2003

ISSUES IN THE CONTROL OF HALFSPACE SYSTEMS

Ramprasad Potluri
University of Kentucky, potluri@engr.uky.edu

ABSTRACT OF DISSERTATION

Ramprasad Potluri

The Graduate School

University of Kentucky

2003

ISSUES IN THE CONTROL OF HALFSPACE SYSTEMS

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By
Ramprasad Potluri

Lexington, Kentucky

Director: Dr. Lawrence E. Holloway,
Professor of Electrical & Computer Engineering,
University of Kentucky,

Lexington, Kentucky

2003

ABSTRACT OF DISSERTATION

ISSUES IN THE CONTROL OF HALFSPACE SYSTEMS

By the name HALFSPACE SYSTEMS, this dissertation refers to systems whose dynamics are modeled by linear constraints of the form $Ex_{k+1} \leq Fx_k + Bu_k$ (where $E, F \in \mathfrak{R}^{m\times n}, B \in \mathfrak{R}^{m\times p}$). This dissertation explores the concepts of BOUNDEDNESS, STABILITY, IRREDUNDANCY, and MAINTAINABILITY (which is the same as REACHABILITY OF A TARGET TUBE) that are related to the control of halfspace systems. Given that a halfspace system is bounded, and that a given static target tube is reachable for this system, this dissertation presents algorithms to MAINTAIN the system in this target tube. A DIFFERENCE INCLUSION has the form $x_{k+1} = Ax_k + B_u u_k$, where $x_k, x_{k+1} \in \mathfrak{R}^n$, $u_k \in \mathfrak{R}^p$, $A \in \mathfrak{R}^{n\times n}$, $B_u \in \mathfrak{R}^{n\times p}$, $A^i \in \mathfrak{R}^{n\times n}, B_j \in \mathfrak{R}^{n\times p}$, and A and B_u belong to the convex hulls of $(A^1, A^2, \ldots, A^q)$ and $(B^1, B^2, \ldots, B^r)$ respectively. This dissertation investigates the possibility that halfspace systems have equivalent difference inclusion representation for the case of $u_k = 0$. An affirmitive result in this direction may make it possible to apply to halfspace systems the control theory that exists for difference inclusions.

KEYWORDS: Control of Halfspace systems, Slack-descriptor systems, Difference Inclusions, Matrix Polytopes, Irredundancy.

Ramprasad Potluri

July – 30 – 2003

ISSUES IN THE CONTROL OF HALFSPACE SYSTEMS

By

Ramprasad Potluri

Dr. Lawrence Emory Holloway
(Director of Dissertation)

Dr. William T. Smith
(Director of Graduate Studies)

July – 30 – 2003
(Date)

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this dissertation for use by its patrons is expected to secure the signature of each user.

Name	Date

DISSERTATION

Ramprasad Potluri

The Graduate School

University of Kentucky

2003

ISSUES IN THE CONTROL OF HALFSPACE SYSTEMS

DISSERTATION

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By
Ramprasad Potluri

Lexington, Kentucky

Director: Dr. Lawrence Emory Holloway,
Professor of Electrical & Computer Engineering,
University of Kentucky,

Lexington, Kentucky

2003

ACKNOWLEDGMENTS

This work has been supported in part by NSF grant ECS-9807106, USARO GRANT daah04-96-1-0399, and the Center for Manufacturing Systems at the University of Kentucky.

I thank the Department of Electrical and Computer Engineering at the University of Kentucky for giving me a research assistantship, a teaching assistantship, and a position of a temporary faculty. Without such sources of support this dissertation would not have been possible. I thank the Center for Robotics and Manufacturing Systems at the University of Kentucky for providing me with computing, printing, and office facilities all through the process of my doctoral studies.

I have had the good fortune to work with and learn from great people at the University of Kentucky. Dr. Vijay Singh and Dr. Bill Smith provided me support as a TA and as a temporary faculty and gave me academic freedom that enabled me to experience myself as a teacher and enjoy the process of teaching. I have taken beautiful courses in Control Theory offered by Dr. Ratnesh Kumar, Dr. Michael Marra, Dr. Raymond Rishel, Dr. Bruce Walcott, and Dr. Yu Ming Zhang. Dr. Kumar, Dr. Rishel, Dr. Richard Stockbridge, Dr. Walcott, and Dr. Zhang have served on my advisory committee. I thank all these people for providing me a rich experience.

Much of the present work is about linear inequalities and convex polyhedra. The many hours of discussions I had with Dr. Carl Lee of the Department of Mathematics at the University of Kentucky gave me a better understanding of these topics. I thank him for these discussions.

Hisham Fadel and Yusuf Matcheswala have been close friends and confidants ever since I came to the USA. The walks and talks with Wilda Moore have been stimulating. Ayisha Siddiqua, Fathema Matcheswala, Jon Atabaev, and Vicky Gentry have provided me an extended family in the USA. Arun and Carol Srinivasan have helped me many times with sound personal and professional advice. Melody Lane

and Penny Moore have provided me a patient ear for whatever I have had to say. I thank all these friends for their support.

My parents have been great sources of support throughout my studies in the USA and before that. They have had to live with many years of separation from me while I have been involved in my academic pursuits in the USA and USSR. My grandparents, relatives, and friends have given me a lot of love without which this work would not have been possible.

Seema, my wife, came into my life when I really needed someone to talk to everyday. Her coming has helped me complete this research a lot more quickly than how I was progressing at the time I met her.

I would like to express my sincerest gratitude to Dr. Larry Holloway for giving me this opportunity to work with him. The present dissertation shows only a fraction of all that he has, through personal example and support, made possible for me to achieve intellectually and spiritually.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FILES

potluri_diss.pdf	**1.24 MB**

Chapter 1

Introduction

The present research was supported by a grant titled ACTIVE SENSING POLICIES FOR SYSTEMS WITH OBSERVATION COST. The primary goal of the grant was to develop sensing policies to minimize some sensing cost for systems modeled and sensed with uncertainty, and in which there may be a cost involved in sensing. A secondary goal of the grant was to develop these policies in the SLACK-DESCRIPTOR framework.

Active sensing, as defined by L.E. Holloway in [Hol93], is different from conventional sensing methods in that the focus is on sensing with a view to conserve system resources as the sensing carries a certain cost. Thus, active sensing involves sensing only when absolutely necessary instead of continuously or frequently or periodically. Work in active sensing was suggested by Holloway in the context of smart sensors [Hol93]. Smart sensors have local memory and processing resources that allow them to store and process sensing information until requested. Thus, they also have the potential capability to provide error and warning messages, and interval-valued observations, which can be expressed through linear constraints. Incorporating such linearly-constrained observations into the system dynamics results in the state of the system being set-valued and linearly constrained. To express the resulting set-valued evolution of the state, [Hol93] proposed the slack-descriptor modeling framework. A set of linear inequalities can be transformed into a set of equations by appending slack variables as is done in linear programming. The resulting system has the form of a descriptor system with slack variables attached. Hence the name 'slack-descriptor' systems. When this model was proposed, it was hoped that some of the established results for descriptor systems could be exploited in working with these models.

This dissertation mainly works with the systems of linear inequalities from which slack-descriptor systems are derived. In this dissertation these systems of linear inequalities are called HALFSPACE SYSTEMS because each linear inequality represents a halfspace.

In the years since [Hol93], some results in active sensing were presented using the slack-descriptor framework [LLH95, LLH96]. However, little has been done to relate active sensing to the control of the plant.

As a first step towards developing control techniques in the halfspace framework, MAINTAINABILITY IN A TUBE is considered as the control specification. Let $Tube$ be a subset of states, and let the control goal be to maintain the state of the system within the set $Tube$. Thus, given a $x_0 \in Tube$, find a control u_0 such that $Reach_1(x_0, u_0) \subseteq Tube$. (Here, $Reach_1(x_0, u_0)$ is the set of next possible states from a given current state x_0 under a control u_0). If such a maintaining control exists for each such x_0 , then the system state can be kept within the tube indefinitely. However, the existence of such controls is not guaranteed, in general, because of uncertainty in the system dynamics. A given tube is called MAINTAINABLE if a maintaining control exists for each $x_0 \in Tube$. Algorithms are presented to test for the maintainability of a slack-descriptor system in a given static target tube, as well as to maintain a slack-descriptor system in a maintainable static target tube.

As the next step towards developing control techniques in the halfspace framework, the possibility that halfspace systems may be related to DIFFERENCE INCLUSIONS is explored. Demonstrating that they are related may make it possible to apply the existing difference inclusion control theory to the control of halfspace systems, while also benefiting the difference inclusion theory.

Set-based state estimates have been considered by several other authors. Early work by Witsenhausen considered state sets as general convex sets [Wit68]. Schweppe focused on ellipsoidal state estimate sets [Sch73]. An overview of recent research is found in the edited volume by Kurzhanski and Veliov [KV94]. Several authors have considered set-based state estimates using linear constraints. Shamma and Tu considered the optimal state estimates for uncertain systems with additive uncertainty [SYT89]. Bertsekas and Rhodes [BR71] considered the problems of

REACHABILITY OF TARGET SET and REACHABILITY OF TARGET TUBE. The MAINTAINABILITY IN A TUBE problem is conceptually similar to their second problem with the difference that the tube in the present work is static. For a given dynamical system, Aubin et al [ALQS02] define a set of states, K, as VIABLE if for all initial conditions in K there exists a solution of the dynamical system that remains in K. Maintainability is the same as viability. The present work differs from the works of these authors in that it uses a new modeling framework which has called for new techniques. Another difference is that, in forming a halfspace model, the modeler may not need to distinguish between plant uncertainty and uncertainty due to external disturbances or noise (since a halfspace model neither assumes nor explicitly represents parametric uncertainty alone). This may mean that a halfspace model can potentially simplify a modeler's job.

This dissertation is organized as follows. Chapter 2 describes the halfspace/slack-descriptor modeling framework with a modeling example, and describes prior work done in this area. Chapters 3 and 4 focus specifically on slack-descriptor systems. Chapter 3 presents a test for boundedness of slack-descriptor systems and briefly discusses the relationship between boundedness and stability. This chapter also defines the problem of maintainability. Chapter 4 treats the problem of maintainability of bounded slack-descriptor systems and shows that the set of maintaining controls can be characterized using a matrix Q. This chapter also presents an algorithm to maintain the given slack-descriptor system in a target tube that has been found to be reachable for this system. This algorithm uses the result of Chapter 5. Chapter 5 studies the problem of irredundancy of a system $Ax \leq b$ of linear inequalities and presents ways to determine the set of all values of vector b for which this system is irredundant. Chapters 6 and 7 focus specifically on halfspace systems. Chapter 6 treats the problem of maintainability of halfspace systems. It presents methods both for testing for the reachability of a target tube and for maintaining a halfspace system in a reachable target tube. Chapter 7 investigates the relationship between halfspace systems and difference inclusions. Chapter 8 concludes the dissertation with a discussion of the results and some directions for future research.

Chapter 2

Halfspace Systems and Slack-Descriptor Systems

2.1 How They Are Obtained

The slack-descriptor modeling framework may be especially useful for complex, hard-to-model systems. This framework was proposed as a potential generalization and extension of a difference inclusion and discrete-time interval systems (Chapter 7 is devoted to this connection). The slack-descriptor framework has its roots in Holloway's doctoral research a decade ago [Hol91] and was later developed in several of his works with co-authors [Hol93, LLH95, LLH96, Che98, PH00].

Notation 2.1 In the present work, instances of variables are represented by appending superscripts to the corresponding variables. For example, an instance of x is x^1; instances of $x_1,\ x_0,\ u_0$ are $x_1^1,\ x_0^\nu,\ u_0^s$ etc. However, the superscript ^{T} is reserved for the matrix and vector transpose operation. $[\Gamma]_i$ will represent the i-th row of Γ, where Γ is any matrix or column vector that will be encountered in this dissertation.

The slack-descriptor model has the following form:

$$Ex_{k+1} = Fx_k + Bu_k + V\nu \tag{2.1}$$

where $E, F \in \mathfrak{R}^{m\times n}$, $B \in \mathfrak{R}^{m\times p}$, $V \in \mathfrak{R}^{m\times q}$, $x_{k+1}, x_k \in \mathfrak{R}^n$, $u_k \in \mathfrak{R}^p$, $\nu \in \mathfrak{R}^q$, and $\nu \geq 0$.

For cases where $V = -I$, where $I \in \Re^{m \times m}$ is the identity matrix, Equation (2.1) becomes:

$$Ex_{k+1} \leq Fx_k + Bu_k \tag{2.2}$$

which will be called a HALFSPACE MODEL in this dissertation. For a given value of the pair (x_k, u_k), each inequality in this system of inequalities represents a halfspace (the term "halfspace" is a standard term in linear algebra). Thus, it can be seen that, in halfspace/slack-descriptor models, restrictions on the set of possible state evolutions are imposed using linear constraints.

The halfspace/slack-descriptor model is useful where, due to the difficulty in understanding the system's parameters/dynamics, or due to the presence of external disturbances, the next state of the system could take on any of a set of possible values determined by the current state and control. In the halfspace/slack-descriptor framework, this set of possible values is expressed as a convex polyhedron.

For example, if the dynamics of some plant can be modeled by the constraints

$$\begin{aligned} c1x1_{k+1} + c2x2_{k+1} &\leq c3x1_k + c4x2_k + c5u_k, \\ c6x1_{k+1} + c7x2_{k+1} &\geq c8x1_k + c9x2_k + c10u_k, \end{aligned}$$

where the c's are constants, possibly all different, the x's are states of the plant and u is the control input, then we have a linear constraint description that can be transformed into a slack-descriptor model by appending slack variables thus:

$$\begin{bmatrix} c1 & c2 \\ -c6 & -c7 \end{bmatrix} x_{k+1} = \begin{bmatrix} c3 & c4 \\ -c8 & -c9 \end{bmatrix} x_k + \begin{bmatrix} c5 \\ -c10 \end{bmatrix} u_k + \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} v; \quad v \geq 0.$$

As another example, it can be shown that interval systems in state space form can be transformed into slack-descriptor systems. A discrete-time interval system in state space form is as follows ([SE89] presents the continuous-time form; the discrete-time form is similar):

$$x_{k+1} = Ax_k + Bu_k,$$

with $A \in [A^-, A^+]$, $B \in [B^-, B^+]$ and where $A \in [A^-, A^+]$ implies $a^-_{i,j} \leq a_{i,j} \leq a^+_{i,j}$, etc, and the following matrices are assumed known: $A^-, A^+ \in \Re^{n \times n}$, $B^-, B^+ \in$

$\mathfrak{R}^{n\times m}$. For $x_k, u_k \geq 0$, this system can be converted into the halfspace form by writing as follows:

$$\begin{aligned} x_{k+1} &\leq A^+ x_k + B^+ u_k, \\ x_{k+1} &\geq A^- x_k + B^- u_k. \end{aligned}$$

The slack-descriptor form is as follows:

$$\begin{bmatrix} I \\ I \end{bmatrix} x_{k+1} = \begin{bmatrix} A^+ \\ A^- \end{bmatrix} x_k + \begin{bmatrix} B^+ \\ B^- \end{bmatrix} u_k + \begin{bmatrix} -I & 0 \\ 0 & I \end{bmatrix} v_k.$$

2.2 Slack-Descriptor Systems in Regular Form

Definition 2.1 A slack-descriptor system is in REGULAR FORM if V in Equation (2.1) is a square diagonal matrix with -1's or 0's on the diagonal.

The regular form is obtained when the plant dynamics are described by constraints such as

$$\begin{aligned} [E]_i x_{k+1} &\leq [F]_i x_k + [B]_i u_k, \quad i = 1, \ldots, m-l; \quad l \leq m \\ [E]_i x_{k+1} &= [F]_i x_k + [B]_i u_k, \quad i = m-l+1, \ldots, m. \end{aligned}$$

The equality $a = b$ is equivalent to the pair of inequalities $a \leq b$ and $a \geq b$. Thus, the equalities in the above system can be expressed as pairs of inequalities. Thus, the above system can be written in the form of Expression (2.2).

2.3 An Example of Modeling in Halfspace Framework

The following modeling example is a fictitious one whose purpose is only to show a possible way in which a halfspace model may be generated. The inspiration for this example came from the example in [AP01, page 1].

2.3.1 A Minicase

Heavy Metal Inc. (HMI) is engaged in the production and sale of two grades of an ore. HMI has leased mines from the state and extracts Grade A and Grade B of this ore. Sales of each ore have always been independent of the other.

Marketing and operating costs are covered by two sources: CAPITAL INVESTMENT and, of course, sales.

The direct operating cost (cost of extracting and/or holding) of Grade A is $30 per ton while that of Grade B is $20 per ton. It is anticipated that 45% of Grade A sales and 30% of Grade B sales made during the current production period are collected during the same period and the cash proceeds will be available to finance the next production period's direct operating costs. Additional funds (comprising 40% of capital investment) may be injected to support the direct operating costs if the need arises. Grade A sells to distributive channels for $50 per ton, and Grade B for $40 per ton.

Those sales that are not collected in the same production period are collected at some point in the future and are converted into the above-mentioned capital investment and into investments into the stock market. Thus, these uncollected sales are also considered as revenue in the current production period.

Of the total ore mined in each production period, at least 10% of Grade A and 15% of Grade B does not sell in the same production period. However, for various technical reasons, HMI cannot afford to adopt JUST-IN-TIME practices to reduce the mining of the ore. So, HMI spends 12% of the capital investment and 15% of the capital investment respectively on Grade A and Grade B to convert the ores into the metal, which is then used in some other operations of HMI's sister concerns. The cost of converting into metal is $40 per ton of Grade A and $50 per ton of Grade B.

2.3.2 The Modeling Problem

Assuming any ore that is either produced in the current production period or left over from the previous period as inventory, develop a model that captures the relation between the inventories of Grade A and Grade B, and the capital investment.

Consider any negative numerical values of the quantities of the ore as reflecting backorder.

2.3.3 Developing the Model

Denote by x_{1k} the inventory in tons of Grade A, by x_{2k} the inventory in tons of Grade B of the ore, and by u_k the capital investment in the k-th production period.

The direct cost of producing and/or holding $x_{1(k+1)}$ tons of Grade A and $x_{2(k+1)}$ tons of Grade B is $30x_{1(k+1)}+20x_{2(k+1)}$. This is covered by the funds that are available for production from the k-th production period. These consist of 40% of capital investment, that is $0.40u_k$ plus the anticipated collections on sales of Grade A of $0.45(50x_{1k}) = \$22.5x_{1k}$ plus those on the sales of Grade B of $0.30(40x_{2k}) = \$12x_{2k}$. Thus, the cash available for production in the $(k+1)$-th production period is $0.40u_k+22.5x_{1k}+12x_{2k}$. This gives the inequality:

$$30x_{1(k+1)} + 20x_{2(k+1)} \leq 22.5x_{1k} + 12x_{2k} + 0.40u_k$$

$0.10x_{1k}$ and $0.15x_{2k}$ are the amounts of ore that do not sell in the same production period. A part of these amounts is converted into metal. The number of tons of metal produced internally from Grade A (with 12% of capital investment at the rate of \$40 per ton) is $0.12u_k/40 = 0.003u_k$ and that from Grade B (with 15% of capital investment at the rate of \$50 per ton) is $0.15u_k/50 = 0.003u_k$. After conversion into metal, the amount of Grade A left over from the k-th production period into the $(k+1)$-th production period is at least $0.10x_{1k} - 0.003u_k$, and that of Grade B is at least $0.15x_{2k} - 0.003u_k$. Thus, these values determine the minimum amounts of Grade A and Grade B inventories in the $(k+1)$-th production period:

$$\begin{aligned} x_{1(k+1)} &\geq 0.10x_{1k} - 0.003u_k \\ x_{2(k+1)} &\geq 0.15x_{2k} - 0.003u_k \end{aligned}$$

Thus, HMI's ore operations are described by the following system of linear con-

straints:

$$\begin{array}{rcrcrcrcl} 30x_{1(k+1)} & + & 20x_{2(k+1)} & \leq & 22.5x_{1k} & + & 12x_{2k} & + & 0.40u_k \\ -x_{1(k+1)} & & & \leq & -0.10x_{1k} & + & & & 0.003u_k \\ & & -x_{2(k+1)} & \leq & & - & 0.15x_{2k} & + & 0.003u_k \end{array}$$

This can be written as a halfspace system as follows:

$$\begin{bmatrix} 30 & 20 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_{1(k+1)} \\ x_{2(k+1)} \end{bmatrix} \leq \begin{bmatrix} 22.5 & 12 \\ -0.10 & 0 \\ 0 & 0.15 \end{bmatrix} \begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix} + \begin{bmatrix} 0.40 \\ 0.003 \\ 0.003 \end{bmatrix} u_k$$

It may seem that HMI's operations need capital investment in every production period, and so this may be a losing business. However, the capital investment comes out of the revenue as mentioned in Subsection "A Minicase". Thus, it may be possible that HMI is a self-sustaining business. Whether this is so may be determined by analyzing the halfspace model that has been developed above.

2.4 Previous Work with Slack-Descriptor Systems

The slack-descriptor model was proposed in [Hol93] as a framework that would help properly utilize the potential capabilities of new sensors called smart/intelligent sensors that were emerging on the scene.

Smart sensors are different from ordinary sensors in that into the sensor unit go, besides the sensing device, also memory and processing capabilities. Thus, such sensors can return not just the values of the variable sensed, but also associated confidence intervals, fault alarms, warning messages et al. They can also store sensing information (subject to memory capabilities) until it is requested by the controller. Holloway saw the potential use of these sensors in active sensing, i.e., sensing so as to minimize any cost associated with the sensing.

The interval-valued observations from these sensors would result in set-valued state estimates. It was thought that to fully utilize the information in interval-valued observations, a new way of modeling would be needed that would cast the

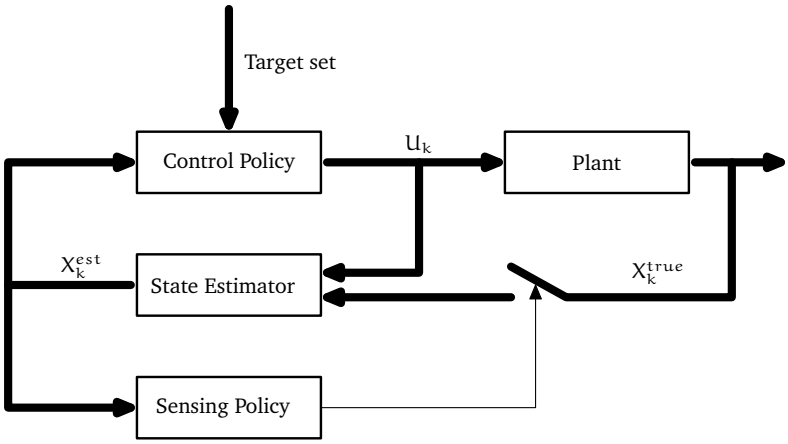


Figure 2.1: The plant under active sensing.

observations and state estimates and state evolution as linearly-constrained. Thus, was born this new modeling framework in which plant dynamics, state estimates, state sets and observed values are described through linear constraints.

In active sensing, three components work together on-line on the plant as shown in Figure 2.1: a control policy, a state estimator, and a sensing policy. The state estimator uses the model of the system dynamics (a slack-descriptor model or some other model) to compute the state sets during the intervals of no-sensing, and supplies the set, X_k^{est}, of possible values of the states to the control policy and to the sensing policy. The control policy uses some approximation techniques to compute the control that should be applied (for example, to lead the system's next state set to the Target set) given that the state could have a value that could lie anywhere in the given set X_k^{est}. The sensing policy checks if the state estimate has exceeded some pre-specified bounds on uncertainty. During the intervals of no-sensing, the system works in open-loop. Thus, the resulting control system works in alternating, and possibly irregular, intervals of open- and closed-loop control.

Lim and Holloway [LLH95, LLH96] proposed sensing policies for active sensing using slack-descriptor models. As mentioned above, the true state of the system at the k-th instant of time can be anywhere in X_k^{est}. This means that the larger the size of X_k^{est}, the more the uncertainty in the true state of the system. [LLH95] pre-

sented two different sensing policies. It was assumed there that each state variable was associated with a different sensor. Under those sensing policies, at each time instant each state variable is maximized and minimized over X_k^{est} (this is a linear programming problem), the difference between the maximum and minimum values for each state variable is computed, and the difference is tested for exceeding a certain threshold value. For whichever variable the corresponding difference exceeds the threshold, the relevant sensor is polled. Thus, different sensors may be polled at different times. This allows for selectively, and possibly infrequently and irregularly, polling the sensors. [LLH96] presented an extension of the results of [LLH95]. It showed that the sensing policies of the type presented in [LLH95] are longest-wait policies in that the sensing does not take place before it is needed.

Later, it was realized that active sensing could work with not just smart sensors, but anywhere where there is uncertainty in the plant dynamics and/or the observations and it is decided to sense irregularly according to a policy. Liu and Holloway [Liu99] proposed active sensing policies for plants modeled not in the slack-descriptor framework, but in the conventional stochastic framework. The plant is modeled by the system of equations:

$$\begin{aligned} Ex_{k+1} &= Ax_k + Bu_k + Gw_k, \\ y_k &= Cx_k + Du_k + Hz_k, \end{aligned}$$

where E, A, B, G, C, D, and H are the parameter matrices of the plant and the x's are states, u's are controls, w's and z's are noises, and y's are the outputs; the matrix E is non-singular and the w's and z's have statistical distributions. The sensing policy requests sensing when the variance of the state estimate crosses a certain threshold.

Rajagopal [Raj95] presented conditions under which a slack-descriptor system will not be bounded. A slack descriptor system is said to be bounded if for any bounded set X_k of states at time instant k and for any bounded set U_k of controls, the corresponding set X_{k+1} is also bounded. He also showed conditions under which a set X_{k+1} would be non-empty for any given nonnegative x_k and u_k, and conditions under which for a given nonnegative x_{k+1} and u_k, there exists some nonnegative x_k

satisfying Equation (2.1). He also tried to relate slack-descriptor models to state-space models. He showed that for $x_k, u_k \geq 0$, the model of Equation (2.1) can be represented as a family of models of the form $x_{k+1} = Ax_k + B_u u_k$.

It was seen above that the state estimator may use the slack-descriptor model to generate X_k^{est}. When the estimator simply uses Equation (2.1) to compute the set of possible current states from the previous set (either X_{k-1}^{est} or X_{k-1}^{true}) of states and the previous inputs, the resulting polyhedron might need a much larger number of constraints to describe it depending on the complexity of the model. Storing all these constraints in the computer will need a lot of memory. Chew [Che98] suggested ways to bound these polyhedra using polyhedra that needed fewer constraints. He characterized the errors that result in the different ways of bounding.

Chew [Che98] used Rajagopal's result — on when one can express a slack-descriptor model as equivalent to a family of $x_{k+1} = Ax_k + B_u u_k$ — to propose a weak way to test for the stability of a slack-descriptor model that satisfies Rajagopal's conditions. He suggested that in some cases it may be possible to test if the eigenvalues of all the (A, B_u) pairs are stable.

Chew [Che98] also presented an analogue for the concept of controllability. He defined a new concept of FULL STATE REACHABILITY: "For any given initial state $x_0 \in \mathfrak{R}^n$ and any target state $x_T \in \mathfrak{R}^n$, a slack descriptor system is full state reachable if there exist a set of inputs that can lead (within n steps and in the absence of sensing) the state x_0 to a state set that includes x_T". He discussed this definition briefly for the case when the model can be expressed as a family of (A, B_u) pairs.

Chew [Che98] also suggested a method for developing a feedback stabilizing controller when the model can be expressed as a family of (A, B_u) pairs.

The problem with the "(A, B_u) family" result is that no clear methods have been suggested by any of the people who worked with slack-descriptor models as to how to find such families for a general model. So, the issues of stability, controllability and stabilizabilty (by developing a feedback controller) of a slack-descriptor model remain open for research.

2.5 Differences between Present Work and Previous Work

The present work differs from previous works done in the slack-descriptor framework in two important aspects:

1. Though the slack-descriptor framework was originally obtained by appending slack variables to linear inequalities, the apparent similarity of the system thus obtained to descriptor systems and the desire to adapt the results existing for descriptor systems to slack-descriptor systems focussed the attention of the previous authors away from the linear inequality origins of the framework and onto the presence of the slack variables. Thus, much of the previous work done in the slack-descriptor framework gave a lot of importance to understanding the effect of the slack variables (for this reason the previous work can be useful for someone interested in understanding the role of slack variables in systems of linear inequalities).

   The present work attempts to understand and exploit the richness of systems of linear inequalities, and the associated theory of convex polyhedra, without the burden of slack variables. When it uses slack variables, it is only to manipulate halfspace models conveniently.

2. Previous works restricted all the states, controls, and slack variables to being nonnegative. Presence of negative variables was dealt with by representing the negative variables as differences of pairs of nonnegative variables, as is common in linear programming. The representation of negative variables in slack-descriptor models is covered in [Che98].

   The present work does not need the states and controls to be nonnegative, except when the halfspace/slack-descriptor systems are obtained from discrete-time interval systems. Most results are developed for all the values of the (x_k, u_k) tuples for which the system of linear inequalities $Ex_{k+1} \leq Fx_k + Bu_k$ is feasible, which may be possible even when x_k and/or u_k are negative.

Chapter 3

Boundedness, Stability, Maintainability

3.1 Reach Sets

Notation 3.1 In the present dissertation, depending on the context, the notations

$$\max_X Y \quad \text{or} \quad \max_X Y$$

read either "maximum of Y over the set X" or "maximize Y over the set X" or "maximize Y subject to the constraints X". Similar notation applies for minimization.

From Expression (2.1) it follows that for a single time-step, a slack-descriptor system can be expressed as

$$Ex_1 = Fx_0 + Bu_0 + Vv_0 \tag{3.1}$$

while a halfspace system can be expressed as

$$Ex_1 \leq Fx_0 + Bu_0 \tag{3.2}$$

Given a state x_0^1 and a control input u_0^1, the set of next states arising from x_0^1 under the control u_0^1 is defined as follows for a slack-descriptor system:

$$\mathrm{Reach}(x_0^1, u_0^1) \triangleq \left\{ x_1 \,\middle|\, \exists v_0 \geq 0 : (x_1, x_0^1, u_0^1, v_0) \text{ is a solution of Equation (3.1)} \right\}$$

and as follows for a halfspace system:

$$\mathrm{Reach}(x_0^1, u_0^1) \triangleq \left\{ x_1 \,\middle|\, (x_1, x_0^1, u_0^1) \text{ is a solution of Equation (3.2)} \right\}$$

In this dissertation, it is assumed that $Reach(x_0^1, u_0^1)$ is bounded and nonempty for all (x_0^1, u_0^1) pairs. The question of boundedness of a slack-descriptor system has been covered in [Raj95]. Section 3.2 will briefly describe an alternative method to test for boundedness.

3.2 Boundedness

A set $S \subset \Re^n$ is BOUNDED if there exists a constant K such that the absolute value of every component of every element of S is less than or equal to K [BT97]. A slack-descriptor system is bounded if for all (x_k, u_k) pairs, $Reach(x_k, u_k)$ is bounded. Remark 5.4 on Page 55 states that the system $Ax \leq b$, $x \in \Re^n$, will be bounded for a bounded b, if the auxiliary system $\{rank(A) = n; A^T z = 0; z \geq 0\}$ is feasible. Applying this result, it can be seen that the halfspace system of Equation (2.2) will be bounded for bounded $Fx_k + Bu_k$ if the auxiliary system $\{rank(E) = n; E^T z = 0; z \geq 0\}$ is feasible. Note that, since $Fx_k + Bu_k$ is only a linear combination of the elements of x_k and u_k, it is always bounded for bounded x_k and u_k.

For this test of boundedness to work for a slack-descriptor system which has been obtained from a halfspace system, and which is not in regular form, the slack-descriptor system must first be transformed into regular form.

3.3 Zero-Input Stability

A discrete-data system is ZERO-INPUT STABLE if the output sequence $y(kT) \in \Re$, $(k = 1, 2, \ldots)$, satisfies the following conditions [Kuo95]:

$$\begin{aligned} &|y(kT)| \leq M < \infty \\ &\qquad \text{and} \\ &\lim_{k \to \infty} |y(kT)| = 0 \end{aligned} \tag{3.3}$$

Classical control theory has other concepts of stability such as ABSOLUTE, ASYMPTOTIC, BIBO, MARGINAL, RELATIVE etc. Analogues of these for halfspace systems

will be studied in a future work. Here, the concept of zero-input stability of halfspace systems will be addressed.

Assuming that the output sequence is the same as the state sequence (that is, $y(kT) = x(kT) \equiv x_k$), we will define the zero-input stability of a halfspace system in regular form as follows.

Definition 3.1 A halfspace system is zero-input stable if for any current state x_k, every next state x_{k+1} that arises from x_k satisfies the following condition (with $\|x\| = \sqrt{x^T x}$ being the distance of x from the origin):

$$\|x_{k+1}\| \leq \|x_k\| \tag{3.4}$$

Note that the condition of Equation (3.4) requires that the distance of the state from the origin be non-increasing, whereas the condition of Equation (3.3) does not require this. In this sense, the condition of Equation (3.4) is stricter than that of Equation (3.3).

Due to the requirement that $\|x_k\|$ be non-increasing, the condition of Equation (3.4) is equivalent to the following condition:

$$\|x_1\| \leq \|x_0\| \tag{3.5}$$

Remark 3.1 Since the next state for a halfspace system is a set, $Reach(x_0, 0)$, the condition of Equation (3.5) is equivalent to the following condition:

$$\|x_0\| \geq \max_{Ex_1 \leq Fx_0} \|x_1\|$$

or equivalently,

$$x_0^T x_0 \geq \max_{Ex_1 \leq Fx_0} x_1^T x_1 \tag{3.6}$$

Definition 3.2 A POLYTOPE is a convex bounded polyhedron.

Definition 3.3 [BT97] Consider a polyhedron P defined by linear equality and inequality constraints, and let x be an element of $\Re^n$.

1. The vector x is a BASIC SOLUTION if:

(a) All equality constraints are active;

(b) Out of the constraints that are active at x, there are n of them that are linearly independent.

2. If x is a basic solution that satisfies all of the constraints, then it is called a BASIC FEASIBLE SOLUTION.

[BT97] first defines VERTEX and BASIC FEASIBLE SOLUTION differently from one another, and then establishes that they are the same. In this dissertation, these two terms will be considered as synonymous to each other.

Lemma 3.1 The point in the polytope P that is farthest from the origin is a vertex of P.

Proof: This is a well-known result. □

Notation 3.2 Let E_n^i (respectively F_n^i) be an $n \times n$ matrix formed of n different rows of E (respectively F), let $I_n \in \Re^{n \times n}$ be the identity matrix, and let

$$\mathbf{C}_n^m = \frac{m!}{(m-n)!n!}.$$

Theorem 3.1 A halfspace system is zero-input stable if, for each invertible E_n^i, $I_n - \left\{(E_n^i)^{-1}F_n^i\right\}^T \left\{(E_n^i)^{-1}F_n^i\right\}$ is positive semidefinite for $i = 1, \ldots, \mathbf{C}_n^m$.

Proof: By Remark 3.1, the halfspace system needs to satisfy the condition of Equation (3.6) for zero-input stability. By Lemma 3.1, this will be true if the vertex of $Reach(x_0, 0)$ that is farthest from the origin satisfies Equation (3.5). This second statement will be true if every vertex of $Reach(x_0, 0)$ satisfies the condition of Equation (3.5). This idea can be developed as follows.

Consider the set $Reach(x_0, 0)$ given by $Ex_1 \leq Fx_0$. Enumerate the basic solutions of this system of constraints. Note that some or all of the basic solutions of $Ex_1 \leq Fx_0$ determine the extreme points of $Reach(x_0, 0)$. Thus, in effect,all the possible vertices of $Reach(x_0, 0)$ are being enumerated. The basic solutions can be

enumerated by simply enumerating all the $\mathbf{C}_n^m$ different subsystems of $Ex_1 \leq Fx_0$ thus:

$$E_n^i x_1 \leq F_n^i x_0, \quad i = 1, \ldots, \mathbf{C}_n^m.$$

Then, the i-th basic solution is $x_1^i = (E_n^i)^{-1} F_n^i x_0$, assuming that E_n^i is invertible.

The condition of Equation (3.5) is equivalent to the following:

$$\|x_1^i\| \leq \|x_0\|, \quad i = 1, \ldots, \mathbf{C}_n^m.$$

This gives the following condition:

$$\|(E_n^i)^{-1} F_n^i x_0\| \leq \|x_0\|, \quad i = 1, \ldots, \mathbf{C}_n^m,$$

that is,

$$x_0^T [((E_n^i)^{-1} F_n^i)^T (E_n^i)^{-1} F_n^i] x_0 \leq x_0^T x_0, \quad i = 1, \ldots, \mathbf{C}_n^m.$$

meaning that $I_n - ((E_n^i)^{-1} F_n^i)^T (E_n^i)^{-1} F_n^i$ is positive semidefinite for $i = 1, \ldots, \mathbf{C}_n^m$. □

3.4 Relationship between Boundedness and Stability

To see the relationship between boundedness and stability, consider a halfspace system where $Reach(x_0^1, 0)$ (x_0^1 belongs to some bounded set Ψ) is bounded but larger than Ψ. The system dynamics evolve into ever-growing but bounded sets of possible next states. Thus, such a slack-descriptor system is not zero-input stable.

If $Reach(x_0^1, u_0^1)$ is unbounded, then the dynamics can evolve without bounds, meaning that the slack-descriptor system is unstable (FOR THIS REASON, ONLY BOUNDED SLACK-DESCRIPTOR SYSTEMS WILL BE STUDIED). Thus, an unbounded slack-descriptor system is clearly unstable, but a bounded slack-descriptor system is not necessarily stable.

3.5 Maintainability

Consider a matrix $T \in \Re^{j\times n}$ and a vector $t \in \Re^j$. Define the subset of states in which we desire to restrict the system as

$$Tube(T,t) \stackrel{\triangle}{=} \{x_k | x_k \text{ satisfies } Tx_k \leq t,\ k = 0,1,\ldots\}.$$

Definition 3.4 If for an x_0^1 there exists a control u_0^1 such that $Reach(x_0^1, u_0^1) \subseteq Tube(T,t)$, then such a control is said to be MAINTAINING for x_0^1. The set of all maintaining controls for an x_0^1 is defined as follows:

$$U(x_0^1) \stackrel{\triangle}{=} \left\{ u_0 \,\middle|\, Reach(x_0^1, u_0) \subseteq Tube(T,t) \right\}.$$

The state set $Tube(T,t)$ is said to be MAINTAINABLE if $\forall x_0 \in Tube(T,t)$, $U(x_0) \neq \emptyset$.

Here is a geometrical explanation for this definition. For any given scalar c_i, consider the set of all points x_1 for which $[T]_i x_1$ is equal to c_i. This is the hyperplane (Figure 3.1) described by the equation $[T]_i x_1 = c_i$. This hyperplane is perpendicular to the vector $[T]_i$. Increasing c_i corresponds to moving the hyperplane along the direction of the vector $[T]_i$. For a given x_0^1, $Reach(x_0^1, u_0^1) \subseteq Tube(T,t)$ means that over the set of all $x_1 \in Reach(x_0^1, u_0^1)$, the maximum value of c_i, denoted by c_i^{max}, in each direction $[T]_i$, is less than or equal to $[t]_i$. That is,

$$\left\{ \begin{array}{rcl} Reach(x_0^1, u_0^1) & \subseteq & Tube(T,t) \\ \forall\ u_0^1 & \in & U(x_0^1) \end{array} \right\} \equiv \left\{ \begin{array}{rcl} c_i^{max} & \leq & [t]_i, \quad i = 1,\ldots,j. \\ \forall\ u_0^1 & \in & U(x_0^1) \end{array} \right\} \quad (3.7)$$

where,

$$c_i^{max} \stackrel{\triangle}{=} \max_{Reach(x_0^1, u_0^1)} [T]_i x_1$$

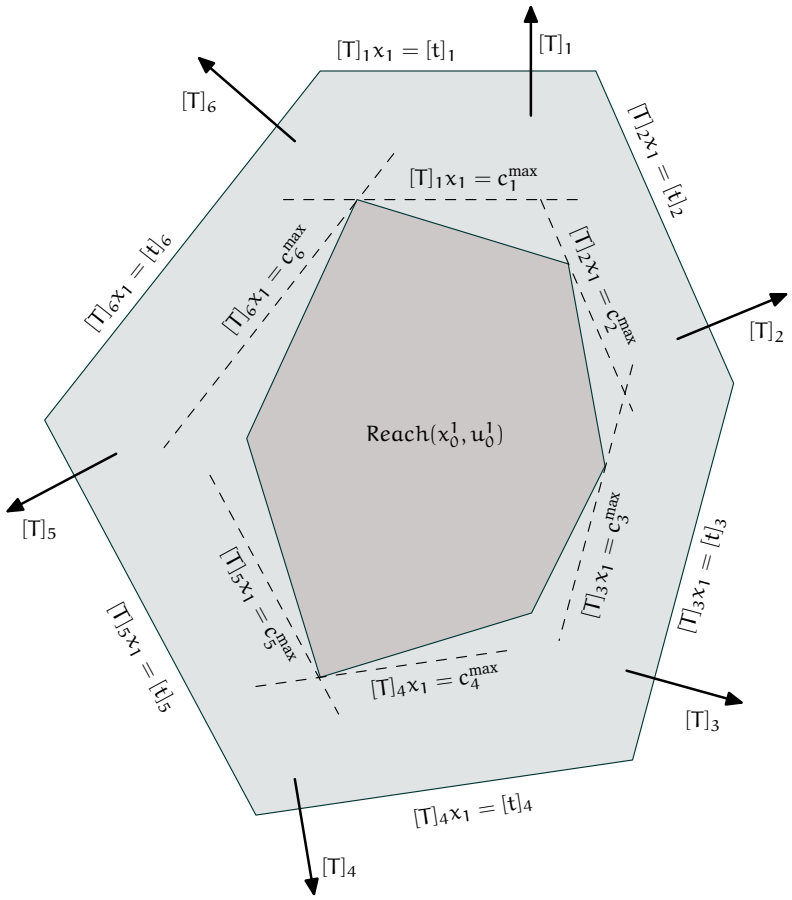


Figure 3.1: Containment of the set $\text{Reach}(x_0, u_0)$ in the tube $Tx_1 \leq t$.

Chapter 4

Maintainability of Slack-Descriptor Systems

4.1 Introduction

The results of this chapter were developed in the early stages of the present research when it was not yet decided to work with halfspace systems. Thus, no assumptions were made about how a slack-descriptor model was obtained — whether through appending slack variables to linear inequalities, or through some other means (as mentioned in Section 2.5, it was later decided to go back to the origins of the SD model and work with linear inequalities). However, it was assumed that they represented convex polyhedral sets. The present chapter discusses maintainability of slack-descriptor systems, while Chapter 6 discusses maintainability of halfspace systems.

4.2 Conditions for Maintainability: Q Matrix

Theorems 4.1 and 4.2 below help develop a test for maintainability of slack-descriptor systems. These theorems use Equation (3.1), and the geometry needed to understand them is provided by Figure 3.1.

