Robust Simulation and Analysis of Nonlinear Systems

Thesis by

Michael James Kantner

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California
1997
Submitted May 13, 1997

To Grandfather

Abstract

For linear systems, robust analysis techniques are well developed. For nonlinear systems, they are not. Most nonlinear analysis techniques use extensive simulation to examine system performance. However, these simulations do not give guarantees, they only describe local performance.

This thesis presents a simulation technique, called robust simulation, that answers the nonlinear robust analysis question. For an uncertain nonlinear system and a set of initial conditions, robust simulation calculates the set of all possible trajectories. By applying a measure to the set of all trajectories, a performance guarantee is obtained. To allow efficient robust simulation, only discrete time piecewise linear systems are considered. This class of systems admits a wide variety of nonlinearities and can approximate generic nonlinear systems to any degree of accuracy. To measure performance, a generalized l_∞ norm is used. As in the linear case, the robust nonlinear analysis question cannot be answered exactly. Instead, upper and lower bounds are calculated. Many techniques, including traditional simulation, exist for finding lower bounds. Robust simulation provides efficient methods for calculating an upper bound.

Robust simulation also supports simulation when multiple models exist for a single system. When modeling a physical system, any amount of complexity is possible. Traditional simulation of these models with different levels of detail yields different individual trajectories. Which is correct? By explicitly quantifying the uncertainty as noise, robust simulation calculates sets of possible trajectories. For each model the result is guaranteed to contain the true output. More detailed models yield smaller sets of possible trajectories.

To test the algorithms, robust simulation is applied to a variety of examples. Algorithm performance is generally very good. Three other applications of robust simulation are also presented. In addition to measuring robust nonlinear performance, robust simulation also generates lower bounds for model predictive control optimizations, verifies the stability of piecewise linear systems, and analyzes gain scheduled systems.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Desktop workstations have greatly changed engineering and design. To reduce both time and cost, engineers now use computer simulations instead of building and testing prototypes. However, as systems become more complex, modeling and analysis quickly become more difficult.

These two areas, modeling and analysis, are central to engineering. A mathematical model of a physical system is constructed, and analysis is performed to predict some quantity. However, the analysis results are highly dependent upon the choice of model. When calculating the number of cars that can fit in a cargo ship, only the mass and volume matter. However, treating a car as just a mass and volume is useless when predicting traffic patterns.

Obviously, the choice of model must be suited to the question at hand. However, complex systems are often constructed by the interconnection of simpler systems, and it is impossible to expect the engineer to know the correct level of detail needed for each component in advance. If too much detail is included, analyses will require extensive computation time. If too little is included, the results will be meaningless.

Having multiple models for a single system introduces another concern. Generally, simulation of two different models will yield two different results. If both models describe the same plant, then this difference must be resolved, or at least understood. One approach to this problem is to quantify the uncertainties in the models and use a simulation method that explicitly accounts for the uncertainties. Instead of a single result, the simulation technique returns a set of possible trajectories. In this setting, more accurate models have less uncertainty and simulation yields a smaller set of possible trajectories.

Even if a suitable model is found, analysis is not a trivial task. In general, one would like to answer the robust analysis question:

> For a system with noise and uncertainty, what is the worst possible performance?

The answer to this question is a guarantee; nothing can be worse than it. However, for practical systems, this question cannot be answered exactly. Computational requirements grow exponentially as the problem size increases.

All approaches taken to answer the robust analysis question share two common themes. First, no attempt is made to find the exact solution. Instead, upper and lower bounds are computed and refined until the gap between the bounds is small. Second, noise descriptions, uncertainty descriptions, system representation, and performance measure are chosen to allow for computationally efficient algorithms. In other words, the model choice is driven by analysis concerns.

For linear systems, both theoretical and numerical results are well established. For example, the μ framework provides an efficient computational approach for finding bounds on the worst-case performance [41]. However, most complex systems are nonlinear and the linear frameworks do not extend to the nonlinear setting. Existing robust nonlinear analysis techniques are either computationally expensive or not applicable to general problems.

Instead of directly analyzing nonlinear a system, simulation is used to characterize its behavior. However, individual simulations only give local information about the plant. By running large numbers of simulations, a feel for the global system response is obtained, but no guarantees can be made.

Though simulation does not answer the robust analysis question, it does give a lower bound. The worst case performance of a system is at least as poor as the performance obtained from any single simulation. To obtain a good lower bound, simulation and local optimization are repeatedly applied and the worst performance is chosen. Recent work by Tierno has combined lower bound ideas from the μ framework with nonlinear simulation to achieve improved lower bounds with reasonable computations [36].

Nonlinear upper bounds have even fewer results. Most results are based upon finding Lyapunov functions. In general, there is no systematic method for finding a Lyapunov function. In the cases where numerical tools are applicable, such as linear parameter-varying (LPV) techniques, the results are often very conservative [5, 21]. The numerically tractable upper bound techniques share a common theme: they cast the nonlinear system into a specific form that is simple to analyze. Again, the analysis technique requires a special type of model.

While the goal of a general nonlinear technique is unrealistic, finding a nonlinear representation that both admits a wide class of nonlinear systems and is numerically analyzable is not. This thesis presents one such class of nonlinear systems and develops the corresponding tools to answer the robust analysis questions. This class of systems also addresses issues related to model libraries and simulation with uncertainty.

Thus, the problem of numerical nonlinear systems modeling and analysis can be broken into two parts:

1. Nonlinear system representation
2. Numerical tool implementation

Before selecting a nonlinear system form, it is instructive to examine linear system representations. Both transfer functions and state space descriptions share a common trait: the complete behavior of the system is defined by a small number of parameters. This finite data size allows the development of efficient algorithms. On the other hand, general nonlinear systems require an infinite amount of data. The flow at every point in time and state space needs to be specified. In order to allow efficient numerical tools, the nonlinear representation must have a finite and small size.

The choice of representation then drives the tool development. Representing linear systems by matrices led directly to eigenvalue and SVD analysis techniques. Likewise, the use of polynomials in transfer functions led to the use of root finding and the Routh criterion. Ideally, the nonlinear form selected will have well developed numerical tools already developed, though their meaning may not be clear.

The framework presented in this dissertation uses a piecewise linear system representation and linear programming. The analysis approach is a modification of traditional simulation ideas. System behavior is analyzed by simulating all possible trajectories at once. This technique, called robust simulation, is the natural extension of simulation to robustness analysis.

1.1 Historical Context

From a control perspective, modeling and analysis to predict system performance dates to the mid 1800s. During this period, the focus was on predicting the stability of steam engine governors. In 1868, Maxwell introduced the idea of modeling governors as linear differential equations. It was known that the stability of a linear differential equation depended upon the value of the real part of the roots of its characteristic equation, and Maxwell stated stability conditions for models up to fifth order. In 1877, Routh developed the Routh-Hurwitz stability criteria which allowed analysis of linear differential equations of arbitrary order [3]. From the beginning, modeling was driven by the ability to analyze the model.

In [4], Bode introduced the notion of robustness by defining gain and phase margin. Though not mathematically rigorous, these practical measures allowed designers to judge their margin for modeling error. In the early 1960s, Zames developed a rigorous definition of robustness and introduced the small gain theorem. Further developments focused on allowing robust analysis with more versatile descriptions of noise and uncertainty and on design to optimize robustness [40].

Extending the linear robust analysis methods to nonlinear problems has yielded mixed results. While the theory extends, the computational properties generally do not. For example, in [23], nonlinear robustness was characterized by the solution of infinite dimensional nonlinear matrix inequalities (NLMIs).

The equivalent linear problem requires solving finite dimensional linear inequalities (LMIs). While solution techniques for LMIs are well developed [13], NLMIs can only be solved in special cases.

While robustness analysis theory was advancing rapidly, modeling theory was not. In general, the linear input-output setting was, with small modifications, sufficient for analysis. However, in 1981, Sontag introduced the idea of piecewise linear modeling of nonlinear systems for control purposes [33]. Rather than extend linear analysis techniques to a nonlinear setting, a linear modeling framework was extended to a nonlinear setting. While piecewise linear analysis theory could be stated elegantly, it was not computationally practical.

Even with a well-developed linear modeling methodology, creating large linear models from the interconnection of simpler systems introduces several difficulties. Model reduction theory has formalized the creation of sets of models of varying complexity. In [22], the issue of choosing the appropriate model components for inclusion in a large system is addressed. However, the results are mainly theoretical and their computation is difficult. This idea of systematic large modeling is a new area of research and is being heavily studied at Caltech under the virtual engineering project.

The ability to construct and analyze large nonlinear systems still remains more art than science. Model creation relies on the skill and intuition of the engineer. Analysis generally consists of large numbers of simulations with randomly selected initial conditions, uncertainty, and noise. Analysis results do not give guarantees; they only give an indication of typical performance. This thesis develops a general modeling framework for which simulation gives guarantees. The tools needed for systematic model creation are also presented. It takes the art and turns some of it into science.

1.2 Contributions and Outline of This Work

This thesis takes several well-known ideas and uses them to numerically solve the nonlinear robustness problem. In the process, a systematic technique for modeling nonlinear systems is developed. This framework and its analysis techniques are inherently suited to working with model libraries. This methodology gives a mathematical foundation to the creation of complex systems from simple subsystems. While each individual idea is not new, their combination represents a large advancement over the state-of-the-art.

Chapter 2 reviews the basics of simulation and introduces the idea of robust simulation. Robust simulation is the simulation of sets through time and allows the calculation of the guarantees required for robust analysis. The modeling framework and basic algorithms needed to support robust simulation are presented in Chapter 3. As with all successful analysis techniques, the modeling framework is chosen to suit analysis. In this case, it happens to be a framework that was first studied over a decade ago, piecewise linear systems.

After establishing the modeling framework, Chapter 4 describes the robust simulation algorithm. Initially, an exact solution that exhibits exponential computational growth is developed. From this exact solution, a conservative approximation is derived. The approximation is refinable; if it is too conservative a better result can be obtained systematically. Other techniques for improving the results are also developed.

Analysis tools based upon robust simulation are developed in Chapter 5. First, a generalized nonlinear cost suited to robust simulation is presented. This cost allows more general performance measures than are available with other techniques. Robust simulation is then applied to obtain an upper bound on the worse case performance, measured in this cost. Several examples are presented.

An analysis tool is only as useful as the models it can analyze. Chapter 6 demonstrates that the piecewise linear framework has practical value. A method for converting from nonlinear to piecewise linear is presented. The piecewise linear form also gives a measure of the degree of nonlinearity of the system. This nonlinear dimension is the key measure affecting the computation time required for analysis. Finally, several common nonlinearities and uncertainties are presented in the piecewise linear framework.

Robust simulation also gives a mathematical foundation for multiresolution simulation. As shown in Chapter 7, robust simulation allows the user to change system models during a simulation and still obtain meaningful results. Since robust simulation gives the set of all possible locations in state space, more accurate models yield smaller sets of possible states. In traditional simulation, two different, unique trajectories would result. Instead, robust simulation calculates two sets of feasible trajectories.

Chapter 8 applies robust simulation and piecewise linear modeling to other problems. Robust simulation can be thought of as a general optimization tool. In this setting, it generates lower bounds for model predictive control design. In a more straightforward application, robust simulation is used to verify the stability of piecewise linear systems. Lastly, the ties between piecewise linear system modeling and traditional gain scheduling are discussed.

This thesis closes with a summary of the results and future directions. One area of future research, the extension to continuous time systems, is discussed in some detail.

Chapter 2
Simulation and Computational Complexity

Simulation is the use of a solution of equations to predict the response of a physical system. Historically, simulation has been used when other analysis techniques cannot be applied. Rather than explicitly calculate system behavior, possible trajectories are used to infer system behavior.

Many forms of simulation exist. Scale models of airfoils are tested in wind tunnels, and the results are used to predict the performance of the full sized wing. Typically, simulation refers to the numerical solution of equations on a digital computer. This notion of simulation will be used through the remainder of this thesis.

Today, simulation is still widely used. Most models of realistic systems are sufficiently nonlinear to preclude analysis. Often, design is done using linear approximations and the final design is verified by simulating the model with all nonlinearities included. The lack of analysis tools, not the benefits of simulation, drive this design procedure.

Extensive simulation tools are available. Some tools, such as SIMULINK, are general purpose tools that allow the simulation of a variety of systems. Other simulation tools are tuned to fill specific needs. However, all tools hide the mechanics of simulation from the user. While simulation has been studied extensively, it is far from fool-proof. Even the simplest simulations can yield misleading results.

This chapter begins with a brief introduction to standard simulation techniques. The ideas are well known and can be found in many texts on numerical analysis, such as [7] and [8].

