. The tasks communicate in a predictable pattern. This pattern is determined by the poset of localization patterns, which are used for the combinatorial root count in sequential implementation. The Pieri homotopies are parallelized by mapping the poset to a tree structure, with the fewest amount of work in the edges closest to the root. The tree structure is proved to be more feasible for parallel computing since the jobs can be separated more easily. In this Chapter, a parallel path tracker in PHCpack with one homotopy will be outlined first, which will be used for solving a specific instance after we have solved a generic problem with similar structure with Pieri homotopies. Our experiments show that the dynamic load balance is more efficient than the static load balance when there is a large variance between the workload of each processor. Then the parallel Pieri homotopy algorithm will be described to solve the pole placement problem in the control of linear system.

3.1 A Parallel Homotopy Path Tracker in PHCpack

Homotopy continuation methods are reliable and powerful methods to compute numerical approximations to all isolated complex solutions. These methods are different from more clas-

sical techniques in that they are global and do not require a choice of initial solution estimates, which is in contrast with local methods, such as Newton's method [35]. We can solve a system $f(\mathbf{x}) = \mathbf{0}$ with the homotopy continuation methods in two stages. We first construct a general system $g(\mathbf{x}) = \mathbf{0}$ for $f(\mathbf{x}) = \mathbf{0}$ and solve the general system. Then $g(\mathbf{x}) = \mathbf{0}$ becomes the start system in the homotopy

$$h(\mathbf{x}, t) = \gamma g(\mathbf{x})(1 - t) + tf(\mathbf{x}) = \mathbf{0}, \quad \gamma \in \mathbb{C}. \tag{3.1}$$

For almost all choices of the complex constant γ, all solutions paths $\mathbf{x}(t)$ are regular and bounded for $t \in [0, 1)$. In the second stage, continuation methods are applied to trace the paths starting at the known solutions of $g(\mathbf{x}) = \mathbf{0}$ to the desired solutions of $f(\mathbf{x}) = 0$, as t goes from 0 to 1. The publicly available software PHCpack [56] implemented the homotopy methods in a sequential version.

The homotopy algorithm is well suited for parallel computing, since the paths can be tracked independently from each other. The efficiency of the algorithms for solving systems of nonlinear equations using probability-one homotopy methods in parallel is discussed in [1] [8] [23]. More recently, in [21] and [55] the authors report on a parallel implementation of polyhedral homotopy methods, which exploits the sparse structure of polynomial systems.

In this section, two load balancing schemes will be described, then the computational experiences of tracking paths defined by the homotopy (Equation 3.1) with academic benchmarks and mechanical applications will be reported.

3.1.1 Static and Dynamic Workload Balance

For best performance, the workload should be distributed evenly among the processors. In the static workload distribution, the paths are distributed evenly to the processors once at the start. While this leads to a minimal communication overhead, the workload for each processor may have a large variance, as paths diverging to infinity require more time. The dynamic workload assignment with a master/slave (or manager/worker) paradigm is usually better. In this more flexible approach, the master keeps a pool of units of work larger than the number of slaves. Each of the slave processors will be given one job at the beginning. After a slave finishes its job, it sends the result to the master, which sends then a new job to the slave. While this requires more communication overhead than the static workload assignment model, we can improve it by overlapping the communication and computation with the non-blocking sending and receiving in the MPI library. This dynamic workload approach, called *self-scheduling*, works well in the presence of tasks of vary sizes and/or workers of varying speeds [54].

3.1.2 Experimental Results and Applications

Our parallel code was developed on a Beowulf cluster with four 2.4GHz processors under Linux. To examine the speedup and the load balancing issues better on larger problems, we ran the code on the Platinum cluster at the National Center for Supercomputing Applications (NCSA).

Example A: An academic benchmark: cyclic 10-roots

The cyclic n-roots problem is widely used as a benchmark for publicly available software ([19], [21], [56]). Computing all cyclic n-roots is hard because the number of paths

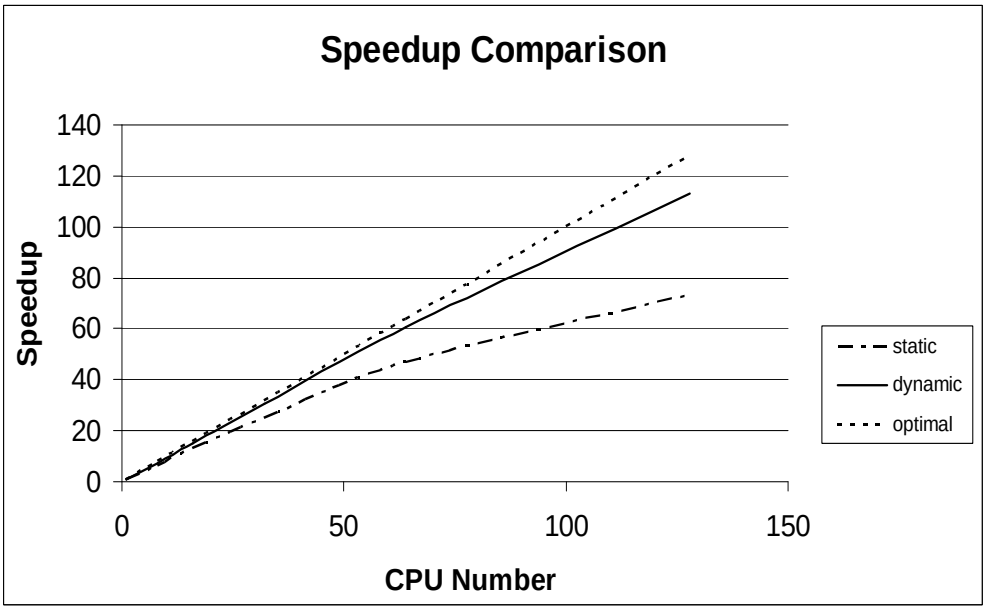

Figure 13. Speedup comparison for the cyclic 10-roots problem

is often too large to be traced by a single computer [12]. For the polynomial system with $n = 10$, we need to trace 35,940 paths. With a given start system, it takes 8 hours with the sequential version of path tracker on a 1GHz computer. Our parallel path tracker traces all 35,940 paths within 5 minutes on 128 1GHz CPUs. The target system for the cyclic 10-roots problem is listed in appendix C. The start system has the same monomials as the target system, but the coefficients are chosen at random.

From Table VI, we see that the dynamic workload balancing improves the total time of the static approach by 10% to 35%. For this problem, the variance of the time needed to trace the paths can be large (one thousand paths diverge). The improvement of using

TABLE VI

SPEEDUP COMPARISON FOR THE CYCLIC 10-ROOTS PROBLEM

N^a	Static		Dynamic		Improvement
	time[b]	speedup	time[b]	speedup	dynamic/static
1	480.0	1.0	480.0	1.0	–
8	75.5	6.4	66.6	7.2	11.75%
16	36.4	13.2	31.7	15.2	12.96%
32	19.0	25.3	15.7	30.7	17.56%
64	10.2	46.9	7.9	60.5	22.48%
128	6.6	73.3	4.3	112.9	35.11%

[a]Number of CPUs.
[b]User CPU time in minutes.

dynamic load balancing is more obvious with more processors since the variance becomes larger for fewer jobs on each processor in the static workload assignment. Figure 13 shows that the speedup is almost optimal for the dynamic model when the number of processors is less than 32. For any other number of processors, dynamic workload model still wins.