Theorem 4.1 If there exists a matrix $Q \in \Re^{j \times m}$ satisfying the conditions that

1. $QE = T$, and

2. QV has all nonpositive elements,

then for a given x_0^1 the following holds:

$$\left\{u_0 \,\middle|\, u_0 \text{ satisfies } QFx_0^1 + QBu_0 \leq t\right\} \subseteq U(x_0^1).$$

Proof: It needs to be shown that if u_0^1 is such that $QFx_0^1 + QBu_0^1 \leq t$, then $u_0^1 \in U(x_0^1)$ (that is, u_0^1 is a maintaining control). To begin, consider any $x_1 \in Reach(x_0^1, u_0^1)$. This x_1 is such that $Ex_1 = Fx_0^1 + Bu_0^1 + V\nu$, and, due to Condition 1, $Tx_1 = QFx_0^1 + QBu_0^1 + QV\nu$. Thus, as $QFx_0^1 + QBu_0^1 \leq t$, so $Tx_1 - QV\nu \leq t$. Condition 2 and the statement in Section 2.1 that $\nu \geq 0$ (slack variables are non-negative) give $QV\nu \leq 0$. Thus, $Tx_1 \leq Tx_1 - QV\nu$. So, $Tx_1 \leq t$. Since this is satisfied for each $x_1 \in Reach(x_0^1, u_0^1)$, then $Reach(x_0^1, u_0^1) \subseteq Tube(T, t)$, so $u_0^1 \in U(x_0^1)$. □

Theorem 4.1 admits the possibility that there exists a u_0^2 such that $u_0^2 \in U(x_0^1)$, but that $u_0^2 \notin \left\{u_0 \,\middle|\, u_0 \text{ satisfies } QFx_0^1 + QBu_0 \leq t\right\}$. Theorem 4.2 shows that if Q satisfies an additional property, then the following equality will hold:

$$\left\{u_0 \,\middle|\, u_0 \text{ satisfies } QFx_0^1 + QBu_0 \leq t\right\} = U(x_0^1).$$

Theorem 4.2 If for a given (x_0^1, u_0^1) pair there exist a matrix Q and slack vectors ν^i, $i = 1, \ldots, m$ satisfying the following conditions:

1. ν^i is such that there exists a tuple $(x_1, x_0^1, u_0^1, \nu^i)$ that satisfies Equation (3.1),
2. Q satisfies the two conditions of Theorem 4.1,
3. $[QV]_i$ is orthogonal to ν^i, $i = 1, \ldots, m$,

then $u_0^1 \in U(x_0^1)$ if and only if u_0^1 satisfies

$$QFx_0^1 + QBu_0^1 \leq t.$$

Proof: The sufficiency part was shown in the proof of Theorem 4.1. The necessity part will be shown here. It will be shown that if $u_0^1 \in U(x_0^1)$ then $QFx_0^1 + QBu_0^1 \leq t$. Suppose that $u_0^1 \in U(x_0^1)$ (that is, u_0^1 is a maintaining control for x_0^1). According to

Definition 3.4 on Page 19, the following is true for u_0^1: $Reach(x_0^1, u_0^1) \subseteq Tube(T, t)$. This implies, according to Expression (3.7), that for u_0^1 the following is true: $c_i^{max} \leq [t]_i,\ i = 1, \ldots, j$. c_i^{max} will be expressed in terms of the other variables and constants. Any $x_1 \in Reach(x_0^1, u_0^1)$ satisfies Equation (3.1) (with $x_0 = x_0^1$ and $u_0 = u_0^1$). Pre-multiplication of Equation (3.1) by Q and application of Condition 1 of Theorem 4.1 shows that such an x_1 satisfies the following equation:

$$Tx_1 = QFx_0^1 + QBu_0^1 + QV\nu \tag{4.1}$$

For each row i of QV, as a consequence of Condition 2 of Theorem 4.1 and the conditions of the present theorem, $[QV]_i\nu$ ranges from a maximum value of zero to some minimum value determined by the system dynamics. So, for any given x_0^1 and u_0^1 the maximum value, c_i^{max}, of $[T]_i x_1 = [QF]_i x_0^1 + [QB]_i u_0^1 + [QV]_i \nu$ is achieved when $[QV]_i \nu = 0$ for some $\nu = \nu^i$, and equals $[QF]_i x_0^1 + [QB]_i u_0^1$. This means that the inequality $c_i^{max} \leq [t]_i$ is equivalent to the inequality $[QF]_i x_0^1 + [QB]_i u_0^1 \leq [t]_i$, for $i = 1, \ldots, m$. Thus, it follows that u_0^1 satisfies $QFx_0^1 + QBu_0^1 \leq t$. □

4.3 Results for Slack-Descriptor Systems in Regular Form

In the rest of this chapter, slack-descriptor systems in regular form (defined in Section 2.2) will be considered. This signifies a move towards the halfspace systems that are studied in the rest of the present dissertation.

4.3.1 Invariance of Q

For a slack-descriptor system in regular form, the $Reach(x_0, u_0)$ sets are polyhedra bounded by the constraints

$$[E]_i x_1 = [F]_i x_0 + [B]_i u_0 + [V]_i \nu, \quad i = 1 \ldots m,$$

where $[V]_i\nu$ goes to zero (meaning that $[\nu]_i$ — the i-th component of ν — equals zero) at points in the x_1 space where the i-th constraint is active (a constraint is said

to be ACTIVE at a point if it is satisfied with equality at that point). The following observations can be made with respect to these $Reach(x_0, u_0)$ sets:

1. Associated with each direction (for example, $[T]_i$) in the x_1 space is a certain group of components of ν that go to zero on the surface of $Reach(x_0, u_0)$.

   For example, consider Figure 4.1. In $Reach(x_0^1, u_0^1)$, as one moves along direction $[T]_2$, the components $[\nu]_1$ and $[\nu]_2$ of ν go to zero.

2. Which components go to zero in a given direction also depends on the (x_0, u_0) pairs because the $Reach$ sets corresponding to different (x_0, u_0) pairs may in general have different shapes while the outward normals of the component constraints are preserved.

   For example, consider Figure 4.1. Both $Reach(x_0^1, u_0^1)$ and $Reach(x_0^2, u_0^2)$ have the same set of outward normals, though in $Reach(x_0^2, u_0^2)$ one of the component constraints is redundant. Along direction $[T]_2$, in $Reach(x_0^1, u_0^1)$ the components $[\nu]_1$ and $[\nu]_2$ of ν go to zero, whereas in $Reach(x_0^2, u_0^2)$ the components $[\nu]_1$ and $[\nu]_3$ of ν go to zero.

3. Equation (4.1) shows that Q represents a series of linear combinations of the rows of Equation (3.1).

4. Assume that there is a vector ν that satisfies Condition 1 of Theorem 4.2. Assume that this ν is associated with the direction $[T]_i$, meaning that if ν corresponds to a point on the surface of the $Reach$ set, then certain components of ν are zeros. Assume that Q satisfies Condition 2 of Theorem 4.2. Then, for this Q to satisfy the third condition of Theorem 4.2, the components of $[QV]_i$ that correspond to the non-zero components of the vector ν, that is associated with the direction $[T]_i$, must be zeros.

   For example, in Figure 4.1 if ν is associated with $[T]_2$ and corresponds to a point on the surface of $Reach(x_0^1, u_0^1)$, then the components ν_1 and ν_2 of ν equal zero. Then, all the components of $[QV]_2$ except the first and the second need to equal zero for this Q to satisfy Condition 3 of Theorem 4.2.

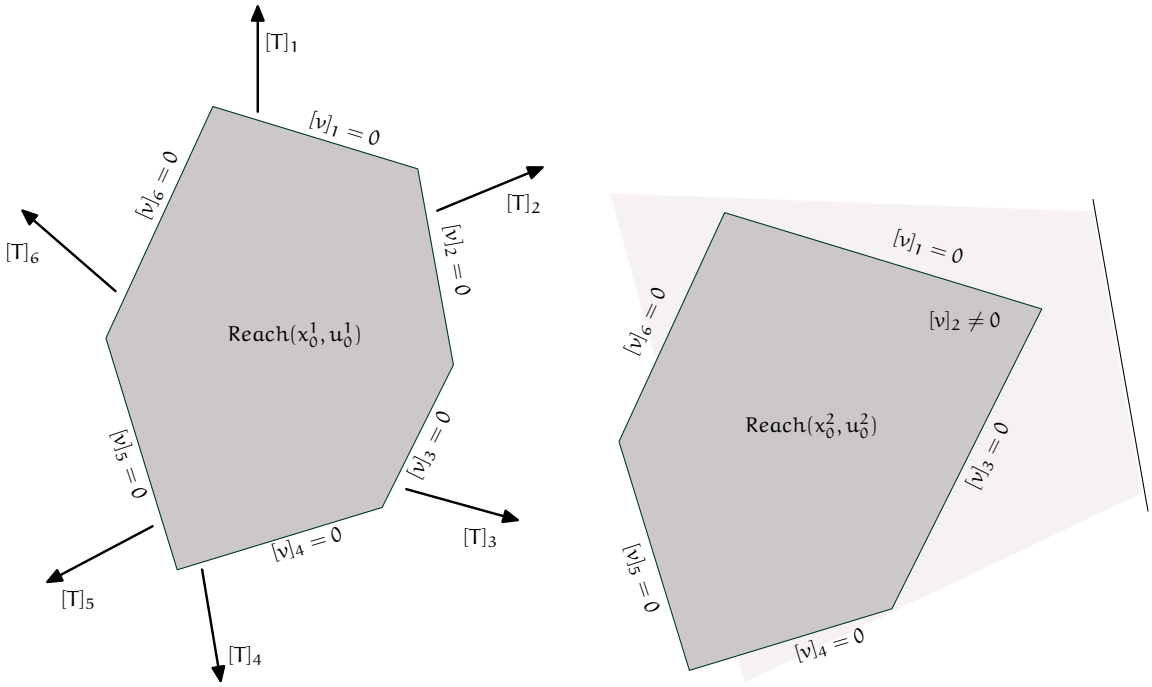


Figure 4.1: Note that every component of ν goes to zero on at least one surface of $Reach(x_0^1, u_0^1)$, but $[\nu]_2$ does not go to zero anywhere in $Reach(x_0^2, u_0^2)$.

In Figure 4.1 if ν is associated with $[T]_2$ and corresponds to a point on the surface of $Reach(x_0^2, u_0^2)$, then the components ν_1 and ν_3 of ν equal zero. Then, all the components of $[QV]_2$ except the first and the third need to equal zero for this Q to satisfy Condition 3 of Theorem 4.2.

5. From the last observation and its example, it follows that a Q matrix that has been determined to satisfy the conditions of Theorem 4.2 for a certain (x_0, u_0) pair may violate them for a different pair.

   For example, with respect to Figure 4.1, a Q satisfying the conditions of The-

orem 4.2 found for (x_0^1, u_0^1) will give the following product for QV:

$$QV = \begin{bmatrix} \sim & 0 & 0 & 0 & 0 & \sim \\ \sim & \sim & 0 & 0 & 0 & 0 \\ 0 & \sim & \sim & 0 & 0 & 0 \\ 0 & 0 & 0 & \sim & \sim & 0 \\ 0 & 0 & 0 & \sim & \sim & 0 \\ \sim & 0 & 0 & 0 & 0 & \sim \end{bmatrix},$$

while a Q found for (x_0^2, u_0^2) will give the following product:

$$QV = \begin{bmatrix} \sim & 0 & 0 & 0 & 0 & \sim \\ \sim & 0 & \sim & 0 & 0 & 0 \\ \sim & 0 & \sim & 0 & 0 & 0 \\ 0 & 0 & 0 & \sim & \sim & 0 \\ 0 & 0 & 0 & \sim & \sim & 0 \\ \sim & 0 & 0 & 0 & 0 & \sim \end{bmatrix}.$$

Here, $\sim$ means that the value of the element that stands in the place of $\sim$ is not important as the component of ν corresponding to that element is equal to zero.

Definition 4.1 For a given slack-descriptor system in regular form, if a single Q is found to satisfy the conditions of Theorem 4.2 for all (x_0, u_0) pairs, then this Q will be called INVARIANT for this slack-descriptor system.

4.3.2 Maintaining Controls under Invariant Q

If Q exists and is invariant, then Theorem 4.2 implies the following:

- For a given x_0,

$$U(x_0) = \{u_0 \,|\, u_0 \text{ satisfies } QFx_0 + QBu_0 \leq t\} \tag{4.2}$$

- The set of all x_0's for which there exist maintaining controls and the set of all maintaining controls are given by

$$QFx_0 + QBu_0 \leq t. \tag{4.3}$$

If no single Q exists that satisfies the conditions of Theorem 4.2 for all (x_0, u_0) pairs, equations (4.2) and (4.3) describe a subset of maintaining controls as per Theorem 4.1.

4.3.3 Test for Maintainability under Invariant Q

Here, a test for the maintainability of a given BOUNDED tube is presented. By definition, a tube is maintainable if for every x_0 in the tube maintaining controls can be found using Equation (4.2). Thus, a test for maintainability might require checking every x_0 in the tube. Clearly, this is impossible. Theorem 4.3 shows, under the assumption of invariance of Q, that if maintaining controls exist for the vertices of the tube, then maintaining controls exist for every point in the tube.

Definition 4.2 Let $P \subseteq \mathfrak{R}^n$ be a polytope. A linear inequality constraint in $\mathfrak{R}^n$ is VALID for P if the constraint is satisfied by all points $x \in P$.

Notation 4.1 Let $P_0 \stackrel{\triangle}{=} \{x_0 | Tx_0 \leq t\}$ be a polytope. Assume that P_0 has a total of L vertices: x_0^1, x_0^2, ..., x_0^L. Let $U(x_0^i) \stackrel{\triangle}{=} \left\{u_0 \middle| QBu_0 \leq t - QFx_0^i\right\}$ be the set of maintaining controls for x_0^i.

Theorem 4.3 Consider Notation 4.1. If $U(x_0^i)$, $i = 1, \ldots, L$, is nonempty, then for any $x_0^p \in P_0$, $U(x_0^p)$ will be nonempty too.

Proof: Any polytope is the convex hull of its vertices. Thus, any point $x_0^p \in P_0$ can be expressed as a convex combination of x_0^i, $i = 1, \ldots, L$. That is,

$$x_0^p = \sum_{i=1}^{L} \gamma_i x_0^i, \text{ where } \sum_{i=1}^{L} \gamma_i = 1, \gamma_i \geq 0. \tag{4.4}$$

It needs to be shown that if the sets $U(x_0^i)$, $i = 1, \ldots, L$, are nonempty, then $U(x_0^p)$ is also nonempty. Multiplying the system of constraints $QBu_0 \leq t - QFx_0^i$ by γ_i, for

$i = 1, \ldots, L$, and adding the L systems we obtain

$$\begin{aligned} \sum_{i=1}^{L} \gamma_i QBu_0 &\leq \sum_{i=1}^{L} \gamma_i t - \sum_{i=1}^{L} \gamma_i QFx_0^i. \qquad (4.5) \\ \Rightarrow QBu_0 \sum_{i=1}^{L} \gamma_i &\leq t \sum_{i=1}^{L} \gamma_i - QF \sum_{i=1}^{L} \gamma_i x_0^i. \\ \Rightarrow QBu_0 &\leq t - QFx_0^p \end{aligned}$$

This last system, by virtue of the non-negative linear combination through which it has been derived, is valid for all $U(x_0^i)$, $i = 1, \ldots, L$, meaning that it is feasible. Since $U(x_0^p) = \{u_0 | QBu_0 \leq t - QFx_0^p\}$, and $QBu_0 \leq t - QFx_0^p$ is feasible, it follows that $U(x_0^p)$ is nonempty too. □

Note that if $U(x_0^i)$ is empty for a vertex x_0^i, then this means that there are no maintaining controls for such an x_0^i. Thus, in such case, an argument based on forming a linear combination such as in Equations (4.5) will not be a valid one, and such a tube is not a maintainable one.

Theorem 4.4 If $x_0^p \in P_0$ is a convex combination of the vertices x_0^i, $i = 1, \ldots, L$, of P_0 as in Equation (4.4), then a convex combination (with γ_i, $i = 1, \ldots, L$ as the coefficients in this combination) of the maintaining controls for these vertices is a maintaining control for x_0^p thus:

$$\left\{ \sum_{1}^{L} \gamma_i u_0^i \,\middle|\, u_0^i \in U(x_0^i),\ i = 1, \ldots, L;\ \gamma_i \text{ satisfy Equation (4.4)} \right\} \subseteq U(x_0^p).$$

Proof: As $u_0^i \in U(x_0^i)$, u_0^i satisfies the inequality $QBu_0 \leq t - QFx_0^i$. This gives the following nonnegative sum in which the γ_i from Equation (4.4) has been used:

$$\sum_{i=1}^{L} \gamma_i QBu_0^i \leq \sum_{i=1}^{L} \gamma_i t - \sum_{i=1}^{L} \gamma_i QFx_0^i.$$

This reduces to the following expression:

$$QB \sum_{i=1}^{L} \gamma_i u_0^i \leq t - QFx_0^p.$$

Thus, it follows that

$$\sum_{i=1}^{L} \gamma_i u_0^i \in U(x_0^p).$$

Hence the proof. □

Using the result of Theorem 4.4 a maintaining control for any $x_0^p \in P_0$ can be determined as follows:

Step 1: Determine maintaining controls for each of the vertices of $P_0 \triangleq \{x_0 | Tx_0 \leq t\}$.

Step 2: Use Equation (4.4) to determine $\gamma = [\gamma_1, \ldots, \gamma_L]$, such that x_0^p is a convex combination of the vertices.

Step 3: Find u_0^p as a convex combination of the vertices' maintaining controls involving this γ.

The next section presents a method to determine Q assuming that it is invariant.

4.4 Finding an invariant Q Matrix

This section presents steps for determining an appropriate Q matrix. The approach is to determine a Q in steps in order to satisfy the conditions of theorems 4.1 and 4.2. In Step 1 a Q is found that satisfies Condition 1 of Theorem 4.1. In Step 2, this Q is modified using vectors from the null space of E^T so that Condition 2 of Theorem 4.1 is also satisfied. By Theorem 4.1, such a Q can be used at least to characterize a subset of $U(x_0)$. Finally, in Step 3, for a given (x_0, u_0) pair, an optimization is performed in order to modify Q until it satisfies the conditions of Theorem 4.2 also, thus allowing the set $U(x_0)$ to be represented exactly. For each step, the properties of the system that allow the transformations of Q to be found are discussed.

Step 1: Begin with any matrix Q_0 such that $T = Q_0 E$.

If E is of full column rank, then such a Q_0 can be determined as $Q_0 = [T \mid A]E_R$ where $A \in \Re^{j\times(m-n)}$ is any matrix and $E_R \in \Re^{m\times m}$ is the transformation matrix that puts the composite matrix $[E \ -F \ -B \ -V]$ into its reduced-row echelon form.

If E is not of full column rank, this means that the given SD system is unbounded as per Section 3.2; such a case will be ignored.

If the row space of T is not in the row space of E, then the dynamics of the system will lead to a next state x_1 that is unbounded in a direction that should be constrained by the tube. In such a case, no Q_0 exists, and the set $U(x_0)$ will be empty.

Step 2: In this step, it will be seen if Condition 2 of the theorem can be satisfied. Define Q_{null} such that the columns of Q_{null}^T are a basis for the null space of E^T. Thus, $Q_{null}E = 0$. If a matrix R_1 can be found such that $(Q_0 + R_1Q_{null})V$ has all nonpositive elements, then Condition 2 will be satisfied, that is, if the feasible region of the following linear constraints is nonempty:

$$\begin{bmatrix} (Q_{null}V)^T & 0 & \cdots & 0 \\ 0 & (Q_{null}V)^T & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & (Q_{null}V)^T \end{bmatrix} \begin{bmatrix} [R_1]_1^T \\ [R_1]_2^T \\ \cdots \\ [R_1]_j^T \end{bmatrix} \leq - \begin{bmatrix} [Q_0V]_1^T \\ [Q_0V]_2^T \\ \cdots \\ [Q_0V]_j^T \end{bmatrix}$$

Choose a matrix R_1 such that its elements satisfy the above constraints. Define $Q_1 = (Q_0 + R_1Q_{null})$. Then,

$$Tx_1 = Q_1Fx_0 + Q_1Bu_0 + Q_1Vv.$$

Note that Q_1 satisfies conditions 1 and 2 of Theorem 4.1. It is thus sufficient to characterize a subset of maintaining controls. In the next step, Q_1 will be modified to satisfy, if possible, the condition of Theorem 4.2.

Step 3: In this step, a random (x_0, u_0) pair is chosen and attempt is made to determine a matrix R_2 such that $Q_2 = Q_1 + R_2Q_{null}$ satisfies all the three conditions of the theorem.

The condition of Theorem 4.2 states that for each row $[QV]_i$, there must exist some ν satisfying Equation (3.1) for the above (x_0, u_0) pair such that ν is orthogonal to $[QV]_i$. Two non-negative vectors are orthogonal to each other if each non-zero element of the first vector has a corresponding zero element in the second vector. This suggests the following strategy. In step 3a, for each direction $[T]_i$, the associated slack vector, which is such that some of its components equal zero, will be determined. Such a slack vector will be called the DESIRED VECTOR associated with the direction $[T]_i$ or DESIRED VECTOR-i for short. Then, in step 3b, each row $[R_2]_i$ of R_2 will be found through optimization so as to minimize the components of $[Q_2V]_i$ corresponding to the nonzero components of the desired vector-i. This strategy is detailed below.

Step 3a: The $Reach(x_0^1, u_0^1)$ polyhedron is described by Equation (3.1) which is in regular form. Thus, any slack variables that equal zero will be associated only with points on the surface of the polyhedron. So, each desired vector will be associated with points that are only on the surface of the polyhedron. The desired vector-i can be determined by solving the linear programming problem

$$\max_{\substack{\text{Equation (3.1)} \\ x_0 = x_0^1 \\ u_0 = u_0^1}} [T]_i x_1$$

This gives a vector $[x_1^T \quad \nu^T]^T$. The ν part of this vector is the desired vector-i. This procedure is repeated for each of the j rows of T.

Step 3b: In Step 3a the desired vectors were determined. In the present step, attempt will be made to determine the matrix R_2 such that the i-th row of the matrix $Q_2V = (Q_1 + R_2Q_{null})V$ can form an orthogonal product with the desired vector-i. This can be done by maximizing the sum of the elements of the i-th row of $(Q_1 + R_2Q_{null})V$ that correspond to the non-zero components of the desired vector-i subject to the constraints that the elements of $(Q_1 + R_2Q_{null})V$ be nonpositive. Maximization brings these elements in the

direction of zero. This is repeated for each of the j rows of $(Q_1 + R_2 Q_{null})V$ to obtain Q_2.

If Step 3b fails, then one can settle for the Q_2 thus determined and one would have a subset of maintaining controls.

4.5 Special Cases where Q is Invariant

Here one general scenario is described in which Q is invariant across all (x_0, u_0) pairs while satisfying the conditions of Theorem 1 and Theorem 2. Q will be invariant when, irrespective of the (x_0, u_0) pairs, in every desired vector associated with any given direction in the x_1 space, the indices of the zero-components are the same. This happens when every constraint from Equation (3.1) forms a face of the $Reach(x_0^1, u_0^1)$ polyhedron for every (x_0, u_0) pair. In other words, none of the constraints becomes redundant, even when the shape changes as (x_0, u_0) pairs change. The solution to the problem of determining the set of all b's for which the system of inequalities $Ax \leq b$ is irredundant will be presented in Chapter 5.

A special case of an irredundant $Reach$ set is one where changing (x_0, u_0) only scales the polyhedron up or down without changing its shape. This happens when all rows of the right hand side of the expression

$$Ex_1 - Vv = Fx_0 + Bu_0,$$

are equally scaled for different (x_0, u_0) pairs. That is,

$$\frac{[F]_k x_0^1 + [B]_k u_0^1}{[F]_k x_0^2 + [B]_k u_0^2} = \frac{[F]_l x_0^1 + [B]_l u_0^1}{[F]_l x_0^2 + [B]_l u_0^2}$$
$$\text{for } k = 1 \ldots m, \;\; l = 1 \ldots m.$$

A little manipulation of the above expression gives the following result:

$$\frac{a^k_{m_1}}{a^k_{n_1}} = \frac{a^l_{m_1}}{a^l_{n_1}} \;\; \text{for } k = 1 \ldots m, \;\; l = 1 \ldots m,$$
$$m_1 = 1 \ldots (n+p), \;\; n_1 = 1 \ldots (n+p).$$

Here, $a^k_{m_1}$ and $a^k_{n_1}$ are respectively the m_1-th and n_1-th elements of the k-th row of the composite matrix $[F \;\; B]$.

4.6 Example

Consider a simple single-state system. Assume that the dynamics of the system can be represented by the following inequalities:

$$0.8x_k + u_k \leq x_{k+1} \leq 0.9x_k + u_k.$$

The corresponding slack-descriptor equation is

$$Ex_{k+1} = Fx_k + Bu_k + Vv,$$

where

$$E = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ F = \begin{bmatrix} 0.9 \\ 0.8 \end{bmatrix},\ B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, V = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Of interest is whether the tube defined by $Tx_k \leq t$, where

$$T = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \text{and} \quad t = \begin{bmatrix} 0.9 \\ -0.8 \end{bmatrix}.$$

is maintainable with respect to this system.

This solution will go through the steps of the algorithm of Section 4.4.

Step 1: Find a Q_0.

$$[T\ \ A] = \begin{bmatrix} 1 & -2 \\ -1 & 2 \end{bmatrix},\ E_R = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}, Q_0 = \begin{bmatrix} -1 & 2 \\ 1 & -2 \end{bmatrix}.$$

Step 2:

$$Q_{null}^T = \begin{bmatrix} -1 \\ 1 \end{bmatrix},\ Q_0V = \begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix},\ (Q_{null}V)^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

On testing the constraints shown in Step 2 of Section 4.4, it can be seen that they are satisfied. Then,

$$R_1 = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

is chosen such that these constraints are satisfied. Then,

$$Q_1 = \begin{bmatrix} 1 & 0 \\ 1 & -2 \end{bmatrix}.$$

Step 3a: The constraints in the slack-descriptor model of the system are linearly independent. So, neither of them becomes redundant for any (x_0, u_0) pair. Thus, any Q matrix that is determined will be invariant as described in Section 4.5. So, $x_0 = 1$ and $u_0 = 1$ are chosen randomly. The desired vector-1 can be found by solving the linear programming problem

$$\max_{\substack{\text{Equation (3.1)} \\ x_0 = 1 \\ u_0 = 1}} x_1$$

This gives the desired vector $[0 \;\; 1]^T$ in the direction $[T]_1 = 1$. Similarly, in the direction $[T]_2 = -1$, the desired vector $[1 \;\; 0]^T$ is obtained.

Step 3b: $Q_2 = Q_1 + R_2 Q_{null}$. Thus, it follows that

$$Q_2 = \begin{bmatrix} 1 - r_1 & r_1 \\ 1 - r_2 & -2 + r_2 \end{bmatrix} \text{ and } Q_2 V = \begin{bmatrix} -1 + r_1 & r_1 \\ -1 + r_2 & -2 + r_2 \end{bmatrix}.$$

Solve the problems

$$\max_{\substack{-1+r_1 \le 0 \\ r_1 \le 0}} r_1 \quad \text{and} \quad \max_{\substack{-1+r_2 \le 0 \\ -2+r_2 \le 0}} -1 + r_2$$

to get

$$R_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

This gives

$$Q_2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \text{ and } Q_2 V = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Thus, $Q = Q_2$. Now, constraints similar to those in Expression (4.3) can be formed:

$$\begin{bmatrix} 0.9 \\ -0.8 \end{bmatrix} x_0 + \begin{bmatrix} 1 \\ -1 \end{bmatrix} u_0 \leq \begin{bmatrix} 0.9 \\ -0.8 \end{bmatrix}.$$

The vertices of the tube are 0.8 and 0.9. Thus, to test if the given tube is maintainable, solve the last expression for controls at these two vertices only (as per Theorem 4.3). It is found that the tube is maintainable. Solving for the set $U(x_0)$ using these constraints shows that u_0 ranges from 0.08 to 0.18 for various values of x_0 in the tube.

4.7 Drawback of using a non-invariant Q

As Q is determined for one particular Reach set (specifically, an irredundant Reach set), $QFx_0^p + QBu_0^p$ may turn out to be not less than or equal to t even when $Reach(x_0^p, u_0^p) \in Tube(T, t)$. This idea is explained in the present section.

Consider figures 4.2 and 4.3. Note that $Reach(x_0^1, u_0^1) \subseteq Tube(T, t)$ and $Reach(x_0^2, u_0^2) \subseteq Tube(T, t)$. Assume that the system $Ex_1 \leq Fx_0 + Bu_0$ comprises six inequality constraints as shown in the figure, with z^1 and z^2 as follows:

$$z^1 = \begin{bmatrix} x_0^1 \\ u_0^1 \end{bmatrix}, z^2 = \begin{bmatrix} x_0^2 \\ u_0^2 \end{bmatrix}$$

When Q is determined for some (x_0^s, u_0^s) for which these six constraints are irredundant, Q is as follows (by inspection of Figure 4.2):

$$Q = \begin{bmatrix} 0 & 0 & 0 & q_{14} & q_{15} & 0 \\ 0 & q_{22} & q_{23} & 0 & 0 & 0 \\ q_{31} & 0 & 0 & 0 & 0 & q_{36} \\ 0 & 0 & 0 & 0 & q_{45} & q_{46} \end{bmatrix}$$

Here, q_{ij} are non-negative numbers. Q is such that $QE = T$.

The dotted lines in figures 4.2 and 4.3 represent the inequalities $QEx_1 \leq QFx_0^1 + QBu_0^1$ and $QEx_1 \leq QFx_0^2 + QBu_0^2$ respectively.

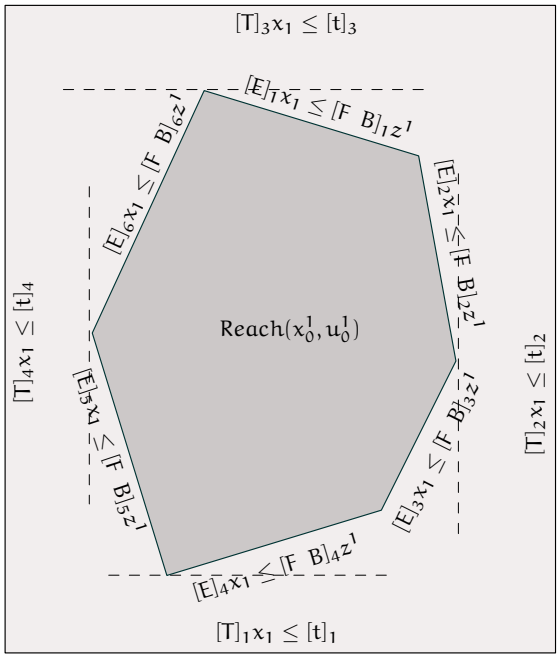


Figure 4.2: Illustration for the drawback of a non-invariant Q. $QFx_0^1 + QBu_0^1 \le t$ and $\mathrm{Reach}(x_0^1, u_0^1) \subseteq \mathrm{Tube}(T, t)$ (the rectangular region is $\mathrm{Tube}(T, t)$).

When this Q is used to test the condition $QFx_0 + QBu_0 \le t$, here is what it tests for $\mathrm{Reach}(x_0^1, u_0^1)$:

$$\begin{aligned}
(q_{14}[F]_4 + q_{15}[F]_5)x_0^1 + (q_{14}[B]_4 + q_{15}[B]_5)u_0^1 &\overset{?}{\le} [t]_1 \\
(q_{22}[F]_2 + q_{23}[F]_3)x_0^1 + (q_{22}[B]_2 + q_{23}[B]_3)u_0^1 &\overset{?}{\le} [t]_2 \\
(q_{31}[F]_1 + q_{36}[F]_6)x_0^1 + (q_{31}[B]_1 + q_{36}[B]_6)u_0^1 &\overset{?}{\le} [t]_3 \\
(q_{45}[F]_5 + q_{46}[F]_6)x_0^1 + (q_{45}[B]_5 + q_{46}[B]_6)u_0^1 &\overset{?}{\le} [t]_4
\end{aligned}$$

This test passes successfully. Here is what it tests for $\mathrm{Reach}(x_0^2, u_0^2)$:

$$\begin{aligned}
(q_{14}[F]_4 + q_{15}[F]_5)x_0^2 + (q_{14}[B]_4 + q_{15}[B]_5)u_0^2 &\overset{?}{\le} [t]_1 \\
(q_{22}[F]_2 + q_{23}[F]_3)x_0^2 + (q_{22}[B]_2 + q_{23}[B]_3)u_0^2 &\overset{?}{\le} [t]_2 \\
(q_{31}[F]_1 + q_{36}[F]_6)x_0^2 + (q_{31}[B]_1 + q_{36}[B]_6)u_0^2 &\overset{?}{\le} [t]_3 \\
(q_{45}[F]_5 + q_{46}[F]_6)x_0^2 + (q_{45}[B]_5 + q_{46}[B]_6)u_0^2 &\overset{?}{\le} [t]_4
\end{aligned}$$

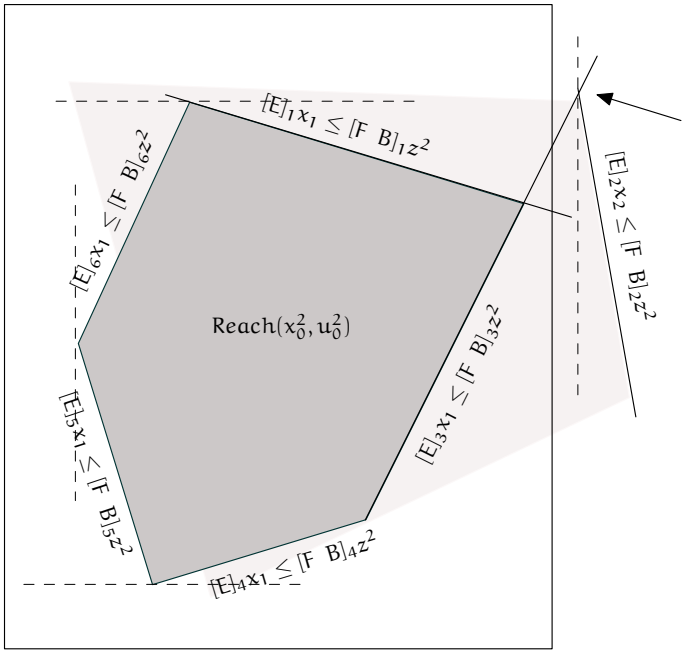


Figure 4.3: Illustration for the drawback of a non-invariant Q. $QFx_0^2 + QBu_0^2 \not\leq t$ even though $Reach(x_0^2, u_0^2) \subseteq Tube(T, t)$ (the rectangular region is $Tube(T, t)$).

Note that, instead of testing whether the intersection of the constraints $[E]_1x_1^2 \leq [F]_1x_0^2 + [B]_1u_0^2$ and $[E]_3x_1^2 \leq [F]_3x_0^2 + [B]_3u_0^2$ is "below" $[T]_2x_1 \leq [t]_2$ (in which case it would be confirmed that $Reach(x_0^2, u_0^2) \subset Tube(T, t)$), this Q tests if the intersection of $[E]_2x_1^2 \leq [F]_2x_0^2 + [B]_2u_0^2$ and $[E]_3x_1^2 \leq [F]_3x_0^2 + [B]_3u_0^2$ (indicated by the arrow in Figure 4.3) is "below" $[T]_2x_1 \leq [t]_2$. Clearly, from Figure 4.3, this test will be answered in the negative.

Thus, in this subsection, an example was presented to illustrate the possibility that a Q determined for an irredundant Reach set $Reach(x_0^1, u_0^1)$ of a given Slack-descriptor system may test correctly whether other irredundant Reach sets of this Slack-descriptor system are subsets of $Tube(T, t)$, but that this Q may identify a redundant Reach set $Reach(x_0^2, u_0^2)$ as not a subset of $Tube(T, t)$ even though $Reach(x_0^2, u_0^2) \subseteq Tube(T, t)$.

4.8 Discussion

In this chapter it was shown that for a system whose dynamics are described through linear constraints, the set of controls that maintain the system's states in a linearly constrained tube can be described by linear constraints if the dynamics satisfy certain conditions. It was explained when this description can be exact and a case was identified for when this is possible (Chapter 5 studies this case in greater depth). A method has been presented to test if a given tube is maintainable for a given slack-descriptor model. Also, an expression has been shown (Equation (4.3)) that can be used on-line to compute controls that would maintain system states within the given maintainable tube.

Note that a slack-descriptor system in regular form is equivalent to a halfspace system. Chapter 6 will study maintainability of halfspace systems without involving a Q.

Chapter 5

Determining the Values of b for which $Ax \leq b$ is Irredundant

5.1 Introduction

The need to study redundancy was motivated in Section 4.5. From Section 3.1, it can be seen that a $Reach_1(x_0^1, u_0^1)$ set is a polyhedron defined by Expression (3.1) as follows:

$$Ex_1 = Fx_0^1 + Bu_0^1 + V\nu. \tag{5.1}$$

The problem studied in Chapter 4 was: given a convex set of states denoted $Tube(T, t) \triangleq \{x \in \Re^n \,|\, Tx \leq t, T \in \Re^{j\times n}, t \in \Re^j\}$ and an $x_0^1 \in Tube(T, t)$, find a u_0^1 such that $Reach_1(x_0^1, u_0^1) \subseteq Tube(T, t)$. If a u_0^1 exists for each x_0^1, then the system states can be maintained in the tube indefinitely.

It was seen in Chapter 4 that if a matrix $Q \in \Re^{j\times m}$ can be found such that through premultiplication by Q, Expression (5.1) can be rewritten as: $Tx_1 = QFx_0^1 + QBu_0^1 + QV\nu$, such that $QV\nu \leq 0$ and each row of $QV\nu$ is assured to take on a value of zero, then this last equation gives: $Tx_1 \leq QFx_0^1 + QBu_0^1$. Then the constraint $QFx_0^1 + QBu_0^1 \leq t$ allows one to determine the values of (x_0^1, u_0^1) pairs for which $Reach_1(x_0^1, u_0^1) \subseteq Tube(T, t)$. This result, by extension, helps determine if the evolution of the states of the system can be maintained in the given tube.

In the method proposed in Chapter 4 to determine Q, the value of Q depends on the value of the (x_0^1, u_0^1) pair. It was concluded there that if Q could somehow be independent of (x_0^1, u_0^1) pairs, then a Q determined for any one pair would work for

the particular slack-descriptor model as a whole. Even if Q were not independent as described, at least the set of all (x_0^1, u_0^1) pairs for which the Expression (2.2) remains irredundant could be determined, and for this set of values a Q could be determined. This was better than finding a Q for a random (x_0^1, u_0^1) pair and not knowing 'how far' it will work. Thus, this was the motivation to study redundancy.

The question of redundancy of a system of linear inequality constraints has been studied for more than a half century now. The first comprehensive survey on this topic was done in [KLTZ83]. Recently [Gre96] did another survey. These and other works have focussed on REDUNDANCY DETECTION — determining if a system of linear inequality constraints is redundant and, if so, which constraints are redundant — and/or REDUNDANCY ELIMINATION — obtaining a minimized (irredundant) system of inequality constraints whose feasible region is the same as that of the original system. In contrast, in this paper, the interest is in characterizing the domain of values over which a system is irredundant.

The techniques used in the literature for redundancy detection and/or elimination fall into two broad categories – DETERMINISTIC TECHNIQUES and PROBABILISTIC TECHNIQUES. Given a system of linear inequality constraints, a deterministic method, in its most naive form, determines if a constraint is redundant by solving a linear programming (LP) problem (LPP) [CBB97]. This means that the computational complexity of the deterministic methods is polynomial. In practice, many simplifying observations are used to classify as many constraints as possible as redundant/irredundant during the solution of each LPP. There are also some non-LP-based deterministic methods which, while not as comprehensive in classifying the constraints as the LP-based methods, are quicker. The probabilistic methods are based on the fact that, if a random line intersects the interior of the feasible region of the given system of constraints, then with probability one the end points of the feasible segment of that line identify necessary or weakly redundant constraints.

Consider the following systems of constraints:

$$\left.\begin{array}{rcl} Ax & \leq & By, \\ y & \in & \mathfrak{R}^{n+p},\ x \in \mathfrak{R}^{n}. \\ \text{Constants: } A & \in & \mathfrak{R}^{m\times n},\ B \in \mathfrak{R}^{m\times(n+p)}. \end{array}\right\} \tag{5.2}$$

Note that Expression (5.2) defines, not one system of constraints, but SYSTEMS, each system being defined by a particular value of y. It is similar to Expression (2.2). It can also model the non-negativity constraints on x. In this paper, the polyhedra defined by this expression are required to be bounded. The problem of determining the set of all (x_0, u_0) pairs for which Expression (3.2) remains irredundant can be formulated more generally as

$$\mathcal{PR}1: \quad \text{Determine } \{y \mid \text{The systems of Expression (5.2) are irredundant}\},$$

or simply as

$$\mathcal{PR}2: \quad \text{Determine } \{b \mid \mathcal{S}(b) \text{ is irredundant}\},$$

where

$$\mathcal{S}(b) \triangleq (Ax \leq b,\ x \in \mathfrak{R}^n), \quad b \in \mathfrak{R}^m. \tag{5.3}$$

The redundancy-related techniques described above process constraints sequentially and, so, are not suited to solve $\mathcal{PR}1$ or $\mathcal{PR}2$.

The work which comes closest to solving $\mathcal{PR}1$ or $\mathcal{PR}2$ is [LW97]. [LW97] presents an algorithm to determine the vertices of a polyhedron, such as defined by Expression (5.2), as functions of the parameter y. That algorithm works approximately as follows. In the first step, it converts the "parametrized polyhedron" of Expression (5.2) into a homogeneous polyhedron of the form $[\ A \quad -B\ ][\ x \quad y\ ]^T \leq 0$. In the second step, the $(n+p)$-FACES of the last polyhedron are determined using a face-enumeration algorithm developed by [LW97]. In the third step, the $(n+p)$-FACES are projected onto the n-dimensional space to obtain the "parametrized vertices". The complexity of this algorithm is bounded by a polynomial in $n+p$. That work can provide a first approximation to our problem as follows: the set of all values of y over which every one of the parametrized vertices is preserved is a subset of the solution to our problem. The solution may not be exact in some cases as can be seen in Figure 5.1: this figure shows polytopes represented by irredundant systems corresponding to Expression (5.2); these polytopes have different number of vertices. In the present chapter, two methods that solve this problem exactly

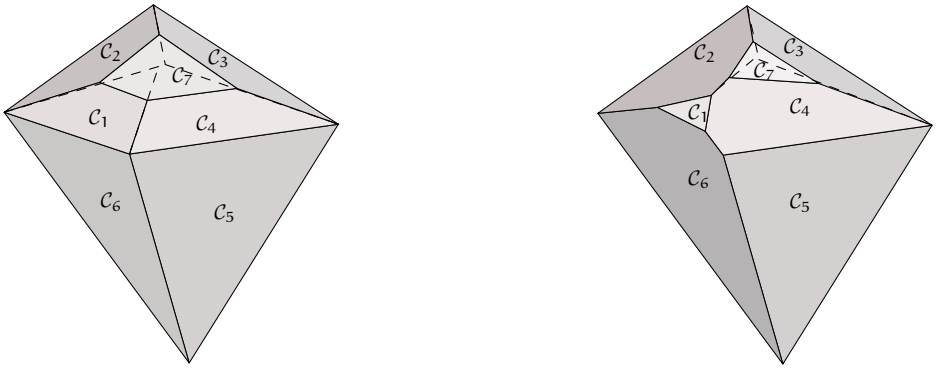


Figure 5.1: Polytopes represented by irredundant systems corresponding to Expression (5.2): these polytopes have different number of vertices.