2.1 Traditional Simulation

All simulation starts with a model of the physical process. Often, this model is a nonlinear partial differential equation (PDE). The solution to an PDE, denoted $u(x,t)$, is defined over the continuum of space and time. Since, except for special cases, analytic solutions cannot be obtained, numerical techniques are needed. However, numerical solutions obtained from a digital computer are fundamentally different from $u(x,t)$.

Digital computers perform operations on finite data sets. The continuum of space and time is infinite dimensional. To account for this limitation, the original problem is discretized over both space and time. Instead of calculating $u(x,t)$ over the continuum, numerical methods return the solution evaluated at a finite set of points.

Spatial discretization changes the PDE to an ordinary differential equation (ODE). Typically, finite element methods are used for the discretization. More advanced methods select the discretization adaptively. For some systems, the spatial discretization step can be avoided by using a lumped parameter approximation. This common approximation uses idealized models, such as pure resistors and massless springs, to directly construct an ODE.

In some cases, the ODE can be solved exactly. For example, the system of linear differential equations

$$\dot{x}(t) = Ax(t),\ \ x(0) = x_0$$

has the solution $x(t) = e^{At}x_0$. Thus, simulating a linear system is as difficult as computing a matrix exponential. However, computing the matrix exponential is numerically challenging. Small changes in At can lead to large changes in e^{At}. In [25], Moler and Van Loan demonstrated nineteen methods for computing the matrix exponential and their shortcomings. Even simple linear simulations can be challenging.

When the ODE cannot be solved analytically, temporal discretization is also required. Like spatial discretization, many methods exist for discretizing time. While the simplest technique, Euler's method, is rarely used, it demonstrates the general idea. Starting with the nonlinear ODE

$$\dot{x}(t) = f(t, x), \; t_0 \leq t \leq t_f, \; x(t_0) = x_0,$$

the objective is to find the solution $x(t)$. Unlike the linear case, $x(t)$ is calculated at a finite number of time samples. The behavior between time samples are approximated by interpolation.

Denoting $w(t)$ as the approximate solution obtained by simulation, Euler's method starts with $w(t_0) = x_0$ and iteratively applies the rule

$$w(t + \Delta t) = w(t) + \Delta t \times f(t, w(t)).$$

This rule is called the difference equation for Euler's method. Euler's method uses only the previous value to compute the solution's next value. During each Δt time step, the derivative, f, is frozen. If f varies quickly relative to Δt, large errors will occur.

Many other techniques have been developed to address the limitations of Euler's method. Taylor methods use higher order derivatives of f to reduce errors. Runge-Kutta methods evaluate f at several t, w pairs when finding $w(t + \Delta t)$. Multi-step methods are similar to Runge-Kutta, but use past values of w to compute the next value. Extensions to these techniques allow varying the step size Δt. When f is changing rapidly, smaller steps allow more accurate solutions.

All simulation methods return a numerical approximation to the true solution. Every method works well for some problems and poorly for others. For highly nonlinear systems, simulation results can be very misleading. For chaotic systems, small changes in initial conditions cause large changes in the solution. Errors due to finite precision computations can even affect the results. When using simulation, it is important to understand both the system being simulated and the simulation method.

2.2 Monte Carlo Analysis

The previously described simulation techniques generate a single approximate solution for a single initial condition. Since initial conditions may vary and are never known exactly, one is usually concerned with system response over a variety of initial conditions. These traditional simulation techniques also assume that the system is completely deterministic. Noise is either ignored or a specific noise signal is chosen for the simulation.

Monte Carlo methods are used to extend traditional ODE simulation techniques to uncertain systems. Consider the more general ODE

$$\dot{x}(t) = f(t, x, n), \ t_0 \leq t \leq t_f, \ x(t_0) \in \mathcal{X}_0, \ n \in \mathcal{N}.$$

This equation allows for both non-deterministic noise and uncertain initial conditions. $\mathcal{X}_0$ and $\mathcal{N}$ are sets of initial conditions and noises that define the parameter space of the simulation. To completely understand system response, the behavior at every x_0, n pair must be calculated.

For most problems, the sets $\mathcal{X}_0$ and $\mathcal{N}$ are continuums and the simulation of every x_0, n pair is impossible. For example, $\mathcal{N}$ is often the set of signals with magnitude less than some bound. Similar to the discretization of PDEs, the simulation parameter space is also discretized to allow analysis by a digital computer. In Monte Carlo analysis, the discretization points are chosen randomly. By simulating a large number of points selected from the simulation parameter space, a description of the typical response is generated.

One example of Monte Carlo analysis is the estimation of π. The area of a circle is πr^2, where $r = \sqrt{x_1^2 + x_2^2}$. By randomly selecting points in the square $|x_1| \leq 1, |x_2| \leq 1$ using a uniform distribution and counting the number of points that lie in the circle, π can be estimated. Since the square has area of 4, one would expect that $\pi/4$ of the points would lie in the circle. Table 2.1 shows the results obtained using MATLAB's uniform random generator for various numbers n of randomly chosen points. As expected, small n do not yield accurate results. However, as n increases, the approximation does not quickly converge to π. Even though this Monte Carlo simulation was constructed to

n	π
10	2.4000
100	3.0400
1000	3.1480
10000	3.1460
100000	3.1464
1000000	3.1451
exact	3.1416

Table 2.1: Monte Carlo estimation of π.

approximate π, it is only as accurate as the random number generator.

While Monte Carlo analysis is a powerful analysis tool, it is important to interpret the results properly. As previously demonstrated, results depend upon the quality of the random number generator. Even with truly random numbers, Monte Carlo analysis does not give global guarantees. The result from the analysis is a set of possible trajectories. Each individual simulation specifies the local behavior of the system. Nothing can be said about the global response. Monte Carlo analysis characterizes the expected behavior of the system. The extreme behavior is still unknown.

2.3 Robust Simulation

Robust simulation addresses the local nature of traditional simulation and Monte Carlo analysis. Instead of calculating a single trajectory, robust simulation finds a set of possible trajectories. For some volume in the simulation parameter space, robust simulation returns all possible trajectories for every point in the volume. From this single simulation, every potential result is generated. Thus, the global behavior of the system can be analyzed.

Robust simulation is the natural extension of simulation to systems with noise and uncertainty. In the linear setting, robust analysis techniques guarantee performance for sets of systems. Robust simulation takes traditional simulation ideas, and applies them to sets of nonlinear systems. While the idea is simple, its execution is not. There has been little success in robust simulation.

One recent success involves the analysis of linear hybrid systems, which have the form

$$\dot{x} = \bar{A}_i \tag{2.1}$$

where the $\bar{A}_i$ are constants defined over different regions of the state space. This type of system has a constant derivative that changes at discrete locations in state space. One example of a linear hybrid system is a billiard ball rolling without friction. The ball has constant velocity until it bounces off the side of the table. After a rebound, the velocity changes to a new constant. For this class of systems, robust simulation algorithms have been developed. The analysis techniques calculate all possible reachable states from a specified set of initial conditions [1].

However, most systems of interest cannot be accurately approximated by systems in the form of (2.1). Around equilibrium points, most systems exhibit linear behavior, not constant behavior. A piecewise linear form of

$$\dot{x} = A_i x + \bar{A}_i \tag{2.2}$$

is much more versatile. Both analytic properties and simulation techniques for this system representation are areas of active research [17, 30].

At the heart of robust simulation is the ability to work with sets of trajectories. By manipulating sets of possible trajectories, global properties can be obtained. Working with sets adds two computational issues: set representation and set mapping. The goal of this work is to present a robust simulation technique that is applicable to a wide class of systems and yields guarantees with reasonable computational cost.

In developing the robust simulation framework, it is important to focus on the added difficulty of robust simulation. All of the previously described simulation techniques require both spatial and temporal discretization. The robust simulation technique developed minimizes those concerns and focuses on the set mapping and representation issues.

Time discretization issues are avoided by considering only discrete time systems. Spatial discretization is addressed by restricting the class of nonlinear systems considered. To achieve efficient robust simulation, discrete time piecewise linear systems are used. As shown in the following chapters, this form allows easy manipulation of sets and efficient computer implementations and admits a very wide class of nonlinearities.

Chapter 3
Piecewise Linear Systems

Piecewise linear (PL) systems are a natural extension of linear system ideas to nonlinear problems. In different regions of state space, the system follows different affine state update laws. This class of systems has several beneficial features. PL systems are easy to implement and simulate, they can exhibit very complex behavior, and they facilitate nonlinear modeling. However, since analyzing PL systems is comparable to analyzing general nonlinear systems, very few analytical results exist.

Though conceptually simple, PL systems are computationally hard. Over a decade ago, Sontag suggested the use of discrete time PL systems for nonlinear regulation [33] and developed a PL algebra [34]. However, no practical computational tools were developed. Recently, Pettit and others have studied continuous time PL systems. While algorithms verifying the stability of these systems have been developed, they can require enormous computation time [30]. In fact, Sontag demonstrated that computing practically all interesting PL system properties is NP-hard [35].

One area where the piecewise linear representation has been successful is circuit analysis. Any continuous resistive nonlinear circuit can be modeled to any accuracy by a continuous piecewise linear equation of the form

$$f(x) = a + Bx + \sum_{i=1}^{\sigma} c_i |\langle \alpha_i, x \rangle - \beta_i| = 0. \tag{3.1}$$

However, not every piecewise linear function can be written in this form [9, 10]. One requirement of (3.1) is that $f(x)$ is continuous. In many control applications, including gain scheduling, $f(x)$ is inherently discontinuous. This continuity restriction makes the circuit analysis approach to piecewise linear analysis unsuitable for general use.

While analyzing general PL systems is very difficult, manipulating them is not. Traditional simulation of discrete time PL systems only requires basic matrix operations. PL systems also have very simple set mapping properties; a convex polyhedron maps to a finite set of convex polyhedra. This property is crucial to efficient robust simulation, and motivates the study of this class of systems. Furthermore, set mapping requires only basic matrix operations and the solution of linear programs.

As described in Chapter 6, PL systems also provide a framework for robustly modeling nonlinear systems and measuring their complexity. A PL system is completely described by a finite amount of data, usually represented as a set of matrices. Though their representation is simple, their behavior can be arbitrarily complex. In fact, almost every nonlinear system can be approximated arbitrarily closely by a PL system. Thus, any nonlinear system can be efficiently represented and manipulated in a digital computer. The PL model also provides a natural measure of the complexity of the system.

Though there is little hope of finding algorithms that give exact solutions to any PL problem in reasonable time, PL systems are still very useful. Almost all computations can be accomplished with widely available software tools, so algorithm development is greatly simplified. PL systems also provide a framework for a variety of complex modeling and simulation tasks. While exact solutions are always prohibitively expensive, approximate solutions are often reasonable. As shown in later chapters, approximate solutions to stability and performance problems are readily obtained.

This chapter focuses on piecewise linear systems and simple manipulations of them. It begins with the definition of a PL system and a discussion of computational complexity issues related to algorithm development. Several efficient basic algorithms for manipulating PL systems are then presented. These are then used to develop the PL system set mapping algorithm. The chapter closes with a set mapping example and a physically motivated PL system.

3.1 Definitions

By convention, all vector spaces are finite dimensional real spaces denoted $\mathcal{R}^n$, where n is the dimension of the space. A separating hyperplane is the set of all points $x \in \mathcal{R}^n$ satisfying the linear mapping $\langle d, x - p \rangle = 0$ with $d, p \in \mathcal{R}^n$ and the usual inner product. With this definition, d is a vector normal to the hyperplane and p is a point on the hyperplane. The positive closed half space is defined as the set of all points satisfying $\langle d, x - p \rangle \geq 0$. The normal vector, d, points into the closed half space.

A closed polyhedron is the intersection of a finite number of positive closed half spaces. By construction, all polyhedra are convex. A bounded closed polyhedron is one with finite volume. For a polyhedron to be bounded, it must be the intersection of at least $n+1$ half spaces (to define a simplex). All polyhedra are assumed to be closed, unless otherwise noted.

A discrete time piecewise linear system is a nonlinear system that follows different affine state update laws in different regions of the state and input space. All PL systems are assumed to be discrete time unless otherwise noted. More specifically, a PL system with n states, m inputs, and p outputs is defined over $Z \subseteq \mathcal{R}^{n+m}$ where Z is the union of a finite number, of closed polyhedra, denoted R_i, $i \in 1 \ldots l$. In region R_i, an affine map denoting the state transition is defined by

$$\begin{aligned} x[k+1] &= A_i x[k] + \bar{A}_i + B_i u[k] \\ y[k] &= C_i x[k] + \bar{C}_i + D_i u[k] \end{aligned}, \quad (x[k], u[k]) \in R_i \tag{3.2}$$

where $u[k] \in \mathcal{R}^m$ is an input vector, $x[k] \in \mathcal{R}^n$ is the state vector, and all matrices are the appropriate sizes. For simplicity, it is assumed that the regions do not intersect, except perhaps on a set of zero measure. In other words, $R_i \cap R_j, i, j \in 1 \ldots l, i \neq j$ has zero volume. While the R_i are typically connected, it is not a requirement.