Example B: An application from mechanism design

This example comes from the geometric design of a five degree-of-freedom robot formed by links connected by revolute, prismatic and spherical joints to form an RPS serial chain [51; 52; 53]. To design this robot one must solve ten polynomial equations in ten unknowns. The homotopy we used (using a linear-product start system as in [52]) led to 9,216 solution paths. As reported in [51], the sequential version of PHCpack takes about

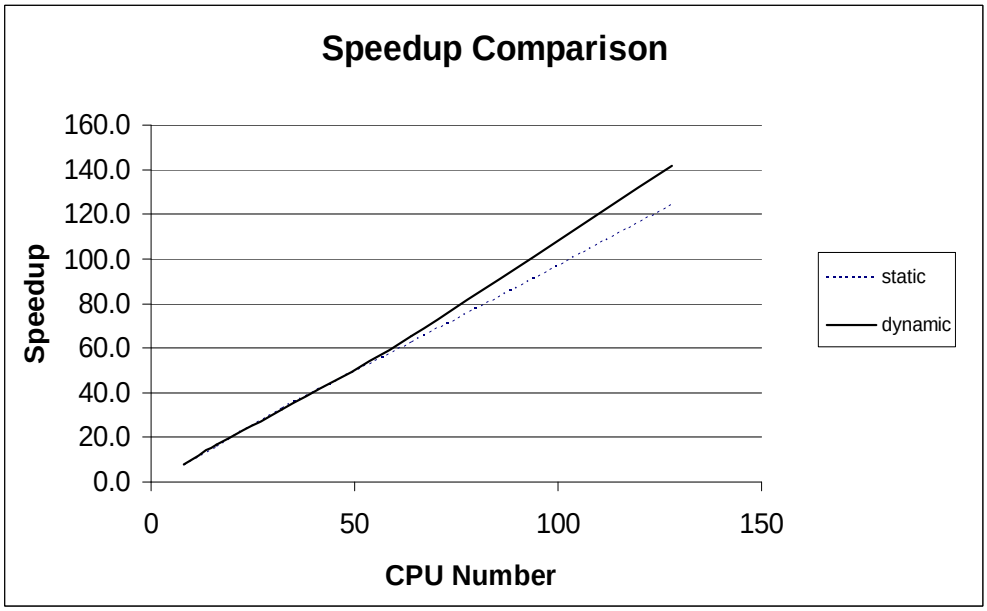

Figure 14. Speedup comparison for a mechanical application

24 hours on a 2.4GHz Pentium IV machine. On 128 1GHz CPUs of the Platinum cluster at NCSA, all paths were traced within 22 minutes. As the time for one 1GHz machine was unavailable, we assumed an initial optimal speedup in the dynamic case, extrapolating to 3111.2 CPU minutes sequential time, obtained as 8×388.9 (see Table VII). While assuming an initial optimal speedup with 8 CPUs is unrealistic, when doubling the number of processors we may finish more than twice as fast when the distribution of the workload is more evenly spread among the processors, see Figure 14.

In Table VII, we can find the improvement of the dynamic over the static balancing model is not obvious here. Since in this example, there are more than eight thousand

TABLE VII

SPEEDUP COMPARISON FOR THE RPS PROBLEM

N^a	Static		Dynamic		Improvement
	time[b]	speedup[c]	time[b]	speedup[c]	dynamic/static
8	417.5	7.5	388.9	8.0	6.84%
16	195.1	15.9	183.7	16.9	5.84%
32	94.7	32.9	96.1	32.4	-1.50%
64	49.8	62.5	47.5	65.5	4.65%
128	25.1	124.0	22.0	141.4	12.43%

[a]Number of CPUs.
[b]User CPU time in minutes.
[c]speedup calculated from extrapolated sequential time.

diverging paths, which dominate the total computation time and each of the diverging path spend almost the same time. So there is no large variance in the workload among the processors in the static model. Moreover, the overhead of the communications decreases the efficiency of the dynamic load balancing model.

We took this application to illustrate what happens when – an unfortunately still too often occurring case – many solution paths diverge to infinity. For this particular system, the mixed volume gives the exact root count of 1024, and thus the polyhedral homotopy (implemented in [19], [21], and [56]) is optimal. The black-box solver of PHCpack gives the complete solution list in 24.6 minutes CPU time on a 2.4GHz Linux machine.

3.2 Parallel Pieri Homotopy Algorithm

As introduced in the first Chapter, the number of solutions will grow exponentially when the degrees of input and output become higher. In this section, we will outline the underlying algorithm of the Pieri homotopy first, then describe the parallel Pieri homotopy algorithm by mapping the posets structure to the Pieri tree structure [59].

3.2.1 Localization Pattern of the Solutions

The pole placement problem we want to solve can be described in the following enumerative geometry way, for which the so-called Pieri homotopies were derived: Let $N = mp + q(m + p)$, for given N general m-planes L_i in $\mathbb{C}^{m+p}$ and N interpolation points $s_i \in \mathbb{C}$, $i = 1, 2, \ldots, N$, we want to compute all polynomial maps $X(s)$ of degree q producing p-planes that meet those given general m-planes L_i at the prescribed interpolation points s_i, i.e.: we are given N intersection conditions:

$$\det(X(s_i)|L_i) = 0, \quad i = 1, 2, \ldots, N. \tag{3.2}$$

These intersection conditions define a polynomial system in the coefficients of the map $X : \mathbb{C} \rightarrow \mathbb{C}^{(m+p)\times p} : s \mapsto X(s)$. In what follows, we show that the problem (Equation 3.2) is well posed: we have N equations in the N variables which define a general map $X(s)$.

We represent $X(s)$ by a localization pattern in $\{0, \star\}^{(m+p)\times p}$ (i.e.: a matrix over $\mathbb{Z}_2$) in which all stars stand for the nonzero coefficients of the generator matrix. A p-plane fits a localization pattern if it can be represented by a matrix of generators with zero entries everywhere the localization pattern prescribes them. For example, in Figure 15, the left picture is the canonical

form of the degree one-map solution localization pattern for $p = 2$, $m = 2$, $q = 1$, where the t is for homogenizing the polynomials to deal with both bottom pivots and top pivots. The middle picture is the concatenated form with $p + N$ stars, where we append the higher degree coefficients below the lower degree coefficients and the degree of freedom is $N = 8$. The right picture is a shorthand notation for the bottom pivots which record the row indices of the bottommost stars.

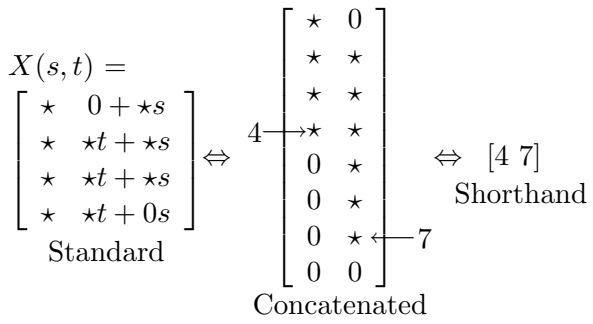

Figure 15. Localization pattern of solutions for $p = 2$, $m = 2$, $q = 1$

A valid bottom pivot localization pattern is defined as below:

1. Let $q = dp + r$ with $d, r \in \mathbb{N}$ and $r < p$. A localization pattern for $(m + p) \times p$-maps of degree q has the first $p - r$ columns with dimension $(d + 1)(m + p)$ and the remaining columns have dimension $(d + 2)(m + p)$.

2. All stars within a column should be contiguous and the row indices in which the bottommost and topmost stars occur strictly increasing as a function of the column index. These indices are called the top and bottom pivots, respectively.

3. No two bottom pivots differ by $m + p$ or more.

The above definition is extracted from [26]. To make it easy to understand and implement, we first consider the top pivots as fixed to $[1\ 2\ \cdots\ p]$.