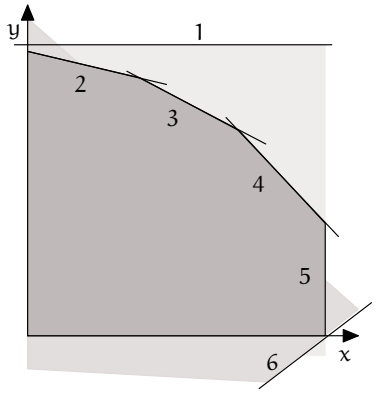


Figure 5.2: Redundant and Necessary Constraints

are discussed. Another promising direction is also shown that needs to be further explored.

This chapter is organized as follows. Section 5.2 presents some definitions related to redundancy and polytopes. Sections 5.3, 5.4, and 5.5 present the first, second, and third solution methods respectively. Section 5.6 presents an example. Section 5.7 concludes the chapter with a discussion of the results.

5.2 Definitions

A few definitions are introduced in this section that also serve the purpose of establishing some of the notation used in this chapter.

Notation 5.1 In Expression (5.3), $A = [\ a_1\ \ \ldots\ \ a_m\ ]^T$, $a_i = [\ a_{i1}\ \ \ldots\ \ a_{in}\ ]^T$, $i = 1 \ldots m$; $b = [\ b_1\ \ \ldots\ \ b_m\ ]^T$. Thus, the i-th constraint in $\mathcal{S}(b)$ is $a_i^T x \leq b_i$. Let $\mathcal{C}_i^=$, $\mathcal{C}_i$, and $\mathcal{C}_i^-$ denote the constraints $a_i^T x = b_i$, $a_i^T x \leq b_i$, and $a_i^T x < b_i$ respectively. Then, $\mathcal{S}(b) = \{\mathcal{C}_i | i = 1 \ldots m\}$.

Definition 5.1 Given that $a_i \in \Re^n$, and $b_i \in \Re$, the set $H_i^= \stackrel{\triangle}{=} \{x \,|\, \mathcal{C}_i^=\}$ is called a HYPERPLANE and the set $H_i \stackrel{\triangle}{=} \{x \,|\, \mathcal{C}_i\}$ is called a CLOSED HALFSPACE with OUTWARD NORMAL a_i. $H_i^- \stackrel{\triangle}{=} \{x \,|\, \mathcal{C}_i^-\}$ is the corresponding OPEN HALFSPACE.

Notation 5.2 For a given b, the feasible region of $\mathcal{S}(b)$ is:

$$P \stackrel{\triangle}{=} \{x \,|\, \mathcal{C}_k,\ 1 \leq k \leq m\}. \tag{5.4}$$

$X \setminus T$ is the set of all elements in the set X that do not belong to the set T. Let $\mathcal{S}_{\setminus i}(b) = \{\mathcal{C}_k | k \in \{1, \ldots, i-1, i+1, \ldots, m\}\}$, $\mathcal{A}_{\setminus i} = \{a_k | \mathcal{C}_k \in \mathcal{S}_{\setminus i}(b)\}$, and $\mathcal{LP}_{\setminus i}^{max} = \max\left\{ a_i^T x \middle| \mathcal{S}_{\setminus i}(b) \right\}$. Let, $P_{\setminus i}$ be the feasible region of $\mathcal{S}_{\setminus i}(b)\}$.

Definition 5.2 [CBB97] A system of constraints is called a REDUNDANT SYSTEM if it contains at least one redundant constraint. In $\mathcal{S}(b)$ of Expression (5.3), the i^{th} constraint is REDUNDANT if $P = P_{\setminus i}$ or $\mathcal{LP}_{\setminus i}^{max} \leq b_i$, WEAKLY REDUNDANT if $\mathcal{LP}_{\setminus i}^{max} = b_i$, STRONGLY REDUNDANT if $\mathcal{LP}_{\setminus i}^{max} < b_i$, and IRREDUNDANT or NECESSARY if it is not redundant.

Figure 5.2 illustrates redundancy of linear constraints w.r.t. the feasible region of a constraint set. Constraints 1 and 6 can be removed without altering the feasible region. So, they are redundant: 1 is strongly so, and 6 is weakly so. The others (2 – 5, $x \geq 0$ and $y \geq 0$) are necessary.

In the following the linear algebraic concepts of subspace, affine subspace, affine hull (denoted $\mathtt{aff}(\cdot)$), polytope, dimension of a polytope, faces and facets of a poly-

tope, cone [BT97, Zie95, GR97, Sch86], and the LP concepts of basic solution and basic feasible solution [BT97] will be used.

Definition 5.3 [Zie95] Let $P \subseteq \Re^n$ be a polytope. A constraint $\mathcal{C}_i$, is VALID for P if it is satisfied for all points $x \in P$.

Definition 5.4 [Sch86] The cone GENERATED by the vectors $X = \{x^1, \ldots, x^m\}$, is the set $cone(X) = cone(\{x^1, \ldots, x^m\}) \stackrel{\triangle}{=} \left\{ \sum_{i=1}^{m} \lambda_i x^i \;\middle|\; \lambda_i \geq 0 \right\}$. The notation $cone(Z)$ will be used to denote $cone(X)$ when $Z = [x^1 \;\; \ldots \;\; x^m]$.

Definition 5.5 In the system of halfspaces $\mathcal{S}(b)$, it will be said that certain constraints SURROUND a particular constraint if the cone generated by the outward normals of these constraints contains the outward normal of the constraint in question. That is, the constraints $\mathcal{C}_1, \ldots, \mathcal{C}_p$ $(p \leq m)$, surround the constraint $\mathcal{C}_i$, if $a_i \in cone(\{a_1, \ldots, a_p\})$, where $i \notin \{1, ..., p\}$. A SURROUNDING CONSTRAINT SET (SCS) of a constraint $\mathcal{C}_i$ is defined as a set of n linearly independent constraints that surround $\mathcal{C}_i$. The set of all SCSs of $\mathcal{C}_i$ will be called the SSCS of $\mathcal{C}_i$.

Definition 5.6 Consider $\mathcal{S}(b^1)$ and $\mathcal{S}_{\setminus i}(b^1)$, where b^1 is an instance of b. Assume that both sets are bounded and that the latter is irredundant. It is known that $\mathcal{C}_i$ in $\mathcal{S}(b^1)$ is redundant if $\mathcal{LP}^{max}_{\setminus i} \leq b^1_i$. Since, $\mathcal{S}_{\setminus i}(b^1)$ is bounded, $\mathcal{LP}^{max}_{\setminus i}$ occurs at a vertex of the feasible region of $\mathcal{S}_{\setminus i}(b^1)$. Assume that this vertex is the point of intersection of the constraints in $\mathcal{S}_n(b^1)$ which is a subset with cardinality n of $\mathcal{S}_{\setminus i}(b^1)$. Consider any $\mathcal{S}(b^2)$ without regard to whether it is redundant (it is not necessary that $b^1 \neq b^2$). In $\mathcal{S}(b^2)$ we will call $\mathcal{S}_n(b^2)$ a REDUNDANCY DEFINING CONSTRAINT SET (RDCS) of $\mathcal{C}_i$. The SRDCS of $\mathcal{C}_i$ is the set of all possible RDCSs of $\mathcal{C}_i$.

For example, in Figure 5.2 $\{Constraint\#2, x \geq 0\}$ is the RDCS of $Constraint\#1$, and $\{Constraint\#5, y \geq 0\}$ is the RDCS of $Constraint\#6$.

Notation 5.3 Let $\rho = [r_1 \;\; \ldots \;\; r_m]^T \geq 0$ be an instance of b. The constraint set

$$\mathcal{S}(\rho) = \left\{ a_j^T x \leq r_j \middle|\; 1 \leq j \leq m \right\}$$

is irredundant and has the origin somewhere in the middle of its feasible region. Let

$$\mathcal{S}_n(\rho) \triangleq \left\{ {a_1^i}^T x \leq r_1^i, \ldots, {a_n^i}^T x \leq r_n^i \right\}, \quad \mathcal{S}'_n(\rho) \triangleq \left\{ {a_1^{i'}}^T x \leq r_1^{i'}, \ldots, {a_n^{i'}}^T x \leq r_1^{i'} \right\}$$

be some subsets of $\mathcal{S}(\rho)$. Let $A_n = \begin{bmatrix} a_1^i & \ldots & a_n^i \end{bmatrix}$, $A'_n = \begin{bmatrix} a_1^{i'} & \ldots & a_n^{i'} \end{bmatrix}$.

5.3 First Method to Solve $\mathcal{PR}1$ and $\mathcal{PR}2$

It is assumed that TRIVIAL REDUNDANCIES such as the simultaneous existence of constraints $a_i^T x \leq b_i$ and $a_i^T x \leq k b_i$, $k \geq 0$, do not occur in our systems, and that $\mathcal{S}(b)$ is bounded.

$\mathcal{PR}1$ and $\mathcal{PR}2$ will be solved using SCSs. Then, this solution will be refined using RDCSs.

5.3.1 Overview of SCS Method

By Theorem 5.1 below, each constraint in the system $\mathcal{S}(b)$ is irredundant if it is irredundant w.r.t. the constraints that SURROUND it. Alternatively, only the constraints that surround a given constraint can make it redundant. It follows from Definition 5.5 that at least two constraints are needed to surround a given constraint. Once the surrounding constraint sets (SCSs) of a given constraint have been identified as shown in Subsection 5.3.3, $\mathcal{PR}1$ and $\mathcal{PR}2$ can be solved as described in Subsection 5.3.4.

5.3.2 Preliminary Results on SCSs

Here are presented some results that will form the foundation for the solution given in Section 5.3.4.

Lemma 5.1 Consider P from Expression (5.4). Assume that $H_i^=$ contains a face F of P. $\mathcal{C}_i$ is valid for P if and only if $\mathcal{C}_i$ is a non-negative linear combination of the constraints that define F.

Proof: If $\mathcal{C}_i$ is not a linear combination (LC) of the constraints that define F, then no point on F will satisfy $\mathcal{C}_i^=$, meaning that $H_i^=$ will not contain F. But, it is given that $H_i^=$ contains F. So, $\mathcal{C}_i$ is an LC of the constraints that define F. It remains to be shown that $\mathcal{C}_i$ is valid for P if and only if this LC is a non-negative one. $\mathcal{C}_i$ can be written thus:

$$\sum_{j=1}^{m} k_j a_j^T x \leq \sum_{j=1}^{m} k_j b_j.$$

Of the constraints $\mathcal{C}_1, \ldots, \mathcal{C}_m$, some may enter this LC multiplied by zeros, while others may simply be linear combinations of some of the other constraints. If F is an l-face in n dimensional space, then this LC is obtained from $n - l$ constraints. Thus, this LC can be seen as obtained from a maximum of n linearly independent constraints that define $aff(F)$. Without loss of generality, for some q, let this LC be as follows:

$$\mathcal{C}_i: \quad \sum_{j=1}^{q} k_j a_j^T x \leq \sum_{j=1}^{q} k_j b_j, \ 1 \leq q \leq n. \tag{5.5}$$

Suppose that $k_1 < 0$ in the above LC. Consider a point x^1 satisfying $\mathcal{C}_1^-$, and $\mathcal{C}_2^=$, $\ldots$, $\mathcal{C}_q^=$. Substituting this x^1 into Expression (5.5) gives $k_1 a_1^T x^1 \leq k_1 b_1$, that is, $a_1^T x^1 \geq b_1$. This is in disagreement with the statement that x^1 satisfies $\mathcal{C}_1^-$. This means that x^1 does not satisfy Expression (5.5). So, it follows that if $\mathcal{C}_i$ is not a non-negative LC of $\mathcal{C}_1, \ldots, \mathcal{C}_m$, then $\mathcal{C}_i$ is not valid for P. To see that if it is a non-negative LC, then it will be valid for P, suppose that all the k_j's in Expression (5.5) are non-negative. Then, any x^2 which satisfies $\mathcal{C}_1, \ldots, \mathcal{C}_q$ also satisfies $\mathcal{C}_i$. □

Theorem 5.1 If constraints in $\mathcal{S}_{\setminus i}(b)$ make $\mathcal{C}_i$ redundant, then $a_i \in cone(\mathcal{A}_{\setminus i})$.

Proof: Without loss of generality, let the constraints in $\mathcal{S}_{\setminus i}(b)$ be irredundant. It is given that the constraints in $\mathcal{S}_{\setminus i}(b)$ make $\mathcal{C}_i$ redundant. This means that a subset $\mathcal{S}_{\subseteq}$ of $\mathcal{S}_{\setminus i}(b)$ determines a face F of P such that for some $\overline{b}_i = \mathcal{LP}_{\setminus i}^{max} \leq b_i$, F is the highest-dimensional face of P contained in the hyperplane $\left\{x \,\middle|\, a_i^T x = \overline{b}_i\right\}$. This means, by Lemma 5.1, that the constraint $a_i^T x \leq \overline{b}_i$ is a non-negative linear

combination of the constraints in $\mathcal{S}_{\subseteq}$. So, $\mathcal{C}_i$ is surrounded by the constraints in $S_{\subseteq}$. So, $a_i \in cone(\mathcal{A}_{\setminus i})$. □

Theorem 5.1 shows that if $a_i \notin cone(\mathcal{A}_{\setminus i})$, then the constraints in $\mathcal{S}_{\setminus i}(b)$ do not make $\mathcal{C}_i$ redundant. Thus, Theorem 5.1 helps confine the redundancy test for $\mathcal{C}_i$ to $\mathcal{C}_i$'s surrounding constraints. But, $\mathcal{C}_i$ may have $n_i \geq 2$ surrounding constraints. To test if $\mathcal{C}_i$ is redundant, it should not be necessary to work with all n_i of them, since only those constraints are needed that determine $\mathcal{LP}^{max}_{\setminus i}$. It is enough to work with just those that determine F (the face on which $\mathcal{LP}^{max}_{\setminus i}$ occurs) or, more concisely, with those that determine $aff(F)$. However, the present chapter is only able to propose a method that works with basic solutions (points of intersection of n linearly independent constraints, such as in an SCS) rather than those that define F or $aff(F)$. There are two possible disadvantages to such an approach: (1) in cases where less than n constraints are enough to surround a constraint, using an SCS MAY BE expensive; (2) in an unbounded system, an SCS may not be found (this is why the present method does not work with unbounded systems), though a set of less than n constraints that surrounds the constraint in question may be found.

Remark 5.1 The SCS method solves $\mathcal{PR}1$ and $\mathcal{PR}2$ by applying the fact, for all the m constraints in the system, that each constraint is irredundant if it is irredundant with respect to all of its respective SCSs (for, if the constraint is not irredundant with respect to any one SCS, then it is redundant).

5.3.3 Determining the SSCS of $\mathcal{C}_i$

An SCS of $\mathcal{C}_i$ can be found by using the fact that for $\mathcal{S}_n(b)$ (see Notation 5.3) to be an SCS of $\mathcal{C}_i$, $a_i \in cone(A_n)$, that is $A_n^{-1}a_i \geq 0$.

Step 1: From the m columns of the matrix A^T form square matrices taking n different columns at a time. There are $\mathbf{C}^m_n$ such matrices.

Step 2: For each a_i from A^T, list the square matrices that do not have a_i as one of the columns.

Step 3: Find the inverses of these square matrices.

Step 4: For each such inverse matrix A_n^{-1}, calculate the product $A_n^{-1}a_i$.

Step 5: Classify $\mathcal{S}_n(b)$ as SCS or not-SCS of $\mathcal{C}_i$.

If a search for the SCS of a constraint turns up empty, then this implies one of two possibilities: (1) the system is unbounded, or (2) the constraint in question is not surrounded by any constraints (for example, in a constraint system that defines a cube in $\Re^3$, none of the constraints has SCSs, but the system is bounded).

5.3.4 Solving $\mathcal{PR}1$ and $\mathcal{PR}2$ using SSCS

Step 1: Let the SCSs of $\mathcal{C}_i$ be (assuming that the SCSs are η_i in number)

$$\left\{ \begin{array}{rcl} {a^i_{1,1}}^T x & \leq & b^i_{1,1} \\ \cdots & \cdots & \cdots \\ {a^i_{n,1}}^T x & \leq & b^i_{n,1} \end{array} \right\}, \ldots\ldots, \left\{ \begin{array}{rcl} {a^i_{1,\eta_i}}^T x & \leq & b^i_{1,\eta_i} \\ \cdots & \cdots & \cdots \\ {a^i_{n,\eta_i}}^T x & \leq & b^i_{n,\eta_i} \end{array} \right\}.$$

These sets are subsets of $\mathcal{S}(b)$ of Expression (5.3). They determine the following intersection points:

$$x^i_k = \left(\left[\begin{array}{ccc} a^i_{1,k} & \cdots & a^i_{n,k} \end{array} \right]^T \right)^{-1} \left[\begin{array}{ccc} b^i_{1,k} & \cdots & b^i_{n,k} \end{array} \right]^T, \quad k = 1, \ldots, \eta_i.$$

Step 2: By Remark 5.1, $\mathcal{C}_i$ is irredundant if $b_i < a_i^T x^i_k$, for $k = 1, \ldots, \eta_i$. Similar conditions have to be satisfied simultaneously by all the constraints, for which SCSs exist, and their respective SCSs.

Step 3: It follows that the solution set of $\mathcal{PR}2$ is defined by the following expression:

$$b_i - a_i^T \left(\left[\begin{array}{ccc} a^i_{1,k} & \cdots & a^i_{n,k} \end{array} \right]^T \right)^{-1} \left[\begin{array}{c} b^i_{1,k} \\ \vdots \\ b^i_{n,k} \end{array} \right] < 0, \qquad (5.6)$$

$$k = 1, \ldots, \eta_i,$$
$$i = 1, \ldots, m.$$

Step 4: The solution to $\mathcal{PR}1$ is obtained by appending the conditions $b = By$ to Expression (5.6). Alternatively, the solution to $\mathcal{PR}1$ is the feasible region of the following system of constraints:

$$\left([B]_i - a_i^T \left(\begin{bmatrix} a_{1,k}^i & \cdots & a_{n,k}^i \end{bmatrix}^T \right)^{-1} \begin{bmatrix} [B]_{1,k}^i \\ \cdots \\ [B]_{n,k}^i \end{bmatrix} \right) y \;<\; 0, \qquad (5.7)$$

$$k = 1, \ldots, \eta_i,$$

$$i = 1, \ldots, m$$

where $[B]_i$ is the i-th row of the matrix B.

5.3.5 Overview of Refinement using RDCSs

By Theorem 5.2 below, if every constraint in $\mathcal{S}(b)$ is irredundant with respect to its respective RDCSs, then $\mathcal{S}(b)$ is irredundant. This means that it is sufficient to work with RDCSs instead of with SCSs, and that the systems of inequalities of expressions (5.6) and (5.7) may have redundancy in them. Identifying the RDCSs will help remove some of this redundancy. In the following, a way is shown to identify some of those constraint sets that cannot be RDCSs of a constraint in $\mathcal{S}(b)$.

5.3.6 Preliminary Results on RDCSs

$\mathcal{LP}_{\setminus i}^{max}$ can occur anywhere in the face F mentioned in the proof of Theorem 5.1. In a bounded system, F contains at least one vertex. Let this vertex be denoted x^v. Then, $\mathcal{LP}_{\setminus i}^{max} = a_i^T x^v$.

For example, in the left hand polytope of Figure 5.1, $\mathcal{LP}_{\setminus 7}^{max}$ is obtained at x^v that is determined by any one of the following 4 sets of constraints: $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$, $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_4\}$, $\{\mathcal{C}_1, \mathcal{C}_3, \mathcal{C}_4\}$, and $\{\mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4\}$.

Given that $\mathcal{C}_i$ has n_i surrounding constraints, and that each x^v is defined by a set of n linearly independent constraints, the maximum number of possible x^v's is

$$\mathbf{C}_n^{n_i} = \frac{n_i!}{(n_i - n)!\, n!}$$

Notation 5.4 In the set $\mathcal{S}_{\setminus i}(b)$, let the j_i-th SCS of $\mathcal{C}_i$ be ${A^{j_i}}^T x \leq b^{j_i}$, with $x^{j_i} \triangleq ({A^{j_i}}^T)^{-1} b^{j_i}$. Here,

$$\begin{aligned}\left\{a_1^{j_i}, \ldots, a_n^{j_i}\right\} &\subset \{a_1, \ldots, a_m\}, A^{j_i} = \left[a_1^{j_i} \ \ldots \ a_n^{j_i}\right], \\ \left\{b_1^{j_i}, \ldots, b_n^{j_i}\right\} &\subset \{b_1, \ldots, b_m\}, b^{j_i} = \left[b_1^{j_i} \ \ldots \ b_n^{j_i}\right]^T.\end{aligned}$$

Lemma 5.2 In the set $\left\{x^{j_i} \middle| 1 \leq j_i \leq \mathbf{C}_n^{n_i}\right\}$ of intersection points, the element that minimizes $a_i^T x$ is the solution to $\mathcal{LP}_{\setminus i}^{\max}$.

Proof: As was stated in the beginning of this subsection, $\mathcal{LP}_{\setminus i}^{\max} = a_i^T x^\nu$, where $x^\nu \in \left\{x^{j_i} \middle| 1 \leq j_i \leq \mathbf{C}_n^{n_i}\right\}$. As the constraints corresponding to A^{j_i} surround the i-th constraint, there exists $\lambda^{j_i} \geq 0$ ($\lambda^{j_i} \in \Re^n$) such that $A^{j_i}\lambda^{j_i} = a_i$. As x^ν satisfies $\mathcal{S}_{\setminus i}(b)$, it follows that ${A^{j_i}}^T x^\nu \leq b^{j_i}$, meaning that ${A^{j_i}}^T x^\nu \leq {A^{j_i}}^T x^{j_i}$. This gives: ${\lambda^{j_i}}^T {A^{j_i}}^T x^\nu \leq {\lambda^{j_i}}^T {A^{j_i}}^T x^{j_i}$, meaning that $a_i^T x^\nu \leq a_i^T x^{j_i}$. □

Remark 5.2 Applying Definition 5.6 it can be seen that in an irredundant system an RDCS of $\mathcal{C}_i$ determines $\mathcal{LP}_{\setminus i}^{\max}$. The proof of Theorem 5.1 has shown that $\mathcal{LP}_{\setminus i}^{\max}$ is determined only by the SCSs of $\mathcal{C}_i$. So, it follows that an RDCS is an SCS.

The number of inequalities involved in expressions (5.6) and (5.7) can be reduced by using the notion of RDCS, as suggested by the following theorem.

Theorem 5.2 If every one of the constraints in $\mathcal{S}(b)$ is irredundant with respect to its respective RDCSs, then $\mathcal{S}(b)$ is irredundant.

Proof: Assume that (**a1**) every constraint in $\mathcal{S}(b)$ is irredundant with respect to its respective RDCSs, but that (**a2**) $\mathcal{C}_i$ is redundant. (a2) means that there is a SCS Ψ_i of $\mathcal{C}_i$ with respect to which the latter is redundant. Thus, if $\mathcal{C}_i$ is moved into the feasible region of $\mathcal{S}(b)$ such that the $\mathcal{C}_i$ is irredundant with respect to Ψ_i, one of the following two things may happen:
(1) $\mathcal{S}(b)$ may become irredundant (this means that Ψ_i was in fact an RDCS of $\mathcal{C}_i$, contradicting (a1)), or
(2) Some other constraint $\mathcal{C}_j$ now becomes redundant with respect to Ψ_j, which is

one of the SCSs of $\mathcal{C}_j$. Now, if $\mathcal{C}_j$ is moved into the feasible region of $\mathcal{S}(b)$ such that it is irredundant with respect to Ψ_j, one of two things may happen: $\mathcal{S}(b)$ becomes irredundant (meaning that Ψ_i and Ψ_j are in fact the RDCSs of $\mathcal{C}_i$ and $\mathcal{C}_j$ respectively; this contradicts (a1)), or some other constraint $\mathcal{C}_k$ now becomes redundant with respect to one of its SCSs, Ψ_k.
Thus, recursive application of the above argument gives one of two possibilities: either (a2) was incorrect and $\mathcal{S}(b)$ was irredundant to start with, or (a1) was incorrect and there were the RDCSs Ψ_i, Ψ_j, Ψ_k, ..., (respectively of $\mathcal{C}_i$, $\mathcal{C}_j$, $\mathcal{C}_k$, ...) that were overlooked when formulating (a1). This completes the proof. □

Lemma 5.3 below identifies some members of the SSCS of $\mathcal{C}_i$ that cannot be RDCSs of $\mathcal{C}_i$ in $\mathcal{S}(b)$. The redundancy-related properties of $\mathcal{S}(b^1)$ — that belongs to the translation class of systems containing $\mathcal{S}(\rho)$ — are the same as those of $\mathcal{S}(\rho)$; so, there is no loss of generality in considering $\mathcal{S}(\rho)$.

Lemma 5.3 (see Notation 5.3 on Page 44) Assume that $\mathcal{S}(\rho)$ is irredundant, $\mathcal{S}_n(\rho)$ and $\mathcal{S}'_n(\rho)$ are SCSs of $\mathcal{C}_i$, and that $cone(A_n) \subset cone(A'_n)$. Then, $\mathcal{S}'_n(\rho)$ is not an RDCS of $\mathcal{C}_i$.

Proof: Suppose that $\mathcal{S}'_n(\rho)$ is an RDCS of $\mathcal{C}_i$. Then, applying Lemma 5.2 and Remark 5.2 we obtain the following inequality:

$$a_i^T \underbrace{\left(A'^T_n\right)^{-1} \left[r_1^{i\nu} \ \dots \ r_n^{i\nu}\right]^T}_{x'_n} \leq a_i^T \underbrace{\left(A_n^T\right)^{-1} \left[r_1^i \ \dots \ r_n^i\right]^T}_{x_n}. \tag{5.8}$$

As $a_i \in cone(A_n)$ and $a_i \in cone(A'_n)$, $\exists \lambda, \lambda' \geq 0 \in \Re^n$ such that $A_n\lambda = a_i$ and $A'_n\lambda' = a_i$. Thus, Expression (5.8) can be written as follows:

$$\lambda^T A_n^T \left(A'^T_n\right)^{-1} \left[r_1^{i\nu} \ \dots \ r_n^{i\nu}\right]^T \leq \lambda^T \left[r_1^i \ \dots \ r_n^i\right]^T.$$

Now, since $\lambda \geq 0$, this inequality will be true only if at least one of the following

inequalities holds:

$$\begin{array}{rcl} a_1^{i\,T}\left(A_n'^{\,T}\right)^{-1}\left[r_1^{i\nu}\ \ldots\ r_n^{i\nu}\right]^T & \leq & r_1^i \\ \ldots\ \ \ldots\ \ \ldots\ \ \ldots\ \ \ldots\ \ \ldots\ \ \ldots & & \\ a_n^{i\,T}\left(A_n'^{\,T}\right)^{-1}\left[r_1^{i\nu}\ \ldots\ r_n^{i\nu}\right]^T & \leq & r_n^i \end{array}$$

As $cone(A_n) \subset cone(A'_n)$, this means that at least one of the constraints in $\mathcal{S}_n(\rho)$ is redundant with respect to the constraints in $\mathcal{S}'_n(\rho)$. However, it is given that $\mathcal{S}(\rho)$ is irredundant. So, Expression (5.8) cannot be true, meaning that $\mathcal{S}'_n(\rho)$ cannot be an RDCS of $\mathcal{C}_i$. □

Corollary 5.1 In an irredundant system $\mathcal{S}(b)$, no two constraints can have the same RDCS.

Proof: This result follows directly from Lemma 5.3. □

5.3.7 Simplification using RDCS

Once all the SCSs in $\mathcal{S}(b)$ have been determined using Subsection 5.3.3, those SCSs that are not the RDCSs can be eliminated using Corollary 5.1. This can be done as follows. Suppose that $\mathcal{C}_i$ and $\mathcal{C}_j$ are members of $\mathcal{S}(b)$. Suppose that $\mathcal{S}_n(b)$ is an SCS of $\mathcal{C}_i$ and $\mathcal{C}_j$. In case there exists among the SCSs of $\mathcal{C}_i$ an SCS which is obtained by replacing one of the constraints in $\mathcal{S}_n(b)$ by $\mathcal{C}_j$, then $\mathcal{S}_n(b)$ is not an RDCS of $\mathcal{C}_i$ and it can be eliminated from the SRDCS of $\mathcal{C}_i$.

5.3.8 Solving $\mathcal{PR}1$ and $\mathcal{PR}2$ using RDCSs

In case the SRDCS of each constraint is given, then the solution to $\mathcal{PR}1$ is exactly as in Section 5.3.4 if the term 'RDCSs' is substituted for 'SCSs'.

5.3.9 Analysis

An m-file (for MATLAB®) has been written to implement the methods of subsections 5.3.3, 5.3.4, and 5.3.7. Instead of computing A_n^{-1} and then the product $A_n^{-1}a_i$, Gaussian elimination is used to solve the system of linear equations $A_n x = a_i$ and obtain x. This method is about 2 – 3 times faster than the one that uses matrix inversion [gen95].

In the method described so far, Subsection 5.3.7 helps reduce only the number of inequalities that describe the solution space of $\mathcal{PR}1$ or $\mathcal{PR}2$. It does not reduce the initial complexity — due to Subsection 5.3.3 — of identifying the SSCS. An efficient way needs to be found that will allow to either identify the SRDCS of $\mathcal{C}_i$ or, at least, to identify those constraints that cannot be the RDCSs of $\mathcal{C}_i$, without going through Subsection 5.3.3.

One potential method to identify the RDCSs of a constraint in $\mathcal{S}(b)$ is to use the concept of an IDEAL SYSTEM — a system $\mathcal{S}(b^s)$ in which all the member constraints are tangent to a sphere. It can be shown that $\mathcal{S}(b^s)$ is irredundant. It is conjectured that, for some classes of systems, the RDCSs of the i-th constraint in $\mathcal{S}(b)$ can be identified from the knowledge of the RDCSs of the i-th constraint in $\mathcal{S}(b^s)$. (Note that this conjecture is not true for all classes of systems. A counter-example for a particular system has been shown by Dr. Vincent Loechner). Identifying classes of systems for which this could work is a topic of current research. The advantage of such a method would be that identifying RDCSs in $\mathcal{S}(b^s)$ involves fewer matrix inverse operations than in $\mathcal{S}(b)$, and so is less expensive computationally. The attempt to exploit the idea of ideal systems is described in Appendix A.

As it stands, the present method is combinatorially complex ($m\mathbf{C}_n^{m-1}$ Gaussian eliminations, each of these being $O(n^3)$ complex). The method presented in the next section is POSSIBLY more efficient as it uses vertex enumeration algorithms that, though not of polynomial complexity, have been found to give acceptable results for some classes of systems [Bre97].

5.4 Second Method to Solve $\mathcal{PR}1$ and $\mathcal{PR}2$

This second method was suggested by Dr. Carl Lee [Lee02] and uses LP duality. It is presented in Appendix B. Please look there for further details.

5.4.1 Analysis

An m-file that uses the extreme point and ray enumeration program CDD [Fuk99] has been written to implement this algorithm.

The complexity of this method is determined by Step 3 which is a vertex enumeration problem. This method solves one problem of vertex enumeration for each constraint. Thus it solves m problems of vertex enumeration overall. Vertex enumeration algorithms are not of polynomial complexity [Bre97].

Step 2 of Subsection 5.3.4 follows from the following observation too: $\mathcal{LP}^{max}_{\setminus i} = a_i^T x^v = \min\left\{a_i^T x_k^i \mid k = 1, \ldots, \eta_i\right\}$ as per Lemma 5.2, and $\mathcal{C}_i$ is irredundant if $b_i < \mathcal{LP}^{max}_{\setminus i}$. In this sense, the method of Subsection 5.3.4 uses the same criteria to test for redundancy as the method shown in this section.

Both methods depend on finding extreme points. When the feasible region of $\mathcal{S}(b)$ does not have extreme points, the SCS method quits even though there may be a finite value for $\mathcal{LP}^{max}_{\setminus i}$. The second method however, finds extreme points in the feasible region of the dual system (dual polyhedron). As the dual polyhedron is in the non-negative orthant, it is guaranteed to have at least one extreme point [BT97]. Thus, the second method works for unbounded systems too.

In this second method, for some i the primal LPP of Step 2 might be unbounded. In such a case the i-th constraint of $\mathcal{S}(b)$ is automatically always irredundant for any choice of b for which $\mathcal{S}(b)$ is feasible. In this case the corresponding dual LPP of Step 2 is infeasible and so no constraints involving b_i are generated.

5.5 Third Method to Solve $\mathcal{PR}1$ and $\mathcal{PR}2$

An additional direction that was explored to solve $\mathcal{PR}2$ is described here. This approach appears promising but is not yet complete. It is presented as a direction for continuing research. It is based on using Minkowski's Theorem for an equilibrated system of vectors.

Theorem 5.3 (Minkowski's Theorem for an equilibrated system of vectors [GH99]) Let $u_1, \ldots, u_m$ be pairwise different unit vectors of Euclidean n-space $\mathfrak{R}^n$ which span $\mathfrak{R}^n$, and let $\mu_1, \ldots, \mu_m$ be positive reals such that $\sum_{i=1}^{m} \mu_i u_i = 0$. Then there exists a polytope P with outer facet normals $u_1, \ldots, u_m$ and corresponding facet volumes $\mu_1, \ldots, \mu_m$. Further, P is unique up to translation.

Let, A_u be a matrix of unit vectors that has been obtained from A (see Notation 5.1) thus: $A_u = \left[\begin{array}{ccc} \frac{a_1}{|a_1|} & \ldots & \frac{a_m}{|a_m|} \end{array} \right]^T$.

Remark 5.3 Minkowski's theorem suggests that each vector z from the set $\Psi = \left\{ z | rank(A_u) = n, A_u^T z = 0, z > 0, z \in \mathfrak{R}^m \right\}$ corresponds to a bounded irredundant system, unique up to translation, represented by Equation (5.3). In particular, each such z is the vector of facet volumes of a polytope represented by Equation (5.3).

It follows that, given A_u and a z from Ψ, if a rule (or an algorithm) that would give the corresponding value of $\hat{b}$ were to be found, then the system $A_u x \leq \hat{b}$ (or, alternatively, $Ax \leq b$) would be irredundant. Here, $\hat{b} = \left[\begin{array}{ccc} \frac{b_1}{|a_1|} & \ldots & \frac{b_m}{|a_m|} \end{array} \right]^T$. Thus, applying the said rule to every z from Ψ, the set $\mathcal{B}$ of all such b would be obtained. Then, the set $\left\{ \mathcal{B} + Ax^1 | x^1 \in \mathfrak{R}^n \right\}$ would be the solution to $\mathcal{PR}2$.

Such a rule or algorithm is the subject of the MINKOWSKI RECONSTRUCTION PROBLEM [GH99]. Such an algorithm does not exist in practice. Even if it existed, it is not practical to apply it to each z from Ψ as there is an infinite number of z.

Currently other ways to implement the idea of finding a correspondence between Ψ and $\mathcal{B}$ are under investigation.

Remark 5.4 Remark 5.3 suggests that $\mathcal{S}(b)$ is bounded if Ψ is non-empty. Thus,

checking for the non-emptiness of Ψ, or of its closure (if a set is non-empty, then its closure is also non-empty) $\mathtt{clo}(\Psi)$, is one test for boundedness of $\mathcal{S}(\mathtt{b})$. Note that while testing $\mathtt{clo}(\Psi)$ for non-emptiness, the case when the only element of $\mathtt{clo}(\Psi)$ is 0 is ignored.

5.6 Example

Consider $\mathcal{S}(\mathtt{b})$ with A as given below:

$$A = \begin{bmatrix} 1.5578 & 0.4142 \\ -2.4443 & -0.9778 \\ -1.0982 & -1.0215 \\ 1.1226 & 0.3177 \\ 0.5817 & 1.5161 \\ -0.2714 & 0.7494 \end{bmatrix}.$$

From Remark 5.4, it follows that $\mathtt{clo}(\Psi)$ is non-empty (this was done using CDD), and thus $\mathcal{S}(\mathtt{b})$ is bounded. Next, from the algorithms of subsections 5.3.3 and 5.3.4 it follows that each constraint in $\mathcal{S}(\mathtt{b})$ has two SCSs and that the solution to $\mathcal{PR}2$ is given by $D\mathtt{b} < 0$ where D is as follows:

$$D = \begin{bmatrix} 1 & -0.0931 & 0 & -1.5903 & 0 & 0 \\ 1 & 0 & -0.0375 & -1.4243 & 0 & 0 \\ 0 & 1 & -2.9296 & 0 & -1.3288 & 0 \\ 0 & 1 & -1.9062 & 0 & 0 & -1.2933 \\ -2.7855 & -2.2245 & 1 & 0 & 0 & 0 \\ 0 & -2.4839 & 1 & -4.4298 & 0 & 0 \\ -0.7154 & 0 & 0 & 1 & -0.0141 & 0 \\ -0.7247 & 0 & 0 & 1 & 0 & -0.0234 \\ -0.6620 & 0 & 0 & 0 & 1 & -1.6571 \\ 0 & 0 & 0 & -0.9135 & 1 & -1.6358 \\ 0 & -0.2701 & 0 & 0 & -0.6685 & 1 \\ 0 & 0 & -0.7913 & 0 & -1.0274 & 1 \end{bmatrix}.$$

Thus, the system $Ax \leq b$ will be irredundant for any b such that $Db < 0$. For comparison, here is the matrix D determined using the method of Appendix B:

$$D = \begin{bmatrix} 1.0000 & 0 & -0.0373 & -1.4239 & 0 & 0 \\ 1.0000 & -0.0928 & 0 & -1.5892 & 0 & 0 \\ 0 & 1.0000 & -1.9059 & 0 & 0 & -1.2944 \\ 0 & 1.0000 & -2.9309 & 0 & -1.3309 & 0 \\ 0 & -2.4840 & 1.0000 & -4.4282 & 0 & 0 \\ -2.7864 & -2.2255 & 1.0000 & 0 & 0 & 0 \\ -0.7155 & 0 & 0 & 1.0000 & -0.0141 & 0 \\ -0.7249 & 0 & 0 & 1.0000 & 0 & -0.0233 \\ 0 & 0 & 0 & -0.9133 & 1.0000 & -1.6358 \\ -0.6620 & 0 & 0 & 0 & 1.0000 & -1.6570 \\ 0 & 0 & -0.7919 & 0 & -1.0282 & 1.0000 \\ 0 & -0.2702 & 0 & 0 & -0.6686 & 1.0000 \end{bmatrix}.$$

The constraint sets which are SCSs of the various constraints can be seen in the

following table which was generated by the m-files mentioned in Subsection 5.3.9:

Constraint	SCS	Is this SCS an RDCS ?
$\mathcal{C}_1$	$\{\mathcal{C}_2 \quad \mathcal{C}_4\}$	No
$\mathcal{C}_1$	$\{\mathcal{C}_3 \quad \mathcal{C}_4\}$	May be
$\mathcal{C}_2$	$\{\mathcal{C}_3 \quad \mathcal{C}_5\}$	No
$\mathcal{C}_2$	$\{\mathcal{C}_3 \quad \mathcal{C}_6\}$	May be
$\mathcal{C}_3$	$\{\mathcal{C}_1 \quad \mathcal{C}_2\}$	May be
$\mathcal{C}_3$	$\{\mathcal{C}_2 \quad \mathcal{C}_4\}$	May be
$\mathcal{C}_4$	$\{\mathcal{C}_1 \quad \mathcal{C}_5\}$	May be
$\mathcal{C}_4$	$\{\mathcal{C}_1 \quad \mathcal{C}_6\}$	No
$\mathcal{C}_5$	$\{\mathcal{C}_1 \quad \mathcal{C}_6\}$	No
$\mathcal{C}_5$	$\{\mathcal{C}_4 \quad \mathcal{C}_6\}$	May be
$\mathcal{C}_6$	$\{\mathcal{C}_2 \quad \mathcal{C}_5\}$	May be
$\mathcal{C}_6$	$\{\mathcal{C}_3 \quad \mathcal{C}_5\}$	No

Some of the redundancy in the description of the solution space of $\mathcal{PR}2$ is removed using this table. The new description of this solution space is obtained as $\widehat{D}b < 0$, where $\widehat{D}$ is the following matrix:

$$\widehat{D} = \begin{bmatrix} 1.0000 & 0 & -0.0375 & -1.4243 & 0 & 0 \\ 0 & 1.0000 & -2.9296 & 0 & -1.3288 & 0 \\ -2.7855 & -2.2245 & 1.0000 & 0 & 0 & 0 \\ 0 & -2.4839 & 1.0000 & -4.4298 & 0 & 0 \\ -0.7154 & 0 & 0 & 1.0000 & -0.0141 & 0 \\ -0.7247 & 0 & 0 & 1.0000 & 0 & -0.0234 \\ -0.6620 & 0 & 0 & 0 & 1.0000 & -1.6571 \\ 0 & -0.2701 & 0 & 0 & -0.6685 & 1.0000 \end{bmatrix}.$$

Thus, the system will be irredundant for any b such that $\widehat{D}b < 0$.

5.7 Discussion

For the system of linear constraints $Ax \leq b$, two methods have been presented to determine the set of all values of b for which the system is irredundant. Also a third promising direction has been described. Currently, the first method works for bounded systems only. The second method works for unbounded systems too.

Chapter 6

Maintainability of Halfspace Systems

6.1 Introduction

In this chapter a linear programming-based test for the maintainability of halfspace systems will be developed. A test for maintainability developed for slack-descriptor systems in regular form will also work for halfspace systems. The complexity of such a test will be compared with the linear programming-based test.

An algorithm for maintaining a halfspace system in a static tube will be presented. An implementation of this algorithm that uses linear programming duality will be shown. This same algorithm can also be used for maintaining a slack-descriptor system in regular form in a static tube. An implementation of this algorithm will be shown. The two implementations will be compared for complexity.