This definition simplifies computer implementation, but leads to a minor technical problem. Since the polyhedra are closed and may share boundaries, the mapping is not always well defined on the boundaries. The mapping is well defined almost everywhere. One solution to this, the approach taken in [33], is to specify the polyhedra by $\langle d, x - p\rangle > 0$ and $\langle d, x - p\rangle = 0$. However, this approach greatly complicates algorithm development and implementation.

Another solution, the one used throughout this thesis, is to allow the mapping to be multivalued on the boundaries of R_i. While it is possible to create a PL system that exploits the behavior on the boundaries, this is generally not the case. For models of physical systems, the flows in each region near a common boundary are often similar. This approach also allows the removal of requirement that $R_i \cap R_j$ has zero volume for $i \neq j$. If this intersection has volume, the mapping for points in the intersection is not uniquely defined. One interpretation of this multivalued update rule is that the system behavior is chosen from a set of possible rules. However, this work does not focus on systems with non-deterministic update laws, and they will not be considered further.

Most of the algorithms developed later also apply to PL time varying systems, where both the update laws, the region boundaries, and the number of regions are functions of k. The notation $\mathcal{S}_j$ denotes a finite set of polyhedra in Z at time j.

3.2 Computational Complexity

In many problems, a formula for the solution cannot be written explicitly. Instead, the problem is considered solved when an efficient algorithm returning the solution exists. For example, many control problems can be written as the solution of LMIs [6]. Since efficient algorithms for solving LMIs have been recently developed [26], posing a problem as the solution to an LMI is equivalent to solving it.

An efficient algorithm is defined as one whose time complexity function grows polynomially with problem size. The time complexity function measures computation time as a function of problem size. A function $f(n)$ is called $\mathcal{O}(g(n))$ whenever there exists a function $g(n)$, constant c, and problem size n_0 such that $f(n)$ satisfies

$$0 \leq f(n) \leq cg(n), \ \forall n \geq n_0 \geq 0.$$

Algorithm			Size n		
Complexity	10	20	30	40	50
$\mathcal{O}(n)$	10 seconds	20 seconds	30 seconds	40 seconds	50 seconds
$\mathcal{O}(n^2)$	10 seconds	40 seconds	1.5 minutes	2.2 minutes	4.2 minutes
$\mathcal{O}(n^3)$	10 seconds	1.3 minutes	4.5 minutes	11 minutes	21 minutes
$\mathcal{O}(n^6)$	10 seconds	11 minutes	2.0 hours	11 hours	1.8 days
$\mathcal{O}(2^n)$	10 seconds	2.8 hours	4.0 months	3.4 centuries	3500 centuries

Table 3.1: Comparison of polynomial and exponential time growth rates.

When $f(n)$ is an algorithm's time complexity function and $g(n)$ is a polynomial function, the algorithm is called a polynomial time, or efficient, algorithm. When the time complexity function cannot be bounded by a polynomial, the algorithm is called an exponential time algorithm [15].

The difference between the two classes of algorithms is profound when solving large problems. Table 3.1 shows computation time as a function of problem size for several algorithm complexities. For polynomial time algorithms, computation time grows reasonably. If a given problem can be solved in seconds, a problem four times larger can typically be solved in minutes. For exponential time algorithms, this is not the case. Even if a small problem can be solved in seconds, a problem four times larger can take centuries to solve.

In order to solve problems of large size, polynomial time algorithms are required. Fortunately, a large number of problems can be solved in polynomial time. Most matrix operations, such as least squares, singular values, and eigenvalues, are $\mathcal{O}(n^3)$, where n is the larger of the number of rows or columns in the matrix [16].

However, a large class of problems, called NP-complete, cannot currently be solved in polynomial time. It is still unknown if polynomial time algorithms exist for NP-complete problems, though it is generally accepted that they do not. This class includes many well known problems, such as the traveling salesman problem and integer linear programming. Another important class of problems is those that are at least as hard as the NP-complete problems. This class, called NP-hard, consists of all problems that include NP-complete problems as special cases. A detailed discussion of NP-complete problems and other computational complexity issues can be found in [15].

The main result from this complexity analysis is that efficient algorithms cannot be found for NP-complete and NP-hard problems. Since these problems cannot be solved efficiently, approximate solutions must be obtained. In [35], Sontag demonstrated that PL analysis problems are NP-hard. Thus, PL algorithm development focuses on finding efficient, approximate solutions.

One other important requirement for efficient analysis is that the problem can be described by a polynomial amount of data. Storage requirements are as important as computation time. Problem representation is especially important when describing nonlinear systems. Describing a flow by individual values at each point in the state space is equivalent to gridding a parameter space. Gridding an n-dimensional space with k grid points in each direction requires k^n values. Small increases in problem dimension lead to unreasonable storage demands.

3.3 Basic Algorithms

Though there is little hope for finding efficient algorithms for PL analysis, inefficient algorithms are easily developed. From these exponential time algorithms, efficient approximations can then be derived. This section presents the building blocks for PL algorithm development. Each algorithm presented efficiently solves a single, simple PL problem. By piecing these algorithms together, more challenging analysis questions can be answered.

The first step in algorithm development is to define the problem representation. A polyhedron is defined by the intersection of c positive half planes, each identified by $d_i, p_i, i \in 1 \ldots c$. After rewriting the definition of a positive half

Name	Size	Description
D	$c \times n$	Vectors normal to each hyperplane bounding the region
P	$c \times 1$	Distances from each hyperplane to the origin
A	$n \times n$	Linear portion of the state update law
$\bar{A}$	$n \times 1$	Constant portion of the state update law
B	$n \times m$	Input portion of the state update law
C	$p \times n$	Linear portion of the output rule
$\bar{C}$	$p \times 1$	Constant portion of the output rule
D	$p \times m$	Feedforward portion of the output rule

Table 3.2: Matrices needed to describe one region of a piecewise linear system.

plane as $\langle d_i, x\rangle \geq \langle d_i, p_i\rangle$ and scaling the normal vectors so that $\langle d_i, d_i\rangle = 1$, the polyhedra can be stored as the two matrices

$$D = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_c \end{bmatrix}, \; P = \begin{bmatrix} \langle d_1, p_1\rangle \\ \langle d_2, p_2\rangle \\ \vdots \\ \langle d_c, p_c\rangle \end{bmatrix} \tag{3.3}$$

where $D \in \mathcal{R}^{c\times n}$ and $P \in \mathcal{R}^{c\times 1}$. This representation minimizes storage requirements and facilitates computations by not explicitly storing the p_i. However, a point on the hyperplane can always be found in $\mathcal{O}(n)$ computations, though the point might differ from the original p_i.

A PL system is merely a set of polyhedra and associated mapping laws. Thus, a PL system can be completely described by specifying the eight matrices in (3.2) and (3.3) for each of the l regions. A summary of these matrices and their sizes is included in Table 3.2. While "D" represents two different matrices, its meaning will always be clear from context. Assuming that $c \sim n$, which is generally true since for simplexes $c = n + 1$ and for hyperrectangles $c = 2n$, and that the system has a reasonable number of inputs and outputs, storage requirements are $\mathcal{O}(ln^2)$.

There are two important features of this representation. First, the complete nonlinear behavior is specified by a finite amount of data. Second, the data size grows polynomially with respect to all system parameters. As the state dimension, grows, storage requirements grow quadratically. As the system becomes more nonlinear (l grows), storage requirements grow linearly. Systems with dozens of states and hundreds of regions can be stored on home computers.

Since this representation consists solely of matrices, most operations on polyhedra and PL systems are naturally written as matrix algebra and solutions to linear programs. Linear programming is an optimization technique that solves

$$\begin{gathered}\min c^t x \\ Dx \geq P\end{gathered}$$

where $Dx \geq P$ is an element-by-element inequality. The set $Dx \geq P$ is identical to the polyhedron definition (3.3). Several techniques exist for solving linear programs. Classical techniques, such as the simplex method, generally require few computations, but exhibit exponential computational growth for worst case problems. The simplex algorithm solves most problems with $\mathcal{O}(n^4)$ computations, though some problems require $\mathcal{O}(n^3 2^n)$ computations. In 1984, Karmarkar introduced an effective interior-point method which calculated the solution in polynomial time [20]. Continued development has led to primal-dual algorithms which solve linear programs with $\mathcal{O}(n^6)$ computations, assuming that the number of constraints in D grows linearly with n. Many primal-dual algorithms are generally much faster than $\mathcal{O}(n^6)$ and are comparable to the typical performance of the simplex algorithm [39].

The following six algorithms are some of the most basic PL system operations. They are used to develop more complicated analysis techniques. The computational complexity of each algorithm is presented as either an operation count or the number of linear program solutions needed. For the complexity measures, it is assumed that $c \sim n$. For all operations, storage requirements are at worst $\mathcal{O}(ln^2)$.

```
Function value=Inside($p$,$P$,$D$)

$P' := D \times p$
For $i$ from 1 to $c$
    If $(P'(i,1) < P(i,1))$
        value = false
        return
value = true
```

Table 3.3: Region identification pseudocode.

Region identification

In order to do any work with PL systems, one must determine if point p lies in polyhedra R. Fortunately, verifying this requires only one matrix multiplication and an element-by-element comparison of two vectors. This calculation, whose pseudocode is given in Table 3.3, requires $\mathcal{O}(n^2)$ operations.

Emptiness of polyhedra

Given a polyhedra defined by D and P, is the polyhedra empty? If there exists a point p such that element-by-element comparison $D \times p \geq P$ is true, then the polyhedra is not empty since p is inside the polyhedra. Finding this p is equivalent to finding a feasible point of a linear program.

Redundant constraints

Identifying redundant constraints in a polyhedra requires finding the optimal solution to a set of linear programs. Given a non-empty polyhedra with c constraints, the object is to find a new representation, D', P', that defines the same volume with fewer constraints. This is accomplished by removing redundant constraints from the original D, P.

The idea of the algorithm, shown in Table 3.4, is to compare the original polyhedra to a new polyhedra D_w, P_w, that has one constraint, $\hat{d}, \hat{p}$, removed. The two polyhedra are identical if

$$\min_{x, D_w \times x \geq P_w} \langle \hat{d}, x \rangle \geq \hat{p}.$$

```
Function $[D', P']$=Redundant($P$,$D$)

$D' = D$ $P' = P$ For $i$ from $c$ to 1
    $D_w = D'$ with constraint $i$ removed.
    $P_w = P'$ with constraint $i$ removed.
    $p = \min_{x, D_w \times x \geq P_w} D'(i,:) \times x$
    If $(p \geq P(i,1))$
        Note: the constraint is redundant
        $D' = D_w$
        $P' = P_w$
```

Table 3.4: Redundant constraint removal pseudocode.

The solution to this linear program, denoted p_{opt}, is the point in the new polyhedra that is as far as possible in the direction opposite the constraint. If $p_{opt} < \hat{p}$, then the optimal solution is a point in the new polyhedra, but not in the original polyhedra, implying that the constraint is not redundant. However, if $p_{opt} > \hat{p}$, then the constraint was not active in the original polyhedra, and can be removed without affecting the volume of the polyhedron.

If the constraint is redundant, then the constraint is removed and the polyhedra D_w, P_w is used for remaining calculations. This process is repeated for each constraint in the polyhedra. The whole calculation requires solving c linear programs.

Traditional simulation

Simulating a PL system also only requires basic matrix operations. However, since the update law may be multivalued at some points, the implementation of the algorithm can affect the results. At each time step, the first task is to choose the state update law. The simplest method for this is to search for regions that contain the current point, and stop after the first region is found. When the regions are searched in a deterministic order, this approach also makes the system deterministic. As implemented in Table 3.5, the regions R_i are searched sequentially and the simulation process is repeatable. A more complex method could find all valid update laws and choose between them

```
Function [states,output]=Simulate($x_0$, $t$, System, u)

states = [ ]
output = [ ]
For $t'$ from 0 to $t$
    $r = 0$
    For $l'$ from 1 to $l$
        If Inside($X(t'), P_{l'}, D_{l'}$)
            $r = l'$
            Break
    If ($r == 0$)
        Error: Point outside of PL system
        Break
    states = [states; $x_c$];
    output = [output; $C_r x_c + \bar{C}_r + D_r u(t')$]
    $x_c = A_r x_c + \bar{A}_r + B_r u(t')$
```

Table 3.5: Traditional simulation pseudocode.

according to some rule. Once the update law is chosen, the law is applied and the process repeats for the next time step. Simulation for t time steps is $\mathcal{O}(tln^2)$.