The special emphasis on the format of $X(s)$ is entirely justified as it leads naturally to a homotopy as follows. The bottommost pivots of X give a recipe (see [26]) for a special m-plane S_X so that $\det(X|S_X) = 0$ if and only if at least one of the entries in X at the bottommost pivots is zero. The Pieri homotopy in Equation 3.3 moves S_X to L_w, where the L_w is the wth general m-plane. For $q > 0$, the map $X(s)$ can meet S_X only at infinity, so the corresponding interpolation point moves from infinity to s_w. To represent infinity properly, we homogenize the polynomials in $X(s)$ using t, and denote the maps as $X(s,t)$. The Pieri homotopy in Equation 3.3 then moves $(s,t) = (1,0)$ to $(s_w, 1)$. Observe the double use of t in Equation 3.3: as continuation parameter and variable added to homogenize the maps.

$$H(X(s,t),s,t) = \begin{cases} \det(X(s,t)|(1-t)S_X + tL_w) = 0 \\ (s-1)(1-t) + (s-s_w)t = 0 \\ \det(X(s_i,t_i)|L_i) = 0, \\ i = 1,2,\ldots,w-1, \end{cases} \quad \text{for } t \in [0,1]. \tag{3.3}$$

The start solutions for the Pieri homotopy all fit in the patterns $Y(s,t)$ obtained from $X(s,t)$ by turning a bottommost star to zero. By induction on w, we assume that all these children $Y(s,t)$ meet already the $w-1$ general m-planes L_i at (s_i,t_i), i.e.: $\det(Y(s_i,t_i)|L_i)=0$, for $i=1,2,\ldots,w-1$. To satisfy the wth intersection condition with L_w at (s_w,t_w), we trace the solution paths defined by the Pieri homotopy in Equation 3.3, as t goes from 0 to 1.

Note that at $t=1$ and $w=N$, we have all the intersection conditions in Equation 3.2 satisfied. In the next section we describe the induction on w from 1 to N, which leads to an efficient way to count all the roots.

3.2.2 Counting Roots by Mapping the Poset to a Tree Structure

The shorthand notation of the bottom pivots in Figure 15 provides a convenient way to count the solution maps and to represent the nested sequences of homotopies needed to compute all solutions.

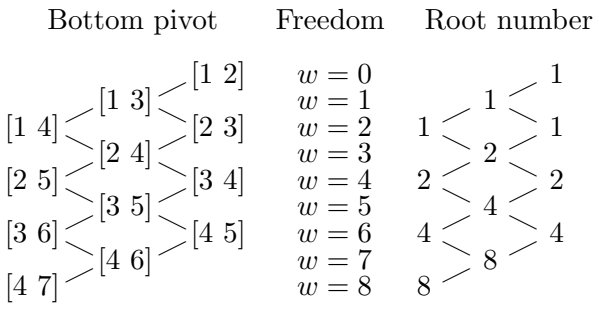

Figure 16. Combinatorial root count for $p=2$, $m=2$, $q=1$ with the poset structure

In Figure 16, the poset structure is described to count the number of the solution planes. It starts from the trivial localization pattern which has its top pivots [1 2 $\cdots$ p], e.g., [1 2], and its bottom pivots [$d(m+p)+m+r+1$ $\cdots$ $d(m+p)+m+p$ $(d+1)(m+p)+m+1$ $\cdots$ $(d+1)(m+p)+m+r$], e.g., [4 7]. The bottom poset structure at the left in Figure 16 is obtained by fixing the top pivots (omitted in the picture) and decreasing a bottommost pivot, which is called a bottom child, to get all the valid localization patterns recursively. The middle of Figure 16 shows the degree of freedom or the number of intersection conditions satisfied. At the right of Figure 16 we see the counting procedure: we start with one solution for $w = 0$; then, for $w > 0$, the number of maps fitting X and meeting w general m-planes equals the sum of the number of solution maps fitting the children of X and meeting $w-1$ general planes. Every link in the poset in Figure 16 corresponds to one instance of the Pieri homotopy.

The combinatorial root count with a poset is implemented in the sequential version of PHCpack [56]. In the parallel version of the Pieri homotopy program, we solve the problem based on a tree, called Pieri trees in [25]. The tree corresponding to the poset in Figure 16 is shown in Figure 17. The number in the brackets are the bottom pivots with top pivots fixed as [1 2]. The bottom trivial localization pattern corresponds to [1 2]. The solutions are counted by starting at the top and finding all the allowable paths to reach the leave [4 7], adding up the number of leaves through different paths yielding 8 solutions.

To see the virtue of Pieri trees for parallel computers, we need to recall the induction on w in the derivation of the Pieri homotopies. In the Pieri tree, each edge represents a job, i.e.: the tracking of one solution path. Two jobs (represented by two edges in the Pieri tree) become

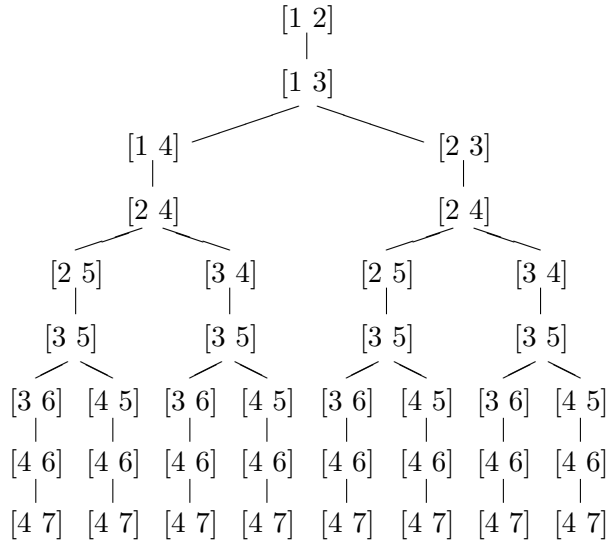

Figure 17. Combinatorial root count for $p = 2$, $m = 2$, $q = 1$ with the Pieri tree.

completely independent from each other once the solution at their common ancestor node has been computed. Compared to posets, the organization of the path tracking along Pieri trees makes them more suitable for parallel computers, since the workload for each of the processors can be balanced better.

In practice, the whole tree is not necessary to be stored during the whole life time. Actually, we only need to store several nodes of the tree at every specific stage of the computation. This can avoid consuming too much memory for very large problem. Every node in the tree is only needed in the computation of the path ending at the node, or in the paths originating from the node, in total no more than $p + 1$ jobs. So in general, the memory occupied by a node in the Pieri tree can be released right after the job ending at the node has been finished and the new jobs originating from the node have been created. In the poset structure, however, the nodes

carry the information of many more paths and need to remain active even if only one job is still not completed. Especially for larger problems, as the number of solutions grows exponentially, the number of internal nodes may also increase dramatically, exhausting all the memory rather quickly.

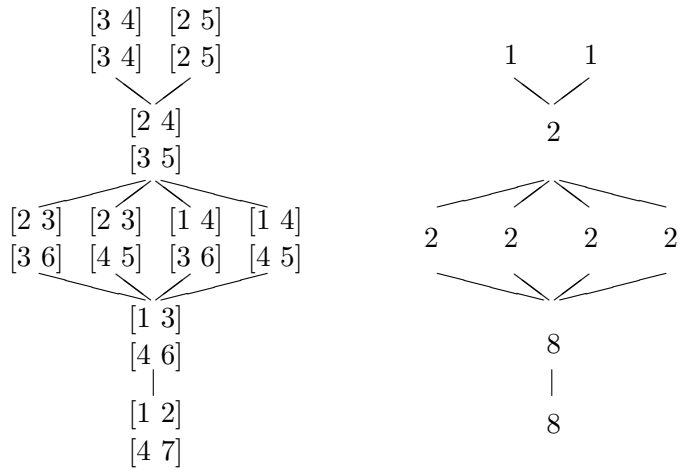

Figure 18. The poset for changing top and bottom pivots simultaneously.