6.2 Test for Maintainability

In Expression (3.7), it was seen that the $\mathcal{R}each$ set from the point x_0^i under the control u_0^i will be inside $\mathcal{T}ube(T, t)$ if

$$[t]_k \geq \max_{\mathcal{R}each(x_0^i, u_0^i)} [T]_k x_1, \quad k = 1, \ldots, j.$$

that is, if

$$[t]_k \geq \max_{Ex_1 \leq Fx_0^i + Bu_0^i} [T]_k x_1, \quad k = 1, \ldots, j. \tag{6.1}$$

The linear programming primal problem

$$\max_{\substack{Ex_1 \leq Fx_0^i + Bu_0^i \\ x_1 \quad free}} [T]_k x_1$$

has the dual [BT97, page 166]

$$\min_{\substack{z \geq 0 \\ z^T E = [T]_k}} z^T (Fx_0^i + Bu_0^i)$$

This dual problem is the same as

$$\max_{\substack{z \geq 0 \\ z^T E = [T]_k}} -z^T (Fx_0^i + Bu_0^i)$$

Thus, Equation (6.1) becomes (with $k = 1, \ldots, j$):

$$[t]_k \geq \max_{\substack{z \geq 0 \\ z^T E = [T]_k}} -z^T (Fx_0^i + Bu_0^i) \tag{6.2}$$

The optimal solution of any linear programming problem occurs at a vertex of the polyhedron that represents the feasible region of the problem. Thus, the solution of the linear programming problem on the right hand side of Expression (6.2) occurs among the vertices of the following polyhedron:

$$\left.\begin{array}{rcl} z & \geq & 0 \\ z^T E & = & [T]_k \end{array}\right\} \tag{6.3}$$

Let $z_{11(k)}^T$, $z_{12(k)}^T$, ..., $z_{1\eta(k)}^T$ be the vertices of the polyhedron of Expression (6.3). Then the following system of inequalities is equivalent to Expression (6.2):

$$\begin{array}{rcl} -z_{11(k)}^T (Fx_0^i + Bu_0^i) & \leq & [t]_k \\ -z_{12(k)}^T (Fx_0^i + Bu_0^i) & \leq & [t]_k \\ \cdots\cdots & \cdots & \cdots \\ -z_{1\eta(k)}^T (Fx_0^i + Bu_0^i) & \leq & [t]_k \end{array}$$

Then, $Reach(x_0^i, u_0^i) \subseteq Tube(T, t)$ if the following system of inequalities holds true:

$$\left.\begin{array}{rcl}
-z_{11(1)}^T(Fx_0^i + Bu_0^i) & \leq & [t]_1 \\
\cdots\cdots & \cdots & \cdots \\
-z_{1\eta(1)}^T(Fx_0^i + Bu_0^i) & \leq & [t]_1 \\
 & & \\
-z_{11(2)}^T(Fx_0^i + Bu_0^i) & \leq & [t]_2 \\
\cdots\cdots & \cdots & \cdots \\
-z_{1\eta(2)}^T(Fx_0^i + Bu_0^i) & \leq & [t]_2 \\
 & & \\
\cdots\cdots & \cdots & \cdots \\
 & & \\
-z_{11(j)}^T(Fx_0^i + Bu_0^i) & \leq & [t]_j \\
\cdots\cdots & \cdots & \cdots \\
-z_{1\eta(j)}^T(Fx_0^i + Bu_0^i) & \leq & [t]_j
\end{array}\right\} \tag{6.4}$$

By definition, a tube is maintainable if for every x_0 in the tube maintaining controls can be found. Thus, a test for maintainability might need every x_0 in the tube to be tested. Clearly, this is impossible. Theorem 6.1 shows that IF MAINTAINING CONTROLS EXIST FOR THE VERTICES OF THE TUBE, THEN MAINTAINING CONTROLS EXIST FOR EVERY POINT IN THE TUBE. Thus, a test for maintainability need look for the existence of maintaining controls only for each vertex of the tube.

Let $Tx_0 \leq t$ and $Tx_1 \leq t$ be the cross sections of the tube at time $t = 0$ and $t = 1$ respectively. Let the extreme points of $Tx_0 \leq t$ be $x_0^1, x_0^2, \ldots, x_0^L$.

Theorem 6.1 Assume that $Tube(T, t)$ is bounded. If for each x_0^i, $i = 1, \ldots, L$, there exists a u_0^i such that Expression (6.4) holds true, then for any point x_0^p in $Tx_0 \leq t$, there exists a u_0^p such that $Reach(x_0^p, u_0^p) \subseteq Tube(T, t)$.

Proof: It needs to be shown that

$$\left.\begin{array}{rcl}
-z_{11(1)}^T(Fx_0^p + Bu_0^p) & \leq & [t]_1 \\
\cdots\cdots & \cdots & \cdots \\
-z_{1\eta(1)}^T(Fx_0^p + Bu_0^p) & \leq & [t]_1 \\
 & & \\
-z_{11(2)}^T(Fx_0^p + Bu_0^p) & \leq & [t]_2 \\
\cdots\cdots & \cdots & \cdots \\
-z_{1\eta(2)}^T(Fx_0^p + Bu_0^p) & \leq & [t]_2 \\
 & & \\
\cdots\cdots & \cdots & \cdots \\
 & & \\
-z_{11(j)}^T(Fx_0^p + Bu_0^p) & \leq & [t]_j \\
\cdots\cdots & \cdots & \cdots \\
-z_{1\eta(j)}^T(Fx_0^p + Bu_0^p) & \leq & [t]_j
\end{array}\right\} \tag{6.5}$$

Since $Tube(T, t)$ is bounded, $Tx_0 \leq t$ is a polytope (that is, a convex bounded polyhedron). Any point x_0^p in $Tx_0 \leq t$ can be expressed as a convex combination of x_0^i, $i = 1, \ldots, L$, as follows:

$$x_0^p = \sum_{i=1}^{L} \lambda_i x_0^i, \quad \sum_{i=1}^{L} \lambda_i = 1, \quad \lambda_i \geq 0. \tag{6.6}$$

Multiplying each inequality in Expression (6.4) by λ_i and summing it over $i = 1, \ldots, L$ gives the following:

$$
\begin{aligned}
\sum_{i=1}^{L} \left\{ -z_{11(1)}^{T} (Fx_0^i + Bu_0^i)\lambda_i \right\} &\leq \sum_{i=1}^{L} \lambda_i [t]_1 \\
&\cdots\cdots \quad \cdots \quad \cdots \\
\sum_{i=1}^{L} \left\{ -z_{1\eta(1)}^{T} (Fx_0^i + Bu_0^i)\lambda_i \right\} &\leq \sum_{i=1}^{L} \lambda_i [t]_1 \\
\sum_{i=1}^{L} \left\{ -z_{11(2)}^{T} (Fx_0^i + Bu_0^i)\lambda_i \right\} &\leq \sum_{i=1}^{L} \lambda_i [t]_2 \\
&\cdots\cdots \quad \cdots \quad \cdots \\
\sum_{i=1}^{L} \left\{ -z_{1\eta(2)}^{T} (Fx_0^i + Bu_0^i)\lambda_i \right\} &\leq \sum_{i=1}^{L} \lambda_i [t]_2 \\
&\cdots\cdots \quad \cdots \quad \cdots \\
\sum_{i=1}^{L} \left\{ -z_{11(j)}^{T} (Fx_0^i + Bu_0^i)\lambda_i \right\} &\leq \sum_{i=1}^{L} \lambda_i [t]_j \\
&\cdots\cdots \quad \cdots \quad \cdots \\
\sum_{i=1}^{L} \left\{ -z_{1\eta(j)}^{T} (Fx_0^i + Bu_0^i)\lambda_i \right\} &\leq \sum_{i=1}^{L} \lambda_i [t]_j
\end{aligned}
$$

Simplification of the above expression gives Expression (6.5) where x_0^p is given by Expression (6.6) and u_0^p is as follows:

$$u_0^p = \sum_{i=1}^{L} \lambda_i u_0^i, \quad \sum_{i=1}^{L} \lambda_i = 1, \quad \lambda_i \geq 0.$$

Expression (6.5) thus obtained holds true because it is a non-negative combination (in fact, a convex combination) of inequalities that hold true. □

6.2.1 Direct Approach to Testing for Maintainability

To test for maintainability of $Tube(T, t)$, for each extreme point x_0^i ($i = 1, \ldots, L$) of $Tx_0 \leq t$, solve Expression (6.4) for the existence of a u_0^i. The tube is not maintainable if there does not exist a u_0^i even for one x_0^i.

The direct approach has the following computational steps:

1. Form Expression (6.4). This is done as follows:

   (a) For $k = 1, \ldots, j$ enumerate the extreme points of the polyhedron of Expression (6.3). That is, solve j extreme-point enumeration problems.

   (b) Enumerate the extreme points of $Tx_0 \leq t$.

2. For $i = 1, \ldots, L$ solve the problem of determining an interior point of the polyhedron of Expression (6.4) to confirm the existence of u for the given possibilities of x's and z's.

6.2.2 Complexity of Direct Approach

Extreme-point enumeration algorithms are not of polynomial complexity except in certain special cases [Bre97].

Determining an interior point is a linear programming problem. In general, a linear programming problem can be solved in time that is a polynomial in the dimension of the problem [BT97]. In our case, the dimension of the problem — which is the dimension of u_0^i — is p.

Thus, the complexity of the direct approach is determined by $j + 1$ non-polynomial time computations plus L polynomial-time computations.

6.2.3 Using the Q Matrix to Test for Maintainability

As described in Chapter 3, to test for maintainability of $\mathrm{Tube}(T, t)$, for each extreme point x_0^i ($i = 1, \ldots, L$) of $Tx_0 \leq t$, solve the expression $QBu_0 \leq t - QFx_0^i$ for u_0. The tube is not maintainable if there does not exist a u_0 even for one x_0^i.

This approach has the following computational steps:

1. Determine Q as follows:

   (a) Determine $\mathcal{B}$ — the set of all (x_0, u_0) tuples for which $\mathrm{Reach}(x_0, u_0)$ is irredundant.

(b) Choose an $(x_0^1, u_0^1) \in \mathcal{B}$.

(c) For this tuple (x_0^1, u_0^1), determine $Reach(x_0^1, u_0^1)$ (that is, evaluate $Fx_0^1 + Bu_0^1$).

(d) For $i = 1, \ldots, j$, where j is the number of rows of T, note the constraints of $Reach(x_0^1, u_0^1)$ that are active at the extreme point of $Reach(x_0^1, u_0^1)$ at which $[T]_i x_1$ is maximum.

By Lemma 5.1, $[T]_i$ is a non-negative linear combination of the direction vectors of these constraints.

(e) Then, the i-th row of Q will have the coefficients of this linear combination in those columns whose indices are the same as the indices of the constraints that were active at this particular extreme point. In the remaining places of the i-th row of Q, there will be zeros.

2. Test whether $Tube(T, t)$ is maintainable as follows:

(a) Enumerate the extreme points x_0^i, $i = 1, \ldots, L$, of $Tube(T, t)$.

(b) For $i = 1, \ldots, L$ determine an interior point u_0^i of the polyhedron $QBu_0 \leq t - QFx_0^i$.

6.2.4 Complexity of the Q-Matrix Approach

Step 1a mainly involves m extreme point enumeration problems if we use the method in Appendix B.
Step 1b is a linear programming problem.
Step 1c involves two matrix multiplication operations and one vector addition operation. The complexity of this step can be ignored.
Step 1d mainly involves j linear programming maximization problems.
Step 1e's complexity can be ignored.
Step 2a involves 1 extreme point enumeration problem.
Step 2b is a linear programming problem.

Thus, the Q matrix approach involves a total of $m+1$ extreme point enumeration problems (non-polynomial time) and $j+1$ linear programming problems (polynomial time).

6.2.5 Comparison of the Two Approaches to Test for Maintainability

In practice, it is expected that $m \approx j$ (the number of constraints defining the $Reach$ sets may be approximately equal to the number of constraints defining the $Tube$). So, the number of extreme point enumeration problems solved by both approaches is approximately the same. Since L may usually be much greater than j (the number of extreme points of a bounded polyhedron are usually greater than the number of constraints defining it), the Q matrix approach may involve fewer linear programming problems.

For those halfspace systems in which the intersection of $\mathcal{B}$ with $\mathcal{A}$ (the set of allowable states and admissible controls) is "sufficiently" large, a matrix Q determined for one irredundant $Reach(x_0, u_0)$ set will work for most other $Reach$ sets of this system. Thus, it can be assumed that Q is invariant for most values of the pair (x_0, u_0).

The disadvantage of the Q matrix approach is that, since it depends upon the invariance of Q, if Q is not invariant, this approach may decide that a $Reach(x_0, u_0)$ set does not lie inside the $Tube$ even when $Reach(x_0, u_0) \subseteq Tube$ (this idea is explained in Section 4.7). Thus, this approach may label a tube as not maintainable even when it is maintainable. However, it will label a tube as maintainable only if it is indeed maintainable.

In contrast, the LP-based approach labels a tube as maintainable if and only if it is indeed maintainable.

6.3 Maintaining the halfspace System in the Tube

Consider the algorithm of Section 6.3.1 for maintaining the system's states inside the tube. This algorithm assumes that

1. Sensing is done in every period.

2. Sensing is accurate and hence point-valued (as opposed to set-valued).

6.3.1 Maintaining Algorithm

INPUT : The pairs (x_0^1, u_0^1), (x_0^2, u_0^2), ..., (x_0^L, u_0^L), each of which represents a vertex of $Tx_0 \leq t$ and a maintaining control for this vertex.

1. IF $Reach(x_0^p, 0) \subseteq Tube(T, t)$, THEN

   No maintaining control needs to be applied.

   ELSE

   (a) Determine $\lambda_1, \lambda_2, \ldots, \lambda_L$ in the following convex combination (this is Expression (6.6)):

   $$x_0^p = \sum_{i=1}^{L} \lambda_i x_0^i, \quad \sum_{i=1}^{L} \lambda_i = 1, \quad \lambda_i \geq 0.$$

   (b) Determine a maintaining control for x_0^p as follows:

   $$u_0^p = \sum_{i=1}^{L} u_0^i \lambda_i.$$

2. Sense the halfspace system's state. Let this state be x_1^p (it is assumed that the sensing is exact).

   SET $x_0^p \Leftarrow x_1^p$.

3. GOTO item 1.

In the maintaining algorithm, the two computations that chiefly determine the complexity of the implementation are:

C1: Testing whether $Reach(x_0^p, 0) \subseteq Tube(T, t)$.

C2: Determining a $[\lambda_1, \lambda_2, \ldots, \lambda_L]$ vector that satisfies Expression (6.6).

6.3.2 Direct Implementation of Maintaining Algorithm

1. C1 can be implemented by testing if Expression (6.4) is satisfied by the given x_0^p and $u_0^p = 0$. This involves multiplying a $[\eta(1)+\eta(2)+\ldots+\eta(j)] \times m$ matrix by an $m \times 1$ vector (Fx_0^p) and then comparing the product with a $[\eta(1)+\eta(2)+\ldots+\eta(j)] \times 1$ vector.

2. C2 can be set up as a linear programming problem as follows:

$$\max_{\substack{x_0^1\lambda_1+x_0^2\lambda_2+\ldots+x_0^L\lambda_L=x_0^p \\ \lambda_1+\lambda_2+\ldots+\lambda_L=1 \\ \lambda_1\geq 0 \\ \ldots\ldots \\ \lambda_L\geq 0}} 0\lambda_1 + 0\lambda_2 + \ldots + 0\lambda_L$$

6.3.3 Implementation of Maintaining Algorithm using Q

1. C1 can be implemented by testing if $QFx_0^p \leq t$. This involves multiplying a $j \times m$ matrix by an $m \times 1$ vector (Fx_0^p) and then comparing the product with a $j \times 1$ vector.

2. Q does not figure in the implementation of C2 and so is no help.

6.3.4 Comparison of the Two Implementations of the Maintaining Algorithm

As $\eta(k) \geq 1$, $k = 1, \ldots, j$, the implementation of C1 involving Q has fewer arithmetic and comparison operations to perform than the direct implementation.

Since Q was determined for one particular *Reach* set (specifically, an irredundant *Reach* set), QFx_0^p may turn out to be not less than or equal to t even when $Reach(x_0^p, 0) \in Tube(T, t)$. But, QFx_0^p will be evaluated as less than or equal to t only if $Reach(x_0^p, 0) \in Tube(T, t)$ (this idea is explained in Section 4.7).

In contrast, in the direct implementation of C1, the pair $(x_0^p, 0)$ will satisfy Expression (6.5) if and only if $Reach(x_0^p, 0) \in Tube(T, t)$.

Chapter 7

Matrix Polytopes, Halfspace Systems, Difference Inclusions

7.1 Introduction

As mentioned in Section 2.1, the halfspace modeling framework was proposed as a potential generalization and extension of a difference inclusion (DI) and discrete-time interval systems (DTIS). DI [CDK02, LS98, APM89, Meg96, KS98] (also known as discrete inclusions) have the form $x_{k+1} = Ax_k + B_u u_k$, where $x_{k+1} \in \Re^n$, $u_k \in \Re^p$, $A \in \Re^{n\times n}$, $B_u \in \Re^{n\times p}$. A can be any matrix in the convex hull of some square matrices $A^1, A^2, \ldots, A^q$, and B_u is any matrix in the convex hull of matrices $B^1, B^2, \ldots, B^r$. In the case of a DTIS, A and B_u are interval matrices (meaning, each element of A or B_u lies in a closed interval of real numbers). Thus, a DTIS is a DI with $q = 2$.

Convex polyhedra have two representations which are duals: implicit representation (or halfspace representation) — as the intersection of a finite number of halfspaces — and Minkowski representation — as a set of lines, rays, and vertices. Implicit representation is as follows: $D = \{x \in \Re^n | Qx \leq s\}$, with $Q \in \Re^{m\times n}$,

$s \in \mathfrak{R}^m$. The Minkowski representation is as follows:

$$D = \left\{ x \in \mathfrak{R}^n \;\middle|\; \begin{array}{c} x = L\lambda + R\mu + V\nu, \\ \forall\lambda \geq 0, \\ \mu \geq 0, \\ \nu \geq 0, \\ \nu^T \mathbb{1} = 1 \end{array} \right\},$$

where L is the matrix containing the lines, R the matrix containing the rays, and V the matrix containing the vertices of the polyhedron, and ν and $\mathbb{1}$ are vectors with appropriate dimensions, and $\mathbb{1}$ is a vector of 1's.

Parametrized convex polyhedra (PCP) also have two representations [Tea02]: implicit representation — as the intersection of a finite number of halfspaces — and Minkowski representation — as a set of lines, rays, and vertices. The implicit representation is as follows:

$$D(p) = \{x \in \mathfrak{R}^n \,|\, Qx \leq Rp + s\}$$

with $Q, R \in \mathfrak{R}^{m \times n}$, $s \in \mathfrak{R}^m$. The Minkowski representation is as follows:

$$D(p) = \left\{ x \in \mathfrak{R}^n \;\middle|\; \begin{array}{c} x = L\lambda + R\mu + V(p)\nu, \\ \forall\lambda \geq 0, \\ \mu \geq 0, \\ \nu \geq 0, \\ \nu^T \mathbb{1} = 1 \end{array} \right\}.$$

For bounded PCP, the Minkowski representation is purely in terms of the parametrized vertices and is as follows:

$$D(p) = \left\{ x \;\middle|\; \begin{array}{c} x = V(p)\lambda, \\ \forall\lambda \geq 0, \\ \lambda^T \mathbb{1} = 1 \end{array} \right\}.$$

Here, $V(p)$ is a matrix containing the vertices of the polyhedron; it depends on parameters p.

It can be seen that the halfspace model is the implicit representation (or halfspace representation) of a PCP in the independent variable $\begin{bmatrix} x_k^T & u_k^T \end{bmatrix}^T$. In contrast, a DI with $u_k = 0$ is a convex hull of parametrized points, and thus is a Minkowski representation (or convex hull representation) of a parametrized polyhedron. Another difference between a DI and a halfspace model is that, while in forming a DI the modeler needs to decide whether the uncertainty in the description of next state is parametric or not (since a DI represents parametric uncertainty only), in forming a halfspace model, the modeler may not need to distinguish between plant uncertainty and uncertainty due to external disturbances or noise (since a halfspace model neither assumes nor explicitly represents parametric uncertainty alone). This may mean that the halfspace model can potentially simplify a modeler's job.

If there exist a halfspace system and a DI that represent the same PCP, then they will be called duals of each other with respect to this particular PCP. There may or may not exist both a halfspace system and a DI that represent the same PCP. Determining whether this is possible is the subject of this chapter. This may give key insights into both halfspace systems and DIs.

This chapter is organized as follows. Section 7.1 explains the notation used in this chapter. Section 7.2 introduces the concept of MATRIX POLYHEDRA, that is polyhedra in $\mathfrak{R}^{n\times n}$, and extends some of the theory of polyhedra in $\mathfrak{R}^n$ to polyhedra in $\mathfrak{R}^{n\times n}$. Section 7.3 introduces the concept of the vertices of a matrix polyhedron and presents some related results that will be used in the following section. This section also shows how DIs are related to matrix polytopes. Section 7.4 presents conjectures on how halfspace and DI modeling frameworks are related. Section 7.5 summarizes the key results of the chapter and outlines the directions for future research.

7.2 Matrix Polyhedra (Polyhedra in $\mathfrak{R}^{n\times n}$)

The definitions and results in this section can be considered as extensions of the results from conventional polyhedral theory to closed convex sets in $\mathfrak{R}^{n\times n}$.

Notation 7.1 $conv(\cdot)$ reads "convex hull of". Let $A = [\ a_1 \quad a_2 \quad \dots \quad a_n\ ] \in \Re^{n\times n}$ be a variable. Let $F = [\ f^1 \quad f^2 \quad \dots \quad f^n\ ] \in \Re^{m\times n}$ and $A^i = \left[\ a_1^i \quad a_2^i \quad \dots \quad a_n^i\ \right] \in \Re^{n\times n}$, $i = 1, \dots, p$, be constants. The a_j's, a_j^i's, and f^j's are columns of A, A^i, and F respectively.

Definition 7.1 A MATRIX CONVEX HULL POLYTOPE (MCHP) is the following convex hull:

$$conv(A^1, A^2, \dots, A^q) \stackrel{\triangle}{=} \left\{ A \;\middle|\; \begin{array}{c} A = \sum_{i=1}^{q} \lambda_i A^i, \\ \sum_{i=1}^{q} \lambda_i = 1, \\ \lambda_i \geq 0, \\ \lambda_i \in \Re, \\ \text{given } A^i \in \Re^{n\times n}, q < \infty. \end{array} \right\}.$$

Definition 7.2 A MATRIX HALFSPACE is the set

$$\left\{ A \in \Re^{n\times n} \,\middle|\, eA \leq f, \text{ given } e, f \in \Re^{1\times n} \right\}.$$

If $A = [\ a_1 \quad a_2 \quad \dots \quad a_n\ ]$, where a_i's are columns of A, and if $f = [\ f^1 \quad f^2 \quad \dots \quad f^n\ ]$, where the f^i's are columns of f, then a matrix halfspace can be viewed as the halfspaces $ea_1 \leq f^1$, $ea_2 \leq f^2$, ..., $ea_n \leq f^n$ stacked on top of each other.

Definition 7.3 A MATRIX HALFSPACE POLYHEDRON (MHP) is a set obtained as the intersection of matrix halfspaces as follows:

$$hal(E, F) \stackrel{\triangle}{=} \left\{ A \in \Re^{n\times n} \,\middle|\, EA \leq F, \text{ given } E, F \in \Re^{m\times n} \right\}.$$

Definition 7.4 A system of the form $EA \leq F$ ($A \in \Re^{n\times n}$, and $E, F \in \Re^{m\times n}$) is said to be bounded if there exists a constant K such that the absolute value of every component of every A that satisfies $EA \leq F$ is less than or equal to K.

Lemma 7.1 An MHP is a convex set.

Proof: Suppose that A^1 and A^2 satisfy $eA \leq f$ of Definition 7.2. So, $eA^1 \leq f$ and $eA^2 \leq f$. Let $\lambda \in [0, 1]$. Then, $e(\lambda A^1 + (1-\lambda)A^2) \leq \lambda f + (1-\lambda)f = f$, which proves

that $\lambda A^1 + (1-\lambda)A^2$ also satisfies $eA \leq f$. So, $eA \leq f$ is a convex set. An MHP is the intersection of convex sets such as $eA \leq f$. The intersection of convex sets is convex [BT97, page 44]. So, an MHP is convex. □

Lemma 7.2 The system $EA \leq F$ is bounded if the auxiliary system $\{rank(E) = n; E^T z = 0; z \geq 0\}$ is feasible.

Proof: The condition $EA \leq F$ is equivalent to the set of conditions $Ea_1 \leq f^1$, ..., $Ea_n \leq f^n$. The proof follows by applying to each of these n conditions the result of Section 3.2. □

In this chapter, unless otherwise stated, it will be assumed that $hal(E, F)$ is bounded.

The concept of a MCHP exists in the theory of difference and differential inclusions, even though the term MCHP may not exist. However, to the knowledge of this author, the concepts of matrix halfspace and MHP do not exist.

7.2.1 Relationship between $conv(A^1, A^2, \ldots, A^q)$ and $hal(E, F)$

The convex hull of $A^1, A^2, \ldots, A^q$ can be written as follows:

$$
\begin{aligned}
& conv(A^1, A^2, \ldots, A^q) \\
= & \left\{ \sum_{i=1}^{q} \lambda_i A^i \;\middle|\; \begin{array}{c} \sum_{i=1}^{q} \lambda_i = 1, \\ \lambda_i \geq 0, \\ \lambda_i \in \Re \end{array} \right\} \\
= & \left\{ \begin{bmatrix} \sum_{i=1}^{q} \lambda_i a_1^i & \ldots & \sum_{i=1}^{q} \lambda_i a_n^i \end{bmatrix} \;\middle|\; \begin{array}{c} \sum_{i=1}^{q} \lambda_i = 1, \\ \lambda_i \geq 0, \\ \lambda_i \in \Re \end{array} \right\} \\
= & \left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \;\middle|\; \begin{array}{c} a_1 = \sum_{i=1}^{q} \lambda_i a_1^i, \\ \ldots \\ a_n = \sum_{i=1}^{q} \lambda_i a_n^i, \\ \sum_{i=1}^{q} \lambda_i = 1, \\ \lambda_i \geq 0, \\ \lambda_i \in \Re \end{array} \right\} \qquad (7.1) \\
\subseteq & \left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \;\middle|\; \begin{array}{c} a_1 \in conv\left(a_1^1, \ldots, a_1^q\right), \\ \ldots \\ a_n \in conv\left(a_n^1, \ldots, a_n^q\right) \end{array} \right\} \qquad (7.2)
\end{aligned}
$$

It is known, from the theory of polyhedra, that a bounded polyhedron has two equivalent representations: as a convex hull of a set of points and as the intersection of a finite number of halfspaces as follows:

$$\left\{ a_j \,\middle|\, a_j \in conv\left(a_j^1, \ldots, a_j^q\right) \right\} = \left\{ a_j \,\middle|\, E^j a_j \leq f^j, \text{ given } E^j, f^j \right\}, \quad j = 1, \ldots, n.$$

Hence, the set on the right hand side of Expression (7.2) equals the following set:

$$\left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \;\middle|\; \begin{pmatrix} E^1 a_1 \leq f^1 \\ \ldots\ldots \\ E^n a_n \leq f^n \end{pmatrix}, \text{ given } E^1, \ldots, E^n, f^1, \ldots, f^n \right\}$$

Thus,

$$\begin{aligned} &conv(A^1, A^2, \ldots, A^q) \\ &\subseteq \left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \middle| \begin{pmatrix} E^1 a_1 \leq f^1 \\ \ldots\ldots \\ E^n a_n \leq f^n \end{pmatrix}, \text{ given } E^1, \ldots, E^n, f^1, \ldots, f^n \right\} \end{aligned} \tag{7.3}$$

Remark 7.1 The condition $E^1 = E^2 = \ldots = E^n = E$ gives

$$\{A \,|EA \leq F\} = \left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \middle| \begin{array}{c} a_1 \in conv\left(a_1^1, \ldots, a_1^q\right), \\ \ldots \\ a_n \in conv\left(a_n^1, \ldots, a_n^q\right) \end{array} \right\} \tag{7.4}$$

and Equation (7.3) becomes:

$$conv(A^1, A^2, \ldots, A^q) \subseteq \left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \middle| E\begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \leq \begin{bmatrix} f^1 & \ldots & f^n \end{bmatrix} \right\}$$

that is,

$$conv(A^1, A^2, \ldots, A^q) \subseteq \{A \,|EA \leq F\} \stackrel{\triangle}{=} hal(E, F)$$

7.2.2 From $conv(A^1, A^2, \ldots, A^q)$ to $hal(E, F)$

Theorem 7.1 For any given A^1, A^2, ..., $A^q \in \Re^{n \times n}$, there exist E, $F \in \Re^{m \times n}$ such that

$$\left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \middle| \begin{array}{c} a_1 \in conv\left(a_1^1, \ldots, a_1^q\right), \\ \ldots \\ a_n \in conv\left(a_n^1, \ldots, a_n^q\right) \end{array} \right\} = hal(E, F)$$

Proof: The matrices E and F can be constructed through the following steps:

1. Use a FACET-ENUMERATION algorithm [Fuk99] to obtain the subsystems $E^1 a_1 \leq f^1$, ..., $E^n a_n \leq f^n$, respectively corresponding to $conv\left(a_1^1, \ldots, a_1^q\right)$, ..., $conv\left(a_n^1, \ldots, a_n^q\right)$.

2. Determine all the row vectors that are in the set

$$\left\{\left[E^1\right]_1, \left[E^1\right]_2, \ldots\right\} \cup \left\{\left[E^2\right]_1, \left[E^2\right]_2, \ldots\right\} \cup \ldots \cup \{[E^n]_1, [E^n]_2, \ldots\}$$

but not in the set

$$\left\{\left[E^1\right]_1, \left[E^1\right]_2, \ldots\right\} \cap \left\{\left[E^2\right]_1, \left[E^2\right]_2, \ldots\right\} \cap \ldots \cap \{[E^n]_1, [E^n]_2, \ldots\}$$

where $\left[E^j\right]_i$ follows Notation 7.1. Let these row vectors be e^{r1}, e^{r2}, ..., e^{rk}. Here, the superscript ^{r} denotes that these row vectors are redundant in the sense described in the next step.

3. Append a SUITABLE subset of the following constraint set to each of the subsystems $E^1a_1 \leq f^1$, ..., $E^na_n \leq f^n$ (choose the subset such that the resulting augmented subsystems all have the same number of rows such that this number is a minimum):

$$\begin{bmatrix} e^{r1} \\ e^{r2} \\ \ldots \\ e^{rk} \end{bmatrix} a_1 \leq \begin{bmatrix} f^r \\ f^r \\ \ldots \\ f^r \end{bmatrix}$$

Here f^r is a scalar chosen such that the above constraint set is redundant for (meaning, it does not change the solution set of) every one of the subsystems $E^1a_1 \leq f^1$, ..., $E^na_n \leq f^n$.

The augmented subsystems constructed above are of the form $Ea_1 \leq f^{1a}$, ..., $Ea_n \leq f^{na}$. Here, the superscript ^{a} denotes that the vectors are augmented versions of f^1, ..., f^n. Thus, $hal(E, F) = \{A \,|EA \leq F\}$, where $F = [\ f^{1a} \quad \ldots \quad f^{na}\ ]$. Hence the proof. □

Remark 7.2 Note that some of the elements in $\left\{a_i^1, \ldots, a_i^q\right\}$ $(i = 1, \ldots, n)$ will be the vertices of $conv\left(a_i^1, \ldots, a_i^q\right)$. So, this last convex hull is equal to the convex hull of its vertices, which, in turn, by the RESOLUTION THEOREM for polyhedra [BT97, page 179], is uniquely equal to $E^ia_i \leq f^i$. Thus, a unique $E^ia_i \leq f^i$ corresponds to $conv\left(a_i^1, \ldots, a_i^q\right)$. However, the matrices E and F are not unique.

7.2.3 Complexity of $conv(A^1, A^2, \ldots, A^q)$ to $hal(E, F)$ Conversion

The complexity of the conversion from $conv(A^1, A^2, \ldots, A^q)$ to $hal(E, F)$ as described in the proof of Theorem 7.1 is mainly determined by step 1 which involves n facet-enumeration problems. Facet-enumeration algorithms are not of polynomial complexity except in certain special cases [Bre97].

7.2.4 From $hal(E, F)$ to $conv(A^1, A^2, \ldots, A^q)$

It can be shown that there exists a unique set of square matrices $A^1, A^2, \ldots, A^q$ for every $hal(E, F)$ such that $hal(E, F) = conv(A^1, A^2, \ldots, A^q)$. This is the subject of the next section.

7.3 Vertices of Matrix Polytopes

The present section describes some properties of the vertices of matrix polytopes. In this chapter, the terms VERTEX and EXTREME POINT will be considered synonymous.

Definition 7.5 A VERTEX OF AN MHP $EA \leq F$ is a point $V = [\ v^1 \quad v^2 \quad \ldots \quad v^n\ ] \in \mathfrak{R}^{n \times n}$ in MHP such that v^1 is a vertex of $Ea_1 \leq f^1$, ..., v^n is a vertex of $Ea_n \leq f^n$.

The following lemma presents an interesting property of the vertex of an MHP.

Lemma 7.3 A vertex V of an MHP $EA \leq F$ satisfies the following condition: there do not exist $A^a, A^b \in \{A \,|EA \leq F\}$ and a $\lambda \in (0, 1)$ such that $A^a \neq V$, $A^b \neq V$, and $\lambda A^a + (1 - \lambda)A^b = V$.

Proof: Any vertex v^i of the polyhedron $Ea_i \leq f^i$ $(i = 1 \ldots n)$, satisfies the property [BT97]:

$$\nexists\ a_i^a, a_i^b \in \left\{ a_i \,\middle|\, Ea_i \leq f^i \right\}, \lambda_i \in (0, 1),$$

such that

$$a_i^a \neq v^i, a_i^b \neq v^i, \lambda_i a_i^a + (1 - \lambda_i) a_i^b = v^i.$$

The proof then follows by letting $A^a = \begin{bmatrix} a_1^a & \dots a_n^a \end{bmatrix}$, $A^b = \begin{bmatrix} a_1^b & \dots a_n^b \end{bmatrix}$, $A = [\ a_1 \ \dots a_n\ ]$, $V = [\ v^1 \ \dots v^n\ ]$, $F = [\ f^1 \ \dots f^n\ ]$, and by observing that if there do not exist $\lambda_1, \lambda_2, \dots, \lambda_n$ — possibly all different — to satisfy any given property, then there cannot exist one single λ to satisfy that same property. □

7.3.1 A Listing of Vertices

Given that

$$\begin{array}{rll}
\left\{a_1^1, \dots, a_1^{q_1-1}, a_1^{q_1}\right\} & \text{is the set of all the different vertices of} & Ea_1 \le f^1, \\
\left\{a_2^1, \dots, a_2^{q_2-1}, a_2^{q_2}\right\} & \text{is the set of all the different vertices of} & Ea_2 \le f^2, \\
\dots & \dots & \dots \\
\left\{a_{n-1}^1, \dots, a_{n-1}^{q_{n-1}-1}, a_{n-1}^{q_{n-1}}\right\} & \text{is the set of all the different vertices of} & Ea_{n-1} \le f^2, \\
\left\{a_n^1, \dots, a_n^{q_n-1}, a_n^{q_n}\right\} & \text{is the set of all the different vertices of} & Ea_n \le f^n,
\end{array}$$

the MHP can be expressed as follows:

$$\begin{aligned}
\{A \,|EA \le F\} &= \left\{ [\ a_1 \ \dots \ a_n\ ] \,\middle|\, Ea_1 \le f^1, \dots, Ea_n \le f^n \right\} \\
&= \left\{ [\ a_1 \ \dots \ a_n\ ] \,\middle|\, \begin{array}{c} a_1 \in \text{conv}\left(a_1^1, a_1^2, \dots, a_1^{q_1}\right), \\ \dots \\ a_n \in \text{conv}\left(a_n^1, a_n^2, \dots, a_n^{q_n}\right) \end{array} \right\}
\end{aligned}$$

A VERTEX of $EA \le F$ is an n-tuple $[\ v^1 \ v^2 \ \dots \ v^n\ ]$ such that $v^1 \in \left\{a_1^1, a_1^2, \dots, a_1^{q_1-1}, a_1^{q_1}\right\}$, $v^2 \in \left\{a_2^1, a_2^2, \dots, a_2^{q_2-1}, a_2^{q_2}\right\}$, …, $v^n \in \left\{a_n^1, a_n^2, \dots, a_n^{q_n-1}, a_n^{q_n}\right\}$. The vertices of $EA \le F$ are obtained as follows:

$$\begin{array}{cccccccccccc}
\begin{bmatrix} a_1^1 & \dots & a_{n-1}^1 & a_n^1 \\ a_1^1 & \dots & a_{n-1}^2 & a_n^1 \end{bmatrix}, & \begin{bmatrix} a_1^1 & \dots & a_{n-1}^1 & a_n^2 \\ a_1^1 & \dots & a_{n-1}^2 & a_n^2 \end{bmatrix}, & \dots, & \begin{bmatrix} a_1^1 & \dots & a_{n-1}^1 & a_n^{q_n} \\ a_1^1 & \dots & a_{n-1}^2 & a_n^{q_n} \end{bmatrix}, \\
\dots \ \dots \ \dots \ \dots & \dots \ \dots \ \dots \ \dots & \dots & \dots \ \dots \ \dots \ \dots \\
\begin{bmatrix} a_1^1 & \dots & a_{n-1}^{q_{n-1}} & a_n^1 \end{bmatrix}, & \begin{bmatrix} a_1^1 & \dots & a_{n-1}^{q_{n-1}} & a_n^2 \end{bmatrix}, & \dots, & \begin{bmatrix} a_1^1 & \dots & a_{n-1}^{q_{n-1}} & a_n^{q_n} \end{bmatrix}, \\
\dots \ \dots \ \dots \ \dots & \dots \ \dots \ \dots \ \dots & \dots & \dots \ \dots \ \dots \ \dots \\
\dots \ \dots \ \dots \ \dots & \dots \ \dots \ \dots \ \dots & \dots & \dots \ \dots \ \dots \ \dots
\end{array}$$

Thus, there are a total of $q = q_1 \times q_2 \times \ldots \times q_n$ different vertices of $EA \leq F$. Let these be denoted A^1, A^2, ..., A^q.

7.3.2 Example 1

Assume that $conv_1$ is as follows:

$$conv_1 = conv\left(\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & -1 \end{bmatrix}\right) \tag{7.5}$$

The polytopes corresponding to

$$conv(a_1^1, a_1^2, a_1^3) = conv\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \quad \text{and}$$

$$conv(a_2^1, a_2^2, a_2^3) = conv\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

are shown respectively in figures 7.1 and 7.2 (the darkly shaded areas). The sets of vertices of these convex hulls are respectively:

$$\left\{\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\} \quad \text{and} \quad \left\{\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right\}$$

The polytopes $E^1a_1 \leq f^1$ (with $a_1 = [x_1\ y_1]^T$) and $E^2a_2 \leq f^2$ (with $a_2 = [x_2\ y_2]^T$) that correspond to these sets of vertices are respectively

$$\begin{array}{rcrcl} x_1 & + & y_1 & \leq & 1 \\ -x_1 & & & \leq & 0 \\ & - & y_1 & \leq & 0 \end{array} \quad \text{and} \quad \begin{array}{rcrcl} -x_2 & - & y_2 & \leq & 1 \\ x_2 & & & \leq & 0 \\ & & y_2 & \leq & 0 \end{array}$$

To obtain the MHP corresponding to these systems, the following systems of constraints (shown using dashed lines in the figures), which are redundant with respect to $E^1a_1 \leq f^1$ and to $E^2a_2 \leq f^2$ respectively, can be appended to each of the systems:

$$\begin{array}{rcrcl} -x_1 & - & y_1 & \leq & 1 \\ x_1 & & & \leq & 1 \\ & & y_1 & \leq & 1 \end{array} \quad \text{and} \quad \begin{array}{rcrcl} x_2 & + & y_2 & \leq & 1 \\ -x_2 & & & \leq & 1 \\ & - & y_2 & \leq & 1 \end{array}$$

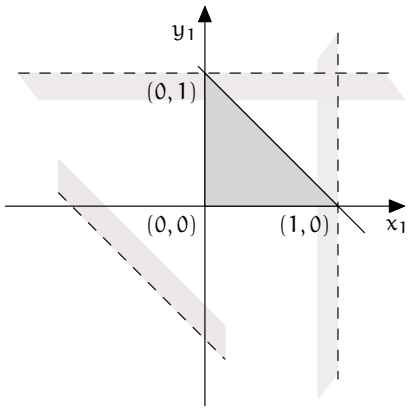


Figure 7.1: $\mathrm{conv}(a_1^1, a_1^2, a_1^3)$ for the example of Subsection 7.3.2.

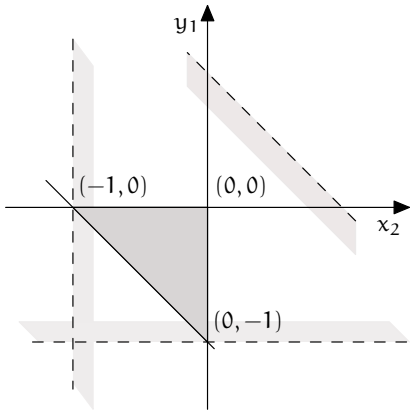


Figure 7.2: $\mathrm{conv}(a_2^1, a_2^2, a_2^3)$ for the example of Subsection 7.3.2.