Intersection of polyhedra

The intersection of two polyhedra is the intersection of the positive half spaces that define each of the polyhedra. For two polyhedra defined by D^1, P^1 and D^2, P^2, the intersection is also a polyhedra and can be written as

$$D = \begin{bmatrix} D^1 \\ D^2 \end{bmatrix}, \; P = \begin{bmatrix} P^1 \\ P^2 \end{bmatrix}.$$

After forming the intersection, it is necessary to check if the intersection is empty and if any constraints are redundant. Assuming that forming the intersection requires copying D^1, P^1 and D^2, P^2 into a new polyhedron, this algorithm is $\mathcal{O}(n^2)$. Under certain circumstances when copying is not required, the intersection can be formed in constant time.

Bounding polyhedra

Though the union of polyhedra is not a polyhedron, the union can be bounded by one. Given a set of j polyhedra denoted D^k, P^k, the objective is to find a polyhedron D, P that satisfies

$$D^k, P^k \subseteq D, P, \ k \in 1 \ldots j.$$

For the bounding polyhedron, the choice of D is arbitrary. In many cases, one would like the convex hull of the D^k, P^k. However, the computations required to find the convex hull of a set of arbitrary polyhedra grows exponentially. Typically, D is chosen to be a hypercube with additional faces. The added hyperplanes can be taken from the D^k or chosen by other means.

Once D is chosen, linear programming is used to find P. Each element of P is given by

$$\langle d_i, p_i \rangle = \min_{k \in 1 \ldots j} \min_{x \in D^k, P^k} \langle d_i, x \rangle.$$

If D has c constraints, finding the bounding polyhedron requires solving cj linear programs.

3.4 Mapping of Sets

The key to robust simulation is the ability to map sets efficiently, and PL systems provide this. A PL system maps a polyhedron to a set of polyhedra with reasonable computational cost. This is because the affine mapping of a polyhedron is a polyhedron.

Affine maps of polyhedra

The first step in deriving a PL mapping algorithm is calculating the mapping of a single polyhedron. Given a polyhedra D, P and an affine map

$$\hat{x} = Ax + \bar{A}, \tag{3.4}$$

find the new, mapped polyhedra $\hat{D}, \hat{P}$.

The mapping algorithm is derived from the definition of the polyhedron. For any hyperplane bounding the polyhedron,

$$\langle d_i, x \rangle \geq \langle d_i, p_i \rangle$$

is satisfied for $x = p_i + d_i^{\perp} + \epsilon d_i$ where $\langle d_i, d_i^{\perp} \rangle = 0$ and $\epsilon \geq 0$. When $\epsilon = 0$, $\langle d_i, x \rangle = \langle d_i, p_i \rangle$ and when $\epsilon > 0$, $\langle d_i, x \rangle > \langle d_i, p_i \rangle$. Applying (3.4) to x yields

$$\langle \hat{d}_i, Ap_i + Ad_i^{\perp} + \epsilon Ad_i + \bar{A} \rangle \geq \langle \hat{d}_i, Ap_i + \bar{A} \rangle, \ \epsilon > 0$$

which reduces to

$$\langle \hat{d}_i, Ad_i^{\perp} \rangle + \epsilon \langle \hat{d}_i, Ad_i \rangle \geq 0, \ \epsilon > 0.$$

This is solved by setting

$$\hat{d}_i = \alpha A^{+t} d_i, \ \alpha > 0$$

where A^{+t} is the transpose of the pseudo-inverse. Since $\langle \hat{d}_i, \hat{d}_i \rangle = 1$ is required, α is chosen such that

$$\hat{d}_i = \frac{A^{+t} d_i}{\|A^{+t} d_i\|_2} \tag{3.5}$$

This step assumes that $\|A^{+t} d_i\|_2$ is nonzero. The case $\|A^{+t} d_i\|_2 = 0$ is discussed later.

When A is invertible, each element of $\hat{P}$ can be calculated directly by the formula $\langle \hat{d}_i, \hat{p}_i \rangle = \langle d_i, p_i \rangle + \langle \hat{d}_i, \bar{A} \rangle$. However, when A is singular, this formula does not hold. For noninvertible A, $\hat{P}$ is constructed through linear programming. Each $\langle \hat{d}_i, \hat{p}_i \rangle$ is the solution to the linear program

$$\langle \hat{d}_i, \hat{p}_i \rangle = \min_{Dx \geq P} \hat{d}_i^t Ax + \langle \hat{d}_i, \bar{A} \rangle. \tag{3.6}$$

This linear program is also valid for invertible A, but the direct calculation is far more efficient.

Two other changes are required when A is not invertible. If any d_i are in the null space of A, (3.5) is undefined. Those d_i in the null space A do not constrain $\hat{D}, \hat{P}$ and are omitted. Additional hyperplanes are also added to

$\hat{D}, \hat{P}$ to account for the left null space of A. Denoting each basis vector of the left null space of A by $\mathcal{N}_i(A^t)$, the mapped volume is given by

$$\hat{D} = \begin{bmatrix} \hat{d}_1 \\ \hat{d}_2 \\ \vdots \\ \hat{d}_{c'} \\ \mathcal{N}_1(A^t) \\ -\mathcal{N}_1(A^t) \\ \vdots \\ \mathcal{N}_r(A^t) \\ -\mathcal{N}_r(A^t) \end{bmatrix}, \quad \hat{P} = \begin{bmatrix} \langle \hat{d}_1, \hat{p}_1 \rangle \\ \langle \hat{d}_2, \hat{p}_2 \rangle \\ \vdots \\ \langle \hat{d}_{c'}, \hat{p}_{c'} \rangle \\ \langle \mathcal{N}_1(A^t), \bar{A} \rangle \\ \langle -\mathcal{N}_1(A^t), \bar{A} \rangle \\ \vdots \\ \langle \mathcal{N}_r(A^t), \bar{A} \rangle \\ \langle -\mathcal{N}_r(A^t), \bar{A} \rangle \end{bmatrix} \tag{3.7}$$

where r is the rank of the left null space of A and $d_i, p_i,\ i \in 1 \dots c'$ are the hyperplanes whose normal vectors are not in the null space of A. Though the mapping was derived for square A, (3.5), (3.6), and (3.7) apply to arbitrary affine maps.

When A is invertible, computing the affine map of a polyhedra is $\mathcal{O}(n^3)$. When A is not invertible, $\mathcal{O}(n)$ linear programs must be solved. If the same mapping is applied repeatedly, the pseudo-inverse and null spaces can be computed once and stored for future use. These calculations require finding the singular value decomposition (SVD), which is also $\mathcal{O}(n^3)$. Storing these intermediate calculations greatly reduces total computations, since calculating the SVD is roughly as expensive as the matrix multiplications needed to form $\hat{D}$ and $\hat{P}$ for invertible A.

Uncertain affine maps of polyhedra

The second step in deriving a PL mapping algorithm is calculating the mapping of a single polyhedron D^x, P^x subject to an additional offset. This offset is unknown, but is contained in the polyhedron D^u, P^u. The result from this mapping is the set of all possible mappings of D^x, P^x for any input u in D^u, P^u. Geometrically, this mapping is the set mapping of $Ax + \bar{A}$ convolved with the set Bu. At each point in the set $Ax + \bar{A}$, a new set is formed by adding all points in the set Bu. Intuitively, this can be thought of as a noisy set mapping. The result consists of all points reachable by the mapping for any allowable noise.

Specifically, given two polyhedra, $D^x, P^x \subset \mathcal{R}^n$ and $D^u, P^u \subset \mathcal{R}^m$ and the affine map

$$\hat{x} = Ax + \bar{A} + Bu, \tag{3.8}$$

find the new polyhedron $\hat{D}, \hat{P}$ that consists of all $\hat{x}$ that satisfy this mapping for $x \in D^x, P^x$ and $u \in D^u, P^u$. This mapping can be rewritten as a new mapping

$$\hat{x} = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + \bar{A} \tag{3.9}$$

of the augmented polyhedra

$$D = \begin{bmatrix} D^x & 0 \\ 0 & D^u \end{bmatrix}, \quad P = \begin{bmatrix} P^x \\ P^u \end{bmatrix}.$$

Written in this form, the uncertain affine polyhedral mapping is a direct application of the previously described affine polyhedral mapping.

Inverse affine maps of polyhedra

Given a polyhedra $\hat{D}, \hat{P}$ and a mapping of the form (3.4) it is possible to find the set D, P that map into $\hat{D}, \hat{P}$. Essentially, finding the inverse affine map is accomplished by running the affine map algorithm in reverse. The same ideas apply and the inverse mapping rules are

$$d_i = \frac{A^t d_i}{||A^t d_i||_2}, \tag{3.10}$$

$$\langle d_i, p_i \rangle = \min_{\hat{D}(Ax+\bar{A}) \geq \hat{P}} d_i^t Ax, \tag{3.11}$$

and

$$D = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{c'} \end{bmatrix}, \quad P = \begin{bmatrix} \langle d_1, p_1 \rangle \\ \langle d_2, p_2 \rangle \\ \vdots \\ \langle d_{c'}, p_{c'} \rangle \end{bmatrix} \tag{3.12}$$

where r is the rank of the null space of A and d_i, p_i, $i \in 1 \dots c'$ are the hyperplanes whose normal vectors are not in the left null space of A. The structure of these equations is similar to that of the forward mapping, but has one key difference. In (3.7), D is augmented with direction in the left null space of A. These vectors correspond to directions that can never be reached through the mapping. Conversely, any direction in the null space of A is unconstrained in the inverse mapping. Those vectors for which (3.10) is undefined are omitted from (3.7).

For uncertain affine maps, this algorithm returns the set of all x, u that can map into $\hat{D}, \hat{P}$. Accounting for input constraints requires intersecting D, P with the input constraints D^u, P^u. Using the notation from (3.9) and denoting the constrained polyhedra by D', P', this is given by

$$D' = \begin{bmatrix} D \\ 0 \ D^u \end{bmatrix}, \quad P' = \begin{bmatrix} P \\ P^u \end{bmatrix}.$$

As when forming the intersection of any polyhedra, it may be necessary to check for redundant constraints.

The inverse uncertain affine mapping has a different meaning than the forward mapping. The forward mapping is the set of all points that can be reached for $x \in D^x, P^x$ and $u \in D^u, P^u$. The inverse mapping returns the set of x such that there is some u that maps x into $\hat{D}, \hat{P}$. For many x, the mapping may lie outside of $\hat{D}, \hat{P}$ for some u.

Calculating the inverse mapping is computationally identical to finding the forward mapping.

Piecewise linear system set mappings

Once the affine mapping of a single polyhedra is defined, the algorithm for PL system set mapping is straightforward. Since a polyhedra may lie in several regions of a PL system, different portions of it are subject to different affine maps. One simply has to calculate the mapping of the polyhedra restricted to each region of the PL system.

The result of the mapping of a single polyhedron is a set of at most l

```
Function [Sets]=Simulate($P$, $D$, System)

Sets = [ ]
For $l'$ from 1 to $l$
    [$P_i$,$D_i$] = Intersect($P$,$D$, $R_{l'}$)
    If (Empty($P_i$,$D_i$)) continue.
    [$\hat{P}_i$,$\hat{D}_i$] = Mapping($P_i$,$D_i$,$A_{l'}$,$\bar{A}_{l'}$)
    Sets = [Sets; $\hat{P}_i$,$\hat{D}_i$]
```

Table 3.6: PL system set mapping pseudocode.

polyhedra. For notational simplicity, consider the set mapping of a piecewise linear system with no inputs. As shown in Table 3.6, the first step of the algorithm is to determine if the original polyhedra intersects a region of the PL system. If the intersection is non-empty, then the mapping is calculated. This process is repeated for each of the l regions in the system.

Computing the PL system map requires forming l intersections of polyhedra and mapping them. Both checking for redundant constraints and the mapping step require the solution $\mathcal{O}(n)$ linear programs. Thus, the algorithm as a whole requires solving $\mathcal{O}(ln)$ linear programs.

3.5 Examples

The following two examples demonstrate many of the algorithms developed. The first example illustrates the differences between the forward and inverse affine maps. Several polyhedral manipulations are described in detail. The second example demonstrates how PL systems arise in practice and examines some PL properties. Both examples demonstrate ideas central to robust simulation, the topic of the next chapter.

A noninvertible affine map and its inverse

To demonstrate the affine set mapping properties, consider the mapping

$$y = \begin{bmatrix} 1 & 1 \end{bmatrix} x + 1$$

of the polyhedron D^x, P^x given by $\|x\|_\infty \le 1$. Written in the form of (3.3),

$$D^x = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}, \; P^x = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}.$$

Applying the forward mapping algorithm given by (3.5), (3.6), and (3.7) to D^x, P^x yields

$$D^y = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \; P^y = \begin{bmatrix} -1 \\ -3 \end{bmatrix}.$$

which can also be written as $-1 \le y \le 3$. For this mapping, A^{+t} has full row rank so no additional constraints are added to D^y, P^y. The inverse mapping of D^y, P^y denoted $D^{x'}, P^{x'}$ is

$$D^{x'} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}, \; P^{x} = \begin{bmatrix} -\sqrt{2} \\ -\sqrt{2} \end{bmatrix}.$$

As shown in Figure 3.1, $D^x, P^x \subset D^{x'}, P^{x'}$. $D^{x'}, P^{x'}$ is the set of all points that map into D^y, P^y while D^x, P^x is a set of points that maps into D^y, P^y. Since A does not have full rank, these set differ.