As pointed out earlier, the Pieri homotopies used above keep their top pivots fixed, we can increase top and decrease bottom pivots simultaneously, hereby satisfying two new intersection conditions with one Pieri homotopy [26]. The combinatorial root count for our running example is shown in Figure 18. The top pivots are displayed above the bottom pivots in the nodes of the poset on the left. At the right we see the corresponding solution count. This scheme needs

in general fewer solution paths than when keeping top pivots fixed. The poset in Figure 18 gives rise to a forest of two trees, shown in Figure 19. Counting the edges in Figure 17 and in Figure 19, we find respectively 37 and 26 path tracking jobs.

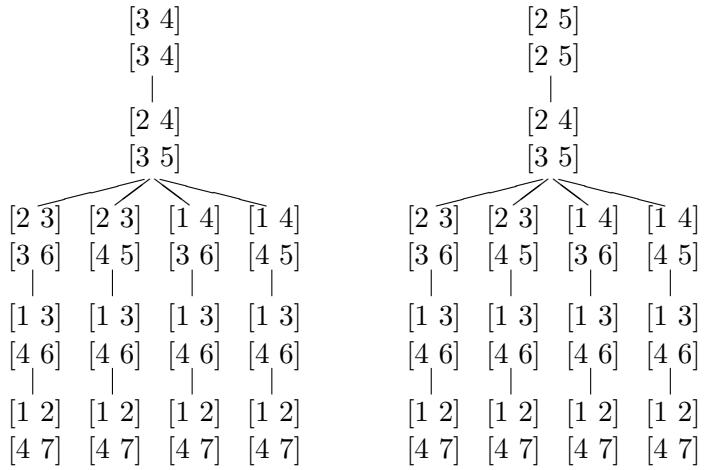

Figure 19. The forest of two trees derived from the poset in Figure 18.

3.2.3 Software Implementation of the Algorithm

Figure 20 illustrates the procedure of the parallel Pieri computation. Here we apply the dynamic workload assignment with a master/slave paradigm, which is proved to be better in the previous section. At the beginning, the master generates (at most p) jobs by increasing the bottom pivots and puts them in the queue. Then the master distributes the available jobs to the slaves, which will finish the computation task. When one of the slave finishes its

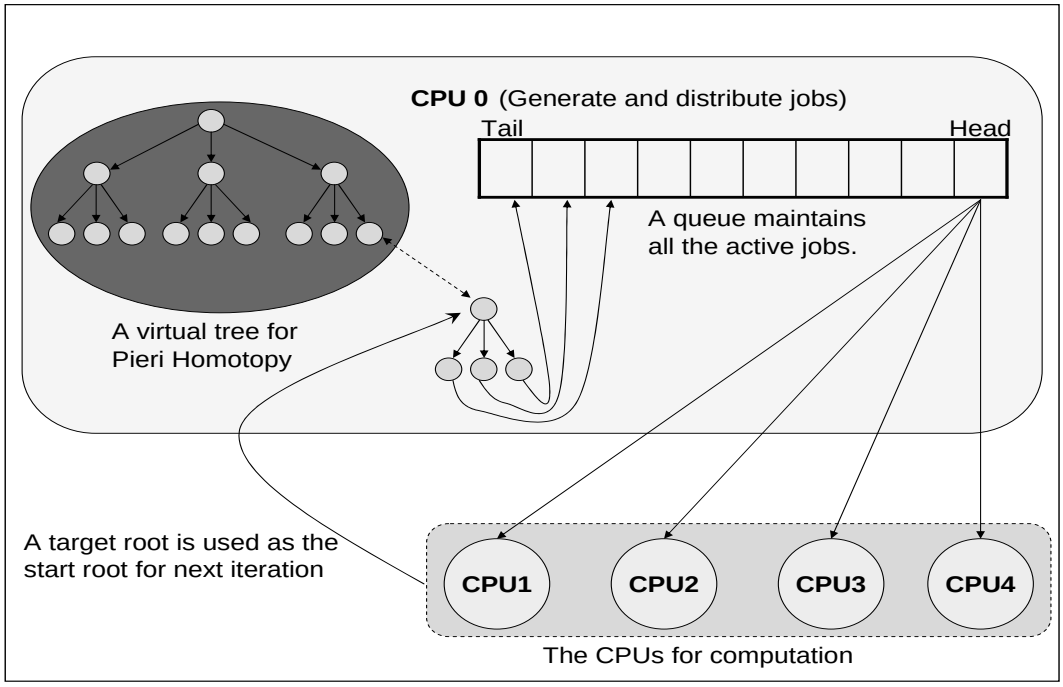

Figure 20. Parallel Pieri homotopy with a virtual tree structure

job, it returns the result to the master. The master generates (at most p) new jobs according to the returned information, which includes the pivot information of the node and the target solution of the previous homotopy, and then puts the jobs in the queue. At first, the jobs are distributed sequentially according to the rank of the slave. After all slave processors are activated, the dynamic workload balance paradigm based on the first-come-first-serve strategy is implemented to compute the remaining jobs.

Since the workload is dynamically distributed, the slave does not know the number of jobs needs to be done in advance. So we need to figure out a way to terminate the computation

subroutine properly. The intuitive idea is when all of the slaves returned leaves, which can not generate any new jobs, we are done. It could be the case, when some of the slaves return leaves and find no job in the queue, they won't work any more, while the other slaves are still working on the internal nodes of the virtual tree. This situation will cause an unbalanced workload distribution, therefore the efficiency will be lower. To avoid this case, we maintain another queue to record which slave has returned a leaf, and activate it again when there are more jobs available to compute. After all the slaves have returned leaves, the master will send a message to each slave to terminate their busy waiting loops.

For more efficient implementation where we simultaneously increase top and decrease bottom pivots (as showed in Figure 19), another queue is used to store the roots of the trees in the forest. Then we build up a virtual tree for each of the roots as we did with the bottom pivot case above.

As the path tracking jobs are subject to a tree hierarchy, every job has to wait till the first path starting at the root node has terminated. Also, every job in the tree has to wait till the job providing its start solution has been finished. So at the start of the of program, only very few processors are active, while most other processors are idle, waiting for their start solutions. Since the jobs highly depend on each other, the efficiency of the parallel computation will usually be affected. Fortunately, the jobs closest to the root are the smallest, as exemplified in Table VIII for $m = 2$, $p = 2$, and $q = 1$. Typically, almost half of the time is spent at the last level, towards the leaves of the Pieri tree. As a result, all of the processors will be assigned a job after trivial time.

TABLE VIII

NUMBER OF PATHS AND USER CPU TIMES

w^a	#paths	user CPU time
1	1	0ms
2	2	0ms
3	3	10ms
4	5	30ms
5	8	80ms
6	13	370ms
7	21	1s 290ms
8	34	3s 830ms
9	55	8s 190ms
10	55	7s 840ms
11	55	16s 570ms
Total	252	38s 350ms

[a]The level of tree

3.3 Applications

Table IX shows the experimental results for different values of m, p, and q. As m, p, and q increase, the number of solutions grows exponentially, for example: 135,660 for $p = 4$, $m = 3$, $q = 1$. The dimension N of the problem grows too, but fortunately as a polynomial, i.e.: $N = mp + q(m + p)$. Nevertheless, the problem of computing **all** solutions quickly becomes intractable for a single processor computer because of limited time and memory.