Then, the MHP $EA \leq F$ is as follows:

$$\begin{bmatrix} 1 & 1 \\ -1 & 0 \\ 0 & -1 \\ -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} \leq \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \tag{7.6}$$

Listed below are all the vertices of this MHP:

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix},$$
$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$
$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & -1 \end{bmatrix}$$

7.3.3 Example 2

Suppose that $EA \leq F$ is an MHP in $\Re^{2\times 2}$, that is, that $A = [\ a_1 \quad a_2\ ] \in \Re^{2\times 2}$. Suppose that the vertices of $Ea_1 \leq f^1$ are four in number:

$$\left\{a_1^1, a_1^2, a_1^3, a_1^4\right\}$$

Suppose that the vertices of $Ea_2 \leq f^2$ are three in number:

$$\left\{a_2^1, a_2^2, a_2^3\right\}$$

Thus, the vertices of the MHP are as follows:

$$\begin{array}{lll}
\left[\begin{array}{cc} a_1^1 & a_2^1 \end{array}\right], & \left[\begin{array}{cc} a_1^1 & a_2^2 \end{array}\right], & \left[\begin{array}{cc} a_1^1 & a_2^3 \end{array}\right], \\
\left[\begin{array}{cc} a_1^2 & a_2^1 \end{array}\right], & \left[\begin{array}{cc} a_1^2 & a_2^2 \end{array}\right], & \left[\begin{array}{cc} a_1^2 & a_2^3 \end{array}\right], \\
\left[\begin{array}{cc} a_1^3 & a_2^1 \end{array}\right], & \left[\begin{array}{cc} a_1^3 & a_2^2 \end{array}\right], & \left[\begin{array}{cc} a_1^3 & a_2^3 \end{array}\right], \\
\left[\begin{array}{cc} a_1^4 & a_2^1 \end{array}\right], & \left[\begin{array}{cc} a_1^4 & a_2^2 \end{array}\right], & \left[\begin{array}{cc} a_1^4 & a_2^3 \end{array}\right].
\end{array}$$

Let these be denoted A^1, A^2, ..., A^{12}. It will be shown that

$$\begin{aligned}
&conv(A^1, A^2, \ldots, A^{12}) = \\
&\quad = \left\{[\ a_1 \quad a_2\ ] \,\middle|\, a_1 \in conv\left(a_1^1, \ldots, a_1^4\right), a_2 \in conv\left(a_2^1, \ldots, a_2^3\right)\right\}
\end{aligned}$$

That is, it will be shown that

$$\left\{ \begin{array}{l} \left[\begin{array}{cc} a_1^1 & a_2^1 \end{array}\right]\lambda_1 + \left[\begin{array}{cc} a_1^1 & a_2^2 \end{array}\right]\lambda_2 + \left[\begin{array}{cc} a_1^1 & a_2^3 \end{array}\right]\lambda_3 + \\ \left[\begin{array}{cc} a_1^2 & a_2^1 \end{array}\right]\lambda_4 + \left[\begin{array}{cc} a_1^2 & a_2^2 \end{array}\right]\lambda_5 + \left[\begin{array}{cc} a_1^2 & a_2^3 \end{array}\right]\lambda_6 + \\ \left[\begin{array}{cc} a_1^3 & a_2^1 \end{array}\right]\lambda_7 + \left[\begin{array}{cc} a_1^3 & a_2^2 \end{array}\right]\lambda_8 + \left[\begin{array}{cc} a_1^3 & a_2^3 \end{array}\right]\lambda_9 + \\ \left[\begin{array}{cc} a_1^4 & a_2^1 \end{array}\right]\lambda_{10} + \left[\begin{array}{cc} a_1^4 & a_2^2 \end{array}\right]\lambda_{11} + \left[\begin{array}{cc} a_1^4 & a_2^3 \end{array}\right]\lambda_{12} \end{array} \,\middle|\, \begin{array}{c} \sum_{i=1}^{12}\lambda_i = 1, \\ \lambda_i \geq 0 \end{array} \right\} =$$

$$\left\{ \left[\begin{array}{cc} a_1^1\beta_1 + a_1^2\beta_2 + a_1^3\beta_3 + a_1^4\beta_4 & a_2^1\gamma_1 + a_2^2\gamma_2 + a_2^3\gamma_3 \end{array}\right] \,\middle|\, \begin{array}{c} \sum_1^4 \beta_j = 1, \\ \sum_1^3 \gamma_j = 1, \\ \beta_i \geq 0, \\ \gamma_j \geq 0. \end{array} \right\}$$

That is, it will be shown that

$$\left\{ \left[\begin{array}{ll} a_1^1(\lambda_1+\lambda_2+\lambda_3)+ & a_2^1(\lambda_1+\lambda_4+\lambda_7+\lambda_{10})+ \\ a_1^2(\lambda_4+\lambda_5+\lambda_6)+ & a_2^2(\lambda_2+\lambda_5+\lambda_8+\lambda_{11})+ \\ a_1^3(\lambda_7+\lambda_8+\lambda_9)+ & a_2^3(\lambda_3+\lambda_6+\lambda_9+\lambda_{12}) \\ a_1^4(\lambda_{10}+\lambda_{11}+\lambda_{12}) & \end{array} \right] \middle| \begin{array}{c} \sum_{i=1}^{12}\lambda_i=1, \\ \lambda_i \geq 0 \end{array} \right\} =$$

$$\left\{ \left[\begin{array}{ll} a_1^1\beta_1+a_1^2\beta_2+a_1^3\beta_3+a_1^4\beta_4 & a_2^1\gamma_1+a_2^2\gamma_2+a_2^3\gamma_3 \end{array} \right] \middle| \begin{array}{c} \sum_1^4\beta_j=1, \\ \sum_1^3\gamma_k=1, \\ \beta_j \geq 0, \\ \gamma_k \geq 0 \end{array} \right\}$$

Thus, it will be shown that, for

$$\sum_{i=1}^{12}\lambda_i=1, \lambda_i \geq 0, \tag{7.7}$$

and

$$\sum_{1}^{4}\beta_j=1, \sum_{1}^{3}\gamma_k=1, \beta_j \geq 0, \gamma_k \geq 0, \tag{7.8}$$

the following system of equations holds true:

$$\left.\begin{array}{rcl} \lambda_1+\lambda_2+\lambda_3 & = & \beta_1 \\ \lambda_4+\lambda_5+\lambda_6 & = & \beta_2 \\ \lambda_7+\lambda_8+\lambda_9 & = & \beta_3 \\ \lambda_{10}+\lambda_{11}+\lambda_{12} & = & \beta_4 \\ \lambda_1+\lambda_4+\lambda_7+\lambda_{10} & = & \gamma_1 \\ \lambda_2+\lambda_5+\lambda_8+\lambda_{11} & = & \gamma_2 \\ \lambda_3+\lambda_6+\lambda_9+\lambda_{12} & = & \gamma_3 \end{array}\right\} \tag{7.9}$$

The system of Equation (7.9) can be written as follows:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \\ \lambda_7 \\ \lambda_8 \\ \lambda_9 \\ \lambda_{10} \\ \lambda_{11} \\ \lambda_{12} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \tag{7.10}$$

One way of viewing what needs to be proved is shown below:

$$\left\{ \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \middle| \begin{array}{c} \sum_{j=1}^{4} \beta_j = 1, \\ \sum_{k=1}^{3} \gamma_k = 1, \\ \beta_j \geq 0, \\ \gamma_k \geq 0 \end{array} \right\} \stackrel{?}{=} conv \left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \right.$$
$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$
$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$
$$\left. \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right).$$

This can be re-written as follows:

$$\mathcal{C}_0 \triangleq \left\{ \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \middle| \begin{array}{rcl} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} & \in & \mathrm{conv}\left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right), \\ \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} & \in & \mathrm{conv}\left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \end{array} \right\} \stackrel{?}{=}$$

$$\stackrel{?}{=} \operatorname{conv}\left(\begin{bmatrix}1\\0\\0\\0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\\0\\0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\0\\0\\1\end{bmatrix}. \right) =$$

$$= \quad conv \left(\begin{array}{cccc} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, & \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, & \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, & \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, & \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, & \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, & \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \end{array} \right) \stackrel{\triangle}{=} \mathcal{C}_1$$

Thus, it needs to be proved that $\mathcal{C}_0 = \mathcal{C}_1$.

Note that, whereas the coefficients of the convex combination of

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

in any element of $\mathcal{C}_0$ may be different from the coefficients of the convex combina-

tion of

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

in $\mathcal{C}_1$, the coefficients of the convex combination of

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$
$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

have to be the same as those of

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$
$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix},$$
$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Thus, it may seem that $\mathcal{C}_1 \subseteq \mathcal{C}_0$. However, in the following, it will be proved that $\mathcal{C}_1 = \mathcal{C}_0$.

Step 1: Note that

$$
\begin{aligned}
\mathcal{C}_1 &= conv\left(\begin{array}{c}
conv\left(\begin{bmatrix}1\\0\\0\\0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\1\\0\\0\end{bmatrix}\right), \\
conv\left(\begin{bmatrix}1\\0\\0\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\0\\1\\0\end{bmatrix}\right), \\
conv\left(\begin{bmatrix}1\\0\\0\\0\\0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\0\\0\\1\end{bmatrix}\right)
\end{array}\right) \\
&= conv\left(\begin{bmatrix}\delta_{11}\\\delta_{21}\\\delta_{31}\\\delta_{41}\\1\\0\\0\end{bmatrix}, \begin{bmatrix}\delta_{12}\\\delta_{22}\\\delta_{32}\\\delta_{42}\\0\\1\\0\end{bmatrix}, \begin{bmatrix}\delta_{13}\\\delta_{23}\\\delta_{33}\\\delta_{43}\\0\\0\\1\end{bmatrix}\right)
\end{aligned}
$$

That is,

$$\mathcal{C}_1 = \left\{ \left[\begin{array}{c} \left[\begin{array}{c} \delta_{11} \\ \delta_{21} \\ \delta_{31} \\ \delta_{41} \end{array} \right] \lambda_1 + \left[\begin{array}{c} \delta_{12} \\ \delta_{22} \\ \delta_{32} \\ \delta_{42} \end{array} \right] \lambda_2 + \left[\begin{array}{c} \delta_{13} \\ \delta_{23} \\ \delta_{33} \\ \delta_{43} \end{array} \right] \lambda_3 \\ \left[\begin{array}{c} 1 \\ 0 \\ 0 \end{array} \right] \lambda_1 + \left[\begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right] \lambda_2 + \left[\begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right] \lambda_3 \end{array} \right] \middle| \begin{array}{c} \sum_{i=1}^{3} \lambda_i = 1, \\ \lambda_i \geq 0 \end{array} \right\}$$

where,

$$\left[\begin{array}{c} \delta_{11} \\ \delta_{21} \\ \delta_{31} \\ \delta_{41} \end{array} \right], \left[\begin{array}{c} \delta_{12} \\ \delta_{22} \\ \delta_{32} \\ \delta_{42} \end{array} \right], \left[\begin{array}{c} \delta_{13} \\ \delta_{23} \\ \delta_{33} \\ \delta_{43} \end{array} \right] \in conv \left(\left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \end{array} \right], \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \end{array} \right], \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \end{array} \right], \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \end{array} \right] \right)$$

Step 2: In $\mathcal{C}_1$, even though the coefficients of the convex combination of

$$\left[\begin{array}{c} 1 \\ 0 \\ 0 \end{array} \right], \left[\begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right], \left[\begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right],$$

are the same as those of

$$\left[\begin{array}{c} \delta_{11} \\ \delta_{21} \\ \delta_{31} \\ \delta_{41} \end{array} \right], \left[\begin{array}{c} \delta_{12} \\ \delta_{22} \\ \delta_{32} \\ \delta_{42} \end{array} \right], \left[\begin{array}{c} \delta_{13} \\ \delta_{23} \\ \delta_{33} \\ \delta_{43} \end{array} \right],$$

since the δ's can be random, the following can be written

$$
\begin{aligned}
\mathcal{C}_1 &= \left\{ \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \middle| \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} \in \operatorname{conv}\left(\begin{bmatrix} \delta_{11} \\ \delta_{21} \\ \delta_{31} \\ \delta_{41} \end{bmatrix}, \begin{bmatrix} \delta_{12} \\ \delta_{22} \\ \delta_{32} \\ \delta_{42} \end{bmatrix}, \begin{bmatrix} \delta_{13} \\ \delta_{23} \\ \delta_{33} \\ \delta_{43} \end{bmatrix} \right) \right. \\
&\qquad \left. \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \in \operatorname{conv}\left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \right\} \\
&= \left\{ \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \middle| \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} \in \operatorname{conv}\left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) \right. \\
&\qquad \left. \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \in \operatorname{conv}\left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \right\} = \mathcal{C}_0
\end{aligned}
$$

7.3.4 Relationship between a Bounded MHP and Convex Hull of its Vertices

Theorem 7.2 A nonempty and bounded MHP is the convex hull of its vertices.

Proof: It needs to be shown that

$$\left\{ A \,\middle|\, EA \leq F, A \in \mathfrak{R}^{n \times n} \right\} = \operatorname{conv}(A^1, A^2, \ldots, A^q), \tag{7.11}$$

that is, it needs to be shown that

$$\operatorname{conv}(A^1, A^2, \ldots, A^q) = \left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \middle| \begin{array}{c} a_1 \in \operatorname{conv}\left(a_1^1, \ldots, a_1^{q_1}\right), \\ \ldots \\ a_n \in \operatorname{conv}\left(a_n^1, \ldots, a_n^{q_n}\right) \end{array} \right\} \tag{7.12}$$

where the MHP $\{A \,|\, EA \leq F, A \in \mathfrak{R}^{n \times n}\}$ and the vertices $A^1, A^2, \ldots, A^q$ have been defined in Section 7.3.1.

The proof is simply a generalization of the proof given in Example 2.

Expression (7.12), with some manipulation similar to that done in Example 2, gives Equation (7.13) which is a generalization of Equation (7.10).

$$
\begin{bmatrix}
\underbrace{\begin{matrix} 1 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots \end{matrix}}_{q_2 \times q_3 \times \dots \times q_n} & & & \\
\overbrace{1 \dots 1}^{q_3 \times \dots \times q_n} \begin{matrix} \dots \\ \dots \\ \dots \end{matrix} \begin{matrix} \\ \\ 1 \dots 1 \end{matrix} & q_2 \left\{ \begin{matrix} 1 \dots 1 & \dots & & \\ & \dots & & \\ & \dots & 1 \dots 1 & \end{matrix} \right. & \begin{matrix} \dots \\ \dots \\ \dots \end{matrix} & \begin{matrix} 1 \dots 1 & \dots & \\ & \dots & \\ & \dots & 1 \dots 1 \end{matrix} \\
\begin{matrix} \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \end{matrix} & \begin{matrix} \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \end{matrix} & \begin{matrix} \dots \\ \dots \end{matrix} & \begin{matrix} \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \end{matrix} \\
\begin{matrix} 1 & \dots & 1 \\ \ddots & \dots & \ddots \\ 1 & \dots & 1 \end{matrix} & q_n \left\{ \begin{matrix} 1 & \dots & 1 \\ \ddots & \dots & \ddots \\ 1 & \dots & 1 \end{matrix} \right. & \begin{matrix} \dots \\ \\ \dots \end{matrix} & \begin{matrix} 1 & \dots & 1 \\ \ddots & \dots & \ddots \\ 1 & \dots & 1 \end{matrix}
\end{bmatrix}
\times
\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \vdots \\ \lambda_q \end{bmatrix}
=
\begin{bmatrix}
\beta_1^1 \\ \beta_1^2 \\ \dots \\ \beta_1^{q_1} \\ \\
\beta_2^1 \\ \beta_2^2 \\ \dots \\ \beta_2^{q_2} \\ \\
\dots \\ \dots \\ \\
\beta_n^1 \\ \beta_n^2 \\ \dots \\ \beta_n^{q_n}
\end{bmatrix}
\tag{7.13}
$$

Note that the pattern of the 1's in the matrix in Equation (7.13) is similar to that in Equation (7.10). The method employed in Step 1 and Step 2 in Example 2 can be applied recursively to this general matrix to arrive at the proof. □

7.3.5 Complexity of $\mathcal{hal}(E, F)$ to $conv(A^1, A^2, \ldots, A^q)$ conversion

The conversion from $\mathcal{hal}(E, F)$ to $conv(A^1, A^2, \ldots, A^q)$ has the following main computational steps:

1. Enumeration of the vertices of the subsystems $Ea_1 \leq f^1$, $Ea_2 \leq f^2$, ..., $Ea_n \leq f^n$.

2. Forming the matrices $A^1, A^2, \ldots, A^q$ from the sets of vertices determined in step 1.

The complexity of the conversion is determined chiefly by step 1 which involves n vertex-enumeration problems. Vertex-enumeration algorithms are not of polynomial complexity except in certain special cases [Bre97].

7.3.6 Obtaining DI from matrix polytopes

Lemma 7.4 Consider the set obtained by multiplying every element of $conv(A^1, A^2, \ldots, A^q)$ by a vector $x_k^1 \in \mathfrak{R}^n$:

$$conv(A^1, A^2, \ldots, A^q) \times x_k^1 = \left\{ Ax_k^1 \middle| A = \sum_{i=1}^{q} \lambda_i A^i, \sum_{i=1}^{q} \lambda_i = 1, \lambda_i \geq 0, \lambda_i \in \mathfrak{R}, \text{ given } A^i \in \mathfrak{R}^{n \times n}, q < \infty \right\}$$

This set is a polytope.

Proof: We have:

$$\left\{ Ax_k^1 \middle| A = \sum_{i=1}^{q} \lambda_i A^i, \sum_{i=1}^{q} \lambda_i = 1, \lambda_i \geq 0, \lambda_i \in \mathfrak{R}, \text{ given } A^i \in \mathfrak{R}^{n \times n}, q < \infty \right\} =$$
$$\left\{ \sum_{i=1}^{q} \lambda_i A^i x_k^1 \middle| \sum_{i=1}^{q} \lambda_i = 1, \lambda_i \geq 0, \lambda_i \in \mathfrak{R}, \text{ given } A^i \in \mathfrak{R}^{n \times n}, q < \infty \right\}.$$

Thus, $conv(A^1, A^2, \ldots, A^1) \times x_k^1$ is the same as $conv(A^1 x_k^1, A^2 x_k^1, \ldots, A^q x_k^1)$ which is a polytope. □

7.4 Future research — relationship between halfspace and DI

It is hoped that the following results will proved in at least some special cases:

Conjecture 7.1 For a bounded halfspace system $Ex_{k+1} \leq Fx_k$, there exist square matrices A^1, A^2, …, A^q, where $q < \infty$, such that

1. The following equation (Equation (7.11)) holds:

$$\left\{A \,\middle|\, EA \leq F, A \in \Re^{n \times n}\right\} = conv(A^1, A^2, \ldots, A^q)$$

2. Over the set of all x_k for which $Ex_{k+1} \leq Fx_k$ is feasible, the DI $x_{k+1} \in conv(A^1, A^2, \ldots, A^q) \times x_k$ and the halfspace model $Ex_{k+1} \leq Fx_k$ are equivalent.

Conjecture 7.2 For any difference inclusion $x_{k+1} = Ax_k$, where $A \in conv(A^1, A^2, \ldots, A^q)$, $q < \infty$, there exists a bounded halfspace model $Ex_{k+1} \leq Fx_k$, such that

1. The following equation (Equation (7.4)) holds:

$$\left\{ \begin{bmatrix} a_1 & \ldots & a_n \end{bmatrix} \,\middle|\, \begin{array}{c} a_1 \in conv\left(a_1^1, \ldots, a_1^q\right), \\ \ldots \\ a_n \in conv\left(a_n^1, \ldots, a_n^q\right) \end{array} \right\} = \{A \,|EA \leq F\}$$

2. Over the set of all x_k for which $Ex_{k+1} \leq Fx_k$ is feasible,

$$\left\{x_{k+1} | x_{k+1} \in conv(A^1 x_k, A^2 x_k, \ldots, A^q x_k)\right\} \subseteq \{x_{k+1} | Ex_{k+1} \leq Fx_k\}.$$

Remark 7.3 The problem of determining the set of all values for which the halfspace system $Ex_{k+1} \leq Fx_k$ is feasible can be solved using the method described in Appendix C.

7.5 Discussion

In the present chapter, the relationship between the halfspace system $Ex_{k+1} \leq Fx_k$ ($E, F \in \mathfrak{R}^{m\times n}$, and $x_k, x_{k+1} \in \mathfrak{R}^n$) and the difference inclusion (DI) $x_{k+1} = Ax_k$, $A \in conv(A^1, A^2, \ldots, A^q)$, has been explored. In this context, the following results have been developed:

1. The concept of a matrix polytope — $EA \leq F$, $A \in \mathfrak{R}^{n\times n}$ — has been motivated and some results from the theory of polyhedra have been extended to matrix polytopes.

2. It has been shown that difference inclusions can be obtained by multiplying a matrix polytope by x_k.

Future work will investigate if a halfspace system can be obtained by multiplying in some manner a matrix halfspace polytope by x_k. Also, the possibility of approximating a halfspace system through a difference inclusion will be investigated. The error in such approximation will need to be quantified. It is hoped that the idea of matrix polytopes developed in the present chapter will prove useful in these investigations.

Chapter 8

Conclusion

The main contributions of the present dissertation were as follows:

1. The linear inequality origins of Slack-descriptor systems have been recalled under the name of Halfspace systems.

2. Techniques from the theory of convex polyhedra and linear programming have been used to develop a rudimentary control theory for halfspace systems. Specifically, the problem of maintainability (which is the same as REACHABILITY OF A TARGET TUBE) has been solved.

3. An example of modeling system dynamics in the Halfspace framework has been presented.

4. The problem of irredundancy of linear inequality constraints has been studied. The possibility of applying Minkowski's theorem for an equilibrated system of vectors to this problem has been proposed. In private communication with Dr. Carl Lee of the Department of Mathematics at the University of Kentucky, it was understood that this Minkowski direction of solving the problem may be of interest to the polytope community.

5. The study of irredundancy has given greater insight into the structure of Halfspace systems.

6. The possibility that Halfspace systems may be related to Difference Inclusions has been mooted.

7. The concept of matrix polytopes has been introduced.

8. Demonstrating a duality between Halfspace systems and Difference Inclusions, for at least a class of systems, may potentially help solve the problem of enumerating the vertices of the parametrized polyhedron $Ax \leq By$ ($A, B \in \Re^{m \times n}$, and $x, y \in \Re^n$). This problem is described in [LW97].

Here are directions for future research:

1. In this dissertation, the concept of reachability of a target tube was developed only for static target tubes. This concept needs to be extended to dynamic target tubes.

2. Relationship between Halfspace systems and Difference Inclusions needs to be studied. Existence of such a relationship may enable the application of the Difference Inclusion control theory to Halfspace systems.

3. More examples of systems modeled by halfspace models need to be found.

4. While the method described in [Lee02] and presented in Appendix B to determine the set of all b's for which $Ax \leq b$ is irredundant is easy to understand and implement, its complexity is not polynomial. Thus, there is value in searching for other computationally less expensive methods. The techniques used in those new methods could themselves be of interest.

5. Application of ideal systems (described in Appendix A) in the study of redundancy needs to be further explored.

6. Application of Minkowski's Theorem for an equilibrated system of vectors needs to be further explored. This particular application may be of interest to the polytope community too.

Appendix A

Using Ideal Systems to Solve the Problem of Irredundancy

The motivation for this chapter comes from the discussion in Subsection 5.3.9 on Page 53. The concept of an ideal system is explained in the following paragraphs.

Lemma A.1 In Expression 5.3, let the system $\mathcal{S}(b^s)$ be such that all its component hyperplanes are tangent to a sphere and the feasible region of this system contains this sphere. Then, every one of the planes forms a facet of the polyhedron $\mathcal{S}(b^s)$.

Proof: Suppose that some constraint $a_i^T x \leq b_i^s$ does not form a facet. This can happen if
(a) there is another constraint $a_i^T x \leq \widehat{b_i^s}$, such that $\widehat{b_i^s} < b_i^s$; but the hyperplane corresponding to such a constraint cannot be tangent to the sphere, it can only intersect the sphere; or
(b) some surrounding constraints make the constraint in question redundant: possible if the locus of intersection of the surrounding constraints is at distance $l \leq r$ from the center of the sphere (r – radius), meaning that the halfspaces corresponding to these surrounding constraints are not tangent to the sphere. □

The properties of $\mathcal{S}(b^s)$ are expressed mathematically as follows:

Suppose that the sphere $(x - x_c)^T (x - x_c) = r^2$ (with center at x_c and radius r) and a constraint of the form $a^T x \leq \alpha$, where a is known, are given. The value of α needs to be determined so that the hyperplane is tangent to the sphere and the halfspace contains the sphere. For this first the point of tangency of the plane

with the sphere is determined — by finding the point on the sphere that is in the direction of a. Then, this value is substituted into the equation of the plane. This gives the value of α. Now, the point on the sphere in the direction of a is $\frac{a}{|a|}r + x_c$. Substituting this into the equation of the plane gives the value $\alpha = a^T(\frac{a}{|a|}r + x_c)$. Thus, the constraint is $a^T x \leq a^T(\frac{a}{|a|}r + x_c)$.

Remark A.1 For a given b, $\mathcal{S}(b)$ of Expression (5.3) on Page 41 will have all its component hyperplanes tangent to a sphere (radius $r > 0$, center x_c) and its feasible region will contain the sphere if and only if $\mathcal{S}(b)$ can be transformed into the form: $a_i^T x \leq a_i^T(\frac{a_i}{|a_i|}r + x_c)$, $1 \leq i \leq m$.

$\mathcal{S}(b^s)$ will be called an IDEAL SYSTEM and the corresponding polytope — an IDEAL POLYTOPE. Next, a method to determine an RDCS for each of the constraints in $\mathcal{S}(b^s)$ is developed. Of interest are only the indices of the elements of the RDCS. For convenience, the origin is translated to the center of the ideal system (this is the same as choosing $x_c = 0$). Thus, the TRANSLATED IDEAL SYSTEM (TIS) corresponding to the ideal system of Remark A.1 is:

$$\mathcal{S}(b^{tis}) \triangleq \left\{ \frac{a_i^T}{|a_i|} x \leq r \;\middle|\; 1 \leq i \leq m \right\} \tag{A.1}$$

In the following, lemmas A.2 through A.4 and Theorem A.1 describe the properties of, and help identify, the SRDCS of a constraint in an ideal system or in a TIS.

In the following, the result of Lemma 5.2 is applied to Expression (A.1).

Notation A.1 Let $\rho = [\ r \quad \dots \quad r\ ]^T \in \Re^n$, and $\mathbb{1} = [\ 1 \quad \dots \quad 1\ ]^T \in \Re^n$. Let

$$\mathcal{S}_n(b^{tis}) \triangleq \left\{ \frac{a_1^{i\,T}}{|a_1^i|} x \leq r, \dots, \frac{a_n^{i\,T}}{|a_n^i|} x \leq r \right\}, \quad \mathcal{S}'_n(b^{tis}) \triangleq \left\{ \frac{a_1^{i'T}}{|a_1^{i'}|} x \leq r, \dots, \frac{a_n^{i'T}}{|a_n^{i'}|} x \leq r \right\}$$

be subsets of $\mathcal{S}(b^{tis})$. The members of $\mathcal{S}_n(b^{tis})$ intersect at $x^n = \left(A_n^T\right)^{-1} \rho$, and those of $\mathcal{S}'_n(b^{tis})$ — at $x^{n'} = (A'^{\,T}_n)^{-1}\rho$.

Lemma A.2 For $\mathcal{S}_n(b^{tis})$ to be an RDCS of the i-th constraint in the TIS of Expression (A.1), A_n must satisfy the following properties:

(1) $A_n^{-1}\frac{a_i}{|a_i|} \geq 0$, and
(2) A_n minimizes $\frac{a_i^T}{|a_i|}\left(A_n^T\right)^{-1}\rho$.

Proof: By Theorem 5.1, for $\mathcal{S}_n(b^{tis})$ to be a candidate for an RDCS of the i-th constraint, $\frac{a_i}{|a_i|}$ must be such that $\frac{a_i}{|a_i|} \in cone(A_n)$, meaning that there must exist $\lambda \geq 0$ such that $A_n\lambda = \frac{a_i}{|a_i|}$. This gives Property (1). Property (2) is obtained by applying Lemma 5.2. □

Lemma A.2 can be used to determine A_n. However, a computationally less expensive method is suggested by the following theorem.

Theorem A.1 Given that the constraints in $\mathcal{S}_n(b^{tis})$ are such that:
(1) $\frac{a_1^{i\,T}}{|a_1^i|}\frac{a_i}{|a_i|}, \ldots, \frac{a_n^{i\,T}}{|a_n^i|}\frac{a_i}{|a_i|}$ are the largest of all $\frac{a_j^T}{|a_j|}\frac{a_i}{|a_i|}$, $j = 1 \ldots m, \quad j \neq i$,
(2) $|A_n| \neq 0$, and
(3) $A_n^{-1}\frac{a_i}{|a_i|} \geq 0$ (i.e., $\exists\lambda \geq 0$ such that $A_n\lambda = \frac{a_i}{|a_i|}$).
Then the n constraints in $\mathcal{S}_n(b^{tis})$ constitute an RDCS of the i-th constraint.

Proof: Consider the constraint sets $\mathcal{S}_n(b^{tis})$ and $\mathcal{S}'_n(b^{tis})$. It is not necessary that $\mathcal{S}_n(b^{tis}) \cap \mathcal{S}'_n(b^{tis}) = \emptyset$. From condition (3) of the theorem it follows that $\exists\lambda \geq 0$ such that $A_n\lambda = \frac{a_i}{|a_i|}$. Assume that $|A'_n| \neq 0$ and $\exists\lambda' \geq 0 : A'_n\lambda' = \frac{a_i}{|a_i|}$. Condition (1) of the theorem means that

$$\frac{a_i^T a_1^i}{|a_i|\left|a_1^i\right|} \geq \frac{a_i^T a_1^{i\prime}}{|a_i|\left|a_1^{i\prime}\right|}, \ldots, \frac{a_i^T a_n^i}{|a_i||a_n^i|} \geq \frac{a_i^T a_n^{i\prime}}{|a_i||a_n^{i\prime}|}.$$

These n inequalities can be combined thus: $\frac{a_i^T}{|a_i|}A_n \geq \frac{a_i^T}{|a_i|}A'_n$. It needs to be shown that

$$\frac{a_i^T}{|a_i|}x^n \leq \frac{a_i^T}{|a_i|}x^{n\prime}, \text{ i.e., } \lambda^T\mathbb{1} \leq \lambda'^T\mathbb{1}$$

(the last inequality has been obtained by using condition (2) of the theorem and applying the definition of x^n and $x^{n\prime}$). Postmultiplying the inequality $\frac{a_i^T}{|a_i|}A_n \geq \frac{a_i^T}{|a_i|}A'_n$ by λ and λ', gives the following comparison:

$$\frac{a_i^T}{|a_i|}A_n\lambda' \geq \frac{a_i^T}{|a_i|}A'_n\lambda' = 1 = \frac{a_i^T}{|a_i|}A_n\lambda \geq \frac{a_i^T}{|a_i|}A'_n\lambda.$$

Thus, $\lambda' = \lambda + \Delta\lambda$ such that $A_n\lambda' = A_n\lambda + A_n\Delta\lambda$ and

$$\frac{a_i^T}{|a_i|}A_n\lambda' = 1 + \frac{a_i^T}{|a_i|}A_n\Delta\lambda.$$

Now $\frac{a_i^T}{|a_i|}A_n\Delta\lambda \geq 0$. Also, $A_n'(\lambda + \Delta\lambda) = \frac{a_i}{|a_i|}$. This gives $\Delta\lambda = {A_n'}^{-1}\frac{a_i}{|a_i|} - \lambda$. Is $\Delta\lambda \geq 0$? That is, is ${A_n'}^{-1}\frac{a_i}{|a_i|} \geq \lambda$? Substituting into the inequality $\frac{a_i^T}{|a_i|}A_n\Delta\lambda \geq 0$ the expression for $\Delta\lambda$ gives $\frac{a_i^T}{|a_i|}A_n{A_n'}^{-1}\frac{a_i}{|a_i|} \geq 1$. This is true only if ${A_n'}^{-1}\frac{a_i}{|a_i|} \geq \lambda$, meaning $\Delta\lambda \geq 0$, i.e., $\lambda' \geq \lambda$. Hence the proof. □

Remark A.2 In Theorem A.1 only the information about the angle that any constraint makes with the i-th constraint was used. However, the information about the angles that the constraints make with one another (contained in the matrices $A_n^TA_n$ and ${A_n'}^TA_n'$) must matter too. This is particularly important in the following scenario: there exist n constraints which make the smallest angles with the i-th constraint while surrounding it, and there exist l ($l \leq (m-1)-n$) more constraints which make the same angles with the i-th constraint as some of those n. In such a case, to determine the RDCSs, a method that directly computes the expression $\frac{a_i^T}{|a_i|}\left(A_n^T\right)^{-1}\rho$ could be applied. However, it may be less expensive to apply a method that uses a fact such as the following

$$f\left(A_n^TA_n\right) \geq f\left({A_n'}^TA_n'\right) \iff \frac{a_i^T}{|a_i|}x^n \leq \frac{a_i^T}{|a_i|}x^{n\prime},$$

where the operation $f(\cdot)$ is computationally less expensive than the operation $(\cdot)^{-1}$. Identifying such an $f(\cdot)$ is still a topic of research.

Lemma A.3 that follows shows that in $\mathcal{S}(b^{tis})$ if $\mathcal{S}_n(b^{tis})$ is an RDCS of $\mathcal{C}_i$, then none of the constraints from the set $\mathcal{S}(b^{tis}) \setminus \{\mathcal{S}_n(b^{tis}) \cup \{\mathcal{C}_i\}\}$ is surrounded by $\mathcal{S}_n(b^{tis})$.

Notation A.2 (Please see notations 5.1 (on Page 43) and A.1 (on Page 101)).

$$A_{nj\setminus k} = \left[\frac{a_1^i}{|a_1^i|} \quad \cdots \quad \frac{a_{k-1}^i}{|a_{k-1}^i|} \quad \frac{a_j}{|a_j|} \quad \frac{a_{k+1}^i}{|a_{k+1}^i|} \quad \cdots \quad \frac{a_n^i}{|a_n^i|}\right].$$

Lemma A.3 In a TIS, assume that given are some $\mathcal{C}_i$, $\mathcal{C}_j \notin \mathcal{S}_n(b^{tis})$ such that $\frac{a_i}{|a_i|}$, $\frac{a_j}{|a_j|} \in cone(A_n)$. Then:
(1) In $\mathcal{S}_n(b^{tis})$ at least one constraint (say $\mathcal{C}_k$) can be found that can be replaced by $\mathcal{C}_j$ giving the new set of n constraints $\mathcal{S}_{nj\backslash k}$ such that $\frac{a_i}{|a_i|} \in cone(A_{nj\backslash k})$.
(2) Any $\mathcal{S}_{nj\backslash k}$ thus formed is such that

$$\frac{a_i^T}{|a_i|}\left(A_{nj\backslash k}^T\right)^{-1}\rho \leq \frac{a_i^T}{|a_i|}\left(A_n^T\right)^{-1}\rho.$$

Proof: Note that

$$\bigcup_{k=1}^{n} cone(A_{nj\backslash k}) = cone(A_n).$$

So,

$$\frac{a_i}{|a_i|} \notin cone(A_{nj\backslash k})\ (\forall k \in \{1,\ldots,n\}) \Longleftrightarrow \frac{a_i}{|a_i|} \notin cone(A_n).$$

This proves statement (1). Next, note that $a_j/|a_j| = A_n\lambda_j$, for some $\lambda_j \geq 0$. So, $A_{nj\backslash k} = A_n\Lambda_j$, where $\Lambda_j = [\ u_1\ \ u_2\ \ \ldots\ \ u_{k-1}\ \ \lambda_j\ \ u_{k+1}\ \ \ldots\ \ u_{n-1}\ \ u_n\ ]$. Here, u_k, $k = 1\ldots n$, is a unit vector with a 1 in the k-th position. Thus, to prove statement (2), it needs to be shown that

$$\frac{a_i^T}{|a_i|}\left((A_n\Lambda_j)^T\right)^{-1}\rho \leq \frac{a_i^T}{|a_i|}\left(A_n^T\right)^{-1}\rho, \ \text{ I.E., that } \lambda_i^T\left(\Lambda_j^T\right)^{-1}\mathbb{1} \leq \lambda_i^T\mathbb{1}$$

(here the fact that $A_n\lambda_i = \frac{a_i}{|a_i|}$, $\lambda_i \geq 0$ has been used). Note that $\Lambda_j^{-1} =$

$$= \begin{bmatrix} 1 & 0 & & \lambda_{j1} & & 0 & 0 \\ 0 & 1 & & \lambda_{j2} & & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots & \vdots \\ 0 & 0 & & \lambda_{jk} & & 0 & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & & \lambda_{j(n-1)} & & 1 & 0 \\ 0 & 0 & & \lambda_{jn} & & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & & -\frac{\lambda_{j1}}{\lambda_{jk}} & & 0 & 0 \\ 0 & 1 & & -\frac{\lambda_{j2}}{\lambda_{jk}} & & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots & \vdots \\ 0 & 0 & & \frac{1}{\lambda_{jk}} & & 0 & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & & -\frac{\lambda_{j(n-1)}}{\lambda_{jk}} & & 1 & 0 \\ 0 & 0 & & -\frac{\lambda_{jn}}{\lambda_{jk}} & & 0 & 1 \end{bmatrix}.$$

The above equality has been written based on [BT97, page 96].

$$\Lambda_j^{-1}\lambda_i = \begin{bmatrix} (\lambda_{i1} - \frac{\lambda_{j1}}{\lambda_{jk}}\lambda_{ik}) \\ \vdots \\ (\lambda_{i(k-1)} - \frac{\lambda_{j(k-1)}}{\lambda_{jk}}\lambda_{ik}) \\ \frac{1}{\lambda_{jk}}\lambda_{ik} \\ (\lambda_{i(k+1)} - \frac{\lambda_{j(k+1)}}{\lambda_{jk}}\lambda_{ik}) \\ \vdots \\ (\lambda_{in} - \frac{\lambda_{jn}}{\lambda_{jk}}\lambda_{ik}) \end{bmatrix}.$$

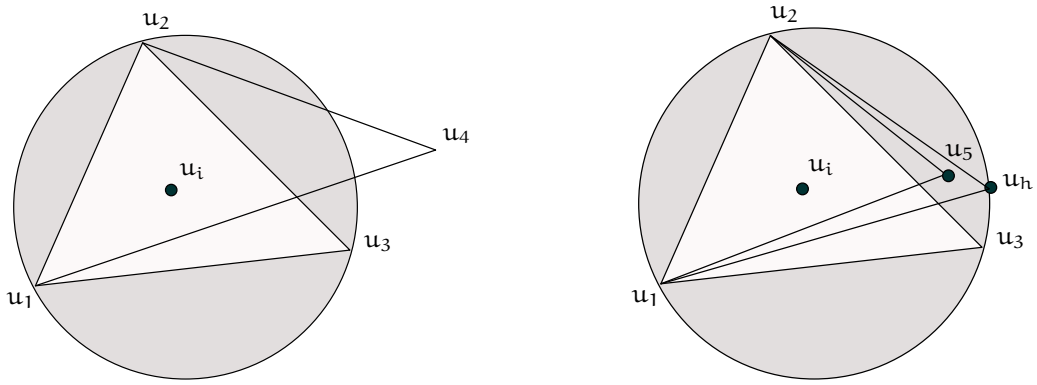
Thus,

$$\begin{aligned} \left(\Lambda_j^{-1}\lambda_i\right)^T \mathbb{1} &= \lambda_{i1} + \ldots + \lambda_{i(k-1)} + \lambda_{ik} + \lambda_{i(k+1)} + \ldots + \lambda_{in} \\ &\quad -\lambda_{ik} - \frac{\lambda_{j1}}{\lambda_{jk}}\lambda_{ik} - \ldots - \frac{\lambda_{j(k-1)}}{\lambda_{jk}}\lambda_{ik} + \frac{1}{\lambda_{jk}}\lambda_{ik} - \frac{\lambda_{j(k+1)}}{\lambda_{jk}}\lambda_{ik} - \ldots - \frac{\lambda_{jn}}{\lambda_{jk}}\lambda_{ik} \\ &= \lambda_i^T\mathbb{1} - \frac{\lambda_j^T\mathbb{1}}{\lambda_{jk}}\lambda_{ik} + \frac{1}{\lambda_{jk}}\lambda_{ik}. \end{aligned}$$

This last expression will not be greater than $\lambda_i^T\mathbb{1}$ if $\frac{\lambda_j^T\mathbb{1}}{\lambda_{jk}}\lambda_{ik} - \frac{1}{\lambda_{jk}}\lambda_{ik} \geq 0$, that is, if $\lambda_j^T\mathbb{1} \geq 1$. Note that the column vectors of A_n are of unit magnitude and hence the points represented by them lie on the surface of the hypersphere $x^Tx \leq 1$, $x \in \Re^n$. So, the set $\Phi \stackrel{\triangle}{=} \left\{A_n\lambda_j \,\middle|\, \lambda_j^T\mathbb{1} = 1, \lambda_j \geq 0\right\}$ is a $(n-1)$-dimensional polytope that lies in this hypersphere on a hyperplane that cuts this hypersphere. Thus, the points $\left\{A_n\lambda_j \,\middle|\, \lambda_j^T\mathbb{1} < 1, \lambda_j \geq 0\right\}$ are closer to the origin than points in Φ and the points $\left\{A_n\lambda_j \,\middle|\, \lambda_j^T\mathbb{1} > 1, \lambda_j \geq 0\right\}$ are farther from the origin than points in Φ. The point $a_j/|a_j|$ is on the surface of the hypersphere. It is a point of the second kind. So, for it $\lambda_j^T\mathbb{1} \geq 1$. Hence the proof. □

Thus far the possibility of the existence of more than one RDCS for each constraint in $\mathcal{S}(b)$ has been admitted. Lemma A.4 that follows shows how all the RDCSs of a constraint in $\mathcal{S}(b^{tis})$ are related to each other.

Lemma A.4 In $\mathcal{S}(b^{tis})$, assume that $\mathcal{S}_n(b^{tis})$ and $\mathcal{S}'_n(b^{tis})$ are RDCSs of the i-th



constraint. Then,

$$\frac{a_k^{i\prime}}{|a_k^{i\prime}|} \in \text{aff}\left(\frac{a_1^i}{|a_1^i|}, \ldots, \frac{a_n^i}{|a_n^i|}\right) \cap \left\{x \,\middle|\, x^T x = 1, x \in \Re^n\right\}, \quad k = 1 \ldots n. \tag{A.2}$$

Proof: (Note that the intersection on the right hand side of the above expression is simply the circumcircle of three points in 3-D, and in higher dimensions — a "hyper-circumcircle".) As both $\mathcal{S}_n(b^{tis})$ and $\mathcal{S}'_n(b^{tis})$ are RDCSs of the i-th constraint, it follows that, for some $\lambda, \lambda' (\in \Re^n) \geq 0$,

$$\begin{aligned} \frac{a_i^T}{|a_i|}(A_n^T)^{-1}\rho &= \frac{a_i^T}{|a_i|}(A_n'^T)^{-1}\rho, \\ \text{i.e., } \lambda^T\rho &= \lambda'^T\rho \end{aligned} \tag{A.3}$$

Let $A_n\lambda^k = a_k^{i\prime}/|a_k^{i\prime}|$, $\lambda^k \in \Re^n$, $k = 1 \ldots n$, and $\Lambda = [\lambda^1 \ldots \lambda^n]$. Then, it follows that $A_n\Lambda = A'_n$. Substituting this last expression into Equation (A.3) gives: $\lambda^T\rho = \lambda^T\Lambda^T\rho$. This gives $\lambda^T(\mathbb{1} - \Lambda^T\mathbb{1}) = 0$. This means that $\mathbb{1}^T\lambda^1 = 1, \ldots, \mathbb{1}^T\lambda^n = 1$. Expression (A.2) follows from this last statement combined with the fact that the points $a_1^{i\prime}/|a_1^{i\prime}|$, ..., $a_n^{i\prime}/|a_n^{i\prime}|$ lie on the unit sphere. □

The affine hull of n points in n-dimensional space is the hyperplane that passes simultaneously through these n points. Thus,

$$\text{aff}\left(\frac{a_1^i}{|a_1^i|}, \ldots, \frac{a_n^i}{|a_n^i|}\right) = \left\{x \,\middle|\, \det\left(\begin{bmatrix} x^T & 1 \\ A_n^T & \mathbb{1} \end{bmatrix}\right) = 0, x \in \Re^n\right\}. \tag{A.4}$$

If $\mathcal{S}_n(b^{tis})$ is such that the only constraint in $\mathcal{S}(b^{tis})$ surrounded by it is the i-th constraint, and $|A_n| \neq 0$, then is it an RDCS of the i-th constraint? Example 1

answers this question in the negative.

Example 1 In the left hand figure, the vectors u_1 – u_4 and u_i begin at the origin and have their ends on the unit sphere in $\Re^3$. The set Σ_1 of constraints corresponding to the set of vectors $\{u_1, u_2, u_3\}$, as well as the set Σ_2 of constraints corresponding to the set $\{u_1, u_2, u_4\}$, are such that the only constraint surrounded by each of them is C_i — the constraint corresponding to u_i. Only the first set is an RDCS. The second set does not satisfy Expression (A.2).

Example 2 This is with respect to Example 1. As Σ_1 is an RDCS of C_i, no vector belonging to its parent system of constraints can be found in the shaded region of left hand figure. This can be understood as follows. Assume for contradiction that there was some vector u_5 in the shaded region (right hand figure). It can be shown, through an argument similar to that used for the proof of statement (1) of Lemma A.3, that we can find at least one vector in the set $\{u_1, u_2, u_3\}$ (in the figure it is u_3) that can be replaced by u_5 such that $u_i \in cone(\{u_1, u_2, u_5\})$. Now, consider a hypothetical vector u_h that satisfies Expression (A.2) such that $u_5 \in cone(\{u_1, u_2, u_h\})$. The set of constraints corresponding to the set $\{u_1, u_2, u_h\}$ is an RDCS of C_i. But by statement (2) of Lemma A.3,

$$u_i^T [u_1\ u_2\ u_5]^{T^{-1}} \mathbb{1} < u_i^T [u_1\ u_2\ u_h]^{T^{-1}} \mathbb{1}.$$

This contradicts the fact that $\{u_1, u_2, u_h\}$ is an RDCS of C_i. So, no vector can be found in the shaded region.