A simple PL system

Piecewise linear systems arise naturally from even the simplest nonlinearities. In Chapter 6, a catalog of common PL nonlinearities is presented. In this example, one of the most common nonlinearities, a saturation, is interconnected with a linear system and properties of the resulting PL system are examined.

The unstable linear system with saturating input

$$\begin{aligned} x[k+1] &= \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} x[k] + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{sat}(u[k]) \\ y[k] &= x[k] \end{aligned}$$

$$\text{sat}(u) = \begin{cases} -1 & \text{if } u < -1, \\ u, & \text{if } -1 \le u \le 1, \\ 1, & \text{if } u > 1 \end{cases}$$

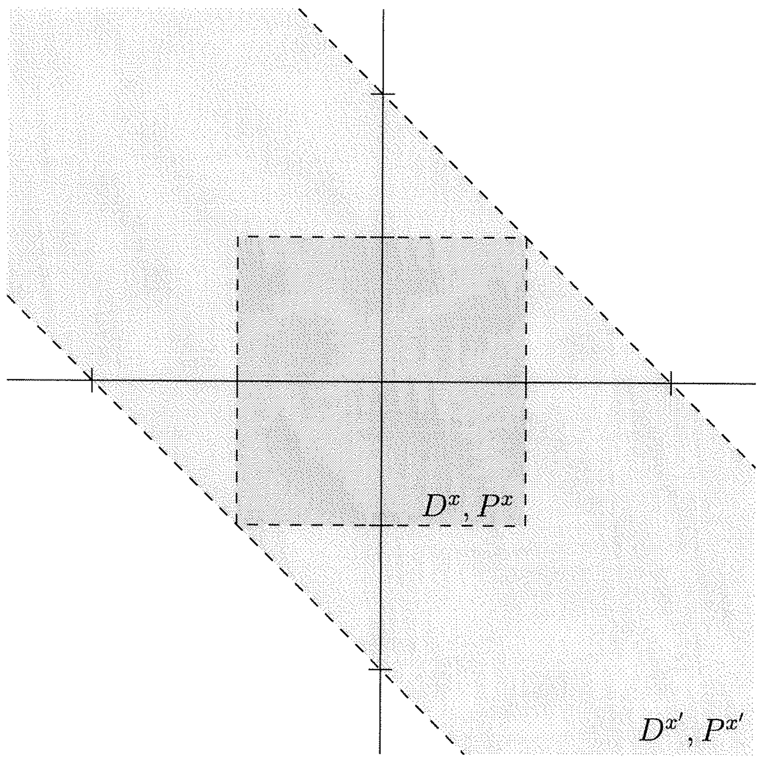


Figure 3.1: Original region and inverse mapped region.

is an unstable PL system with $l = 3$. One might attempt to stabilize the system by ignoring the saturation and designing a controller for the resulting linear system. One such controller,

$$u[k] = -\begin{bmatrix} 0.5 & 1.5 \end{bmatrix} y[k]$$

yields the closed loop system

$$x[k+1] = \begin{cases} \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} x[k] + \begin{bmatrix} 0 \\ 1 \end{bmatrix}, & \begin{bmatrix} 0.5 & 1.5 \end{bmatrix} x[k] < -1 \quad (R_1) \\ \begin{bmatrix} 1 & 2 \\ -0.5 & -0.5 \end{bmatrix} x[k], & -1 \leq \begin{bmatrix} 0.5 & 1.5 \end{bmatrix} x[k] \leq 1 \quad (R_2) \\ \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} x[k] + \begin{bmatrix} 0 \\ -1 \end{bmatrix}, & \begin{bmatrix} 0.5 & 1.5 \end{bmatrix} x[k] > 1 \quad (R_3) \end{cases} \tag{3.13}$$

$$y[k] = x[k]$$

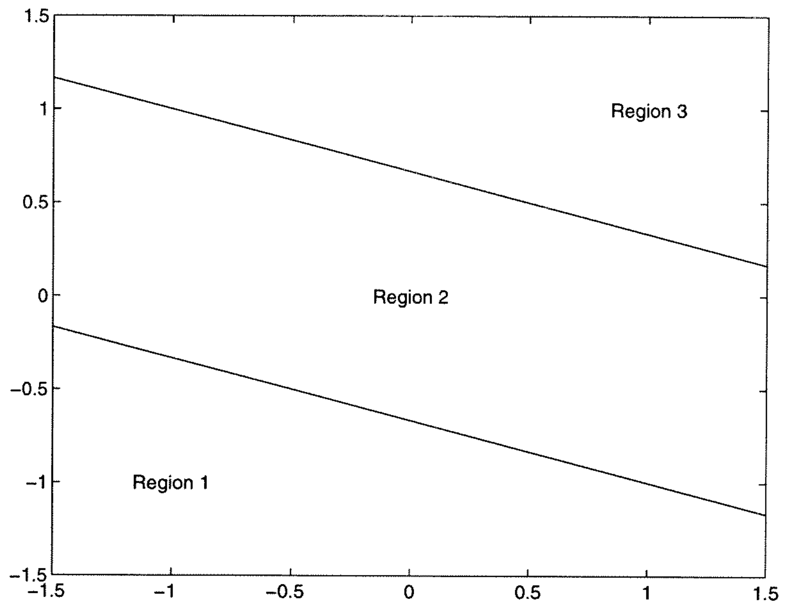


Figure 3.2: Piecewise linear regions.

The closed loop PL system also has three regions, as shown in Figure 3.2. While the open-loop system's regions were aligned with the coordinate axis corresponding to u, that is not the case for the closed-loop system. For this example, a coordinate transformation could realign the region boundaries with the coordinate axes. In general, realignment is not possible.

Due to the nature of the piecewise linear nonlinearities, there are an infinite number of controllers that yield the same close-loop system. For example, the controller

$$u[k] = \begin{cases} -1, & \begin{bmatrix} 0.5 & 1.5 \end{bmatrix} x[k] > 1 \\ -\begin{bmatrix} 0.5 & 1.5 \end{bmatrix} y[k], & -1 \leq \begin{bmatrix} 0.5 & 1.5 \end{bmatrix} x[k] \leq 1 \\ 1, & \begin{bmatrix} 0.5 & 1.5 \end{bmatrix} x[k] < -1 \end{cases}$$

also gives the same closed loop equations. Another feature, which will be shown in Chapter 6, is that the interconnection of PL systems is a PL system.

Traditional simulation of the system, shown in Figure 3.3, is straightforward; one first determines the applicable state update rule and then applies it. For example, the initial condition $x[0] = [-1 - 1.5]^t$ is in region 1 and follows the update rule in (3.13). Thus, $x[1] = [-4 - 0.5]^t$. This trajectory, shown for

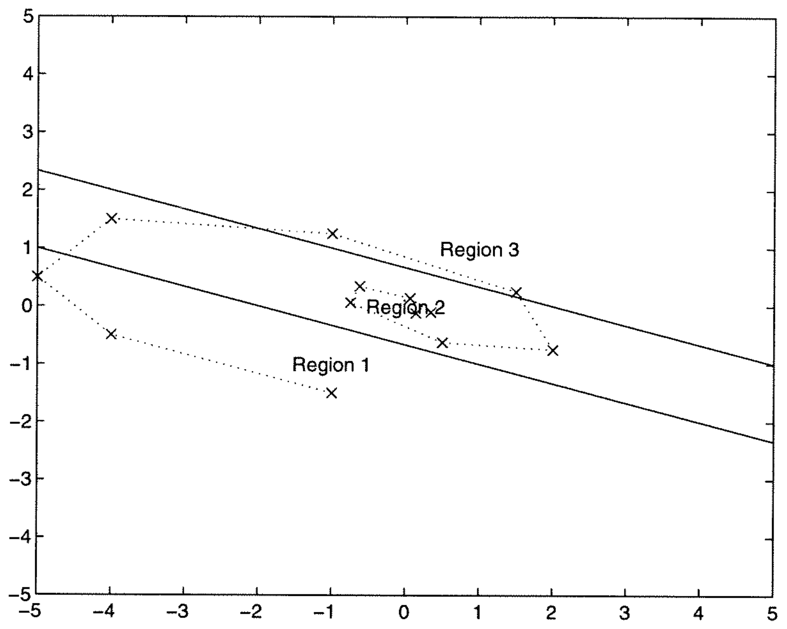


Figure 3.3: Traditional simulation of a PL system.

13 time steps, traverses all of the regions before settling in region 2. Though this system converges to the origin exponentially, stability issues will not be addressed until Chapter 8.

To complete the example, a PL set mapping is demonstrated. The initial set $\mathcal{S}_0$, shown by dotted lines in Figure 3.4, is the square $|x|_\infty \leq 1$. Since $\mathcal{S}_0$ lies in all three regions, its mapping, denoted $\mathcal{S}_1$, is a set of three polyhedra. The set $\mathcal{S}_0 \cap R_1$ is a triangle and maps to the shifted and stretched triangle shown by a dashed line. The set $\mathcal{S}_0 \cap R_2$ is a parallelogram and maps to the parallelogram shown by the dash-dotted line. The set $\mathcal{S}_0 \cap R_3$ is a triangle and maps the the stretched triangle shown by a dash-dotted line.

In this example, $\mathcal{S}_0$ maps to a connected set of polyhedra. This occurs because the state update law is continuous along switching boundaries. In general, the state update law is discontinuous along the boundaries and the mapping results in polyhedra that do not share boundaries.

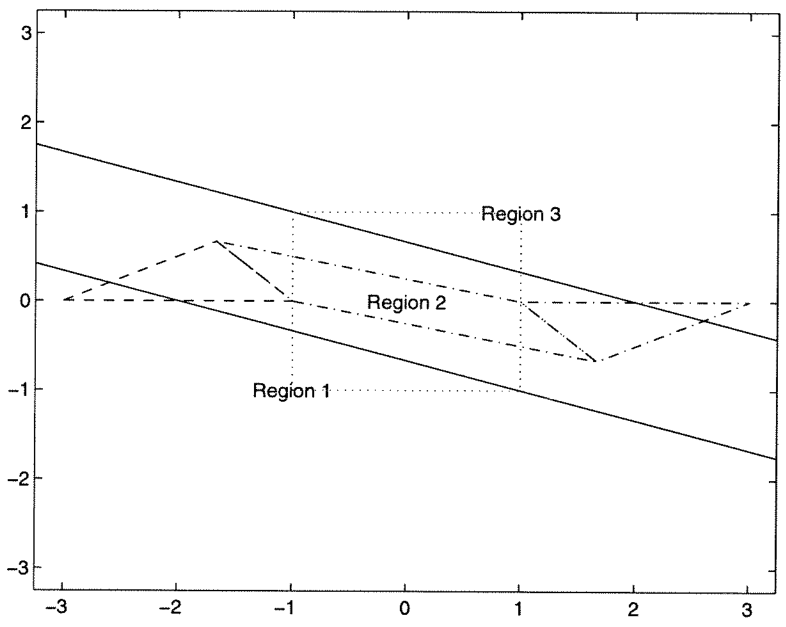


Figure 3.4: Piecewise linear system set mapping.

Chapter 4 Robust Simulation

As introduced in Chapter 2, robust simulation is the extension of traditional simulation ideas to systems with uncertainty and noise. While traditional simulation maps a single initial condition and noise signal into a unique trajectory, robust simulation generates all possible trajectories for a set of initial conditions and noise signals. By calculating all possible trajectories, many problems can now be solved through simulation. In Chapter 5, robust simulation is used to obtain nonlinear performance guarantees. In Chapter 7, meaningful results are obtained when simulating multiple models of the same system. These types of questions cannot be answered with traditional simulation techniques.

The key difference between robust simulation and traditional simulation is the mapping of sets. In fact, in its simplest form, robust simulation can be viewed as an exercise in set manipulation. As time flows, regions of state space are propagated into new regions of state space, and the results are stored. While intuitively simple, the process is technically daunting. Both system description and set representation play large roles in determining the efficiency of the algorithms.

Two commonly used set descriptions are both compact and easy to manipulate. Ellipsoids can be represented by a point and a matrix. Convex polyhedra can be represented by a list of hyperplanes. Both of these representations generally require $\mathcal{O}(n^2)$ numbers to describe a set. Other common representations, such as defining a region by its vertices, can require huge amounts of resources. For example, defining a hypercube by its vertices requires $\mathcal{O}(n2^n)$ storage.