Our 2.4GHz PC under Linux can only solve some low dimensional problems in hours. On the Platinum cluster at NCSA, we improved the time from hours to minutes for some lower

TABLE IX. SOLVING PIERI HOMOTOPIES ON A 2.4GHz PC AND 64 1GHz CPUs OF PLATINUM CLUSTER AT NCSA[a]

		$q=0$			$q=1$			$q=2$			$q=3$		
			time(s)			time(s)			time(s)			time(s)	
p	m	#Sols	PC	Cluster	#Sols	PC	Cluster	#Sols	PC	Cluster	#Sols	PC	Cluster
2	2	2	0.2	–	8	0.9	–	32	18.4	–	128	218.3	19.1
3	2	5	0.2	–	55	38.4	–	610	2331.7	137.2	6765	N/A	4749.0
3	3	42	8.8	–	2730	7663.8	327.7	17462	N/A	–			
4	3	462	638.7	52.4	135660	N/A	–						
4	4	24024	N/A	1891.2[b]									

[a]Times are user CPU time in seconds. #Sols = number of Solutions.

[b]Done on 256 CPUs

dimensional problems and solved some higher dimensional problems which can not be solved by our PC. The table is drawn in an upper triangular format to show the limit size of the problem which our PC can solve.

For concrete applications of Pieri homotopies to control linear systems, please refer to Chapter 2 and [57; 58].

3.4 Future Work

Although we can solve some high dimensional problems with a parallel computer in a reasonable time, the performance of the program still needs to be improved.

For our current implementation, we maintain a queue of active jobs with a linked list structure. The memory for each node is allocated and deallocated dynamically in the heap for its lifetime. The advantage of this technique is we can allocate as much memory as we need until the heap becomes "full", so we can deal with the problem with very large number of jobs. While the frequent allocation and deallocation of memory will affect the efficiency, we can consider to take advantage of other techniques to improve it. Hanson and Sorensen employed recycling job queues in [22] to enhance the limit number of processes that the pre-PVM parallel programming package SCHEDULE (see [14] and [22]) can handle. The recycling use of the queues replaced the cumulative number of processes limits with only an active number of processes limit. Since the limit is given at the beginning according to the size of the problem, we can avoid tedious memory management during the computation and get better efficiency.

Another issue is the dependency between the jobs at different levels of the virtual tree. As a job can't start until its predecessor has been done, it could cause an unbalanced situation

near the end of the computation, i.e.: some branches of the tree can be finished earlier than the other branches which will make some processors have no job to do. As mentioned in the previous section, we have implemented a separate queue to avoid some processors "retire" early after finishing a job from a leave. A priority queue with the level of the node in the tree as the key can be employed to solve the problem in essence and get better balance.

CHAPTER 4

SOFTWARE FOR NUMERICAL SMITH NORMAL FORM

The Smith normal form has many applications in group theory, module theory and number theory, especially in the linear system control for our case. The algorithms of efficiently computing the Smith norm form of large polynomial matrices can be found in [27] and [60]. While our implementation emphasizes on the Smith normal form computation of the polynomial matrices with approximate coefficients, as this is more practical situation for the design of a control system for a physical problem. The basic operation for computing the Smith normal form is the computation of GCD of two polynomials. It is well known that the GCD problem is sensitive to the perturbation of the coefficients. In the Chapter 2, we have given a stable algorithm to compute the extended GCD of two polynomials, which is used to eliminate the row or column to make the matrix become diagonal. In this Chapter, we will introduce the self-contained software for computing the numerical Smith normal form and illustrate its usage with a numerical example.

4.1 Organization of the Software

We distinguish two levels in the development of this portable C software:

(1) the basics:

dcmplx : defines complex numbers of double floats

dc_roots[1]: approximates all roots of a polynomial

ts_roots[2] : tests root finding operations

dc_matrix : defines basic operations on complex matrices

dc_inverse : inverse of a complex matrix

ts_dc_inverse : test on inverse matrix computation

(2) routines for numerical Smith normal form:

poly_dcmplx[3]: arithmetic of complex polynomials

dc_interpolation : interpolation with divided differences

poly_gcd : defines extended GCD calculations

ts_interpolation : tests interpolation routines

ts_gcd : tests GCD computations

poly_matrix : defines operations on matrices of polynomials

poly_hermite : Hermite normal form of polynomial matrix

poly_smith : Smith normal form of polynomial matrix

ts_hermite : test on the Hermite normal form

ts_poly_inverse : test on inverse of polynomial matrix

---

[1]dc = double complex

[2]ts = test facility

[3]poly = one variable polynomial

ts_smith : test on the Smith normal form

The level (1) of the software is tailored towards our specific application. We include it to make our software self-contained. At level (2), we implement the original algorithm in section 2.2.1 to calculate the Smith normal form of a matrix of univariate polynomials with approximate complex coefficients (see [58]). Every function in the software is also self-contained with its own test function. So it can also be used as a general polynomial matrix operation software.

4.2 Availability and Usage of the Software

The software for computing the numerical Smith normal form can be downloaded from `http://www.math.uic.edu/~ywang25/download.html`. User can either download the executable file for their preferred operation systems (Windows and Linux available), or use the makefile to generate their own executable file. It is recommended to prepare an input file, which contains the information of a polynomial matrix, before we can test the program.

For example, if we want to compute the Smith normal form of the polynomial matrix:

$$\begin{bmatrix} 3s^2+s+2 & 6s^2+2s+4 \\ s+2 & 2s^2+4s+6 \end{bmatrix}. \tag{4.1}$$

User can prepare an input file as below:

```
2 2            // the number of rows and columns of the matrix
2              // choose the input matrix:
```

```
                1 random matrix.
                2 user's own matrix.

2           // the degree of the polynomial for (0, 0) entry
2 0         // the complex coefficients of the polynomial
1 0            from low degree to high degree
3 0

2           // the degree and coefficients for (0, 1) entry
4 0
2 0
6 0

1           // the degree and coefficients for (1, 0) entry
2 0
1 0

2           // the degree and coefficients for (1, 1) entry
6 0
4 0
2 0
```

Then type "./ts_smith < input_file" in the command line. The Smith normal form of the given matrix returns within a second:

$$\begin{bmatrix} 1 & 0 \\ 0 & -6s^4 - 8s^3 - 12s^2 - 6s - 4 \end{bmatrix}, \tag{4.2}$$

with unimodular matrices:

$$P = \begin{bmatrix} 8.33 \times 10^{-2} & -0.25s + 0.417 \\ s + 2 & -3s^2 - s - 2 \end{bmatrix} \tag{4.3}$$

and

$$Q = \begin{bmatrix} 1 & 0.5s^3 - 0.333s^2 - 0.333s - 2.83 \\ 0 & 1 \end{bmatrix}. \tag{4.4}$$

The original output of the example above can be found in appendix D. User can also choose interactive input in the command line, keeping in mind that the matrix input is row-wised and the complex coefficients of a polynomial are read from low degree to high degree. The usage of the other functions in the software, such as computing the Hermite normal form and inverse of a polynomial matrix, is similar.

CHAPTER 5

CONCLUSIONS AND FUTURE RESEARCH

We showed the practical feasibility of computing dynamic feedback laws using numerical homotopy algorithms and successfully turned the solutions of enumerative geometry to the output feedback laws of the pole placement problem, which was stated as an open problem. Since the numeric calculation of the Smith normal form is essential for minimal realization of the output feedback laws and the inverse of a polynomial matrix, an original algorithm of computing numerical Smith normal form was presented and implemented through computing the extended GCD of two polynomials with the root matching method. While the stability of the algorithm highly depends on the underlying root-finder, more advanced root-finders, such as MPSolve [4] and Eigensolve [17] can be employed to improve our primitive implementation of Weierstrass method. Presently, we resolve the situation when the number of interpolation conditions is more than number of eigenvalues to assign by choosing additional input planes to the geometric problem. Recent advances with homotopies (see e.g. [47] [48] [49] [50]) allow to treat positive dimensional solution sets.