It can be seen that a formal version of the argument presented in Example 2 can be used as a proof for Theorem A.1.

Lemma A.2 and Theorem A.1 allow the determination of an RDCS for each of the constraints in $\mathcal{S}(b^{tis})$ for which there exists an RDCS. Lemma A.3 shows that if $\mathcal{S}_n(b^{tis})$ is an RDCS of the i-th constraint in $\mathcal{S}(b^{tis})$, then this is the only constraint in $\mathcal{S}(b^{tis})$ surrounded by $\mathcal{S}_n(b^{tis})$. So, it follows that the i-th constraint in $\mathcal{S}(b^0)$ is the only constraint surrounded by $\mathcal{S}_n(b^0)$. Thus, in $\mathcal{S}(b^0)$, either $\mathcal{S}_n(b^0)$ is an RDCS

of the i-th constraint, or it is not an RDCS of any constraint at all. Next, of interest is the question whether $\mathcal{S}_n(b^0)$ is necessarily an RDCS of the i-th constraint in $\mathcal{S}(b^0)$ given that $\mathcal{S}_n$ is an RDCS of the i-th constraint in $\mathcal{S}(b^{tis})$. The possibilities to check are:

Q1: If $\mathcal{S}_n(b^{tis})$ is an RDCS of the i-th constraint in $\mathcal{S}(b^{tis})$, then is it possible that $\mathcal{S}_n(b^0)$ is not an RDCS of the i-th constraint in $\mathcal{S}(b^0)$?

Q2: If $\mathcal{S}_n(b^{tis})$ is not an RDCS of the i-th constraint in $\mathcal{S}(b^{tis})$, then is it possible that $\mathcal{S}_n(b^0)$ is an RDCS of the i-th constraint in $\mathcal{S}(b^0)$?

A counter-example was given for these questions. Thus, it remains to be seen if they may be answered in the affirmative for a class of systems.

Appendix B

Second Method to Solve $\mathcal{PR}1$ and $\mathcal{PR}2$

Notation

$\mathcal{PR}1$ and $\mathcal{PR}2$ are defined on Page 41.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}; \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_m \end{bmatrix}.$$

$A_{\setminus i}$ is the $(m-1) \times n$ matrix that is obtained by deleting a_i (see Notation 5.1 on Page 43) from A. $b_{\setminus i}$ is the $(m-1) \times 1$ vector that is obtained by deleting b_i from b. Thus,

$$A_{\setminus i} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{(i-1)1} & a_{(i-1)2} & \cdots & a_{(i-1)n} \\ a_{(i+1)1} & a_{(i+1)2} & \cdots & a_{(i+1)n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}; \quad b_{\setminus i} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_{(i-1)} \\ b_{(i+1)} \\ \cdots \\ b_m \end{bmatrix}.$$

The Problem

Given the system of constraints $Ax \leq b$, determine the set of all b for which this system is irredundant.

The Solution

The following solution was suggested by Carl Lee[Lee02] and uses LP duality.

Step 1: The constraint $a_i^T x \leq b_i$ is irredundant with respect to the system $A_{\setminus i} x \leq b_{\setminus i}$ if and only if $b_i < \mathcal{LP}_{\setminus i}^{\max}$. Thus, for the system $Ax \leq b$ to be irredundant, this condition must be satisfied for $i = 1, \ldots, m$.

Step 2: The linear programming primal problem

$$\begin{array}{ll} \max & a_i^T x \\ A_{\setminus i} x \leq b_{\setminus i} & \end{array}$$

has the dual

$$\min_{\substack{A_{\setminus i}^T y = a_i \\ y \geq 0}} b_{\setminus i}^T y$$

So, the condition for irredundancy of Step 1 can be translated into the new condition "b_i should be strictly less than the minimum of $b_{\setminus i}^T y$ over the system $A_{\setminus i}^T y = a_i, y \geq 0$".

Step 3: Enumerate the extreme points — say, y_1^i, y_2^i, ..., $y_{\nu_i}^i$ — of the polyhedron

$$\begin{array}{rcl} A_{\setminus i}^T y & = & a_i \\ y & \geq & 0 \end{array}$$

Step 4: Then, for irredundancy, the following inequality must be true:

$$b_i < \min \left\{ \begin{array}{cccc} b_{\setminus i}^T y_1^i, & \ldots, & b_{\setminus i}^T y_{\nu_i}^i \end{array} \right\}$$

Step 5: This inequality is equivalent to the following system of inequalities:

$$\begin{array}{rcl} b_i & < & b_{\setminus i}^T y_1^i \\ b_i & < & b_{\setminus i}^T y_2^i \\ \ldots & \ldots & \ldots \\ b_i & < & b_{\setminus i}^T y_{\nu_i}^i \end{array}$$

Step 6: From Step 5, it follows that the solution to $\mathcal{PR}2$ is given by the following system:

$$
\begin{array}{lcl}
b_1 & < & b_{\setminus 1}^T y_1^1 \\
b_1 & < & b_{\setminus 1}^T y_2^1 \\
\dots & \dots & \dots \\
b_1 & < & b_{\setminus 1}^T y_{\nu_1}^1 \\
 & & \\
b_2 & < & b_{\setminus 2}^T y_1^2 \\
b_2 & < & b_{\setminus 2}^T y_2^2 \\
\dots & \dots & \dots \\
b_2 & < & b_{\setminus 2}^T y_{\nu_2}^2 \\
 & & \\
\dots & \dots & \dots \\
\dots & \dots & \dots \\
 & & \\
b_m & < & b_{\setminus m}^T y_1^m \\
b_m & < & b_{\setminus m}^T y_2^m \\
\dots & \dots & \dots \\
b_m & < & b_{\setminus m}^T y_{\nu_m}^m
\end{array}
$$

The solution to $\mathcal{PR}1$ can be obtained by appending the condition $b = By$ to the above system of constraints.

Appendix C

Determining the Values of b for which $Ax \leq b$ is Feasible

The Problem

Given the system of constraints $Ax \leq b$, determine the set of all b for which this system is feasible (notation same as in Appendix B).

The Solution

The solution is based on the method shown by Dr. Carl Lee[Lee02] to determine the set of all b for which the system $Ax \leq b$ is irredundant. Indeed, our solution is a very slight modification of the solution that he gave for the problem of irredundancy.

Step 1: The constraint $a_i^T x \leq b_i$ is feasible with respect to the system $A_{\setminus i} x \leq b_{\setminus i}$ if and only if $b_i \geq \mathcal{LP}_{\setminus i}^{\min}$. Thus, for the system $Ax \leq b$ to be feasible, this condition must be satisfied for $i = 1, \ldots, m$ (notation same as in Appendix B).

Step 2: The linear programming primal problem

$$\min_{A_{\setminus i} x \leq b_{\setminus i}} a_i^T x$$

has the dual (HERE IS THE DIFFERENCE BETWEEN THE IRREDUNDANCY AND THE FEASIBILITY PROBLEMS)

$$\max_{\substack{A_{\setminus i}^T y = a_i \\ y \leq 0}} b_{\setminus i}^T y$$

So, the condition for feasibility of Step 1 can be translated into the new condition "b_i should be greater than or equal to the maximum of $b_{\setminus i}^T y$ over the system $A_{\setminus i}^T y = a_i, y \leq 0$".

Step 3: Enumerate the extreme points — say, y_1^i, y_2^i, ..., $y_{\nu_i}^i$ — of the polyhedron

$$\begin{aligned} A_{\setminus i}^T y &= a_i \\ y &\leq 0 \end{aligned}$$

Step 4: Then, for feasibility, the following inequality must be true:

$$b_i \geq \min \left\{ b_{\setminus i}^T y_1^i, \quad \ldots, \quad b_{\setminus i}^T y_{\nu_i}^i \right\}$$

Step 5: This inequality is equivalent to the following system of inequalities:

$$\begin{aligned} b_i &\geq b_{\setminus i}^T y_1^i \\ b_i &\geq b_{\setminus i}^T y_2^i \\ \ldots &\quad \ldots \quad \ldots \\ b_i &\geq b_{\setminus i}^T y_{\nu_i}^i \end{aligned}$$

Step 6: From Step 5, it follows that the solution to $\mathcal{PR}2$ is given by the following

system:

$$
\begin{array}{lcl}
b_1 & \geq & b_{\setminus 1}^T y_1^1 \\
b_1 & \geq & b_{\setminus 1}^T y_2^1 \\
\cdots & \cdots & \cdots \\
b_1 & \geq & b_{\setminus 1}^T y_{\nu_1}^1 \\
 & & \\
b_2 & \geq & b_{\setminus 2}^T y_1^2 \\
b_2 & \geq & b_{\setminus 2}^T y_2^2 \\
\cdots & \cdots & \cdots \\
b_2 & \geq & b_{\setminus 2}^T y_{\nu_2}^2 \\
 & & \\
\cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots \\
 & & \\
b_m & \geq & b_{\setminus m}^T y_1^m \\
b_m & \geq & b_{\setminus m}^T y_2^m \\
\cdots & \cdots & \cdots \\
b_m & \geq & b_{\setminus m}^T y_{\nu_m}^m
\end{array}
$$

The One-Dimensional Case

Consider the one-dimensional system ($x \in \Re$):

$$
\left.
\begin{array}{lcl}
a_1 x & \leq & b_1 \\
a_2 x & \leq & b_2
\end{array}
\right\} \tag{C.1}
$$

We will determine the set of all $[\ b_1 \quad b_2\ ]^T$ for which this system is FEASIBLE. Note that this is the same as the set of all values of $[\ b_1 \quad b_2\ ]^T$ for which this system is IRREDUNDANT, as this is a one-dimensional system.

Step 3 of Section C for this system is as follows. The extreme point of the system

$$\begin{aligned} a_2 y &= a_1 \\ y &\leq 0 \end{aligned}$$

is $y = a_1/a_2$. The extreme point of the system

$$\begin{aligned} a_1 y &= a_2 \\ y &\leq 0 \end{aligned}$$

is $y = a_2/a_1$. Note that, if the system of Equation (C.1) is bounded, then $a_2/a_1 < 0$. So, it is not impossible to obtain $a_2/a_1 < 0$.

Step 4 of Section C for this system is as follows.

$$\begin{aligned} b_1 &\geq \frac{a_1}{a_2} b_2 \\ b_2 &\geq \frac{a_2}{a_1} b_1 \end{aligned}$$

Since both the inequalities are equal, the set of all values of the vector $[\ b_1 \quad b_2\ ]^T$ for which the system of Equation (C.1) is feasible is given by the following inequality:

$$-b_1 + \frac{a_1}{a_2} b_2 \leq 0$$

This also gives us the set of all values of $[\ b_1 \quad b_2\ ]^T$ for which this system is irredundant.

Question

If we were to attempt to apply Carl Lee's method for determining the set of all b for which the system of constraints $Ax \leq b$ is irredundant, as shown in Appendix B, to the one-dimensional system presented in Equation (C.1), we would arrive at the following inequalities:

$$b_1 < \max_{a_2 x \leq b_2} a_1 x$$

and

$$b_2 < \max_{a_1 x \leq b_1} a_2 x$$

Applying linear programming duality to the above inequalities, we get the following:

$$b_1 < \min_{\substack{a_2 y = a_1 \\ y \geq 0}} b_1 y$$

and

$$b_2 < \min_{\substack{a_1 y = a_2 \\ y \geq 0}} b_1 y$$

We note in these last pair of inequalities that the constraints $y = a_1/a_2$, $y \geq 0$ (similarly $y = a_2/a_1$, $y \geq 0$) are infeasible if the system of Equation (C.1) should be bounded, because for bounded systems $a_1/a_2 < 0$.

Does this mean that the methods presented in Appendix B do not work when $Ax \leq b$ is one-dimensional? This question needs to be studied.

Appendix D

MATLAB Program Listings for First Method for Irredundancy

```
--------------------------How to use the folder SCS--------------------------

File_Name : README.txt
Author : Ramprasad Potluri
e-mail : potluri@engr.uky.edu
Date : July-03-2002.

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

The folder SCS has the following m-files in it:

A.m
C.m
f_D_DB.m (originally f_make_D.m, but extended on July-29-2002)
f_mcn.m
f_refine.m
f_scs_i.m
f_subst.m
f_xplain.m
&
f_scs.m -- This file is a very slight modification of scs.m.
           It is simply scs.m made into a function.
           Created on August 06, 2002.

These files comprise the program to find the SSCSs, solve PR1 & PR2,
and refine this solution.

The user needs to run the files A.m and C.m first and f_scs(.) next.

The files named "f_..."  contain functions that will be called by
f_scs.m.

Each of these files has more information about how and what it does.
```

The file mcntemp.m is created by f_scs_i.m.
If you **delete** mcntemp.m, then please ignore the result of the first pass of f_scs_i.m; its second pass onwards, f_scs_i.m seems to work fine. However, **if** you have a file mcntemp.m in this folder before you start f_scs(.), then f_scs_i.m works fine.

The file Full_scs.txt shows example of a 2–D system every one of whose constraints was found to have SCSs. In my trial runs, this was the only instance where **all** the constraints in a system had SCSs.

The user can also compare the results of scs.m with that of carl_lee.m.

I wish you an enjoyable experience using this program.

Date : July–29–2002.

The m–file f_make_D.m was extended and renamed as f_D_DB.m.
The m–file f_D_DB.m solves PR1 besides performing the tasks that f_make_D.m performed.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_scs.m : Implementation of sections 3.3 and 3.4 of the paper
% titled "Determining the Values of the Right-Hand Vector of an
% Irredundant System of Linear Inequalities"
% by Ramprasd Potluri and L.E.Holloway.
%
% INPUT : Matrices A and C which define the system Ax <= Cy.
%
% OUTPUT : The matrices rev_DB and rev_DB which define the solution
%          spaces PR1 and PR2 as rev_DB*x < 0 and rev_D*x < 0.
%
% USES : f_mcn(.), f_scs_i(.), f_explain(.), f_D_DB(.) ( this one
%        is  an extension of f_make_D(.) ), f_refine(.) ( in turn
%        uses f_subst(.) ).
%
% PRECONDITION : x is at least 2-D.
%
% Created on : June 29, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
% This m-file was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [rev_D,rev_DB] = f_scs(A,C)


Atran = A';

% It is necessary to test for the boundedness of the system Ax<=
% b. For this, the function f_bound(A) can be run.
% Note: f_bound(.) uses cdd.exe.

%-------------------------------------------------------------------------

% List the mCn ("m choose k") different combinations of size n
% from the set {1, 2, ..., m} using the function f_mcn(m,n).

m = size(Atran,2);
n = size(Atran,1);
indices = f_mcn(m,n);

% TO DO: Write a more compact code for f_mcn(m,n).
```

```
%-------------------------------------------------------------------

% For each constraint, obtain a matrix of the indices of SSCS,
% and a matrix containing the product vector(s) inv(An)*a_i. The
% function [scs_inds,invAn_ai] = f_scs_i(i,indices,Atran) is used.

catalog = [];
SSCS_indices = [];
invAn_ai_all = [];
for i = 1:m,
 [scs_inds,invAn_ai] = f_scs_i(i,indices,Atran);
 if scs_inds ~= []
  catalog = [catalog; i  size(scs_inds,1)];
  SSCS_indices = [SSCS_indices; scs_inds];
  invAn_ai_all = [invAn_ai_all  invAn_ai];
 end
end

%-------------------------------------------------------------------

catalog;
% The matrix 'catalog' has two columns. In each row of catalog,
% the first column contains the index of a constraint that has
% SCSs, and the second column contains the number of SCSs of
% this constraint.

SSCS_indices;
% The matrix 'SSCS_indices' has n columns and number of rows equal
% to the sum of the elements in the second column of 'catalog'
% (that is, SSCS_indices has as many rows as there are SCSs
% overall).

invAn_ai_all;
% The matrix 'invAn_ai_all' has n rows and number of columns equal
% to the number of rows in SSCS_indices.

%-------------------------------------------------------------------

% Explain how the matrices catalog, SSCS_indices, and invAn_ai_all
% store the informaton about SSCSs.

% f_xplain(catalog,SSCS_indices,invAn_ai_all);

%-------------------------------------------------------------------

% Build the matrices D (which defines the solution space
% D*b < 0 of PR2) and DB (which defines the solution space
% DB*y < 0 of PR1) for the systems Ax <= b and Ax <= Cy
```

```
% respectively.

[D,DB] = f_D_DB(catalog,SSCS_indices,invAn_ai_all,m,C);

% CHECK: The value of D given by this routine can be compared with
% that given by Carl Lee's method. It was found that they are equal.
% As for the value of DB given by this routine, it should match the
% value of DB obtained as D*C (because, D*b < 0, and C*y = b,
% imply D*C*y < 0). It does.

%-------------------------------------------------------------------

% In preparation for the next part (removing those SCSs that cannot
% be RDCSs), merge the information in the matrices 'catalog' and
% 'SSCS_indices' into one matrix 'SSCS_inds', and add one "flag
% column" in the front of SSCS_inds and two "block start & end
% marking columns" at the end of SSCS_inds:

temp_col = [];
l = 1;

for j = 1:size(catalog,1),
 col_ones = ones(catalog(j,2),1);
 temp = [catalog(j,1)*col_ones, col_ones*[l, l+catalog(j,2)-1] ];
 temp_col = [temp_col; temp  ];
 l = l + catalog(j,2);
end

SSCS_inds = [ones(size(SSCS_indices,1),1) SSCS_indices temp_col];
% This first column will serve as a "flag column" in the function
% f_refine.

%-------------------------------------------------------------------

% Refine the solution by removing those SCSs that cannot be RDCSs.

[rev_SSCS,rev_D,rev_DB] = f_refine(SSCS_inds,D,DB);

% CHECK: rev_DB and rev_D*C are equal.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_D_DB.m : Implementation of the function
%            f_D_DB(catalog,SSCS_indices,invAn_ai_all,m,C).
%
%            For an explanation of how the first 3 arguments of this
%            function store the information, use the function
%            f_xplain(catalog,SSCS_indices,invAn_ai_all).
%
% INPUT : m, C, and matrices catalog, SSCS_indices, invAn_ai_all.
%         These values are calculated by scs.m
%
% OUTPUT : The matrices D and DB which respectively
%          define the solution spaces of PR2 and PR1.
%
% Created on : July 29, 2002 (Extension of f_make_D.m).
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [D,DB] = f_D_DB(catalog,SSCS_indices,invAn_ai_all,m,C)


temp = zeros(size(SSCS_indices,1),m);
DB = [];
k = 1;

for i = 1:size(catalog,1),
 for j = k : k + catalog(i,2) - 1,
  temp(j,catalog(i,1)) = 1;

  temp(j,SSCS_indices(j,:)) = - invAn_ai_all(:,j)';

  if size(C,1) > 0, % This if condition was added on Aug-14-2002.
    DB(j,:) = C(catalog(i,1),:) - invAn_ai_all(:,j)'* ...
              C(SSCS_indices(j,:),:);
  end

 end
 k = k + catalog(i,2);
end

D = temp;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_make_D.m : Implementation of the function
% f_make_D(catalog,SSCS_indices,invAn_ai_all.m).
%
% Function f_make_D(catalog,SSCS_indices,invAn_ai_all,m).
%
% For an explanation of how the first 3 arguments of this
% function store the information, use the function
% f_xplain(catalog,SSCS_indices,invAn_ai_all).
%
% Input : The m and matrices catalog, SSCS_indices, invAn_ai_all.
%         These values are calculated by scs.m
%
% Output : The matrix D which defines the solution space of PR2.
%
% Created on : June 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function D = f_make_D(catalog,SSCS_indices,invAn_ai_all,m)


temp = zeros(size(SSCS_indices,1),m);
k = 1;

for i = 1:size(catalog,1),
 for j = k : k + catalog(i,2) - 1,
  temp(j,catalog(i,1)) = 1;

  temp(j,SSCS_indices(j,:)) = - invAn_ai_all(:,j)';
 end
 k = k + catalog(i,2);
end

D = temp;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_mcn.m : Implementation of the function f_mcn(m,n) that lists
%           the mCn ("m choose k") different combinations of size
%           n from the set {1, 2, ..., m}.
%
% INPUT : m, n;
%
% OUTPUT : Matrix r of dimensions mCn x n. Each row of r shows one
%          way of choosing n elements out of the list {1,2,...,m}.
%
% PRECONDITION : m >= n.
%
% Created on : June 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
% TO DO : Write a more compact code for f_mcn(m,n) that does not
% print a file.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% D = [1 2 ... m];
%
% For a matrix such as D this function generates a file named
% mcntemp.m that contains commands of the type shown below.
%
% bb = [];
% x1 = 0;
% for x2 = x1 + 1 : m-n+1,
%  for x3 = x2 + 1 : m-n+2,
%   for x4 = x3 + 1 : m-n+3,
%     .............
%     .............
%      for x(n+1) = xn + 1 : m-n+n,
%       bb = [ bb; x2 x3 ... x(n+1) ];
%      end
%     .............
%     .............
%  end
% end
%
% Here, [x2 x3 ... x(n+1)] is the n-tuple that represents the
% different combinations of size n.
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function r = f_mcn(m,n)


space = [];
fid = fopen('mcntemp.m','wt');

fprintf(fid, '%s\n', '% mcntemp.m: file generated by f_mcn.m');
fprintf(fid, '%s\n', '                                      ');

fprintf(fid, '%s\n', 'aa=[];');
fprintf(fid, '%s\n', 'x1 = 0;');
for t = 2:n+1,
 temp1 = [space 'for x'...
    num2str(t) ' = x' num2str(t-1) ' + 1 : ' num2str(m-n+t-1) ','];
 space = [space ' '];
 fprintf(fid, '%s\n', temp1);
end

u = [];
for t = 2 : n+1,
 u = [ u 'x' num2str(t) ' ' ];
end

u = ['[ ' u ']'];
temp2 = [space 'bb =' u ';'];
fprintf(fid, '%s\n', temp2);

temp3 = [space  'aa = [aa; bb];'];
fprintf(fid, '%s\n', temp3);

for t = 1:n,
 temp4 = [space(:,1:size(space,2)-t) 'end'];
 fprintf(fid, '%s\n', temp4);
end

fclose(fid);

mcntemp;

r = aa;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_refine.m : Implementation of of the function
%              f_refine(SSCS_inds,D,DB).
%
% INPUT : The matrices 'D', 'DB' and 'SSCS_inds' generated by
%         f_D_DB(.).
%
% OUTPUT : Matrices rev_D, rev_DB, and rev_sscs --- revised
%          versions of the input matrices. Those SCSs that cannot
%          be RDCSs are removed from D, DB and SSCS_inds and the
%          result is rev_D, rev_DB, and rev_sscs.
%
% USES : f_subst(.)
%
% Created on : July 1, 2002.
% Extended on : July - 29 - 2002 to include DB.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless maybe if he wishes to use this m-file on non- Windows 98
% machines or with a later (and less bugged) version of Matlab.
% For example, in MATLAB v.4 student edition, negation operator
% '~' is not working OK. So, I wrote a longer piece of code (than
% if this operator were working OK) to work around this problem.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%-----------------------EXPLANATION FOR THIS PROGRAM-----------------------
%
% Here is an example matrix SSCS_inds.
%
% SSCS_inds =
%
%     1     2     3     1     1     4
%     1     2     5     1     1     4
%     1     3     4     1     1     4
%     1     4     5     1     1     4
%     1     1     5     3     5     7
%     1     2     5     3     5     7
%     1     4     5     3     5     7
%     1     1     2     4     8    10
%     1     2     3     4     8    10
%     1     2     5     4     8    10
%
% In this matrix, there are 3 "blocks" as shown below. Each block
```

```
% corresponds to the SSCS of one constraint. The index of a
% constraint to which a block corresponds is the element in the
% third-but-last column. The first column is a "flag" column.
% SSCS_inds is supplied to the function f_refine(.) with the
% flag column set to 1's. When the function determines that a
% particular SCS is not an RDCS, then, the flag before that SCS
% is set to 0. Each row of SSCS_inds corresponds to an SCS. The
% SCS-indices are situated in this matrix from the second to the
% fourth-but-last positions.
%
% Thus, in our example, Block 1 corresponds to Constraint # 1.
% The first row of SSCS_inds shows that [2,3] is the index set of
% one SCS of Constraint # 1. The last two columns in Block 1 show
% that the SSCS of Constraint # 1 occupies 1st through 4th rows
% of Block 1.
%
% The second row in Block 2 shows that [2,5] is the index
% set of one SCS of Constraint # 3, and that the SSCS of
% Constraint # 3 begins at row 5 and ends at row 7.
%
%-------------------------------------------------------
%     1     2     3     1     1     4
%     1     2     5     1     1     4
%     1     3     4     1     1     4       BLOCK 1
%     1     4     5     1     1     4
%-------------------------------------------------------
%     1     1     5     3     5     7
%     1     2     5     3     5     7       BLOCK 2
%     1     4     5     3     5     7
%-------------------------------------------------------
%     1     1     2     4     8    10
%     1     2     3     4     8    10       BLOCK 3
%     1     2     5     4     8    10
%-------------------------------------------------------
%
% The function f_refine(.) takes the SCS-index set of a constraint
% and tests if this SCS is an RDCS. If it is not, then a 0 is
% placed in the first cell of the corresponding row.
%
% For example, f_refine(.) takes up the index set [2,5] of
% Constraint # 1. With Row 2 of SSCS_inds as its reference row,
% f_refine(.) then begins with the first row of Block 2
% and scrolls down SSCS_inds testing if there is another index set
% [2,5] in any of the rows. It finds one in Block 2. It sees that
% [2,5] is also the SCS of Constraint # 3. f_refine(.) uses the
% function f_subst(.) thus: [a,b] = f_subst([2,5],3) to obtain
% a = [2,3] and b = [3,5]. If there is a 1 in the first cell of its
% reference row, f_refine(.) checks if a or b is present in the
% home block of the reference row, i.e., Block 1. Sure enough, a is
```

```
% in Block 1. This means (according to the theory in our paper)
% that [2,5] is not an SCS of Constraint # 1. So, f_refine(.) sets
% the first element of its reference row to 0. Next, f_refine(.)
% again uses f_subst(.) thus: f_subst([2,5],1) to obtain a = [] and
% b = [1,5]. f_refine(.) then checks if [1,5] is in Block 2. Sure
% enough it is. So, f_refine sets the first element of row 1 of
% Block 2 to 0.
%
% Then, f_refine(.) scrolls the blocks after Block 2 and finds [2,5]
% first in Block 3. f_refine(.) uses f_subst(.) thus:
% [a,b] = f_subst([2,5],1). It obtains a = [] and b = [1,5].
% f_refine(.) checks if a or b is present in Block 3. Neither is.
% So, the 1 in the first cell of the row in which [2,5] was in
% Block 3 remains. f_refine(.) also wants to use f_subst([2,5],4)
% and test if [2,5] from Block 1 can be marked with a 0 in the first
% cell. But, it does not test because, that cell already has a 0.
% Thus, f_refine has reached the end of the table in its search with
% ([2,5],1) from Block 1.
%
% Now, f_refine(.) repeats this operation with ([3,4],1)from Block 1.
% Now, Row 3 of SSCS_inds is the reference row.
%
% And so on.
%
% f_refine(.) does not search for the occurence of [3,4] in Block 1
% because, according to how we constructed SSCS_inds, not more than
% one instance of [3,4] can exist in any block.
%
% f_refine(.) searches in the top-to-bottom direction only and not
% backwards. Because of this rule, it picks up an SCS (to proceed
% to search from that position) only in the first to
% last-but-one blocks of SSCS_inds. Thus, when f_refine has finished
% the above procedure for the last element of the last-but-one block,
% then f_refine stops and returns control to the calling program.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [rev_sscs,rev_D,rev_DB] = f_refine(SSCS_inds,D,DB)


rows = size(SSCS_inds,1);
cols = size(SSCS_inds,2);

n = size(SSCS_inds(1,2:cols-3),2);
% n is the column size of each SCS.

k2 = SSCS_inds(rows,cols-1);
% k2 is the number of the row where begins the last block.
```

```
l = 1;
% l stores the pointer to the reference row.

while l < k2

 temp = SSCS_inds(l,2:n+1);
 % temp stores the SCS-index set from the reference row.

 k3 = SSCS_inds(l,cols) + 1;
 % k3 is used to point to the beginning of the next block.

 while k3 <= rows
 % This condition tests for end of SSCS_inds matrix.

   if norm( temp - SSCS_inds(k3,2:n+1) ) > 0
   % This condition is a substitute for the test
   % "if SSCS_inds(k3,2:n+1) ~= temp", as in my copy of Matlab,
   % the ~ operator is not working correctly.

      % Do nothing.

   else
   % Meaning: "if SSCS_inds(k3,2:n+1) == temp"

    if SSCS_inds(l,1) == 1
     [a,b] = f_subst( temp, SSCS_inds(k3,n+2) );
     for k4 = [ SSCS_inds(l,cols-1) : l-1 , l+1 : SSCS_inds(l,cols) ],
      if (SSCS_inds(k4,2:n+1) == a)|(SSCS_inds(k4,2:n+1) == b)
       SSCS_inds(l,1) = 0;
       k4 = SSCS_inds(l,cols)+1; % Provides exit from the FOR loop.
      end % for the IF loop
     end % for the FOR loop
    end % for the IF loop

    if SSCS_inds(k3,1) == 1
     [a,b] = f_subst( SSCS_inds(k3,2:n+1), SSCS_inds(l,n+2) );
     for k5 = [ SSCS_inds(k3,cols-1) : k3-1 , k3+1 :SSCS_inds(k3,cols) ],
      if (SSCS_inds(k5,2:n+1) == a)|(SSCS_inds(k5,2:n+1) == b)
       SSCS_inds(k3,1) = 0;
       k5 = SSCS_inds(k3,cols)+1; % Provides exit from the FOR loop.
      end % for the IF loop
     end % for the FOR loop
    end % for the IF loop

   end % for the big IF - ELSE loop

   k3 = k3 + 1;

 end % for the WHILE k3 < rows loop
```

```
 l = l + 1;

end % end of the WHILE l < k2 loop

% Now, with the information we have in the first column of the
% matrix SSCS_inds, we will revise the matrix D. This amounts to
% removing those rows of D which correspond to rows of SSCS_inds
% with 0's for their first elements.

for i = 1:rows,
 if SSCS_inds(i,1) == 0
  D(i,:) = [];
  if size(DB,1)>0 % This if condition was added on Aug-14-2002.
    DB(i,:) = []; % This line was added on July - 29 - 2002.
  end
 end
end

rev_sscs = SSCS_inds(:,1:n+2);
rev_D = D;
rev_DB = DB;  % This line was added on July - 29 - 2002.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_scs_i.m : Implementation of the function
%             f_scs_i(i,indices,Atran).
%
% INPUT : The index i, the mCn x n matrix of indices that lists all
%         the n-tuples of different indices from the index set of
%         the matrix Atran, and the matrix Atran;
%
% OUTPUT : For each constraint, a matrix of the indices of SSCS and,
%          a matrix containing the product vector(s) inv(An)*a_i
%          are output.
%
% Created on : June 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non- Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [scs_ind,invAn_ai] = f_scs_i(i,indices,Atran)


lambda = [];
temp = [];

for j = 1 : size(indices,1),
 tf = (i == indices(j,:));

 if tf == zeros(1,size(indices,2))

 % If this is true, it means that i is not in indices(j,:). So,
 % Atran(:,indices(j,:)) is a potential SCS of Atran(:,i).

  x = Atran(:,indices(j,:))\Atran(:,i);

 % Here, we have used the equation x = A\b instead of x = inv(A)*b.
 % Both ways solve the system of linear equations Ax = b.
 % But, the first way is about 2-3 times faster and more accurate.
 % [Page 468. The student edition of Matlab: version 4 : User's
 % guide/ the Mathworks Inc. 1995.]

  if x >= 0
   temp = [temp; indices(j,:)];
   lambda = [lambda  x];
```

```
    end

  end

end

scs_ind = temp;
invAn_ai = lambda;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_subst.m : Implementation of of the function f_subst(SCS_i,C_i).
%
% INPUT : The row vector SCS_i of indices of constraint C_i,
%         and the index i of the constraint C_i.
%
% OUTPUT : Row vectors SCS_i_max, and SCS_i_min.
%
% PRECONDITION : 'i' cannot be an element of SCS_i.
%
% Created on : July 1, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% If SCS_i has two successive elements between which i lies,
% then SCS_i_max is obtained by substituting i for the greater
% of these two elements and SCS_i_min is obtained by substituting
% i for the lesser of these two elements.
% Else, if i < min(SCS_i) (meaning, i < first element of SCS_i),
% then SCS_i_min is formed and SCS_i_max is empty, and
% if i > max(SCS_i) (meaning, if i > last element of SCS_i),
% then SCS_i_max is formed and SCS_i_min is empty.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [SCS_i_max,SCS_i_min] = f_subst(SCS_i,i)

n = size(SCS_i,2);

tf = SCS_i==i;

if tf == zeros(1,n)
 if (i < SCS_i(:,1))
  SCS_i(:,1) = i;
  temp_min = SCS_i;
  temp_max = [];
 elseif (i > max(SCS_i))
  SCS_i(:,n) = i;
  temp_min = [];
  temp_max = SCS_i;
```

```
 else
  k = 1;
  while i > SCS_i(:,k)
  k = k + 1;
  end
  temp_min = [ SCS_i(:,1:k-2)  i  SCS_i(:,k:n) ];
  temp_max = [ SCS_i(:,1:k-1)  i  SCS_i(:,k+1:n) ];
 end
else
 disp('  ');
 disp('WARNING : i cannot be an element of SCS_i');
end

SCS_i_min = temp_min;
SCS_i_max = temp_max;

    %-------------------------------------------------------------------
    %
    % NOTE: In the above code, we could have used the test
    %
    %                 if tf~=zeros(1,n)
    %
    % But, MATLAB did not respond to such a test. Maybe, this is a
    %
    % bug in the version (MATLAB 4) that I have.
    %
    %-------------------------------------------------------------------
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_xplain.m : Implementation of the function
%              f_xplain(catalog,SSCS_indices,invAn_ai_all).
%
% INPUT : The matrices catalog, SSCS_indices, invAn_ai_all.
%
% OUTPUT : The information in these matrices is interpreted and
%          output.
%
% Created on : June 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ ] = f_xplain(catalog,SSCS_indices,invAn_ai_all)


disp('    ');
disp('    ');
disp('Displaying the indices of the SCSs (row vectors) and');
disp('the products inv(An)*ai (column vectors) for each constraint.');
disp('    ');

l = 1;
for i = 1:size(catalog,1),
 temp1 = ['The indices of the SSCS of constraint # '...
          num2str(catalog(i,1)) ' are:'];
 disp(temp1);
 disp('    ');
 disp(SSCS_indices(l:catalog(i,2)+l-1,:))
 disp('    ');

 temp2 = ['The products for constraint # '...
          num2str(catalog(i,1)) ' are:'];
 disp(temp2);
 disp('    ');
 disp(invAn_ai_all(:,l:catalog(i,2)+l-1))
 disp('    ');

 l = catalog(i,2)+l;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_mcn.m : Implementation of the function f_mcn(m,n) that lists
%           the mCn ("m choose k") different combinations of size
%           n from the set {1, 2, ..., m}.
%
% INPUT : m, n;
%
% OUTPUT : Matrix r of dimensions mCn x n. Each row of r shows one
%          way of choosing n elements out of the list {1,2,...,m}.
%
% PRECONDITION : m >= n.
%
% Created on : June 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
% TO DO : Write a more compact code for f_mcn(m,n) that does not
% print a file.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% D = [1 2 ... m];
%
% For a matrix such as D this function generates a file named
% mcntemp.m that contains commands of the type shown below.
%
% bb = [];
% x1 = 0;
% for x2 = x1 + 1 : m-n+1,
%  for x3 = x2 + 1 : m-n+2,
%   for x4 = x3 + 1 : m-n+3,
%     .............
%     .............
%      for x(n+1) = xn + 1 : m-n+n,
%       bb = [ bb; x2 x3 ... x(n+1) ];
%      end
%     .............
%     .............
%  end
% end
%
% Here, [x2 x3 ... x(n+1)] is the n-tuple that represents the
% different combinations of size n.
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function r = f_mcn(m,n)


space = [];
fid = fopen('mcntemp.m','wt');

fprintf(fid, '%s\n', '% mcntemp.m: file generated by f_mcn.m');
fprintf(fid, '%s\n', '                                      ');

fprintf(fid, '%s\n', 'aa=[];');
fprintf(fid, '%s\n', 'x1 = 0;');
for t = 2:n+1,
 temp1 = [space 'for x'...
    num2str(t) ' = x' num2str(t-1) ' + 1 : ' num2str(m-n+t-1) ','];
 space = [space ' '];
 fprintf(fid, '%s\n', temp1);
end

u = [];
for t = 2 : n+1,
 u = [ u 'x' num2str(t) ' ' ];
end

u = ['[ ' u ']'];
temp2 = [space 'bb =' u ';'];
fprintf(fid, '%s\n', temp2);

temp3 = [space  'aa = [aa; bb];'];
fprintf(fid, '%s\n', temp3);

for t = 1:n,
 temp4 = [space(:,1:size(space,2)-t) 'end'];
 fprintf(fid, '%s\n', temp4);
end

fclose(fid);

mcntemp;

r = aa;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_mcn.m : Implementation of the function f_mcn(m,n) that lists
%           the mCn ("m choose k") different combinations of size
%           n from the set {1, 2, ..., m}.
%
% INPUT : m, n;
%
% OUTPUT : Matrix r of dimensions mCn x n. Each row of r shows one
%          way of choosing n elements out of the list {1,2,...,m}.
%
% PRECONDITION : m >= n.
%
% Created on : June 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
% TO DO : Write a more compact code for f_mcn(m,n) that does not
% print a file.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% D = [1 2 ... m];
%
% For a matrix such as D this function generates a file named
% mcntemp.m that contains commands of the type shown below.
%
% bb = [];
% x1 = 0;
% for x2 = x1 + 1 : m-n+1,
%  for x3 = x2 + 1 : m-n+2,
%   for x4 = x3 + 1 : m-n+3,
%     .............
%     .............
%      for x(n+1) = xn + 1 : m-n+n,
%       bb = [ bb; x2 x3 ... x(n+1) ];
%      end
%     .............
%     .............
%  end
% end
%
% Here, [x2 x3 ... x(n+1)] is the n-tuple that represents the
% different combinations of size n.
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function r = f_mcn(m,n)


space = [];
fid = fopen('mcntemp.m','wt');

fprintf(fid, '%s\n', '% mcntemp.m: file generated by f_mcn.m');
fprintf(fid, '%s\n', '                                      ');

fprintf(fid, '%s\n', 'aa=[];');
fprintf(fid, '%s\n', 'x1 = 0;');
for t = 2:n+1,
 temp1 = [space 'for x'...
    num2str(t) ' = x' num2str(t-1) ' + 1 : ' num2str(m-n+t-1) ','];
 space = [space ' '];
 fprintf(fid, '%s\n', temp1);
end

u = [];
for t = 2 : n+1,
 u = [ u 'x' num2str(t) ' ' ];
end

u = ['[ ' u ']'];
temp2 = [space 'bb =' u ';'];
fprintf(fid, '%s\n', temp2);

temp3 = [space  'aa = [aa; bb];'];
fprintf(fid, '%s\n', temp3);

for t = 1:n,
 temp4 = [space(:,1:size(space,2)-t) 'end'];
 fprintf(fid, '%s\n', temp4);
end

fclose(fid);

mcntemp;

r = aa;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Full_scs.txt : Contains an example result of running f_scs(.)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
EDU>> A

A =

    1.5578    0.4142
   -2.4443   -0.9778
   -1.0982   -1.0215
    1.1226    0.3177
    0.5817    1.5161
   -0.2714    0.7494

EDU>> scs

catalog =

     1     2
     2     2
     3     2
     4     2
     5     2
     6     2


D =

    1.0000   -0.0931         0   -1.5903         0         0
    1.0000         0   -0.0375   -1.4243         0         0
         0    1.0000   -2.9296         0   -1.3288         0
         0    1.0000   -1.9062         0         0   -1.2933
   -2.7855   -2.2245    1.0000         0         0         0
         0   -2.4839    1.0000   -4.4298         0         0
   -0.7154         0         0    1.0000   -0.0141         0
   -0.7247         0         0    1.0000         0   -0.0234
   -0.6620         0         0         0    1.0000   -1.6571
         0         0         0   -0.9135    1.0000   -1.6358
         0   -0.2701         0         0   -0.6685    1.0000
         0         0   -0.7913         0   -1.0274    1.0000


elapsed_time =

    0.3300
```

```
rev_SSCS =

     0     2     4     1
     1     3     4     1
     0     3     5     2
     1     3     6     2
     1     1     2     3
     1     2     4     3
     1     1     5     4
     0     1     6     4
     0     1     6     5
     1     4     6     5
     1     2     5     6
     0     3     5     6


rev_D =

    1.0000         0   -0.0375   -1.4243         0         0
         0    1.0000   -2.9296         0   -1.3288         0
   -2.7855   -2.2245    1.0000         0         0         0
         0   -2.4839    1.0000   -4.4298         0         0
   -0.7154         0         0    1.0000   -0.0141         0
   -0.7247         0         0    1.0000         0   -0.0234
   -0.6620         0         0         0    1.0000   -1.6571
         0   -0.2701         0         0   -0.6685    1.0000


elapsed_time =

    0.0500
```

Here is the D matrix generated using f_carlee.m:

```
D =

    1.0000         0   -0.0373   -1.4239         0         0
    1.0000   -0.0928         0   -1.5892         0         0
         0    1.0000   -1.9059         0         0   -1.2944
         0    1.0000   -2.9309         0   -1.3309         0
         0   -2.4840    1.0000   -4.4282         0         0
   -2.7864   -2.2255    1.0000         0         0         0
   -0.7155         0         0    1.0000   -0.0141         0
```

−0.7249	0	0	1.0000	0	−0.0233
0	0	0	−0.9133	1.0000	−1.6358
−0.6620	0	0	0	1.0000	−1.6570
0	0	−0.7919	0	−1.0282	1.0000
0	−0.2702	0	0	−0.6686	1.0000

Appendix E

MATLAB Program Listings for Second Method Method for Irredundancy

```
----------How to use the folder carl_lee----------

File_Name : README02.txt
Author : Ramprasad Potluri
e-mail : potluri@engr.uky.edu
Date : August-06-2002.

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

This README02.txt file is a modification of README01.txt.
It summarizes the modifications done to the files in this
folder to make them into functions.

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

Please read README01.TXT that is in the subfolder oldfiles.

The files listed in README01.TXT were modified slightly and
made into functions.