The system representation is also critical to algorithm success. Ideally, set representation is preserved after a set mapping. For example, linear systems map ellipsoids to ellipsoids and polyhedra to polyhedra. Nonlinear systems map ellipsoids into arbitrary shapes. As previously shown, discrete time piecewise linear systems map polyhedra into sets of polyhedra.

The choice between continuous and discrete simulation is also important. On digital computers, continuous simulations are actually discrete simulations with small time steps. The choice of time step size plays a large role in the accuracy of the simulation. Various schemes exists for selecting time step size. Some give a uniform time step, while others adaptively choose and appropriate step size. By considering only discrete simulations, this issue is removed.

In the following, robust simulation algorithms are developed for discrete time piecewise linear systems. As shown in Chapter 6, this class of systems allows a very rich class of nonlinearities, yet has attractive representation and set mapping properties. With this choice, one can focus on set mapping, which is the key difference between robust simulation and traditional simulation.

This chapter begins by defining the robust simulation problem. From the definition, an exact, exponential time algorithm is constructed. Next, the exact algorithm is used to derive efficient approximations. Methods for improving the approximation are then presented. The chapter closes with an example that demonstrates the exact algorithm and the conservatism added by the approximate algorithm.

4.1 Problem Statement

Robust simulation answers the following question:

> For a set of initial conditions and noise signals, what is the set of all reachable states?

Specifically, for a discrete time piecewise linear system of the form (3.2), with initial conditions in the set of bounded polyhedra $\mathcal{S}_0$ and inputs in the time-varying convex polyhedron $\mathcal{U}[k]$, what is the set of all possible final conditions after f time steps, denoted $\mathcal{S}_f$?

By solving this problem at each time t, $0 < t \leq f$, the set of all possible trajectories is calculated. Though the initial condition set $\mathcal{S}_0$ is allowed to be a set of polyhedra, it may also be a unique point. While $\mathcal{U}[k]$ allows time-varying input constraints, often it is a ball in l_∞.

By robustly simulating in reverse time, the following question can also be answered:

> For a set of final conditions and noise signals, what are all possible initial states that can reach this set?

This is equivalent to the previous problem, except time flows backwards. For a given final condition set $\mathcal{S}_f$, $\mathcal{S}_0$ is calculated to be the set of all points x for which there exists a $u[k] \in \mathcal{U}[k]$ that maps x into $\mathcal{S}_f$ in f time steps. As shown later, solving the reverse time problem is nearly identical to solving the forward time problem.

The notation $\mathcal{S}_k$ denotes a set of convex polyhedra at time k. Some of these polyhedra may have zero volume, or even be individual points. $\widehat{\mathcal{S}_k}$ is an approximation such that $\mathcal{S}_k \subseteq \widehat{\mathcal{S}_k}$. T denotes a single polyhedron in the set $\mathcal{S}_k$. The mapping of a single polyhedron restricted to region i of the PL system, $T \cap R_i$, is called T_i. $T_{j,i}$ is the mapping of $T_i \cap R_j$ by the region j's update rule.

4.2 Exact Solution

Calculating $\mathcal{S}_f$ from $\mathcal{S}_0$ requires the repeated calculation of $\mathcal{S}_{k+1}$ from $\mathcal{S}_k$. This is a direct application of the PL system set mapping algorithm derived in Section 3.4. At each time step, the set of all reachable states is found. This set is then used to find the set of all reachable states at the next time step. Robust simulation in reverse time, which requires calculating $\mathcal{S}_{k-1}$ from $\mathcal{S}_k$, is identical except the PL system inverse set mapping algorithm from Section 3.4 is used.

While the exact solution can be computed, it has exponential growth. In one time step, a PL system can map a single polyhedron into a set of l polyhedra. Assuming that $\mathcal{S}_0$ is a single polyhedron, $\mathcal{S}_1$ may contain l polyhedra.

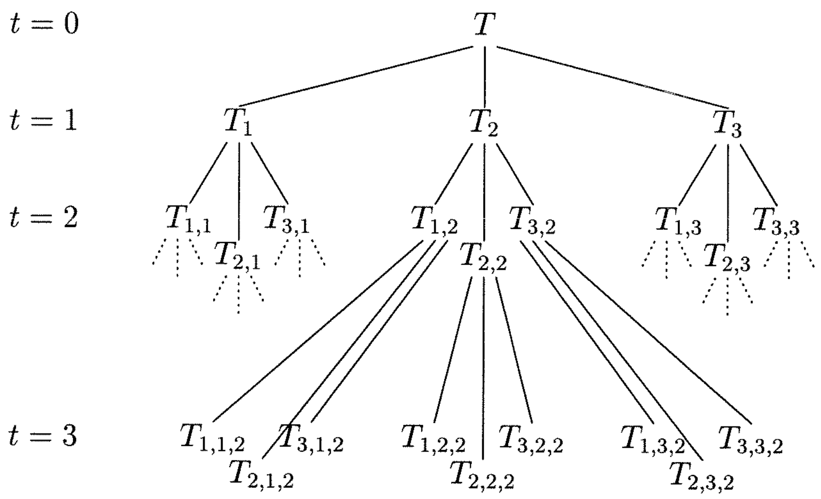


Figure 4.1: Exact robust simulation algorithm example.

Thus, $\mathcal{S}_2$ can contain up to l^2 polyhedra, and in general, $\mathcal{S}_k$ can contain l^k polyhedra.

An example of this behavior for a PL system with 3 regions is depicted in Figure 4.1. $\mathcal{S}_0$ consists of a single polyhedron, T, that intersects all 3 regions of the PL system. In one time step, each of the $T \cap R_i$ maps to a new polyhedron, yielding 3 polyhedra at time $t = 1$. Each of the three polyhedra in $\mathcal{S}_1$ also intersects all regions of the PL system. Repeating the mapping step gives $\mathcal{S}_2$ which contains 9 polyhedra. Assuming that each polyhedron in $\mathcal{S}_2$ intersects all regions of the system, $\mathcal{S}_3$ contains 27 polyhedra.

In general, exponential growth is unacceptable. A practical algorithm must be able to solve a slightly larger problem with a reasonable growth in computation time. For robust simulation of PL system with l regions and n states over t time steps, computational cost is $\mathcal{O}(l^t n^5)$ and storage requirements are $\mathcal{O}(l^t n^2)$.

4.3 Approximate Solution

To avoid exponential growth, an approximate solution is derived that both returns all possible trajectories and has polynomial computational growth. In the exact solution, the exponential growth is caused by the increasing number of polyhedra in $\mathcal{S}_k$. Rather than allowing l^k polyhedra in $\mathcal{S}_k$, the number of

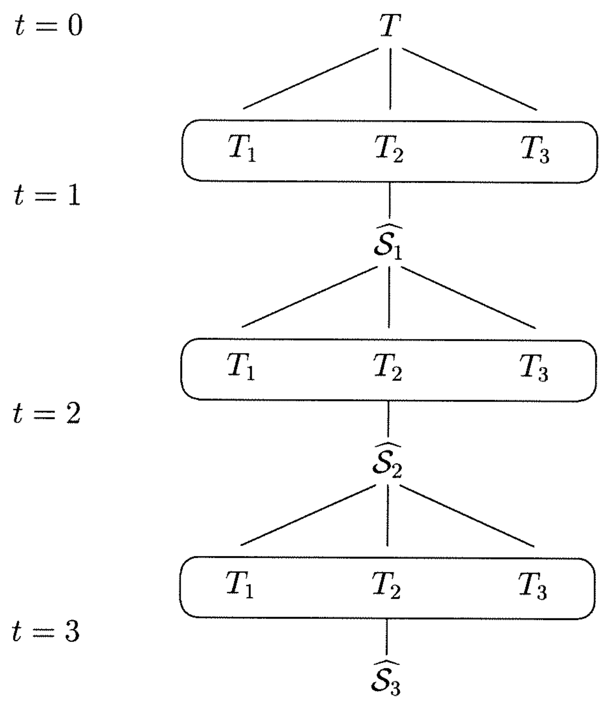


Figure 4.2: Approximate robust simulation algorithm example.

polyhedra is required to be no larger than a given bound, denoted Γ.

At each time step, $\widehat{\mathcal{S}_k} \supseteq \mathcal{S}_k$ is formed before the mapping step is applied. $\widehat{\mathcal{S}_k}$ is restricted to have at most Γ polyhedra. When $\mathcal{S}_k$ contains more than Γ polyhedra, some polyhedra are combined to form $\widehat{\mathcal{S}_k}$. Heuristics guide both the choice of polyhedra to be combined and the method of combination. Any technique that ensures $\mathcal{S}_k \subseteq \widehat{\mathcal{S}_k}$ is allowed.

Figure 4.2 demonstrates the approximate algorithm for a PL system with 3 regions and $\Gamma = 1$. Since $\mathcal{S}_0$ consists of a single polyhedron, T, $\mathcal{S}_0 = \widehat{\mathcal{S}_0}$. Identical to the exact solution example, T intersects all 3 regions of the PL system and, in one time step, maps to three new polyhedra. Instead of mapping each of the three polyhedra in $\mathcal{S}_1$ to form $\mathcal{S}_2$, the three polyhedra are combined into one new polyhedra, $\widehat{\mathcal{S}_1}$. $\mathcal{S}_2$ is then formed from the mapping of $\widehat{\mathcal{S}_1}$. This process is repeated until the final time is reached.

At each time step, the number of polyhedra in $\mathcal{S}_k$ is limited to Γl. This limitation ensures polynomial growth, but also adds conservatism to the algorithm. When combining polyhedra to form $\widehat{\mathcal{S}_k}$, additional volume is added to the polyhedra. $\widehat{\mathcal{S}_k}$ contains all states reachable at time k plus additional regions of the state space that cannot be reached. The added conservatism is directly related to the additional volume in $\widehat{\mathcal{S}_k}$.

Each polyhedron in $\widehat{\mathcal{S}_k}$ is calculated by applying the bounding polyhedra algorithm given in Section 3.3 to a set of polyhedra chosen from $\mathcal{S}_k$. Assuming that the number of constraints in each bounding polyhedron is proportional to n, forming $\widehat{\mathcal{S}_k}$ requires solving $\mathcal{O}(\Gamma ln)$ linear programs. Calculating $\mathcal{S}_{k+1}$ from $\widehat{\mathcal{S}_k}$ is a direct application of the piecewise linear system set mapping algorithm presented Table 3.6. Since this mapping is performed for each of the Γ polyhedra in $\widehat{\mathcal{S}_k}$, this step also requires solving $\mathcal{O}(\Gamma ln)$ linear programs. When applied over k time steps, the approximate robust simulation algorithm requires the solution of $\mathcal{O}(\Gamma kln)$ linear programs. Since robust simulation at time k is based only on the results from time $k-1$, storage requirements do not grow with time. Storage requirements, dictated by the size of the results from the piecewise linear system set mapping step, are $\mathcal{O}(\Gamma ln^2)$.

These computation and storage requirements assume that the polyhedra have a fixed number of constraints. In general, the PL system set mapping adds constraints to each polyhedron at each time step. Assuming that the number of constraints defining each region R_i of the PL system is $\mathcal{O}(n)$, the polyhedra in $\mathcal{S}_k$ will have $\mathcal{O}(kn)$ constraints. However, when $\widehat{\mathcal{S}_k}$ is formed by the bounding algorithm, the number of constraints can be selected. By choosing a fixed number, the polyhedra in $\mathcal{S}_k$ will have $\mathcal{O}(n)$ constraints and this additional computational dependence upon time is eliminated.

4.4 Algorithm Refinements

One of the main features of the approximate solution is that it can be systematically refined to obtain the exact solution. Less conservative solutions are obtained by increasing both Γ and the number of constraints in the polyhedra $\widehat{\mathcal{S}_k}$. Assuming that $\mathcal{S}_0$ is a single polyhedron, the exact solution is obtained when $\Gamma = l^k$ and each polyhedra is allowed to have an unlimited number of constraints. Intermediate solutions are obtained by letting $\Gamma = l^\gamma,\ 0 \leq \gamma \leq k$.

The choice of polyhedra for combination during the formation of $\widehat{\mathcal{S}_k}$, is also dictated by γ. In essence, γ is a memory parameter. Polyhedra with similar history are combined, and γ determines the length of history considered. Polyhedra with the same mapping history are combined when creating $\widehat{\mathcal{S}_k}$. γ is the number of mappings considered during the combination step. Using the individual polyhedral notation $T_{i,j,\ldots}$, polyhedra having identical indices $T_{i_1,i_2,\ldots,i_\gamma}$ are combined. In Figure 4.2, $\gamma = 0$ and all polyhedra are combined at each time step; the mapping history is completely ignored.