Homotopy methods to solve polynomial systems are well suited for parallel computing because the solution paths defined by the homotopy can be tracked independently. Both the static and dynamic load balancing models were implemented in C with MPI, adapting PHCpack written in Ada using gcc, and tested on academic benchmarks and mechanical applications. The dynamic load balancing was proved to be more efficient when there is a large variance of work

load between different processors for the static case, where only minimum communications are required. We studied the parallelization of Pieri homotopies to compute all feedback laws to control linear systems. To distribute the workload, we mapped the poset onto a tree. As the dimensions of the Pieri homotopies grow incrementally from the root to the leaves in the tree, we found the Pieri homotopies well suited for parallel computing. A parallel Pieri homotopy algorithm to solve the enumerative geometry problem with a very large number of solutions was designed, which made our method feasible for higher dimension problems.

Both the realization algorithm and parallel Pieri homotopies are implemented in PHC-pack [56], a software package for solving polynomial system with homotopy methods. Although our approach for solving output feedback control pole placement problem has been successfully applied to several applications in the literature, the stability of the Pieri homotopies still needs to be improved with more efficient algorithm [33] for some large systems, e.g., the turbo-generator problem (10 internal states with 2 inputs and 2 outputs) and the model of a Boeing 767 aircraft at a flutter condition (55 internal states with 2 inputs and 2 outputs) in [6].

APPENDICES

Appendix A

MAPLE CODE FOR THE COEFFICIENT-SENSITIVE GCD EXAMPLE

```
> # This example is from Pan's paper (Example8.1)

> restart;

> UseHardwareFloats := true:

> # Use hardware floating-point computation environment

> Digits:=16:

> # Load Zeng's uvGCD package.

> read("c:/wang/Courses/mcs563/pj3/Zeng/loadgcd.txt"):

> loadgcd("c:/wang/Courses/mcs563/pj3/Zeng/"):

> n:=32: # Choose n as a large and even number

> epsilon:=2.0^(-n); # The tolerance when n=32

               epsilon := 2.328306436538696 10^(-10)

> Ux:=x^n-epsilon: Vx:=(x-1)^n: # The given polynomials

> f:=Ux: g:=Vx-epsilon:

> # The polynomials in the same group of the given polynomials

> # within the tolerance

> u1,v1,w1,p,q,res,cond := pgcd(Ux,Vx,x,epsilon):

> u1;  # The GCD computed with the package

-0.1250000002471365 + 1.000000000000000022 x^3  -
```

Appendix A (Continued)

```
1.48078528137636622 x^2 + 0.740392641176121358 x

> solve(u1); # The roots of the GCD

0.4903926405233433 - 0.09754516107924724 I,

0.4903926405233433 + 0.09754516107924724 I,

0.5000000003296794

> cond; # The condition number

                    6.359589448796864 10^(16)

> with(RootFinding): with(plots):

> # Plot the roots for the given polynomials

> zeros_U := RootFinding:-Analytic(Ux,x,re=-1.1..1.1,im=-1.1..1.1):

> P:=plots[complexplot]([zeros_U],style=point,symbol=CROSS,

>                       axes=boxed,color=black,legend="u(x)"):

> zeros_V := solve(Vx,x):

> Q:=plots[complexplot]([zeros_V],style=point,symbol=CIRCLE,

>                       axes=boxed,color=black,legend="v(x)"):

> display({P,Q}):

>

> # Plot the roots for the perturbed polynomials within the tolerance

> zeros_f := RootFinding:-Analytic(f,x,re=-1.1..1.1,im=-1.1..1.1):

> P:=plots[complexplot]([zeros_f],style=point,symbol=CROSS,

>                       axes=boxed,color=black,legend="u*(x)"):
```

Appendix A (Continued)

```
> zeros_g := solve(g,x):

> Q:=plots[complexplot]([zeros_g],style=point,symbol=CIRCLE,

>                       axes=boxed,color=black,legend="v*(x)"):

> display({P,Q}):
```

Appendix B

SATELLITE TRAJECTORY CONTROL DATA

```
4 2 2 1 -1 0 1


  0       0     1     0    0   0    0      0
0.3578    0     0     0    0   0  0.8525   0
  0       0     0     0    0   0    1      0
  0       0  -0.5596 0    0   0    0      0


  0       0     0     0
1.3411    0     0     0
  0       0     0     0
  0       0   1.0867  0


0.1234    0   0.7934  0   0.8322  0   0.7979  0
0.3434    0   0.4232  0   0.0233  0   0.9232  0


-0.8945    0.4472
-0.8945   -0.4472
-5.0       0
```

Appendix B (Continued)

```
-7.0       0

-3.0       0


------------------------------------------------

Application of Satellite Trajectory Control


The meaning of the input parameter(in order):

The number of the internal states for the given plant (A, B, C) n = 4

The system's input dimension m = 2.

The system's output dimension p = 2.

The number of the internal states for the dynamic compensators q = 1.

Give the number of solutions wanted (<0 for all) : -1

Type 0, 1, 2, or 3 to select output level :

  0. no intermediate output;

  1. only final determinant validation;

  2. + validation of all intermediate determinants;

  3. + intermediate output of all path trackers.

The amount of the intermediate output: 0

Type 0, 1, 2 to select input :

  0. interactive input of real numbers

  1. interactive input of complex numbers
```

Appendix B (Continued)

```
  2. random input of complex numbers

  3. random input of real numbers

  4. interactive input of real numbers for matrices and complex numbers for poles

The selected input is: 1

The given matrix A(4*4)

The given matrix B(4*2)

The given matrix C(2*4)

The given poles
```

Appendix C

CYCLIC 10-ROOTS PROBLEM

```
Target system:


  z0 + z1 + z2 + z3 + z4 + z5 + z6 + z7 + z8 + z9;


  z0*z1 + z1*z2 + z2*z3 + z3*z4 + z4*z5 + z5*z6 + z6*z7 + z7*z8 + z8*z9 + z9*z0;


  z0*z1*z2 + z1*z2*z3 + z2*z3*z4 + z3*z4*z5 + z4*z5*z6 + z5*z6*z7
+ z6*z7*z8 + z7*z8*z9 + z8*z9*z0 + z9*z0*z1;


  z0*z1*z2*z3 + z1*z2*z3*z4 + z2*z3*z4*z5 + z3*z4*z5*z6 + z4*z5*z6*z7
+ z5*z6*z7*z8 + z6*z7*z8*z9 + z7*z8*z9*z0 + z8*z9*z0*z1 + z9*z0*z1*z2;


  z0*z1*z2*z3*z4 + z1*z2*z3*z4*z5 + z2*z3*z4*z5*z6 + z3*z4*z5*z6*z7
+ z4*z5*z6*z7*z8 + z5*z6*z7*z8*z9 + z6*z7*z8*z9*z0 + z7*z8*z9*z0*z1
+ z8*z9*z0*z1*z2 + z9*z0*z1*z2*z3;


  z0*z1*z2*z3*z4*z5 + z1*z2*z3*z4*z5*z6 + z2*z3*z4*z5*z6*z7 + z3*z4*z5*z6*z7*z8
+ z4*z5*z6*z7*z8*z9 + z5*z6*z7*z8*z9*z0 + z6*z7*z8*z9*z0*z1 + z7*z8*z9*z0*z1*z2
```