Thus, we now have the following functions:

f_carlee.m --> f_bound(.) --> f_makine
                          --> cdd/cddf+
           --> f_step3(.) --> f_makine(.)
                          --> cdd / cddf+
           --> f_step56(.)
(optional) --> f_pbcone(.) --> f_makine(.)
                           --> cdd / cddf+

f_pbcone.m --> f_makine(.)
           --> cdd / cddf+

f_step3.m --> f_makine
```

```
          --> cdd/cddf+

f_step56.m

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

Also present in this folder are the following files:

cddf+.exe       (this is invoked by the m-files to enumerate
                 extreme points and rays)

cygwin1.dll     (this is necessary for cddf+.exe is work)

carl_lee.pdf    (contains the algorithm; same as Appendix B
                 of the dissertation)

README01.txt      (this file)

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

To use this program start with f_carlee(.).
This is the function that needs to be invoked to execute
Carl Lee's algorithm. It calls f_step3(.), f_step56(.),
and f_pbcone(.) in that order.

f_pbcone(.) is not essential to solving PR2.
It was only created for using Minkowski's Theorem
to solve the problem of Irredundancy.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_carlee.m : An implementation of the algorithm shown
%              by Dr. Carl Lee to determine the set of all values of
%              b for which Ax <= b is irredundant.
%
%              A description of the algorithm can be found in
%              carl_lee.pdf.
%
% INPUT : The matrix A.
%
% OUTPUT : The matrix D. Also generates the files pbcone.ine and
%          pbcone.ext that contain the description of the
%          pointed-b-cone.
%
% USES : cdd/cdd+, f_bound(.) (which in turn uses cdd/cdd+),
%        f_step3(.) (which in turn uses cdd/cdd+ and f_makine(.) ),
%        f_step56(.), f_pbcone(.) (which in turn uses cdd/cdd+
%        and f_makine(.) ).
%
% Created on : April 27, 2002.
% Modified on : August 06, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non- Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [] = f_carlee(A)

disp('  ');
disp('WARNING:');
disp('  ');
disp('As part of its operation, this program launches some MS-DOS');
disp('windows. The first time an MS-DOS window opens and does not');
disp('automatically close after the program that launched it');
disp('has completed execution, the user can click on an icon that');
disp('is at the top of that DOS window. This will open a');
disp('"properties" pop-up window. Checking the "close on exit"');
disp('box will close all future MS-DOS windows after the programs');
disp('that launch them complete execution.');
disp('    ');
disp('Now, please press any character key to continue.');
disp('   ');
```

```
pause

flag = f_bound(A);
if flag == 0
 disp('The system Ax <= b is unbounded');
 disp(' ');
end

Atran = A';
f_step3(Atran);
% Executes Step 3 of carl_lee.pdf: Forms the polyhedron and
% enumerates its extreme points.

N = size(Atran,2);
D = f_step56(N);
% Executes steps 5 and 6 of carl_lee.pdf to form the b-cone.

f_pbcone(A,D);
% Forms the pointed-b-cone corresponding to the b-cone obtained
% above and uses cddf+.exe to obtain the V-format of the
% pointed-b-cone.

disp('    ');
disp('    ');
disp('The "b-cone" is D*b < 0, where D is as follows:');

D

disp('   ');
disp('The reader can find the H-format of the closure of the');
disp('pointed-b-cone in the file pbcone.ine.');
disp('    ');
disp('The reader can find the V-format of the closure of the');
disp('pointed-b-cone in the file pbcone.ext.');
disp(' ');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_step3.m: m-file that implements Step3 of carl_lee.pdf.
%
% INPUT : The matrix Atran which is the transpose of A.
%
% OUTPUT : None. Generates the files constrI.ext (I = 1,2,...)
%          which contain the extreme points as described in Step3
%          of carl_lee.pdf.
%
% USES : cdd (cdd+), f_makine(.).
%
% Created on : April 27, 2002.
% Modified on : August 06, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Let C be an m x k matrix, and let d be a column m-vector.
% The Polyhedra format (H-format) of the system Cx <= d
% of m inequalities in k variables x = (x1, x2, ...., xd)^T is
%-----------------------------------
% various comments
% H-representation
% begin
% m   k+1    numbertype
% d      -C
% end
% various options
%-----------------------------------
%
% The above is the format of the data that is presented
% in *.ine file ("ine" stands for "inequality").
%
% This m-file generates *.ine files.
%
% This m-file was written in Matlab 4. Maybe the later versions of
% Matlab have more features that allow for simplification of this
% code.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Example:    Assume that we are given the following A matrix.
%
% A = [a11, a12;
%      a21, a22;
%      a31, a32;
%      a41, a42;
```

```
%      a51, a52]
%
% Convert to the following format:
%
% [a21  a31  a41  a51] y  =  [a11]
% [a22  a32  a42  a52]       [a12]
%                      y  >=  0
%
% This is the same as
%
% [ a21   a31   a41   a51]        [ a11]
% [ a22   a32   a42   a52]        [ a12]
% [-a21  -a31  -a41  -a51] y  <=  [-a11]
% [-a22  -a32  -a42  -a52]        [-a12]
%
%                         -y  <=   0
%
% Now this information needs to be placed in a *.ine file. This job
% is performed by this m-file.
%
% From the A matrix in this example, 5 such systems, and hence 5
% such *.ine files, can be formed.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point, unless
% he wishes to use this m-file on non-Windows 98 machines.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [] = f_step3(Atran)


dd = [];
CC = [];
N = size(Atran,2);

options = [];
c = [];
% The function f_makine(.) will be used. 'options' and 'c' are two
% of its inputs.

x1 = 'constr';
x3 = '.ine';
% The *.ine files will be named constr1.ine, ..., constrN.ine.

for I = 1:N,
% Begin forming contsrI.ine file.
% The file constrI.ine contains the H-format description of
```

```
% the polyhedron shown in Step 3 of carl_lee.pdf.

 dd = [ Atran(:,I);
        -Atran(:,I);
        -zeros(N-1,1) ];
 CC = [ Atran(:,1:I-1), Atran(:,I+1:N);
        -Atran(:,1:I-1), -Atran(:,I+1:N);
        -eye(N-1)                        ] ;

 x2 = num2str(I);
 filename = [x1 x2 x3];
 % In these 2 lines, the name of the file has been created.

 filename = deblank(filename);
 % Remove any blanks spaces from the file's name. Actually,
 % we get the desired filename even without this step.

 % Place this information in a constrI.ine file:

 f_makine(filename,options,c,CC,dd);

end

% Thus far, we have obtained the *.ine files. Next, we will run
% cddf+.exe on these *.ine files to generate the extreme points of
% each of the polyhedra therein.

fid = fopen('process.bat','wt');
% A file process.bat is opened for writing.
% The reason for creating process.bat is because we want the end
% result of the following FOR-loop and the command
% '!cddf+.exe filename' will not give that same result.

for I = 1:N,
 x2 = num2str(I);
 x3 = '.ine';
 filename = [x1 x2 x3];
 fprintf(fid, '%s\n', ['cddf+.exe' ' ' filename]);
end
% The commands "cddf+.exe constr1.ine",..., "cddf+.exe constrN.ine"
% have been placed in process.bat.

fclose(fid);
% The file process.bat is closed.

!process.bat

% End of f_step3(.)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_step56.m: An m-file to generate the description of the
%              "b-cone". This m-file implements steps 5 and 6 shown
%              in carl_lee.pdf.
%
% INPUT : N - the number of the *.ext files named constr1.ext, ...,
%          constrN.ext that were generated by f_step3(.).
%
%          f_step56.m reads the information from those .ext files.
%
% OUTPUT : The matrix D such that D*b < 0 is the solution set of PR2
%           as shown in Step 6 of carl_lee.pdf.
%
% Created on : April 27, 2002.
% Modified on : August 06, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The following note is extracted from cdd/cdd+ Reference Manual.
%
% Let P be represented by n extreme points and s rays
% as P = conv(v1, ..., vn) + nonneg(r1, ..., rs).
% Then the Polyhedra V-format for P (as presented in *.ext files)
% is defined as
%-------------------------------------
% various comments
% V-representation
% begin
% n+s    d+1     numbertype
% 1       v1
% .       .
% .       .
% .       .
% 1       vn
% 0       r1
% .       .
% .       .
% .       .
% 0       rs
% end
% various options
%-------------------------------------
% Here d is the dimension of the full-dimensional polyhedron.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% The above is the format of the data present in *.ext file.
%
% The function f_step56(.) implemented here reads those portions
% of the *.ext file that are situated beginning two lines after the
% line on which the 'begin' statement is and have a 1 as their
% first element.
%
% This m-file was written in Matlab 4. Maybe the later versions of
% Matlab have more features that will allow for simplification of
% this code.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point, unless
% he wishes to use this m-file on non-Windows 98 machines. This
% m-file creates and then runs an MSDOS batch file.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% In the file FGETL.M, the original value of BLOCKSIZE as set by
% the distribution of Matlab was 128. This was preventing the lines,
% that were being read in from files by the FGETL command,
% to be displayed correctly. The first several characters
% of each line would go missing. So, I decided to see if there
% was some parameter in the FGETL.M file that I could play with
% to get it to work correctly. Interestingly, I hit upon BLOCKSIZE
% right away. Increasing the value of BLOCKSIZE
% worsens the problem. Decreasing the value helps.
% - RAMPRASAD POTLURI. potluri@engr.uky.edu. April 29, 2002.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function D = f_step56(N)


x1 = 'constr';
x3 = '.ext';

D = [];
% D is the matrix that represents the b-cone thus: D*b < 0.

for I = 1:N,

 x2 = num2str(I);
 filename = [x1 x2 x3];
 % In these 2 commands, the name of the file has been created.

 fid=fopen(filename, 'rt');
 % open the file constrI.ext for reading.
```

```
line = fgetl(fid);
% read in the first line of the file.

K1 = findstr(line,'begin');
% find the string 'begin' in the line read above.

while K1 == [],
% while 'begin' is absent in the line, do this loop.
% This loop executes until 'begin' is found in the file.

  line = fgetl(fid);
  % read the next line.

  K1 = findstr(line,'begin');
  % check for the presence of 'begin' in this line.

end

% Now, we have completed reading the line that contains 'begin'.
% I thought that now, if we execute fscanf(), we will read this
% same line. But, fscanf() reads the next line which contains
% 'number number numbertype'.
% So, it means that after fgetl() finishes reading a line,
% it positions the counter at the beginning of the next line.

K2 = fscanf(fid,'%i',1);
% Read the first number which is in the line that comes after the
% line that contains 'begin'. We are actually interested in the
% next number. So, this is just a step to reach it.

K2 = fscanf(fid,'%i',1);
% Read the second number which is in the line that comes after the
% line that contains 'begin'. This second number indicates the
% number of columns that the matrix K5 will have.

line = fgetl(fid);
% Position the counter at the beginning of the next line.

K3 = fscanf(fid,'%i',1);
% Read the first element of the matrix of 1's, v's, 0's, and r's.

% The following while-loop checks if K3 = 0. If it is, then this
% loop skips to the next line of the matrix of 1's, v's, 0's, r's
% and again checks if the first element is 0. If the first element
% is 0 in all the rows, then the matrix K5 will not be formed.
% Actually, if the first element of a row is 0, then every first
% element of the subsequent rows in *.ext matrices will be 0, and
% we don't have to perform the following while loop.
```

```
while K3 ~= 1,
 line = fgetl(fid);
 % Finish reading the rest of the line and position the counter at
 % the beginning of the next line.

 K3 = fscanf(fid,'%i',1);
 % Read the first element of this line of matrix of 1's, v's, 0's,
 % and r's.

end

i = 0;
K5 = [];
% K5 is an intermidiate matrix in the formation of D. K5 is such
% that K5*b < 0 is the system shown in Step 5 of carl_lee.pdf.
% D is such that D*b < 0 is the system shown in Step 6 of
% carl_lee.pdf.

while K3 == 1,
   i = i+1;
   K4 = fscanf(fid,'%g',[1,K2-1]);
   % The K2-1 elements that follow the first element (which is a 1)
   % of a row are read into K4.

   K5(i,1:I-1) = -K4(1,1:I-1);
   K5(i,I) = 1;
   K5(i,I+1:K2) = -K4(1,I:K2-1);
   % Through the above 3 commands, the elements of -K4 are
   % successively placed in all but the I-th position of the i-th
   % row of K5.

   line = fgetl(fid);
   % Move to the next line of the *.ext file.

   K3 =  fscanf(fid,'%i',1);
end

fclose(fid);
% close the file

D = [D;K5];
% The matrix D is formed by placing the K5 matrices one below the
% other.

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_pbcone.m: This m-file forms the file pbcone.ine. The file
%             pbcone.ine contains the H-format of the
%             "pointed-b-cone". This m-file also runs cddf+.exe on
%             pbcone.ine to generate the V-format of the
%             pointed-b-cone.
%
%             "pointed-b-cone" is a term coined by Dr. Carl Lee. The
%             pointed-b-cone is {b | D*b < 0, A'*b = 0}.
%
% INPUT : The matrices D (D*b < 0 is the set of all b for which Ax
%         <= b is irredundant) and A.
%
% OUTPUT : None. The V-format of the pointed-b-cone can be found in
%          the file pbcone.ext.
%
% USES : cdd / cddf+
%
% Created on : April 27, 2002.
% Modified on : August 06, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [] = f_pbcone(A,D)


Atran = A';

C1 = [ D; Atran; -Atran  ];

% The matrix D (that describes the b-cone) was formed in the file
% f_step56.m.

d1 = [zeros(size(C1,1),1)];
options = [];
c = [];

% Place this information in the file pbcone.ine.

f_makine('pbcone.ine',options,c,C1,d1);

% Launching cddf+.exe to work on pbcone.ine:

!cddf+.exe pbcone.ine

% End of function f_pbcone(.).
```

Appendix F

MATLAB Program Listings for the Q-Matrix Method of Maintainability of Halfspace Systems

```
————————Description of the contents of the folder TstMaint————————

Author : Ramprasad Poturi
E–mail : potluri@engr.uky.edu
Date : August – 10 – 2002.

_____________________________________________________________________

The folder TstMaint containts the following m–files
("–––>" reads "uses"):

f_active.m––> f_makine
          ––> cdd/cddf+

f_bound.m––> f_makine
         ––> cdd/cddf+

f_cddX.m––> f_inter ––> f_makine
                    ––> cddf+

f_EFB.m––> f_bound ––> f_makine
                   ––> cdd/cddf+

f_findQ.m––> f_pickX ––> f_square ––> f_mcn
                     ––> f_cddX ––> f_inter ––> f_makine
                                            ––> cddf+
         ––> f_active ––> f_makine
                      ––> cdd/cddf+

f_inter ––> f_makine
        ––> cddf+
```

```
f_makine.m

f_max.m --> f_makine
        --> cdd/cddf+

f_pickX.m --> f_square --> f_mcn
          --> f_cddX --> f_cddX --> f_makine
                                --> cddf+

f_square.m --> f_mcn

f_TinPR1.m --> f_max --> f_makine
                     --> cdd/cddf+

f_tstmnt.m --> f_inter --> f_makine
                       --> cdd/cddf+

f_tube.m --> f_bound --> f_makine
                     --> cdd/cddf+
         --> f_scs --> f_mcn
                   --> f_scs_i
                   --> f_explain (optional)
                   --> f_D_DB
                   --> f_refine --> f_subst

f_verts.m --> f_makine
          --> cdd/cddf+

pre_pros.m --> f_EFB --> f_bound --> f_makine
                                 --> cdd/cddf+
           --> f_tube --> f_bound --> f_makine
                                  --> cdd/cddf+
                      --> f_scs --> f_mcn
                                --> f_scs_i
                                --> f_explain (optional)
                                --> f_D_DB
                                --> f_refine --> f_subst
           --> f_scs --> f_mcn
                     --> f_scs_i
                     --> f_explain (optional)
                     --> f_D_DB
                     --> f_refine --> f_subst
           --> f_TinPR1 --> f_max --> f_makine
                                  --> cdd/cddf+
           --> f_findQ --> f_pickX --> f_square --> f_mcn
                                   --> f_cddX --> f_inter --> f_makine
                                                          --> cddf+
                       --> f_active --> f_makine
                                    --> cdd/cddf+
```

```
        --> f_verts --> f_makine
                    --> cdd/cddf+
        --> f_tstmnt --> f_inter --> f_makine
                                 --> cddf+
```

OTHER FILES:

The program generates the following temporary intermediate files:

MCNTEMP.m

MYTEMP.m

If they are absent, the program may complain the first time it tries to use either of them and does not **find** it. However, this **error** may be ignored and the program may be run a second time. The second pass onwards, the program will work fine.

Optional Components.

Added on August 15, 2002.

Besides the above files, The following files are also present in this folder:

f_poly2D.m--> f_trunc

f_trunc.m

```
ptope.m--> f_tube --> f_bound --> f_makine
                              --> cdd/cddf+
                  --> f_scs --> f_mcn
                            --> f_scs_i
                            --> f_explain (optional)
                            --> f_D_DB
                            --> f_refine --> f_subst
        --> f_verts.m--> f_makine
                     --> cdd/cddf+
        --> f_poly2D --> f_trunc
```

These functions **help** visualize the tube. Given a randomly ordered list of the tube's vertices, f_poly2D(.) first reorders this list so that

adjacent elements in the list represent adjacent vertices, and then plots the polytope.

ptope.m is an m-file to generate a random bounded and irredundant tube (using f_tube(.)), obtain this tube's vertices (using f_verts(.)), and **plot** this tube using f_poly2D(.).

Z.m – This was written to test some parts of the program. This is not a part of the program itself.

Besides the above files, the program uses the **function** f_scs(.) **which** is in the folder SCS.

HOW THIS PROGRAM WORKS

pre_pros.m is the file to run.

pre_pros.m runs in succession the functions f_EFB(.) and f_tub(.) (to generate the specifications of a bounded EFB system and a bounded irredundant Tube), f_scs(.) (to determine the **set** of (x0,u0) pairs **for which** the Reach(x0,u0) sets **for** this EFB system are irredundant), f_TinPR1(.) (to see **if** Tube is a subset of PR1), f_findQ(.) (to **find** a suitable Q matrix), f_verts(.) (to determine the vertices of the tube), and f_tstmnt(.) (to test **if** the tube is maintainable).

This program uses cddf+. cdd does not have **all** the options (e.g. find_interior) that we need to run this program successfully (please see the next section **for** a list of the options used in this program). So, the user needs to download cddf+ (or some other variant of cdd+) to use this program.

IMPORTANT: It seems like cddf+.exe needs to be in the same folder from where it is being invoked. Else, you may see errors.

THE OPTIONS USED IN OUR PROGRAM FOR CDD / CDDF+ AND THEIR AVAILABILITY FOR EACH OF CDD & CDDF+:

OPTION	AVAILABLE FOR	
	cdd	cddf
maximize	yes	yes
dynout_off	yes	yes
stdout_off	yes	yes
logfile_off	yes	yes
find_interior	NO	yes
incidence	yes	yes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_active.m: An m-file to determine the indices of those
%             constraints in the bounded system Ax <= b that are
%             active at each of this system's extreme points.
%
% INPUT : A, b.
%
% OUTPUT : The matrices 'verts' and 'act_constrs'. 'verts' contains
%          the indices of the vertices of Ax <= b. 'act_constrs'
%          contains the indices of 'n' constraints that are active
%          at each of the vertices referenced by 'verts'. Here n is
%          the dimension of x.
%
%          In case a vertex of Ax <= b has more than n active
%          constraints, f_active shows just n of these constraints
%          that are linearly independent.
%
% USES : f_makine(.) which in turn uses cdd/cddf+.
%
% Created on : August 08, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% We use cddf+.exe with the 'incidence' option specified.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [verts,act_constrs] = f_active(A,b)


options = ['incidence  '
           'dynout_off '
           'stdout_off '
           'logfile_off' ];
c = [];
name = 'Reach.ine';

% Place this information in the file Reach.ine:

f_makine(name,options,c,A,b);

% Launching cddf+.exe to work on Reach.ine:

!cddf+.exe Reach.ine

    %-------------------------------------------------------------------
```

```
    %
    % Two files --- Reach.ext and Reach.ecd --- have been generated
    % by cdd. Next, we will read the coordinates of the vertices
    % from the file Reach.ext into the matrix 'verts', and the
    % indices of the active constraints from the file Reach.ecd
    % into the matrix 'act_constrs'.
    %
    %-------------------------------------------------------------------

%-------------------------------First, the file Reach.ext-----------------

fid=fopen('Reach.ext', 'rt');
% open the file Reach.ext for reading.

line = fgetl(fid);
% read in the first line of the file.

K0 = findstr(line, 'Number of Vertices');
% find the string 'Number of Vertices' in the line read above.

while K0 == [],
 line = fgetl(fid);
 K0 = findstr(line, 'Number of Vertices');
end

% Now that we have found the line that contains the number of
% vertices that have been enumerated, in the following two command
% lines, we will extract this number of vertices and assign it to
% the variable 'N'.

token = strtok(line,',');
N = str2num(token(1,22:size(token,2)));

% Next, we will move 3 lines past the line that contained "Number
% of Vertices".

for i = 1:3,
  line = fgetl(fid);
end

% Next, we will read the contents of the next N lines. These next N
% lines give us the coordinates of the vertices.

n = size(A,2);
temp = fscanf(fid,'%f',[n+1,N]);

fclose(fid);

verts = temp(2:n+1,:)';
```

```
%————————————————————Next, the file Reach.ecd————————————————————

% open the file Reach.ecd for reading.
fid=fopen('Reach.ecd', 'rt');

% read in the first line of the file.
line = fgetl(fid);

% find the string 'begin' in the line read above.
K0 = findstr(line, 'begin');

while K0 == [],
 line = fgetl(fid);
 K0 = findstr(line, 'begin');
end

% Now, the pointer is on the line that contains 'begin'. Next, we
% will move the pointer one more line down.
line = fgetl(fid);

% Now, we will read in the entries on the next N lines.

for i = 1:N,

  line = fgetl(fid);

  [token,rem] = strtok(line);
  % Read the contents of 'line' upto and not including the
  % delimiter ' ' into the variable 'token', and read the rest of
  % the string beginning with and including ' ' upto the end of the
  % line, not including the end of line character and the carriage
  % return, into the variable 'rem'.

  firstitem = str2num(token);
  % This extracts the first item in 'line'.

  rem = rem( 5:size(rem,2) );
  % The line we read in is, for example, like this: '2 :  2 3'. A
  % single space after the first item and before the ':', then a
  % double space after the ':' and before the next numeral is the
  % format of the data in a *.ecd file. We assigned the first item
  % --2-- to the variable 'firstitem' in the above instructions.
  % The variable 'rem' is ' :  2 3'. Then, we removed the first 5
  % places from the beginning of 'rem'.

  if firstitem > n
    temp = str2num(rem);
```

```
    % Convert 'rem' into a numeric matrix containing N elements.

    r = f_mcn(firstitem,n);
    % Determine the firstitem-Choose-n different combinations of
    % the elements of the set {1,2,...,firstitem}.

    j = 1;

    flag = (rank( A( temp( r(j,:) )',: ) ) < n);
    % Suppose that firstitem = 3, n = 2, and temp = [2 4 7]. Then,
    % r = [1 2;1 3;2 3]. So, r(1,:) = [1 2], temp(r(1,:))=[2 4].
    % So, A( temp( r(1,:) ) ) will extract the rows number 2 and 4
    % of A. If the rank of A([2 4]',:) is less than n, then it
    % means that we have to try an n-subset of temp that is
    % different from [2 4]. That is what we do in the following
    % 'while' loop.

    while flag > 0,

      j = j + 1;
      flag = (rank( A( temp( r(j,:) ) ) ) < n);

    end

    act_constr(i,:) = temp(r(j,:));

  else

    act_constrs(i,:) = str2num(rem);

  end

end

fclose(fid);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_bound.m: An m-file to test for the boundedness of a given
%            system of inequalties Ax <= b.
%
% INPUT : The matrix A.
%
% OUTPUT : The value of 'flag' is set to 1 or 0 depending on whether
%          Ax <= b is bounded or unbounded respectively.
%
% PRECONDITION : A is an m x n matrix where m > n.
%
% USES : f_makine(.) which in turn uses cdd/cddf+.
%
% Created on : July 29, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The system Ax <= b (x is in the set of reals), will be bounded
% for a bounded b, if and only if the auxiliary system
% { rank(A) = n; A^Tz = 0; z > 0 } is feasible.
%
% This result (with "z > 0" replaced by "z >= 0") is used by
% function f_bound(A) to test for the boundedness of Ax <= b as
% follows:
%
% After checking if A is of full rank, f_bound(A) forms the system
% {A^Tz = 0; z >= 0}. This system is called 'z-cone' because it
% represents a cone for the variable z.
%
% f_bound(A) then uses the program cdd to test if the z-cone is
% non-empty. The z-cone is empty if the only element in it is 0.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function flag = f_bound(A)


if rank(A) == size(A,2)

 options = ['dynout_off '
            'stdout_off '
            'logfile_off'];
 c = [ ];

 Atran = A';
```

```
C2 = [ Atran;
       -Atran;
       -eye(size(Atran,2))];
d2 = [zeros(size(C2,1),1)];

% Place this information in the file bound.ine.
f_makine('bound.ine',options,c,C2,d2);

% Launching cddf+.exe to work on bound.ine.
!cddf+.exe bound.ine

   %-------------------------------------------------------------------
   %
   % Next, we will see if the number of extreme rays of the z-cone
   % is greater than 0. If it is, then z-cone is non-empty, and so
   % Ax <= b is bounded. The information about the number of
   % extreme rays of the z-cone is present in the file bound.ext
   % in the line '*Number of Rays = '..... So, we will see if
   % this line contains a number other than 0.
   %
   %-------------------------------------------------------------------

% open the file Ax_le_b.ext for reading.
fid=fopen('bound.ext', 'rt');

% read in the first line of the file.
line = fgetl(fid);

% find the string '*Number of Rays =' in the line read above.
K1 = findstr(line,'*Number of Rays =');

while K1 == [],
% while '*Number of Rays =' is absent in the line, do this loop.
% This loop executes until '*Number of Rays =' is found in the file.

  % read the next line.
  line = fgetl(fid);

  % check for the presence of '*Number of Rays =' in this line.
  K1 = findstr(line,'*Number of Rays =');

end
fclose(fid);

if strcmp(line,'*Number of Rays = 0') == 0
 flag = 1;
else
 flag = 0;
end
```

```
else
  flag = 0;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_cddX.m : Uses cdd to find a vector (x0,u0) that lies in the
%            feasible region of PR1.
%
%            PR1 is explained in the paper titled "Reachability of
%            target tube in a new class of uncertain systems
%            represented by linear constraints"
%            by Ramprasd Potluri and L.E.Holloway.
%
% INPUT :  Matrices 'rev_DB', 'T', 't'.
%
% OUTPUT : Vector 'x' such that rev_DB*x < 0.
%
% USES : f_inter(.) which in turn uses f_makine(.) which in turn
%        uses cddf+.
%
% Created on : June 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
% This m-file was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% We choose an (x0,u0) from the intersection of PR1 and Tube(T,t)
% because cddf+ does not work with homogeneous systems
% of inequalities.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  x = f_cddX(rev_DB,T,t);


m1 = size(rev_DB,1); % Number of rows of rev_DB
n1 = size(rev_DB,2); % Number of columns of rev_DB
m2 = size(T,1);
n2 = size(T,2);

A = [ rev_DB ; [T, zeros(m2,n1-n2)] ];

b = [zeros(m1,1); t];

x = f_inter('fcddx',A,b);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_EFB.m : Creat a slack-descriptor model (SD model).
%
% INPUT : m (number of rows of EFB), n_E (number of columns of E or
%         F; n_E is the same as the dimension of the state space),
%         n_B (number of columns of B; n_B is the same as dimension
%         of the control space).
%
% OUTPUT : E,F,B matrices such that the SD system Ex1 <= Fx0 + Bu0
%          is bounded, and the constant 'EFBtries'. Since, we
%          randomly generate the E,F,B matrices, 'EFBtries' gives
%          the number of trials made before finding a bounded EFB
%          system.
%
% USES : f_bound(.).
%
% Created on : July 29, 2002.
% Modified on : August 14, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [E,F,B,EFBtries] = f_EFB(m,n_E,n_B)


E = randn([m,n_E]);
tries = 1;

flag = f_bound(E);

while flag == 0

 E = randn([m,n_E]);
 flag = f_bound(E);
 tries = tries+1;

end

F = randn([m,n_E]);
B = randn([m,n_B]);
EFBtries = tries;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_findQ.m : Finds a Q matrix as explained in the paper titled
%             "Reachability of target tube in a new class of
%             uncertain systems represented by linear constraints"
%             by Ramprasd Potluri and L.E.Holloway.
%
% INPUT : rev_DB,T,t,E,F,B.
%
% OUTPUT : Q.
%
% USES : f_pickX(.), which inturn use f_square(.) and f_cddX(.),
%        which inturn uses cddf+, and f_active(.) which inturn uses
%        cddf+.
%
% Created on : June 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
% This m-file was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Q = f_findQ(rev_DB,T,t,E,F,B)


% To determine a matrix Q we proceed as follows:

% Choose some (x0,u0) tuple from PR1.

disp(' ');
disp('An (x0,u0) tuple that satisfies both PR1 and Tube(T,t)');
disp('is as follows:');
disp(' ');

% We choose an (x0,u0) from the intersection of PR1 and Tube(T,t)
% because cddf+ does not work with homogeneous systems of
% inequalities. So, if ask it to find an interior point from, say,
% Ax <= 0, it won't find one.

x0u0 = f_pickX(rev_DB,T,t)

disp(' ');
disp('CHECK : For this (x0,u0), the value of the product');
disp('rev_DB*[x0^T  u0^T]^T  must be negative:');
```

```
disp(' ');

revDBx0u0 = rev_DB*x0u0

% For this value of x0u0, determine Reach(x0,u0):

disp(' ');
disp('Reach(x0,u0) for this (x0,u0) is given by E <= [F B]*x0u0');
disp(' ');

% Enumerate the extreme points of Reach(x0,u0) and the constraints
% that are active at each of the extreme points of Reach(x0,u0):

A = E;
b = [F B]*x0u0;
[verts,constrs] = f_active(A,b);

% For i = 1,...,m_T (m_T = size(T,1) --- the number of rows of T),
% determine the extreme point from the above list at which [T]_ix1
% is maximum. Note the constraints of Reach(x0,u0) that are active
% at this extreme point.

Q = zeros( size(T,1),size(E,1) );
for i = 1:size(T,1),

  temp = verts*T(i,:)';
  [y,ind] = max(temp);

  % This assigns to 'y' the maximum value in the column vector temp
  % and to 'ind' the index of this maximum value. The significance
  % of 'ind' is that vert(ind,:) is the vertex of Reach(x0,u0) at
  % which [T]_ix1 ( represented here by T(i,:) ) attains
  % maximum. The constraints of Reach(x0,u0) which intersect at
  % vert(ind,:) have the indices constrs(ind,:), and their
  % direction vectors are given by:

  temp1 = E( constrs(ind,:)',: )'\T(i,:)';

  % [T]_i is a non-negative linear combination of the direction
  % vectors of these constraints.
  %
  % Then, the i-th row of Q will have the coefficients of this
  % linear combination in places whose indices are the same as the
  % indices of the constraints that were active at this particular
  % extreme point. In the remaining places of the i-th row of Q,
  % there will be 0's:

  Q(i,constrs(ind,:)) = temp1';
```