In addition to giving the exact solution after a finite number of refinements, this approach also has intuitive justification. If the PL system has nice dynamics, then an initial condition set distorts slowly. For a discretization of a continuous time system, the next state is very similar to the previous state. States that follow similar mappings have similar values, so their combination should add little extra volume.

4.5 Heuristics

Two of the steps in the approximate solution are heuristic based. The first heuristic step, the selection of polyhedra for combination, has already been discussed. One systematic method for selection was presented, though other methods are equally valid.

The second heuristic step is calculating the combination of the polyhedra. Ideally, the combination step forms the convex hull of the polyhedra. The convex hull adds the smallest possible volume and thus is the least conservative. However, finding the convex hull of a set of polyhedra exhibits exponential growth. One exception is if each polyhedra in the set is a simplex, then the convex hull can be found in polynomial time. However, simplexes are rarely used in practice.

Since the convex hull cannot be found with reasonable computations, a bounding polyhedra is formed. Both the number and choice of bounding hyperplanes have a large impact on results. Generally, one includes as many surfaces from the original polyhedra as possible. However, when bounding l

polyhedra, each additional hyperplane requires solving l linear programs. A large number of bounding hyperplanes can yield a very good approximation of the convex hull, but extensive computations are required. A small number of bounding hyperplanes is simple to compute, but can lead to very conservative results.

Several hyperplane selection methods have been tested, and no method is always the best. However, all successful methods share some common traits. When $\gamma > 0$, some mappings of the boundaries of the regions R_i are included. The first step of the mapping algorithm is the formation $T \cap R_i$. Whenever $T \cap R_i \neq T$, some of R_i's boundaries are active constraints. Since some boundaries are usually active, their mappings are likely to be constraints on some of the polyhedra being combined to form $\widehat{\mathcal{S}_k}$. As γ grows, multiple mappings of boundaries are included.

However, only a few multiple mappings are included. Repeated application of (3.5) in the same region is a power iteration for calculating the eigenvector associated with the maximum eigenvalue of A_i^{-t}. Repeated mappings yield very similar directions which cause numerical sensitivity problems in linear programming algorithms. Including all bounding hyperplanes for each polyhedron in the combination can also exhibit similar problems.

In general, one wants to include a large number of hyperplanes that form the convex hull. This requires including the hyperplanes from the original polyhedra that form the surface of the convex hull. Calculating which hyperplanes are on the convex hull is prohibitively expensive. However, including too many hyperplanes leads to both excessive computations and numerical instabilities. In practice, including mappings of region boundaries along with all hyperplanes from a few polyhedra yields good results.

Another method for reducing the conservatism is to split $\mathcal{S}_0$ into separate polyhedra and perform robust simulation separately on each polyhedron. The results from each simulation are then combined to form the final result. This idea is the basis of branch and bound, and is discussed in more detail in Section 5.3.

One final note is that for a linear system, the exact solution is obtained for

$\Gamma = 1$. For a linear system, a single polyhedra maps to a single polyhedra so there is no growth in the number of polyhedra at each time step. In addition, the system has only one region, so no additional bounding hyperplanes from region boundaries are added during the mapping step.

4.6 Example

The critical step of the robust simulation algorithm is the combination of polyhedra to avoid exponential growth. This step adds conservatism to the result while ensuring that the result contains all reachable states. This example demonstrates that process.

Consider the robust simulation of

$$x[k+1] = \begin{cases} \begin{bmatrix} 1 & 0 \\ 0 & 0.75 \end{bmatrix} x[k] + \begin{bmatrix} 0.25 \\ 0 \end{bmatrix} & \text{if } [1\ 0]x < 0, \quad (R_1) \\ \begin{bmatrix} 0.5 & -0.5 \\ 0.5 & 0.5 \end{bmatrix} x[k] + \begin{bmatrix} -2 \\ 0 \end{bmatrix} & \text{if } [1\ 0]x \geq 0, \quad (R_2) \end{cases}$$

for 2 time steps. This system has no underlying physical meaning. It is selected to demonstrate the potential for exponential growth in the exact solution and the conservatism added during combination step. For this example, $\Gamma = 1$.

The initial condition set, $\mathcal{S}_0$, shown in Figure 4.3, is $||x||_\infty \leq 1$. In one time step, this maps to two overlapping rectangles, shown by dashed lines in Figure 4.3. The next step of the robust simulation algorithm is to form $\widehat{\mathcal{S}_1}$ from $\mathcal{S}_1$.

One possible $\widehat{\mathcal{S}_1}$, denoted $\widehat{\mathcal{S}_{1_a}}$, is formed by bounding $\mathcal{S}_1$ by a rectangle whose sides are aligned with the coordinate axes. As shown in Figure 4.4a, this bounding set contains $\mathcal{S}_1$ (the shaded area) but also contains large additional areas of state space. With only four constraints, this is one of the simplest bounding polyhedra. However, it is also very conservative.

A second, less conservative possibility, denoted $\widehat{\mathcal{S}_{1_b}}$, is obtained by including all directions contained in the $\mathcal{S}_1$. This set of directions also includes mappings of the boundary between R_1 and R_2. This polyhedron is constrained by the

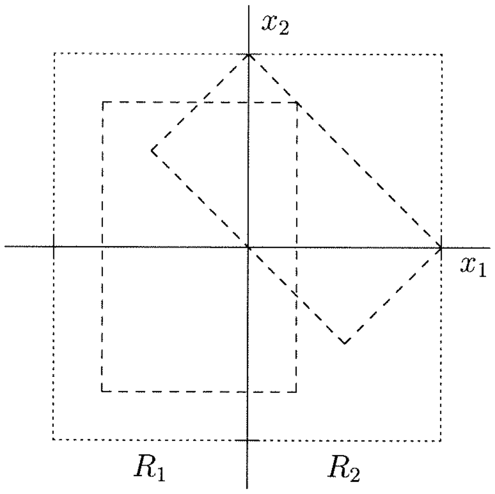


Figure 4.3: $\mathcal{S}_0$ (dotted) and $\mathcal{S}_1$ (dashed).

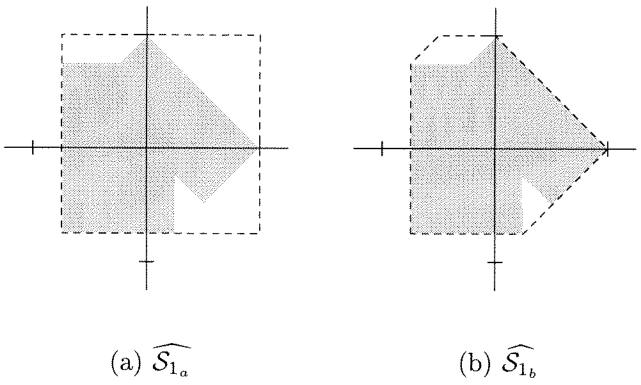

(a) $\widehat{\mathcal{S}_{1_a}}$
(b) $\widehat{\mathcal{S}_{1_b}}$

Figure 4.4: Two possibilities for $\widehat{\mathcal{S}_1}$.

eight directions

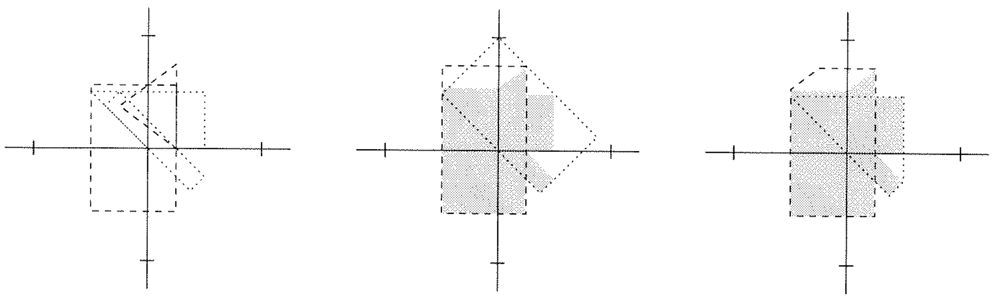
$$D = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \tag{4.1}$$

The first four directions are from the mapping of $\mathcal{S}_0 \cap R_1$ and the second four directions come from $\mathcal{S}_0 \cap R_2$. As shown in Figure 4.4b, two of the constraints are redundant; $\widehat{\mathcal{S}_{1_b}}$ has only six active constraints. This set adds much less extra area, and is far less conservative.

One possible measure of conservatism is obtained by comparing the areas of $\mathcal{S}_k$ and $\widehat{\mathcal{S}_k}$. $\mathcal{S}_1$ has an area of 2.0625. The simple bounding approximation, $\widehat{\mathcal{S}_{1_a}}$, has an area of 3.0625, nearly 50% larger than the true value. $\widehat{\mathcal{S}_{1_b}}$, with four additional constraints, has an area of 2.25, less than 10% larger than the true value.

Figure 4.5a shows the calculation of $\mathcal{S}_2$ from $\mathcal{S}_1$. Since both rectangles in $\mathcal{S}_1$ intersect R_1 and R_2, the exact solution has four polyhedra in $\mathcal{S}_2$. In the figure, the dashed lines are the mapping of $\mathcal{S}_1 \cap R_1$ and the dotted lines show the mapping of $\mathcal{S}_1 \cap R_2$. This mapping demonstrates two features of robust simulation. First, though $\mathcal{S}_1$ consisted only of rectangles, $\mathcal{S}_2$ has two rectangles, a triangle, and a trapezoid. The PL system maps polyhedra to polyhedra; it does not preserve their shape. Second, each of the four polyhedra in $\mathcal{S}_2$ intersects both R_1 and R_2. If the robust simulation continued for another time step, the exact solution for $\mathcal{S}_3$ would contain eight polyhedra. This is the exponential growth problem previously described.

The exponential growth is avoided by calculating $\mathcal{S}_2$ from $\widehat{\mathcal{S}_1}$. Figure 4.5b shows the calculation of $\mathcal{S}_2$ from $\widehat{\mathcal{S}_{1_a}}$. The shaded area is the exact solution for $\mathcal{S}_2$ and the dashed and dotted lines are the mapping $\widehat{\mathcal{S}_{1_a}}$ intersected with R_1



(a) Exact (b) From $\widehat{\mathcal{S}_{1_a}}$ (c) From $\widehat{\mathcal{S}_{1_b}}$

Figure 4.5: The three potential sets for $\mathcal{S}_2$.

and R_2, respectively. As intuitively expected, the large areas of unreachable state space in $\widehat{\mathcal{S}_{1_a}}$ map to large areas of unreachable space in the approximation of $\mathcal{S}_2$. Using the same notation as Figure 4.5b, Figure 4.5c shows the calculation of $\mathcal{S}_2$ from $\widehat{\mathcal{S}_{1_b}}$. Since this mapping started with the less conservative set $\widehat{\mathcal{S}_{1_b}}$, it contains much less unreachable region of state space.

The main property of the polynomial growth robust simulation algorithm is that the results contain all reachable states. As shown in Figure 4.5, the approximate solution contains the exact solution. Different heuristics for calculating $\widehat{\mathcal{S}_k}$ affect the conservatism in the results, but this property is preserved. Measuring the conservatism in the algorithm is difficult. For this two-dimensional example, the exact solution and areas of polyhedra are simple to calculate. For systems with larger state dimension, these calculations are prohibitively expensive. The next chapter introduces methods for measuring system performance through robust simulation. These methods also indirectly measure the conservatism of robust simulation, and in general, the results are very good.

Chapter 5
Nonlinear Performance Analysis

For most linear and nonlinear systems, it is impossible to answer the robust analysis question. Finding the noise and uncertainty that yield the worst performance is generally a non-convex optimization problem. Instead of solving the problem directly, numerical approximations are used. Traditional simulation is one common technique that is used to characterize performance. Any system can be analyzed through simulation, though interpreting the results can be difficult.

Traditional simulation describes the system's behavior for one initial condition, uncertainty value, and noise signal. Little can be inferred about the system response for other values of noise, uncertainty, and initial state. Each individual simulation is a lower bound on the worst system performance. The only information known about the worst possible performance is that it is at least as bad as any single, traditional simulation.

An extension of the traditional simulation idea is Monte Carlo analysis. The system is repeatedly simulated with randomly chosen initial condition, noise, and uncertainty values. Each simulation gives a lower bound on the worst case performance and the set of simulations gives a picture of typical performance. However, no guarantees can be made. In [36], the worst Monte Carlo performance was consistently 20% more conservative than lower bounds found by other techniques.

Neither Monte Carlo analysis nor other lower bound techniques answer the robust analysis question. Monte Carlo analysis generates results like

For 95% of all noise signals and uncertainty values, the performance

> is less than J_c.

Other lower bound techniques, such as gradient descent and power iteration, state

> There exists a noise signal and uncertainty value with performance of J_c.

These statements about performance are not guarantees. The desired result is

> For all noise signals and uncertainty values, the performance is less than J_c.