Appendix C (Continued)

```
+ z8*z9*z0*z1*z2*z3 + z9*z0*z1*z2*z3*z4;


  z0*z1*z2*z3*z4*z5*z6 + z1*z2*z3*z4*z5*z6*z7 + z2*z3*z4*z5*z6*z7*z8
+ z3*z4*z5*z6*z7*z8*z9 + z4*z5*z6*z7*z8*z9*z0 + z5*z6*z7*z8*z9*z0*z1
+ z6*z7*z8*z9*z0*z1*z2 + z7*z8*z9*z0*z1*z2*z3 + z8*z9*z0*z1*z2*z3*z4
+ z9*z0*z1*z2*z3*z4*z5;


  z0*z1*z2*z3*z4*z5*z6*z7 + z1*z2*z3*z4*z5*z6*z7*z8 + z2*z3*z4*z5*z6*z7*z8*z9
+ z3*z4*z5*z6*z7*z8*z9*z0 + z4*z5*z6*z7*z8*z9*z0*z1 + z5*z6*z7*z8*z9*z0*z1*z2
+ z6*z7*z8*z9*z0*z1*z2*z3 + z7*z8*z9*z0*z1*z2*z3*z4 + z8*z9*z0*z1*z2*z3*z4*z5
+ z9*z0*z1*z2*z3*z4*z5*z6;


  z0*z1*z2*z3*z4*z5*z6*z7*z8 + z1*z2*z3*z4*z5*z6*z7*z8*z9
+ z2*z3*z4*z5*z6*z7*z8*z9*z0 + z3*z4*z5*z6*z7*z8*z9*z0*z1
+ z4*z5*z6*z7*z8*z9*z0*z1*z2 + z5*z6*z7*z8*z9*z0*z1*z2*z3
+ z6*z7*z8*z9*z0*z1*z2*z3*z4 + z7*z8*z9*z0*z1*z2*z3*z4*z5
+ z8*z9*z0*z1*z2*z3*z4*z5*z6 + z9*z0*z1*z2*z3*z4*z5*z6*z7;


  z0*z1*z2*z3*z4*z5*z6*z7*z8*z9 - 1;
```

Appendix D

OUTPUT OF SMITH NORMAL FORM EXAMPLE

```
Give number of rows :

Give number of columns :

Please choose the test matrix:

1 random matrix.

2 input your own matrix.

Give entries of 2-by-2 matrix :

Please input the degree of the Polynomial ( 0, 0 )

Please input the polynomial:

Please input the degree of the Polynomial ( 0, 1 )

Please input the polynomial:

Please input the degree of the Polynomial ( 1, 0 )

Please input the polynomial:

Please input the degree of the Polynomial ( 1, 1 )

Please input the polynomial:

The original matrix A :

( 0, 0 )

2.000000000000000e+000  0.000000000000000e+000*i

1.000000000000000e+000  0.000000000000000e+000*i
```

Appendix D (Continued)

```
3.000000000000000e+000  0.000000000000000e+000*i


( 0, 1 )

4.000000000000000e+000  0.000000000000000e+000*i

2.000000000000000e+000  0.000000000000000e+000*i

6.000000000000000e+000  0.000000000000000e+000*i


( 1, 0 )

2.000000000000000e+000  0.000000000000000e+000*i

1.000000000000000e+000  0.000000000000000e+000*i


( 1, 1 )

6.000000000000000e+000  0.000000000000000e+000*i

4.000000000000000e+000  0.000000000000000e+000*i

2.000000000000000e+000  0.000000000000000e+000*i


The Smith form of matrix A is :

( 0, 0 )

1.000000000000000e+000  0.000000000000000e+000*i


( 0, 1 )
```

Appendix D (Continued)

```
0.000000000000000e+000  0.000000000000000e+000*i


( 1, 0 )

0.000000000000000e+000  0.000000000000000e+000*i


( 1, 1 )

-4.000000000000000e+000  -2.220446049250313e-016*i

-6.000000000000000e+000  -3.330669073875470e-016*i

-1.200000000000000e+001  -6.661338147750939e-016*i

-8.000000000000000e+000  -4.440892098500626e-016*i

-6.000000000000000e+000  -3.330669073875470e-016*i


The calculated Smith form with P*A*Q is:

( 0, 0 )

1.000000000000000e+000  0.000000000000000e+000*i


( 0, 1 )

0.000000000000000e+000  0.000000000000000e+000*i


( 1, 0 )

0.000000000000000e+000  0.000000000000000e+000*i
```

Appendix D (Continued)

```
( 1, 1 )

-4.000000000000000e+000  -2.220446049250313e-016*i

-6.000000000000000e+000  -3.330669073875470e-016*i

-1.200000000000000e+001  -6.661338147750939e-016*i

-8.000000000000000e+000  -4.440892098500626e-016*i

-6.000000000000000e+000  -3.330669073875470e-016*i


 P is:

( 0, 0 )

8.333333333333336e-002  0.000000000000000e+000*i


( 0, 1 )

4.166666666666667e-001  2.775557561562891e-017*i

-2.500000000000001e-001  0.000000000000000e+000*i


( 1, 0 )

2.000000000000000e+000  0.000000000000000e+000*i

1.000000000000000e+000  0.000000000000000e+000*i


( 1, 1 )
```

Appendix D (Continued)

```
-2.000000000000000e+000  0.000000000000000e+000*i

-1.000000000000000e+000  0.000000000000000e+000*i

-3.000000000000000e+000  0.000000000000000e+000*i


The determinant of P is:

-1.000000000000000e+000  -8.326672684688674e-017*i

 Q is:

( 0, 0 )

1.000000000000000e+000  -5.551115123125783e-017*i


( 0, 1 )

-2.833333333333334e+000  -1.665334536937735e-016*i

-3.333333333333330e-001  -1.110223024625157e-016*i

-3.333333333333333e-001  -5.551115123125783e-017*i

5.000000000000001e-001  0.000000000000000e+000*i


( 1, 0 )

0.000000000000000e+000  0.000000000000000e+000*i


( 1, 1 )

1.000000000000000e+000  5.551115123125783e-017*i
```

Appendix D (Continued)

```
The determinant of Q is

1.000000000000000e+000  0.000000000000000e+000*i
```

CITED LITERATURE

1. D.C.S. Allison, A. Chakraborty, and L.T. Watson. Granularity issues for solving polynomial systems via globally convergent algorithms on a hypercube. *J. of Supercomputing*, 3:5–20, 1989.

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

3. B. Beckermann and G. Labahn. A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials. *J. Symb. Comp.*, 26:691-714,1998.

4. D. Bini and G. Fiorentino. Design, analysis and implementation of a multi-precision polynomial rootfinder. *J. Symbolic Computation* 33:627-646, 2000.

5. 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.

6. J.V. Burke, A.S. Lewis and M.L. Overton. A nonsmooth, nonconvex optimization approach to robust stabilization by static output feedback and low-order controllers. Proceedings of ROCOND 2003, Milan, 2003.

7. 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.

8. A. Chakraborty, D.C.S. Allison, C.J. Ribbens, and L.T. Watson. The parallel complexity of embedding algorithms for the solution of systems of nonlinear equations. *IEEE Transactions on Parallel and Distributed Systems.* 4(4), 1993.

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

10. M.J. Corless and A.E. Frazho. *Linear Systems and Control.* Marcel Dekker, Inc., 2003 .

11. R. M. Corless, S. M. Watt, and L. Zhi. QR factoring to compute the GCD of univariate approximate polynomials. *IEEE Transactions on Signal Processing,* 52(12):3394-3402, December 2004.

12. Y. Dai, S. Kim and M. Kojima. Computing all nonsingular solutions of cyclic-n polynomial using polyhedral homotopy continuation methods. *J. Comput. Appl. Math.* 152(1-2):83–97, 2003.

13. D.C. Dorf and R.H. Bishop. *Modern Control Systems*. Ninth edition. Addison-Wesley, 1998.

14. J.J. Dongarra and D.C. Sorensen. SCHEDULE: tools for developing and analyzing parallel FORTRAN programs. *ANL-MCS-TM-86*, Argonne National Laboratory, Argonne, IL 60439, 1986.

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

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

17. S. Fortune. An Iterated eigenvalue algorithm for approximating roots of univariate polynomials. *J. Symbolic Computation* 33:627-646, 2002.

18. I. Foster. Designing and Building Parallel Programs. Addison-Wesley Publishing 1995. Available at `http://www-unix.mcs.anl.gov/dbpp/`.