end

```
% $$$ disp(' ');
% $$$ disp('The matrix Q is as follows:');
% $$$ disp(' ');
% $$$ Q
% $$$ disp(' ');
% $$$ disp('CHECK: Q*E == T?');
% $$$ disp(' ');
% $$$ QE = Q*E
% $$$ T
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_inter.m : Finds an interior point of the given polyhedron
%             Ax <= b.
%
% INPUT :  Matrices 'A', 'b' such that Ax <= b. 'filename' which is
%          a character string and contains the name, say, "myfile"
%          etc which will be used by f_inter(.) to create the file
%          myfile.ine.
%
% OUTPUT : Column vector 'x' which is an interior point of Ax <= b.
%          x = [] means that no interior point of Ax <= b was
%          found.
%
% USES : f_makine(.) which in turn uses cddf+.
%
% Created on : August 13, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
% This m-file was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  x = f_inter(filename,A,b);


c = [];
options = ['find_interior'
           'dynout_off   '
           'stdout_off   '
           'logfile_off  '];
name = [filename '.ine'];

% Place this information in the file given by 'name':

f_makine(name,options,c,A,b);
% An .ine file whose name is given by 'name' (for example,
% myfile.ine if filename = 'myfile') has been created.

%-------------- Launching cddf+.exe to work on filename ---------------------

fid = fopen('mytemp.m','wt');
% A file mytemp.m is opened for writing.
% The reason for creating mytemp.m is because we want the end
```

```
% result of the following command (whereas the command
% '!cddf+.exe filename' will not give that same result):

fprintf(fid, '%s\n', ['!cddf+.exe' ' ' name]);

% The command "!cddf+.exe myfile.ine" (if filename = 'myfile')
% has been placed in mytemp.m.

fclose(fid);

mytemp;
% This executes the command "!cddf+.exe myfile.ine" (assuming
% filename = 'myfile').

    %-------------------------------------------------------------------
    %
    % Next, we will read the file myfile.lps to find the interior
    % point that has been determined by cdd+. If cdd+ finds an
    % interior point, it outputs the statment "LP status: a dual
    % pair (x, y) of optimal solutions found". Beginning the third
    % line after this statement, the coordinates of the interior
    % point are presented in the form of a vector which has a
    % length of size(A,2)+1. We will extract the first size(A,2)
    % elements from this vector. If the said statement is absent in
    % interior.lps, then we return x = [].
    %
    %-------------------------------------------------------------------

lpsname = [filename '.lps'];

% open the file, say, myfile.lps for reading.
fid=fopen(lpsname, 'rt');

% read in the first line of the file.
line = fgetl(fid);

% find the string 'optimal solutions found' in the line read above.
K0 = findstr(line, 'optimal solutions found');

K1 = findstr(line, 'begin');

while (K0 == [])&(K1 == []),
 line = fgetl(fid);
 K0 = findstr(line, 'optimal solutions found');
 K1 = findstr(line, 'begin');
end

if K0 == []
```

```
  x = [];

else

  while K1 == []
   line = fgetl(fid);
   K1 = findstr(line, 'begin');
  end

  % move down one line after the line in which the string
  % "begin" was found. This positions the pointer at the beginning
  % of the first element of the vector of the interior point.
  line = fgetl(fid);

  for i = 1 : size(A,2)
   line = fgetl(fid);
   [token,rem] = strtok(line);
   x(i,:) = str2num(rem(1,5:size(rem,2)));
  end

end

fclose(fid);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_makine.m: An m-file to form the file Ax_le_b.ine for use by cdd.
%
% INPUT : The matrix A, column vector 'b', matrices 'options' and
%         'inename', and the row vector c = [c0 c1 c2 ... ck] of the
%         objective function (c0 + c1*x1 + c2*x2 + ... + ck*xk) in
%         case the word 'maximize' or 'minimize' is one of the
%         elements of 'options'.
%
% OUTPUT : The file Ax_le_b.ine is created.
%
% PRECONDITION : If 'maximize' or 'minimize' is an option, then it
%                should be the first row of the matrix 'options'.
%
%                All rows of 'options' must be in small letters.
%
%                'inename' should contain a name and the extension
%                '.ine'.
%
% CAUTION : f_makine(.) has been designed to accept only the
%           following  options: hull, verify_input, dynout_off,
%           stdout_off, logfile_on, incidence, #incidence,
%           input_incidence, nondegenerate, adjacency,
%           input_adjacency, postanalysis, lexmin, lexmax,
%           minindex, mincutoff, maxcutoff, mixcutoff, random,
%           initbasis_at_bottom, maximize, minimize, find_interior,
%           facet_listing, tope_listing.
%
%           If the user wishes to use the options
%           partial_enumeration, equality, linearity,
%           strict_inequality, preprojection, zero_tolerance,
%           round_output_off, and output_digits, then he needs to
%           modify the function definition for f_makine(.) such
%           that it will allow for the inputs that accompany these
%           options.
%
%           The options vertex_listing cannot be placed in an *.ine
%           file, so though f_makine won't complain if this option
%           is input to it, cdd most certainly will.
%
%           If the options maximize or minimize are input, then a
%           valid vector 'c' that specifies the objective function
%           also must be input. If either one of these options is
%           not chosen, then input 'c = [ ]'.
%
%           Some of the options listed above can be used only with
%           cdd+ and not with cdd. For an explanation of the
%           options that can be used with cdd (cdd+), please read
```

```
%            cdd's user manual.
%
% Created on : August 06, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [] = f_makine(inename,options,c,A,b)


C2 = A;
d2 = b;
d2C2 = [d2  -C2];
width = 5;

% Place this information in the file whose name is in inename:

fid = fopen(inename,'wt');

fprintf( fid, '%s\n', ['*File: ' inename] );
fprintf(fid, '%s\n', 'H-representation');
fprintf(fid, '%s\n', 'begin');

% The following 2 commands form a line such as "8   5   real"
% that one would find in cdd's *.ine files.

temp = [num2str(size(C2,1)) '   ' num2str(size(C2,2)+1) '   ' 'real'];
fprintf(fid,'%s\n',temp );

% The following piece of code until the end of j's FOR-loop writes
% the contents of d2C2 to the file whose id is fid (i.e.,
% Ax_le_b.ine).

H = [];
for j = 1:size(d2C2,1),

  H = [];
  for k = 1:size(d2C2,2),

    if d2C2(j,k) >= 0,

      temp = blanks(width-1-size(num2str(d2C2(j,k)),2));
      item = [' ' num2str(d2C2(j,k),width) temp];
      % Adding the single space in the beginning of item helps
      % left-justify the digits in positive numbers with those of
      % negative numbers.

    else
```

```
      temp = blanks(width-size(num2str(d2C2(j,k)),2));
      item = [num2str(d2C2(j,k),width) ];

    end

    H = [H  item '  '];

  end

  fprintf(fid, '%s\n',H);
  % One row of d2C2 has been written to the file.

end
% All rows of d2C2 have been written to the file.

fprintf(fid, '%s\n', 'end');

if size(options,1) > 0

  k = 1;
  if findstr( options(1,:) , 'maximize' )

    fprintf(fid, '%s\n', 'maximize');

    % Next, place the vector 'c' into the file whose name is in
    % inename.
    H = [];
    for i = 1:size(c,2),
      H = [H '  ' num2str(c(:,i),width)];
    end

    fprintf(fid, '%s\n', H);
    k = 2;

  end

  for i = k:size(options,1)
    fprintf( fid, '%s\n', options(i,:) );
  end

end

fclose(fid);
% End of placing this information in the file whose name is in
% inename.

% End of the function f_makine(.)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_max.m: An m-file to solve the linear programming problem of
%          maximizing an objective function over a system of
%          inequality constraints Ax <= b.
%
% INPUT : The row vector c = [c0 c1 c2 ... ck] of the objective
%         function (c0 + c1*x1 + c2*x2 + ... + ck*xk),
%         the matrix A, the column vector b.
%
% OUTPUT : The maximum value of c*x.
%
% USES : f_makine which inturn uses cdd (cddf+).
%
% Created on : July 29, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% We use cddf+.exe with the 'maximize' option specified.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function max_val = f_max(c,A,b)


options = ['maximize    '
           'dynout_off  '
           'stdout_off  '
           'logfile_off'];

% Place this information in the file maxim.ine:

f_makine('maxim.ine',options,c,A,b);

% Launching cddf+.exe to work on maxim.ine:

!cddf+.exe maxim.ine

    %-----------------------------------------------------------------
    %
    % Next, we will read the file maxim.lps to find the optimal
    % value. The *.lps file presents this information in either one
    % of the two statements:  'unbounded direction for the primal
    % LP' or 'optimal_value : .....'. In the second statement, some
    % number is present in the place of the dots.
    % The following piece of code checks if the first statement is
    % present. If it is, then the code returns max_val = inf, where
```

```
    % 'inf' is a standard MATLAB value. Else, if the first
    % statement is not present, then the code looks for the second
    % statement (without the ':') and extracts the number that is
    % present in that statement. The code then returns this number
    % as max_val.
    %
    %-------------------------------------------------------------------

fid=fopen('maxim.lps', 'rt');
% open the file maxim.lps for reading.

line = fgetl(fid);
% read in the first line of the file.

K0 = findstr(line, 'LP is inconsistent');
% find the string 'LP is inconsistent' in the line read above.
K1 = findstr(line,'unbounded direction for the primal LP');
K2 = findstr(line,'begin');

while (K0 == []) & (K1 == []) & (K2 == []),
  line = fgetl(fid);
  K0 = findstr(line, 'LP is inconsistent');
  K1 = findstr(line,'unbounded direction for the primal LP');
  K2 = findstr(line,'begin');
end

if K0 == []

  if K1 == [],

    s = 'optimal_value';
    K3 = findstr(line,'end');
    K4 = findstr(line,s);

    while (K4 == [])&(K3 == []),
    % This condition is similar to the above while loop
    % that uses K1 and K2. Please read that loop's explanation.

      % read the next line.
      line = fgetl(fid);

      K3 = findstr(line,'end');
      K4 = findstr(line,s);

    end

    [token,rem] = strtok(line);
    max_val = str2num(rem(1,4:size(rem,2)));
```

```
    else
      max_val = inf;
    end

else

    max_val = [];

end

fclose(fid);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_pickX.m:  Choose some (x0,u0) tuple from PR1.
%
% INPUT : The matrix rev_DB that gives the solution of PR1 as
%         rev_DB*x < 0, and the tube parameters T and t.
%
% OUTPUT : A vector 'x' such that x = [x0^T u0^T]^T is in PR1.
%
% USES : Functions f_square(.), f_cddX(.) which inturn uses f_max(.)
%       which inturn uses cdd (cddf+).
%
% Created on : July 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x = f_pickX(rev_DB,T,t)


x = [];
temp1 = -ones( size(rev_DB,1),1 );
rownum = size(rev_DB,1);
colnum = size(rev_DB,2);

if ( rank(rev_DB) == min(rownum,colnum) )&( rownum < colnum )
% Meaning, if rev_DB is full ranked, and
% if # of rows of rev_DB < # of columns of rev_DB

   col_inds = f_square(rev_DB);
   % Obtain the indices of some 'rownum' linearly independent
   % columns of rev_DB.

   x      = zeros( colnum,1 );
   x(col_inds,1) = rev_DB(:,col_inds)\temp1;

   % What we have done in the above 3 lines is this: Since PR1
   % needs that rev_DB*x be less than 0, we have arbitrarily
   % decided to set rev_DB*x to temp1 which is a vector of -1's.
   % Next, we have arbitrarily chosen those elements of x to be
   % zeros whose indices are absent in col_inds.
   % I didn't know how to implement the idea of this last sentence.
   % So, I set all elements of x to 0, and then assigned - to those
   % of its elements that are indexed by col_inds - the value of
   % the product inv( rev_DB(:,col_inds) )*temp1.
   % Now, if we compute the product rev_DB*x, we should get a value
   % equal to temp1, which is strictly less than 0.
   %
```

```
    % Here, we have used the equation x = A\b instead of
    % x = inv(A)*b.
    % That is, instead of
    % x(col_inds,1) = inv(rev_DB(:,col_inds))*temp1;
    % we have used
    % x(col_inds,1) = rev_DB(:,col_inds)\temp1;
    %
    % Both ways solve the system of linear equations Ax = b.
    % But, the first way is about 2-3 times faster and more accurate.
    % [Page 468. The student edition of Matlab: version 4 : User's
    % guide/ the Mathworks Inc. 1995.]

elseif ( rank(rev_DB) == min(rownum,colnum) )&( rownum == colnum )
% Meaning, rev_DB is a full-ranked square matrix.

   x = rev_DB\temp1;

else

   x = f_cddX(rev_DB,T,t);

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_poly2D.m : M-file to draw a 2-D polytope (a bounded convex
%              polyhedron).
%
% INPUT : Matrix 'verts' of dimension m x 2.
%
% OUTPUT : Matrix 'adjverts' of dimension m x 2. Also a 2-D
%          polytope is drawn to screen. Please see below for a
%          description of 'adjverts'.
%
% PRECONDITION : The vertices should belong to a polytope.
%
% Created on : August 15, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% This M-file works as follows: It creates a matrix 'adjverts' of
% dimension m x 2. 'adjverts' has been obtained by reordering the
% rows of 'verts' such that the i-th row of 'adjverts' represents a
% vertex that is adjacent to the vertex represented by the (i-1)-th
% row of 'adjverts'. Also, the vertices represented by the 1-st and
% m-th row of 'adjverts' represent vertices that are adjacent.
%
% The information in 'adjverts' is used by MATLAB's fill(.)
% function to draw the polytope.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function adjverts = f_poly2D(verts)


m = size(verts,1);

% Initialize adjverts:

adjverts = [];

for k1 = 1 : m-2

  % Initialize all local variables:

  a = [];
  b = [];
  c = [];
  d = [];
  b1 = [];
```

```
b2 = [];
k2 = [];
temp = [];
tempvect = [];

% Among those elments that are after the k1-th, beginning with the
% element number k1+1, search for and find an element that is
% adjacent to the k1-th element:

flag = 0; % This is simply to get the program to go into the
          % while loop.
k2 = k1+1;
while (flag == 0)&(k2 <= m)

  % For the k1-th element of 'verts', obtain the coefficients of
  % the equation of the line through this element and the k2-th
  % element of 'verts'. If (x1,y1) and (x2,y2) are the elements,
  % then the equation of the line through them is as follows:
  %
  %       y - ( (y1-y2)/(x1-x2) )*x = (y2*x1-y1*x2)/(x1-x2)
  %
  % Denote a = (y1-y2)/(x1-x2), b = (y2*x1-y1*x2)/(x1-x2).

  a = ( verts(k1,2)-verts(k2,2) )/( verts(k1,1)-verts(k2,1) );
  b1 = ( verts(k2,2)*verts(k1,1)-verts(k1,2)*verts(k2,1) );
  b2 = verts(k1,1)-verts(k2,1);
  b = b1/b2;

  % Test if all the elements of 'verts' are either on this line
  % or on only one side of this line (if they are not, then the
  % k1-th and the k2-th elements do not represent adjacent
  % vertices):

  tempvect = verts(:,2)-a*verts(:,1);
  % tempvect = y-a*x.

  c = f_trunc(tempvect,4) <= f_trunc(b,4);
  d = f_trunc(tempvect,4) >= f_trunc(b,4);

  % The function f_trunc(val,num) trucates and rounds 'val' to
  % number of decimal places equal to 'num'.
  % 'c' is a column vector. 'c' will have 0's corresponding to
  % those elements of 'tempvect' that are greater than 'b'.

  if ( min(c) == 0 )&( max(c) == 1 )&( min(d) == 0 )&( max(d) == 1)
    flag = 0;
    k2 = k2 + 1; % Try another k2.
  else
    flag = 1;
```

```
    end

  end

  % Swap the (k1+1)-th row of 'verts' with the k2-th row of 'verts':

  temp = verts(k1+1,:);
  verts(k1+1,:) = verts(k2,:);
  verts(k2,:) = temp;

end

adjverts = f_trunc(verts,4);

fill(adjverts(:,1),adjverts(:,2),'r');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_square.m: Returns the indices of those columns (or rows) of a
%             rectangular matrix that are linearly independent.
%
% INPUT : The matrix A
%
% OUTPUT : The vector 'indices' that contains the indices of the
%          columns (or rows) that are linearly independent.
%
% PRECONDITION : A must be of full rank and not-square.
%
% USES : The function f_mcn(.).
%
% Created on : July 30, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function indices = f_square(A)


cols = size(A,2);
rows = size(A,1);
indexes = [];

if rows < cols

  indexes = f_mcn(cols,rows);

  i = 1;
  while rank( A( :,indexes(i,:) ) ) < rows
    i = i + 1;
  end

elseif cols < rows

  indexes = f_mcn(rows,cols);

  i = 1;
  while rank( A( indexes(i,:),: ) ) < cols
    i = i + 1;
  end

end

indices = indexes(i,:);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_TinPR1.m : An m-file implementation of the function f_TinPR1(.).
%              Tests if Tube(T,t) lies completely within the
%              projection of PR1 onto the x0 coordinates.
%
%              Tube(T,t) is in PR1 if
%              [t]_i = max{[T]_ix | x \in closure(PR1), x \in Tube}
%              is true for i = 1,...,m_T. Here [X]_i is the i-th
%              row of X, m_T is number of rows of T.
%
%              For details please see the paper titled
%              "Reachability of target tube in a new class of
%              uncertain systems represented by linear constraints"
%              by Ramprasd Potluri and L.E.Holloway.
%
% INPUT : The matrices rev_DB, T, and t which are such that
%         rev_DB*x < 0 is the solution set of PR1, and Tx <= t is
%         the Tube(T,t).
%
% OUTPUT : a scalar x. x = 0 if Tube(T,t) does not intersect
%          proj_x0{PR1}, x = 1 if proj_x0{PR1} does not completely
%          contain Tube(T,t), and x = 2 if proj_x0{PR1} contains
%          Tube(T,t).
%
% USES : f_max(.) which inturn use cdd (cddf+), and the m-file scs.m.
%
% Created on : August 06, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point, unless he
% wishes to use this m-file on non-Windows 98 machines.  This m-file
% was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x =  f_TinPR1(rev_DB,T,t);


% Compare [t]_i with max of [T]_ix over rev_DB*x < 0, Tx \le t.

m1 = size(rev_DB,1); % Number of rows of rev_DB
n1 = size(rev_DB,2); % Number of columns of rev_DB
m2 = size(T,1);
n2 = size(T,2);
```

```
A = [ rev_DB;
      [T, zeros( m2, n1-n2 )] ];
b = [ zeros(m1,1);
     t ];
c = [ 0, T(1,:), zeros( 1, n1-n2 ) ];

max_val = f_max( c , A , b );

if max_val == []      % IF#3
  x = 0;
  % Tube does not intersect PR1.
else

  if t(1) == max_val % IF#4

    i = 2;
    temp = 0;
    while (i <= m2) & (temp == 0)

      if t(i) == f_max( [0,T(i,:),pad_row], A , b )
        i = i + 1;
      else
        temp = 1;
      end

    end

  else                % IF#4

    i = 1;
    temp = 1;

  end                 % IF#4

  if i == m2 + 1    % IF#5
    x = 2;
    % proj_x0{PR1} contains Tube completely.
  else                % IF#5
    x = 1;
    % proj_x0{PR1} contains Tube INcompletely.
  end                 % IF#5

end                   % IF#3
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_trunc.m : M-file to truncate and round numbers to specified
%             numbers of decimal places.
%
% INPUT : 'val' - the number which needs to be truncated, 'num' -
%         the number of decimal places to which 'val' needs to be
%         truncated.
%
% OUTPUT : 'truncval' - the turncated number.
%
% Created on : August 15, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function truncval = f_trunc(val,num)


truncval = round(val*(10^num))/(10^num);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_tstmnt.m : Determines if the given Tube(T,t) is maintainable as
%              explained in the paper titled "Reachability of
%              target tube in a new class of uncertain systems
%              represented by linear constraints" by Ramprasad
%              Potluri and L.E.Holloway.
%
% INPUT : verts ( generated by f_verts(.) ),Q,t,F,B.
%
% OUTPUT : x = 0 if Tube(T,t) NOT maintainable.
%          x = 1 otherwise.
%          vertcntrls = [] if Tube(T,t) is not maintainable.
%          If Tube(T,t) is maintainable, then the i-th row of the
%          matrix 'vertcntrls' represents a maintaining control for
%          the i-th vertex of Tube(T,t).
%
% USES : f_makine(.), cddf+.
%
% Created on : August 08, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
% This m-file was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x,vertcntrls] = f_tstmnt(verts,Q,t,F,B)


N = size(verts,1);

% Test if QBu0 <= t-QFx0i is feasible ( x01,x02,..,x0N are the
% vertices of the tube):

for i = 1:N,

  A = Q*B;
  b = t - Q*F*verts(i,:)';

  name = ['ftstmnt' num2str(i)];
  temp = f_inter(name,A,b);

  if temp == []
    % Meaning, no interior point was found for QBu0 <= t-QFx0i
```

```
    x = 0;
    vertcntrls = [];
    break;
  else
    x = 1;
    vertcntrls(i,:) = temp';
  end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_tube.m : Gives a bounded irredundant
%            Tube(T,t) := { x | Tx <= t }.
%
% INPUT : mT (number of rows of T and t), nT (number of columns
%         of T; nT is the same as the dimension of the state
%         space); numties is the maximum number of random values of
%         't' to try.
%
% OUTPUT : T, t such that Tx <= t is bounded and irredundant, and
%          the constant 'tubetries'. Since, we randomly generate
%          T and t, 'tubetries' gives the number of trials made
%          before we find a bounded, irredundant system Tx <= t. If
%          no valid 't' is found in numtries number of attempts,
%          then T = [], t = [] are returned.
%
% PRECONDITION : mT >= nT+1.
%
% USES : f_bound(.) (which in turn uses cdd/cddf+) and f_scs(.).
%
% Created on : July 29, 2002.
% Modfied on : August 14, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [T,t,tubetries] = f_tube(mT,nT,numtries)


% First, let us determine a T which will give a bounded Tube.

T = randn([mT,nT]);
tubetries = 1;

flag = f_bound(T);

while flag == 0
  T = randn([mT,nT]);
  flag = f_bound(T);
  tubetries = tubetries+1;
end

% Next, let us determine a 't' which will give an irredundant tube.
% For this, we will first determine the set of all values of t for
% which Tx <= t will be irredundant.
% Then we will randomly generate t until we find one that is an
% element of the said set.
```

```
%
% Note that, if x is 1-D, then mT = 2, nT = 1, and the tube is
% always irredundant. Since, f_scs(.) that we use below requires
% that x be at least 2-D, we will test for the irredundancy of the
% tube only if we have at least a 2-D system.

if nT >= 2

  [rev_D,rev_DB] = f_scs(T,[]);
  % We have used the function f_scs(.). For a null second argument
  % in the input, this function returns a null value of rev_DB.
  % The set of all values of t for which Tx <= t is irredundant is
  % given by rev_D*t < 0.

  t = randn([mT,1]);

  flag = (rev_D*t < 0);

  while (min(flag) == 0)&(tubetries < numtries)
    t = randn([mT,1]);
    flag = (rev_D*t < 0);
    tubetries = tubetries+1;
  end

  if min(flag)==0
    T = [];
    t = [];
  end

else

  t = randn([mT,1]);

  % Test for the satisfaction of the condition
  % [-1, a1/a2]*[b1;b2] <= 0:

  flag = ([-1, T(1,1)/T(2,1)]*t <= 0);
  while (flag == 0)&(tubetries < numtries)
    t = randn([mT,1]);
    flag = ([-1, T(1,1)/T(2,1)]*t <= 0);
    tubetries = tubetries + 1;
  end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_verts.m : Enumerates the vertics of Ax <= b.
%
% INPUT : A,b.
%
% OUTPUT : Matrix 'verts' whose rows represents the vertices.
%
% PRECONDITION : Ax <= b must be bounded. You can use f_bound(.) to
%                test this. Neither of A and b should be empty.
%
% USES : f_makine(.), cdd/cddf+.
%
% Created on : August 12, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
% This m-file was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function verts = f_verts(A,b)


options = ['dynout_off '
           'stdout_off '
           'logfile_off' ];
c = [ ];
name = 'verts.ine';

% Place this information in the file verts.ine:

f_makine(name,options,c,A,b);

% Launching cddf+.exe to work on verts.ine:

!cddf+.exe verts.ine

% A file -- verts.ext has been generated by cddf+.

%-----------Get vertices of Ax <= b from the file verts.ext-------

fid=fopen('verts.ext', 'rt');
% open the file Reach.ext for reading.
```

```
line = fgetl(fid);
% read in the first line of the file.

K0 = findstr(line, 'Number of Vertices');
% find the string 'Number of Vertices' in the line read above.

while K0 == [],
  line = fgetl(fid);
  K0 = findstr(line, 'Number of Vertices');
end

% Now that we have found the line that contains the number of
% vertices, in the following two command lines, we will extract
% this number of vertices and assign it to the variable 'N'.

token = strtok(line,',');
N = str2num(token(1,22:size(token,2)));

if N > 0

  % Next, we will move 3 lines past the line that contained "Number
  % of Vertices".

  for i = 1:3,
    line = fgetl(fid);
  end

  % Then, we will read the contents of the next N lines. These next
  % N lines give us the coordinates of the vertices.

  n = size(A,2);
  temp = fscanf(fid,'%f',[n+1,N]);

  verts = temp(2:n+1,:)';

else

  verts = [];

end

fclose(fid);

% Now, the matrix 'verts' contains the coordinates of the vertices
% of Ax <= b. Each row of verts represents one vertex of Ax <= b.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% mcntemp.m: file generated by f_mcn.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

aa=[];
x1 = 0;
for x2 = x1 + 1 : 4,
 for x3 = x2 + 1 : 5,
  bb =[ x2 x3 ];
  aa = [aa; bb];
 end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% mytemp.m : File created by f_inter.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


!cddf+.exe ftstmnt5.ine
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% pre_pros.m : Accomplishes the preprocessing needed to test the
%              theory of maintainability presented in the paper
%              titled "Reachability of target tube in a new class
%              of uncertain systems represented by linear
%              constraints" by Ramprasd Potluri and L.E.Holloway.
%
% USES : f_EFB(.), f_tube(.), f_scs(.), f_TinPR1(.), f_findQ(.),
%        f_verts(.), f_tstmnt(.) which inturn use other functions.
%
%        Please see README.txt for a complete tree of which
%        functions use which.
%
% Created on : June 29, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point, unless he
% wishes to use this m-file on non-Windows 98 machines.  This m-file
% was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

diary control1.txt
tic

% Create a bounded EFB system in regular form.

m  = 5; % number of rows of EFB.
nE = 2; % number of columns of E or F.
nB = 2; % number of columns of B.

[E,F,B,EFBtries] = f_EFB(m,nE,nB);

disp(' ');
disp('The matrices E, F, B in the SD system Ex1 <= Fx0 + Bu0 are:');
disp(' ');
disp(' ');
E
F
B
disp(' ');
disp(' ');
disp('This SD system is bounded.');

%-------------------------------------------------------------------------
```

```
% Create a bounded irredundant tube Tube(T,t) := { x | Tx <= t }
% and test if it is bounded.

mT = 5;
nT = nE;
numtries = 50; % Maximum number of random values of 't' to try.

[T,t,tubetries] = f_tube(mT,nT,numtries);

% If a tube is not found in a maximum of numtries number of tries,
% then f_tube(.)  returns T = [] and t = [].

if (size(T,1)==0)       % IF #1

  disp(['A tube was not found in ' num2str(numtries) ' attempts']);

else                    % IF #1
  disp(' ');
  disp('The tube Tx1 <= t is given by the following T, t:');
  disp(' ');
  T
  t
  disp(' ');
  disp('This tube is bounded and irredundant');
  disp(' ');

toc

%-------------------------------IF #1-------------------------------------

tic

% Determine the set PR1 of all (x0,u0) tuples for which
% Reach(x0,u0) is irredundant.

A = E;
C = [F B];

if nE == 1

  rev_D = [-1, A(1,1)/A(2,1)]; % The system Ax <= b, where 'b'
                               % belongs to the set of all 'b' such
                               % that rev_D*b <= 0, is irredundant.
  rev_DB = rev_D*C; % The system Ax <= Cy, where y belongs to the
                    % set of all 'y' such that rev_DB*y <= 0, is
                    % irredundant.

else
```

```
  % This can be done with the help of the program scs.m that is in
  % the folder C:\MATLAB\Matfiles\research\SCS.
  % I have set the appropriate path in the file
  % C:\MATLAB\Matlabrc.m.
  % The m-file scs.m works on the system Ax <= Cy.
  % In the case of our EFB system, this is
  % Ex1 <= [F B]*[x0^T u0^T]^T.

  % A and C are used by scs.m.

  [rev_D,rev_DB] = f_scs(A,C);

end

disp(' ');
disp('The set PR1 of all (x0,u0) tuples for which Reach(x0,u0)');
disp('is irredundant is given by rev_DB*[x0^T u0^T]^T < 0,');
disp('where rev_DB is as follows:');
disp(' ');

rev_DB

%-------------------------------------------------------------------------

% Test if Tube(T,t) lies completely within the projection of PR1
% onto the x0 coordinates. Tube(T,t) is
% in PR1 if [t]_i = max{ [T]_ix | x \in closure(PR1), x in Tube }
% is true for i = 1,...,m_T. Here [X]_i represents the i-th row of
% X, m_T is number of rows of T. (m_T is also specified in the
% file spec_tub.m).

% Compare [t]_i with max of [T]_ix over rev_DB*x < 0, Tx \le t :

x = f_TinPR1(rev_DB,T,t);

if x == 0
  disp('Sorry! The Tube does not intersect proj_x0{PR1}');

  % One reason for why we care for whether Tube intersects
  % proj_x0{PR1} is because in the function f_findQ(.) below, we
  % choose an (x0,u0) from the intersection of PR1 and Tube(T,t)
  % because cddf+ does not work with homogeneous systems of
  % inequalities. So, if we ask it to find an interior point from,
  % say, Ax <= 0, it won't find one.

elseif x == 1
  disp(' ');
  disp('Well! proj_x0{PR1} contains this Tube INCOMPLETELY.');
```

```
  disp(' ');
else
% x = 2
  disp(' ');
  disp('Congratulations! The Tube is a subset of PR1');
  disp(' ');
end

%-------------------------------------------------------------------------

if (x == 1)|(x == 2)

  Q = f_findQ(rev_DB,T,t,E,F,B)
  verts = f_verts(T,t)
  [x1,vertcntrls] = f_tstmnt(verts,Q,t,F,B)
  % x = 0 if Tube is not maintainable, x = 1 if maintainable.

  if x1 == 0
    disp('Sorry! Tube(T,t) is not maintainable.');
    dips('Try tweaking "t". ');
  else
    disp('Congratulations! Tube(T,t) is maintainable.');
  end

end

%-------------------------------------IF #1-------------------------------

end                        % IF #1

toc

diary off
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ptope.m : An example of obtaining an irredundant tube in 2D and
%           plotting its cross-section.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[T,t,tubetries] = f_tube(5,2,50);

if size(T,1) > 0

  tic
  verts = f_verts(T,t)
  toc

  adjverts = f_poly2D(verts)

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Z.m : This was written to test some parts of the program. This is
%       not a part of the program itself.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mt = 7;
nt = 3;

Z = randn(mt,nt);

while rank(Z) < min( mt,nt )
  Z = randn(mt,nt);
end

Z
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% control1.txt : Example result of pre_pros.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

The matrices E, F, B in the SD system Ex1 <= Fx0 + Bu0 are:


E =

   -1.4462   -0.3600
   -0.7012   -0.1356
    1.2460   -1.3493
   -0.6390   -1.2704
    0.5774    0.9846


F =

   -0.0449    0.5135
   -0.7989    0.3967
   -0.7652    0.7562
    0.8617    0.4005
   -0.0562   -1.3414


B =

    0.3750   -0.3229
    1.1252    0.3180
    0.7286   -0.5112
   -2.3775   -0.0020
   -0.2738    1.6065


This SD system is bounded.

The tube Tx1 <= t is given by the following T, t:


T =

    0.8476   -0.5571
    0.2681   -0.3367
   -0.9235    0.4152
   -0.0705    1.5578
    0.1479   -2.4443
```

```
t =

    0.5360
    0.2985
    0.2840
    0.9597
    2.0876


This tube is bounded and irredundant


elapsed_time =

    4.3900


The set PR1 of all (x0,u0) tuples for which Reach(x0,u0)
is irredundant is given by rev_DB*[x0^T u0^T]^T < 0,
where rev_DB is as follows:


rev_DB =

    1.4915   -0.3074   -1.7071   -0.9583
   -0.7737    0.2004    0.9491    0.4061
  -16.0079   24.4836   52.8296  -38.1064
    1.4199   -0.6291   -3.2475    0.6693
    3.1656   -1.0148   -5.3703   -0.3307


Well! proj_x0{PR1} contains this Tube INCOMPLETELY.

An (x0,u0) tuple that satisfies both PR1 and Tube(T,t)
is as follows:


x0u0 =

   -0.0090
   -0.8484
    0.1621
   -0.0147


CHECK : For this (x0,u0), the value of the product
rev_DB*[x0^T  u0^T]^T  must be negative:
```

```
revDBx0u0 =

   -0.0152
   -0.0152
  -11.5059
   -0.0152
   -0.0331


Reach(x0,u0) for this (x0,u0) is given by E <= [F B]*x0u0


Q =

         0         0    0.5764         0    0.2242
         0         0    0.2273    0.0236         0
         0    1.8772         0         0    0.6802
         0    1.5828         0         0    1.8002
         0         0    0.7156    1.1639         0


verts =

    0.1045   -0.8032
    1.0690    0.6644
   -0.0312    0.6146
    0.0439   -0.8514
   -0.7109   -0.8971


x1 =

     1


vertcntrls =

   -1.0270    0.1189
   -0.3627    1.0515
   -0.6292    0.7197
   -1.0555    0.0805
   -1.2428   -0.1428

Congratulations! Tube(T,t) is maintainable.

elapsed_time =

    8.7900
```

Appendix G

MATLAB Program Listings for the Q-Matrix Method of Maintaining a Halfspace System in a Tube

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% control.m : To maintain system states in the tube, this m-file
%             executes the control alogrithm developed in the paper
%             titled "Reachability of target tube in a new class of
%             uncertain systems represented by linear constraints" by
%             Ramprasd Potluri and L.E.Holloway.
%
%             Also, performs sensing when required.
%
% INPUT : The matrices 'verts' (each row of this matrix represents a
%         vertex of the tube Tx <= t), 'vertcntrls' (the i-th row of
%         'vertcntrls' gives a maintaining control for the i-th row of
%         'verts'), 'E', 'F', 'B' (these three matrices specify the
%         EFB system), 'T', 't'(these two matrices specify the tube),
%         'Q', 'rev_DB' (rev_DB defines the set of all (x0,u0) tuples
%         for which the system Ex1 <= Fx0 + Bu0 is irredundant), 'x0'
%         (an intial state which has maintaining controls), 'horizon'
%         (this specifies the time horizon over which we want to
%         maintain the system's states in the tube, 'step' (this is
%         the length of the time-step for the control).
%
%         All the above matrices can be generated by running the
%         m-file pre_pros.m
%
% USES : The functions f_inter(.), f_genx(.)
%
% Created on : August 13, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
% This m-file was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

diary control2.txt

vnum = size(verts,1); % vnum is the number of vertices of the tube.

n = size(verts,2); % n is the dimension of the state-space.

%-------------------------------------------------------------------------

% Generate a random initial state x0p which is in Tube(T,t). This
% random state is generated by taking a random mean of all the
% vertices as follows:

randinds = randperm(vnum); % The MATLAB function randperm(n) gives
                           % a random permutation of the elements
                           % of 1:n. 'randinds' is a row vector.

x0p = (1/vnum)*(verts'*randinds'); % x0p is a column vector.

%-------------------------------------------------------------------------

numsteps = 50; % Total number of time steps over which we want to
               % control the system.

states(:,1) = x0p; % The matrix 'states' stores the information
                   % about the states.

for i = 1:numsteps

% So far, we have generated a random initial state x0 that is
% inside the tube.
%
% Next, we will first see if, in the absence of any control,
% Reach(x0,0) will be inside the tube. If it will not be, then we
% will compute a control u0 such that Reach(x0,u0) will be inside
% the tube.
%
% Then, we will store the definition of this Reach set in X1.

if (Q*F*x0p <= t)
% Meaning, if Reach(x0p,0) is in Tube(T,t), then:

%  disp('Reach(x0p,0) is in Tube(T,t)');
```

```
  % Denote Rch := Reach(x0p,0):
  Rch = [E,F*x0p];
  cntrls(1:size(F,2),i) = zeros(size(F,2),1);

else

  A = [verts'
       ones(1,vnum)
       -verts'
       -ones(1,vnum)
       -eye(vnum)    ];
  b = [x0p
       1
       -x0p
       -1
       zeros(vnum,1) ];

  filename = 'lambdas';

tic
  lambda = f_inter(filename,A,b); % f_inter.m is a file in the
                                  % folder TstMaint. f_inter(.)
                                  % returns a column vector.
toc
  % f_inter(.) took 1.04 seconds to find an interior point of
  % Ax <= b.

  % Determine a maintaining control for x0:
  u0p = vertcntrls'*lambda; % u0p is a column vector.

  % Denote Rch := Reach(x0p,u0p):
  Rch = [E,F*x0p+B*u0p];
  cntrls(1:size(F,2),i) = u0p;

end

%-------------------------------------------------------------------------

% Check if the one-step evolution of the system from the Reach set
% given by X1 in the absence of a control will be within the tube.
%
% For this, we can test if the Reach sets from each of the extreme
% points of X1 in the absence of a control will be inside the
% tube. However, this method will need us once more to resort to
% cdd / cddf+ to find the extreme points of X1. For a 5x2 system Tx
% <= t, using cdd/cddf+ takes 0.88 - 1.2 seconds on my Windows 98
% HP Pavilion 8660C PC which has 128 MB of RAM and a processor
% speed of 533 MHz.
```

```
%
% Instead, we can take the basic solutions of X1 that will be
% obtained through the linear combinations of its rows if we were
% to use Q. Then, from these basic solutions, we can consider the
% one-step evolution of the system (Reach sets) as mentioned in the
% previous paragraph. This, will be a "harsher" test as explained
% in Chapter 6 of the dissertation.
%
%-------------------------------------------------------------------
%
% flag = 0, if the one-step evolution will not be inside the tube.
% flag = 1, otherwise.
%
%-------------------------------------------------------------------

%if flag = 0

   x1 = f_genx(Rch); % x1 is a column vector.

   % f_genx(.) generates a state that is in Reach(x0,u0) using the
   % information in Rch.
   %
   % In practice, we will use f_sense(.) which will give us the
   % system's state as obtained from sensing. However, for the
   % purposes of this algorithm, we use f_genx(.).

   % Assign this x1 to x0:
   x0p = x1;

   % Also store this x1 in states:
   states(:,i) = x1;

%else
%
% Here, one may add functionality to bring the evolution back into
% the tube. This functionality has not be implemented in this
% dissertation.
%
%end

end  % End of FOR.

states;
cntrls;

%-------------------------------------------------------------------

diary off
```

```
% Next, we want to plot the data we obtained in the matrix
% 'states'.

i = 1:numsteps;
for j = 1:n
  figure
  subplot(2,1,1)
  plot(i,states(j,:)), xlabel('k'), ylabel(['x' num2str(j)]);
  subplot(2,1,2)
  plot(i,cntrls(j,:)), xlabel('k'), ylabel(['u' num2str(j)]);
%  print -deps exmaint.eps
end

% plotting controls needs to be seperate from plotting states. The
% number of controls may be different from the number of states.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% f_genx.m : Generates a state that is in Reach(x0,u0).
%
% INPUT : The matrix 'Rch' which stores the information
%         Ex1 <= Fx0+Bu0 as shown in the m-file control.m.
%
% OUTPUT : A row vector x1 which satisfies Ex1 <= Fx0+Bu0.
%
% PRECONDITION : The last column of Rch is the vector Fx0+Bu0. Rch
%                is of the form Rch = [E, Fx0+Bu0].
%
% USES : f_bound(.) , f_max(.) both of which inturn use cdd (cddf+),
%        and f_scs(.).
%
% Created on : August 10, 2002.
% Author : Ramprasad Potluri.
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User does not need to modify anything beyond this point,
% unless he wishes to use this m-file on non-Windows 98 machines.
% This m-file was written for MATLAB v.4 Student Edition.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function x1 = f_genx(Rch)


filename = 'genx';
n = size(Rch,2);

x1 = f_inter(filename,Rch(:,1:n-1),Rch(:,n));
% f_inter.m is a file in the folder TstMaint.
% f_inter(.) returns a column vector.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% spec.m : A sample specifications file to test the maintainability
%          algorithm.
%
% Author : Ramprasad Potluri
% E-mail : potluri@engr.uky.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

E = [-1.4462   -0.3600
     -0.7012   -0.1356
      1.2460   -1.3493
     -0.6390   -1.2704
      0.5774    0.9846];


F = [-0.0449    0.5135
     -0.7989    0.3967
     -0.7652    0.7562
      0.8617    0.4005
     -0.0562   -1.3414];


B = [0.3750   -0.3229
     1.1252    0.3180
     0.7286   -0.5112
    -2.3775   -0.0020
    -0.2738    1.6065];


T = [0.8476   -0.5571
     0.2681   -0.3367
    -0.9235    0.4152
    -0.0705    1.5578
     0.1479   -2.4443];


t = [0.5360
     0.2985
     0.2840
     0.9597
     2.0876];


rev_DB = [ 1.4915   -0.3074   -1.7071   -0.9583
          -0.7737    0.2004    0.9491    0.4061
         -16.0079   24.4836   52.8296  -38.1064
           1.4199   -0.6291   -3.2475    0.6693
```

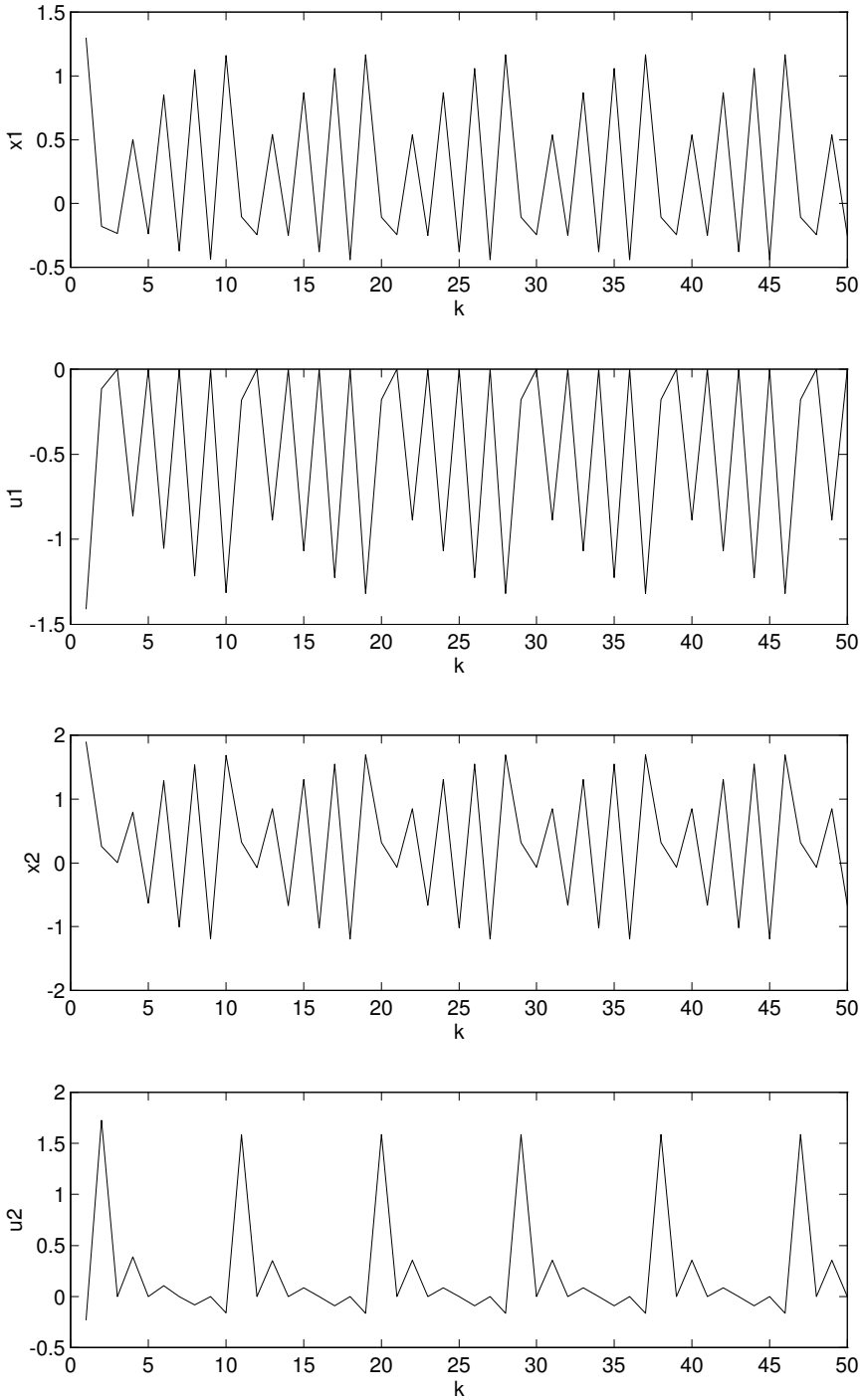
```
            3.1656   −1.0148   −5.3703   −0.3307];


Q = [0         0    0.5764         0    0.2242
     0         0    0.2273    0.0236         0
     0    1.8772         0         0    0.6802
     0    1.5828         0         0    1.8002
     0         0    0.7156    1.1639         0];


verts = [ 0.1045   −0.8032
          1.0690    0.6644
         −0.0312    0.6146
          0.0439   −0.8514
         −0.7109   −0.8971];


vertcntrls = [−1.0270    0.1189
              −0.3627    1.0515
              −0.6292    0.7197
              −1.0555    0.0805
              −1.2428   −0.1428];
```

The following figures illustrate maintaining a system in a tube. The specifications can be found in the m-file *spec.m*. $x = [x_1 \;\; x_2]^T$ is the state, and $u = [u_1 \;\; u_2]^T$ is the control.



Bibliography

[ALQS02] Jean-Pierre Aubin, John Lygeros, Marc Quincampoix, and Shankar Sastry. Impulse differential inclusions: A viability approach to hybrid systems. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, 47(1), January 2002.

[AP01] Dimitris Alevras and Manfred W. Padberg. LINEAR OPTIMIZATION AND EXTENSIONS: PROBLEMS AND SOLUTIONS. Springer-Verlag, 2001.

[APM89] Ye. S. Pyatnitskiy A. P. Molchanov. Criteria of asymptotic stability of differential and difference inclusions encountered in control theory. SYSTEM AND CONTROL LETTERS, 13:59–64, 1989.

[BR71] D. P. Bertsekas and I. B. Rhodes. On the minimax reachability of target sets and target tubes. AUTOMATICA, 7:233–247, 1971.

[Bre97] David Bremner. ON THE COMPLEXITY OF VERTEX AND FACET ENUMERATION FOR CONVEX POLYTOPES. PhD thesis, School of Computer Science, McGill University, Montréal, QC, 1997.

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. INTRODUCTION TO LINEAR OPTIMIZATION. Optimization and Neural Computation. Athena Scientific, Belmont, Massachusetts, 1997.

[CBB97] Richard J. Caron, Arnon Boneh, and Shahar Boneh. Redundancy. In Tomas Gal and Harvey J. Greenberg, editors, ADVANCES IN SENSITIVITY ANALYSIS AND PARAMETRIC PROGRAMMING, chapter 13, pages 13. 1–13. 41. Kluwer Academic Publishers, 1997.

[CDK02] M. Cannon, V. Deshmukh, and B. Kouvaritakis. Nonlinear model predictive control with polytopic invariant sets. In PROCEEDINGS OF THE 15TH TRIENNIAL WORLD CONGRESS, Barcelona, Spain, 2002. IFAC.

[Che98] Hian Suan Chew. Approximation and controllability issues of slack descriptor systems. Master's thesis, University of Kentucky, USA, November 1998.

[Fuk99] Komei Fukuda. CDD/CDD+. EPFL, Lausanne, Switzerland, http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html, 1999.

[gen95] THE STUDENT EDITION OF MATLAB: VERSION 4: USER'S GUIDE. The Math Works Inc., 1995.

[GH99] Peter Gritzmann and Alexander Hufnagel. On the algorithmic complexity of Minkowski's reconstruction theorem. JOURNAL OF THE LONDON MATHEMATICAL SOCIETY, 59(3):1081–1100, 1999.

[GR97] Jacob E. Goodman and Joseph O' Rourke, editors. HANDBOOK OF DISCRETE AND COMPUTATIONAL GEOMETRY. CRC Press, New York, 1997.

[Gre96] Harvey J. Greenberg. Consistency, redundancy, and implied equalities in linear systems. ANNALS OF MATHEMATICS AND ARTIFICIAL INTELLIGENCE, (17):37–83, 1996.

[Hol91] Lawrence E. Holloway. ON-LINE FAULT DETECTION IN DISCRETE OBSERVATION SYSTEMS USING BEHAVIORAL MODELS. PhD thesis, Carnegie Mellon University, 1991.

[Hol93] Lawrence E. Holloway. The challenge of intelligent sensing for monitoring and control. In INTERNATIONAL SYMPOSIUM ON INTELLIGENT CONTROL, pages 139–143, Chicago, Illinois, USA, 1993.

[KLTZ83] Mark H. Karwan, Vahid Lotfi, Jan Telgen, and Stanley Zionts. REDUNDANCY IN MATHEMATICAL PROGRAMMING: A STATE-OF-THE-ART SUR-

VEY. Lecture ABSTRACTs in Economics and Mathematical Sciences; 206. Springer-Verlag, Berlin, first edition, 1983.

[KS98] P. E. Kloeden and B. Schmalfuss. Asymptotic behaviour of nonautonomous difference inclusions. SYSTEMS AND CONTROL LETTERS, 33:275–280, 1998.

[Kuo95] Benjamin C. Kuo. AUTOMATIC CONTROL SYSTEMS. Prentice-Hall, 7th edition, 1995.

[KV94] A. B. Kurzhanski and V. M. Veliov, editors. MODELING TECHNIQUES FOR UNCERTAIN SYSTEMS. Birkhauser, 1994.

[Lee02] Carl Lee. Determining the values of the right-hand vector of an irredundant system of linear inequalities using linear programming duality. In Private Discussions, 2002. Department of Mathematics, University of Kentucky, Lexington, KY, USA.

[Liu99] Shuo Liu. Active sensing policy for stochastic systems. Master's thesis, University of Kentucky, 1999.

[LLH95] Hang Lian Lim and Lawrence E. Holloway. Active sensing policies for slack-descriptor systems. In 34TH CONFERENCE ON DECISION AND CONTROL, pages 2360–2365, New Orleans, LA, USA, December 1995.

[LLH96] Hang Lian Lim and Lawrence E. Holloway. Active sensing for uncertain systems under bounded-uncertainty sensing goals. In IFAC, 13TH TRIENNIAL WORLD CONGRESS, San Francisco, USA, 1996.

[LS98] Suttipan Limanond and Jennie Si. Neural-network-based control design: An LMI approach. IEEE TRANSACTIONS ON NEURAL NETWORKS, 9(6):1422–1429, November 1998.

[LW97] Vincent Loechner and Doran K. Wilde. Parametrized polyhedra and their vertices. INTERNATIONAL JOURNAL OF PARALLEL PROGRAMMING, 25(6), December 1997.

[Meg96] Alexandre Megretski. Integral quadratic constraints derived from the set-theoretic analysis of difference inclusions. In PROCEEDINGS OF THE 35TH CONFERENCE ON DECISION AND CONTROL, pages 2389–2394, Kobe, Japan, December 1996.

[PH00] Ramprasad Potluri and Lawrence E. Holloway. Control of uncertain systems represented by linear constraints: Maintainability. In AMERICAN CONTROL CONFERENCE, pages 1899–1903, June 2000.

[Raj95] Venkitaramani Rajagopal. Properties of slack-descriptor systems for modeling systems with linearly constrained uncertainty. Master's thesis, University of Kentucky, USA, 1995.

[Sch73] Fred C. Schweppe. UNCERTAIN DYNAMIC SYSTEMS. Prentice-Hall Inc., New Jersey, 1973.

[Sch86] Alexander Schrijver. THEORY OF LINEAR AND INTEGER PROGRAMMING. Discrete mathematics and optimization. Wiley-Interscience, Great Britain, 1986.

[SE89] Yeng C. Soh and Robin J. Evans. Characterization of robust controllers. AUTOMATICA, 25(1):115–117, 1989.

[SYT89] Jeff S. Shamma and Kuang Yang Tu. Set-valued observers and optimal disturbance rejection. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, (44(2)):253–264, February 1989.

[Tea02] Polylib Team. POLYLIB USER'S MANUAL. http://www.irisa.fr/polylib/DOC/document.html, April 2002.

[Wit68] H. S. Witsenhausen. Sets of possible states of linear systems given perturbed observations. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, (AC-22(3)):556–558, October 1968.

[Zie95] Günter M. Ziegler. LECTURES ON POLYTOPES. Graduate Texts in Mathematics. Springer-Verlag, New York, 1995.

Vita

Date and place of birth

February - 09 - 1970, Andhra Pradesh, India.

Educational institutions attended and degrees awarded

M.S.E.E., Saint Petersburg State Technical University, Saint Petersburg, Russia.
Specialized in Electric Drives and Industrial Automation.
Project: Electric Drive of Mechatronic Turntable.

Professional positions held

1. Spring 2003, Summer 2003.

   Temporary Faculty, Department of Electrical Engineering, University of Kentucky.

2. Fall 2002, Spring 2002, Fall 2001, Spring 2001.

   Teaching Assistant, Department of Electrical Engineering, University of Kentucky.

3. Summer 2002, Summer 2001, August 1997 – January 2001.

   Research Assistant, Department of Electrical Engineering, University of Kentucky.

4. September 1995 – January 1996, December 1994 – May 1995.

   Engineer (full time), Leningrad Machine Tool Factory, Saint Petersburg, Russia.

Scholastic and professional honors

1. July 1997.

   Awarded a full scholarship (a Research Assistantship) by the Department of Electrical Engineering, University of Kentucky, to pursue Ph.D. studies.

2. April 1996.

   Invited into the Ph.D. program of the Department of Automatic Control, Faculty of Technical Cybernetics, Saint Petersburg State Technical University, Saint Petersburg, Russia, for success at studies.

Journal submissions

1. Ramprasad Potluri and Larry Holloway. HALFSPACE REPRESENTATION OF DIFFERENCE INCLUSIONS. Submitted to SIAM Journal on Control and Optimization.

2. Ramprasad Potluri and Larry Holloway. DETERMINING THE VALUE OF THE RIGHT HAND VECTOR OF AN IRREDUNDANT SYSTEM OF LINEAR INEQUALITIES. Submitted after revision to SIAM Journal on Optimization.

Conference papers

1. Ramprasad Potluri and Larry Holloway. MODELING AND CONTROL OF HALFSPACE SYSTEMS. 35-th IEEE Southeastern Symposium on System Theory, West Virginia University, Morgantown, WV, USA, 2003.

2. Ramprasad Potluri and Larry Holloway. CONTROL OF UNCERTAIN SYSTEMS REPRESENTED BY LINEAR CONSTRAINTS: MAINTAINABILITY. American Control Conference, Chicago, IL, USA, 1999.

3. Ramprasad Potluri. ELECTRIC DRIVE OF MECHATRONIC TURNTABLE (In Russian). Master's Project. Saint Petersburg State Technical University, Russia, 1996.