19. T. Gao and T.Y. Li. Mixed volume computation for semi-mixed systems. *Discrete Comput. Geom.* 29(2):257-277, 2003. Available at `http://www.csulb.edu/~tgao` and at `http://www.math.msu.edu/~li`.

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

21. T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa, and T. Mizutani. PHoM – a polyhedral homotopy continuation method for polynomial systems. *Computing.* 73(4):329-348, 2004. Available at `http://www.is.titech.ac.jp/~kojima`.

22. F. B. Hanson and D. C. Sorensen. The SCHEDULE Parallel Programming Package with Recycling Job Queues and Iterated Dependency Graph. *Concurrency: Practice and Experience* 2(1):33-53, 1990.

23. S. Harimoto and L.T. Watson. The granularity of homotopy algorithms for polynomial systems of equations. In G. Rodrigue, editor, *Parallel processing for scientific computing*, pages 115–120. SIAM, 1989.

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

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

26. 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.

27. G. Jäger. Parallel Algorithms for Computing the Smith Normal Form of Large Matrices. In Jack Dongarra, Domenico Laforenza, and Salvatore Orlando, editors, Proceedings of 10th European PVM/MPI 2003, volume 2840 of Lecture Notes in Computer Science, p. 170-179, Springer-Verlag, Berlin-Heidelberg, 2003.

28. C.-P. Jeannerod and G. Labahn. The SNAP package for arithemetic with numeric polynomials. In International Congress of Mathematical Software, World Scientific, pages 61-71, 2002.

29. T. Kailath. *Linear Systems.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1980.

30. J.Kautsky, N.K. Nichols, and P. Van Dooren Robust pole assignment in linear state feedback *Int. J. Control.* 41:1129-1155, 1985.

31. T.Y. Li. Numerical solution of polynomial systems by homotopy continuation methods. In *Handbook of Numerical Analysis. Volume XI. Special Volume: Foundations of Computational Mathematics*, edited by F. Cucker, pages 209–304, 2003.

32. 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.

33. 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.

34. The MathWorks. *Control System Toolbox. For Use with MATLAB. Getting Started.* Version 5. June 2001.

35. A. Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems.* Prentice-Hall, 1987.

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

37. W.F. Moss. Vertical stabilization of a rocket on a movable platform. In *Applied Mathematical Modeling. A multidisciplinary approach*, edited by D.R. Shier and K.T. Wallenius, pages 363–381. Chapman & Hall/CRC 2000.

38. K. Ogata. *Modern Contol Engineering.* Prentice-Hall, 2nd ed., 1990.

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

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

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

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

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

44. J. Rosenthal, J.M. Schumacher, and J.C. Willems. Generic eigenvalue assignment by memoryless real output feedback. *System and Control Letters* 26:253-260, 1995.

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

46. 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.

47. 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.

48. 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.

49. 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.

50. 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.

51. H.J. Su and J.M. McCarthy. Kinematic synthesis of RPS serial chains. In *Proceedings of the ASME Design Engineering Technical Conferences* (CDROM). Paper DETC03/DAC-48813. Chicago, IL, Sept.02-06, 2003.

52. H.J. Su, J.M. McCarthy and L.T. Watson. Generalized linear product polynomial continuation and the computation of reachable surfaces. Technical Report TR-03-24, Computer Science, Virginia Tech.

53. H.J. Su, C.W. Wampler and J.M. McCarthy. Geometric Design of Cylindric PRS Serial Chains. In *Proceedings of the ASME Design Engineering Technical Conferences* (CDROM). Chicago, IL, Sep 2-6, 2003.

54. T. Sterling. *Beowulf Cluster Computing with Windows.* The MIT Press, 2001.

55. A. Takeda, M. Kojima, and K. Fujisawa. Enumeration of all solutions of a combinatorial linear inequality system arising from the polyhedral homotopy continuation method. *J. of Operations Society of Japan* 45:64–82, 2002.

56. J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.* 25(2):251–276, 1999. Software available at `http://www.math.uic.edu/~jan`.

57. 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.

58. J. Verschelde and Y. Wang. Computing Dynamic Output Feedback Laws. *IEEE Trans. Automat. Contr.* 49(8):1393-1397, 2004.

59. J. Verschelde and Y. Wang. Computing Feedback Laws for Linear Systems with a Parallel Pieri Homotopy. Proceedings of the 2004 International Conference on Parallel Processing Workshops: High Performance Scientific and Engineering Computing, Pages 222-229, Montreal, Quebec, Canada, August 15-18, 2004.

60. G. Villard. Fast parallel computation of the Smith normal form of polynomial matrices. In International Symposium on Symbolic and Algebraic Computation, ACM Press, Pages 312 - 317, Oxford, UK, July, 1994.

61. X.A. Wang. Pole Placement by Static Output Feedback. *Journal of Mathematical Systems, Estimation, and Control* 2(2):205-218, 1992.

62. X.A. Wang. Grassmannian, central projection, and output feedback pole assignment of linear systems. *IEEE Trans. on Automatic Control* 41(6):786-794, 1996.

63. Z. Zeng. The approximate GCD of inexact polynomials. Part 1: a univariate algorithm. to appear.

64. Z. Zeng and B.H. Dayton. The approximate GCD of inexact polynomials. Part 2: a multivariate algorithm. Proceedings of ISSAC 2004, Pages 320-327, Santander, Spain, July 4-7, 2004.

VITA

NAME: Yusong Wang

EDUCATION:

* B.E., Applied Physics, University of Science and Technology Beijing, Beijing, P.R. China, 1997
* M.S., Mathematical Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2002
* Ph.D., Mathematical Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2005

EXPERIENCE:

* Research Assistant, Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 2001-2005
* Givens Associate, Mathematics and Computer Science Division, Argonne National Laboratory, Summer, 2004

HONORS:

* Scholarship, University of Science and Technology Beijing, 1994, 1996
* Givens Award, Argonne National Laboratory, 2004

PROFESSIONAL MEMBERSHIP:

* American Mathematical Society
* Society for Industrial and Applied Mathematics

PUBLICATIONS:

* Jan Verschelde and Yusong Wang: Computing Dynamic Output Feedback Laws. IEEE Transactions on Automatic Control 49(8):1393-1397, 2004.
* Jan Verschelde and Yusong Wang: Computing Feedback Laws for Linear Systems with a Parallel Pieri Homotopy. Proceedings of the 2004 International Conference on Parallel Processing Workshops. 15-18 August 2004. Montreal, Quebec, Canada. High Performance Scientific and Engineering Computing. Edited by Yuanyuan Yang. Pages 222-229, IEEE Computer Society, 2004.
* Jan Verschelde and Yusong 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.

SOFTWARE DEVELOPMENT:

* Realization of output feedback, parallel path tracker and parallel Pieri homotopies in PHCpack (ACM TOMS 795)
* Numerical Smith normal form computation for polynomial matrices

PRESENTATIONS:

* Applying Pieri Homotopies to Compute Dynamic Output Feedback Laws on a Parallel Computer. Jointly in the special sessions on "Modern Schubert Calculus" and "Solving Polynomial Systems", 2004 Fall AMS Central Section Meeting, Northwestern University (Evanston, IL), October 23-24, 2004.
* Computing Feedback Laws for Linear Systems with a Parallel Pieri Homotopy. 2004 International Conference on Parallel Processing Workshops, Montreal, Quebec, Canada, August 15-18, 2004.
* A Parallel Path Tracker in PHCpack. SIAM Conference on Parallel Processing and Scientific Computing, San Francisco, February 25-27, 2004.
* Numerical Homotopy Algorithms for Satellite Trajectory Control by Pole Placement. Mathematical Theory of Networks and Systems (CDROM), Notre Dame, August 12-16, 2002.

Index