If the set of possible performances is viewed as a probability distribution, then the robust analysis question is concerned with the tail. Monte Carlo analysis gives the shape of the distribution. Other lower bound techniques find individual points in the tail. The robust analysis question finds the end of the tail.

Solving the robust analysis question requires making a statement about all possible trajectories. Robust simulation provides all possible trajectories. By attaching a measure to the robust simulation results, a global performance guarantee is obtained.

As with all analysis techniques, the performance measure used is compromise between physically motivated traits and mathematical convenience. The first part of this chapter describes tube cost, an extension of linear norm ideas to the PL setting. This measure fits naturally within the robust simulation framework and includes the l_∞ norm as a special case.

The second part of this chapter describes nonlinear performance guarantees and their interpretation. These guarantees are an upper bound on the worst possible performance and also provide a method for evaluating the conservatism of the robust simulation algorithm. This conservatism and limitations in lower bound algorithms can lead to a large gap between the bounds. One method for reducing this gap is presented. The chapter closes with some examples.

5.1 The Tube Cost

The tube cost is a nonlinear measure of signal size. Though not a norm in the traditional sense, it can be viewed as a generalization of the l_∞ norm to the piecewise linear setting. It allows nonlinear scalings; doubling a signal may not double its cost. It allows sets of signals to have the same value; many signals can have 0 tube cost. Due to its piecewise linear nature, the tube cost can be computed with only basic matrix operations and linear programming.

The tube cost is defined as

$$||u||_t = \max\{0, \max_{i,k} \langle d_i[k], u[k] - u_i[k] \rangle\} \tag{5.1}$$

where the d_i are a set of direction vectors and the u_i are a a set of nominal bounding trajectories. The $d_i[k], u_i[k]$ can be thought of as a set of hyperplanes. When the trajectory lies in the negative half space defined by the pair, no penalty is assessed. When the trajectory lies in the positive half space, it may contribute to the tube cost since $\langle d_i[k], u[k] - u_i[k] \rangle \geq 0$.

While the tube cost gives a useful measure of the size of a signal, it does not satisfy the norm properties described in [12]. In fact, the only property it satisfies is $||u||_t \geq 0$. All trajectories contained in the closed negative halfspaces defined by $d_i[k], u_i[k]$ have $||u||_t = 0$. Because of this property, neither the linear scaling property or parallelogram relationship hold. Though $||u||_t = 0$, $||u + u||_t = ||2u||_t$ may be much larger than zero. This nonlinear behavior gives the tube cost much of its flexibility.

Though the tube cost is not suited to traditional analysis techniques, it fits nicely within the robust simulation framework. The definition is based upon polyhedra, and applying the cost to sets requires the solution of linear programs. For ease of computer implementation, the tube cost parameters are usually written as two matrices

$$D[k] = \begin{bmatrix} d_1[k] \\ d_2[k] \\ \vdots \\ d_c[k] \end{bmatrix}, \; U[k] = \begin{bmatrix} u_1[k] \\ u_2[k] \\ \vdots \\ u_c[k] \end{bmatrix}.$$

```
Function value=Tubecost($S$,$D$,$U$)

value = 0
For $i$ from 1 to $c$
    test = $\max_{x \in S} \langle d_i, x - u_i \rangle$
    If test > value
        value = test
```

Table 5.1: Tube cost pseudocode.

Calculating the tube cost requires solving c linear programs for each polyhedra in the set of reachable states at each time step. An algorithm for calculating the tube cost for a single polyhedron S at one time step is given in Table 5.1. Calculating the tube cost for a single trajectory only requires basic matrix operations.

Example: a step response

Measuring the performance of a step response demonstrates the versatility of the tube cost. In many applications, one initially wants to allow limited overshoot and undershoot without penalty. Later, small errors are lightly penalized while large errors are heavily penalized. For this example, the requirements are as follows:

- Unit penalty for overshoots larger that 20% for $k \in 0 \dots 5$.
- Double penalty for $u[k] < 0$ for $k \in 0 \dots 5$.
- Unit penalty for errors greater than 5% for $k \in 6 \dots 100$.
- Triple penalty for errors greater than 10% for $k \in 6 \dots 100$.

These requirements can be written as

$$D[k] = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \; U[k] = \begin{bmatrix} 1.2 \\ 0 \end{bmatrix}, \; k \in 0 \dots 5$$

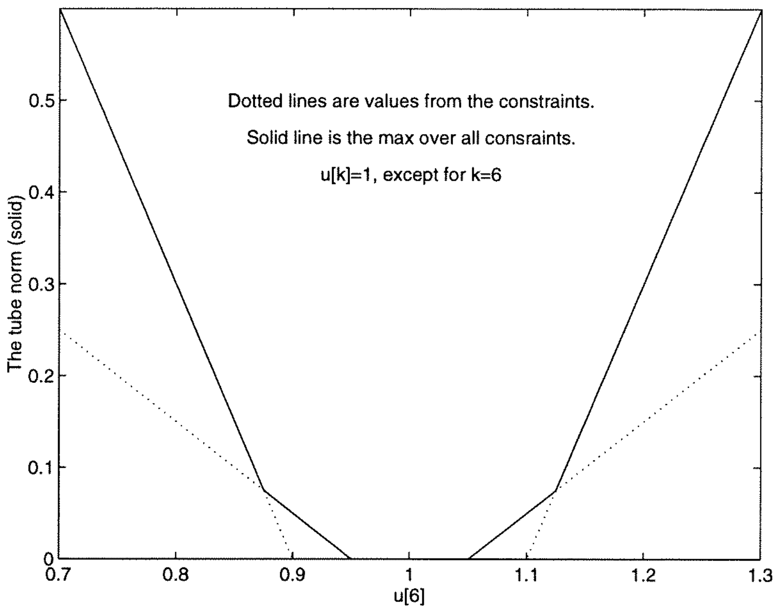


Figure 5.1: The tube cost for various step responses.

and

$$D[k] = \begin{bmatrix} 1 \\ -1 \\ 3 \\ -3 \end{bmatrix}, \; U[k] = \begin{bmatrix} 1.05 \\ 0.95 \\ 1.1 \\ 0.9 \end{bmatrix}, \; k \in 6 \ldots 100. \tag{5.2}$$

Thus, any trajectory in the set

$$0 \leq u[k] \leq 1.2, \; k \in 0 \ldots 5$$
$$0.95 \leq u[k] \leq 1.05, \; k \in 6 \ldots 100$$

has tube cost of zero. Figure 5.1 shows the tube cost for a step response that is 1 everywhere except at $k = 6$. Each row in (5.2) corresponds to a linear segment in Figure 5.1. The maximum over all segments, shown by a solid line, is the tube cost. For $1.05 \leq u[6] \leq 1.125$, $||u||_t = u[6] - 1.05$. For $u[6] \geq 1.125$, $||u||_t = 3(u[6] - 1.1)$. This piecewise linear error is the added flexibility from the tube cost. True norms cannot give this behavior and satisfy the linear scaling property.

Example: the l_∞ norm

The l_∞ norm, defined as

$$||u||_\infty = \max_k |u[k]|,$$

is a special case of the tube cost obtained by setting

$$D[k] = \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix},\ U[k] = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \forall k,$$

with **I** being the identity matrix and **0** being the zero matrix, both of appropriate sizes.

5.2 Nonlinear Performance

Since the nonlinear robust analysis question cannot be solved exactly, upper and lower bounds on the worst performance are computed. Several techniques exist for efficiently calculating good lower bounds. Robust simulation provides an efficient algorithm for calculating an upper bound.

The upper bound, denoted J_u, is a statement about the global performance of the system. This statement must hold for all possible trajectories. Since the result of robust simulation is the set of all possible trajectories, a global performance guarantee is obtained by applying a measure to these results.

The choice of measure must be compatible with the robust simulation framework. The tube cost is one such measure. Other common measures are either incompatible or may lead to extremely conservative results. Many common norms, such as l_1 and l_2, require knowing the entire trajectory to calculate the norm. However, robust simulation gives sets of reachable states, not information about individual trajectories. While a bound on the l_1 norm can be found by selecting the worst point at each time step, this may be very conservative since it assumes that the worst trajectory attains the worst value at every time step. Finding the worst l_2 signal suffers from the same limitation, and in addition, finding the largest l_2 point in a set is a non-convex optimization problem which is not computationally practical. Finding the l_∞ norm does not suffer from these limitations. The l_∞ norm depends only upon the largest value of the signal and the largest l_∞ point in a set can be found through linear programming. As previously shown, then l_∞ norm is a special case of the tube cost.

Even when using the tube cost, J_u may still be very conservative. Since the results from robust simulation include infeasible trajectories, J_u may be substantially larger than the worst possible signal. While there is no feasible trajectory with performance worse than J_u, there may be no feasible trajectory with performance of ϵJ_u, for an arbitrarily small $\epsilon > 0$.

To help measure the conservatism in the upper bound, a lower bound, denoted J_l, is also calculated. Lower bounds can be obtained by a variety of methods. Traditional simulation of a random initial condition and noise signal gives one. Improved lower bounds can be found through gradient descent algorithms and Monte Carlo techniques. Recently, better nonlinear lower bounds have been obtained through power iteration methods [36]. However, lower bounds never give guarantees on worst case performance. They are simply the value obtained from a single traditional simulation.

Given a required performance threshold J_c, there are three possible results. If $J_\mathrm{u} \leq J_\mathrm{c}$, then the performance threshold is achieved for all possible signals. If $J_\mathrm{l} > J_\mathrm{c}$, then there exists at least one noise and initial condition that does meet the requirement. When $J_\mathrm{l} \leq J_\mathrm{c} < J_\mathrm{u}$, no conclusion can be drawn. The worst known signal meets the performance requirement, but there is no guarantee that all signals meet it.

The gap between J_u and J_l is important. When $J_\mathrm{l} \approx J_\mathrm{u}$, the worst known signal has performance roughly equal to the guarantee. When the gap is large, little can be said about the true worst case performance. When $J_\mathrm{l} \leq J_\mathrm{c} < J_\mathrm{u}$, the gap is critical. In order to determine if the performance requirement is met, the gap must be reduced.

5.3 Branch and Bound

The gap between J_l and J_u depends upon both the lower bound and the upper bound. As described in Section 4.4, increasing γ is one systematic method for reducing the conservatism in robust simulation which leads to a better upper bound. However, it is also very expensive. Increasing γ by 1 increases the number of polyhedra allowed in the set of reachable states $\widehat{\mathcal{S}_k}$ and the total computation time by a factor of l. Changing γ also has no effect on the lower bound. If the upper bound is good and the lower bound is poor, increasing γ will have little effect on the gap.

Branch and bound is one method for improving both the lower and upper bounds that generally gives good results without excessive computational growth. While branch and bound only guarantees convergence with exponential computational growth, in many instances it performs well. More details about the algorithm and proofs of convergence properties can be found in [28]. Only the basic algorithm is presented here.

The main idea behind branch and bound is that multiple applications of the bounding algorithms over smaller parameter spaces are less conservative than a single application over a larger parameter space. Instead of looking at a single large problem, many small problems are considered and the results are combined.

To present the algorithm, several definitions are needed. P_0 is the set of initial conditions and all possible noises and uncertainty. This polyhedra is the parameter space upon which branching is performed. $\mathcal{P}$ is a set of polyhedra that may contain the worst case signal. P_i is a single polyhedra contained in $\mathcal{P}$. The upper and lower bounds for a single polyhedron are $J_{\mathrm{u}}(P_i)$ and $J_{\mathrm{l}}(P_i)$, respectively. $\hat{J}_{\mathrm{u}}$ and $\hat{J}_{\mathrm{l}}$ are largest upper and lower bounds of all $P_i \in \mathcal{P}$.

There are three main steps to the algorithm, which is given in Table 5.2. The stopping criterion determines when the algorithm terminates. As presented, the algorithm stops when the gap between the bounds is ϵ. If meeting a performance threshold J_{c} is the objective, then the algorithm terminates when $\hat{J}_{\mathrm{u}} \leq J_{\mathrm{c}}$ or $\hat{J}_{\mathrm{l}} > J_{\mathrm{c}}$. In practice, the algorithm also terminates after a specified number of branches.

The next major step is to find all regions where the global maximum cannot occur. If $J_{\mathrm{u}}(P_i) < \hat{J}_{\mathrm{l}}$, then it is impossible for the worst signal to lie within P_i and P_i is removed from $\mathcal{P}$. This pruning step is the key difference

```
$\mathcal{P} = P_0,\ \hat{J}_\mathrm{u} = \infty,\ \hat{J}_\mathrm{l} = 0$
While $\hat{J}_\mathrm{u} - \hat{J}_\mathrm{l} > \epsilon$, (stopping criterion)
    $\hat{J}_\mathrm{u} = \max_i J_\mathrm{u}(P_i)$
    $\hat{J}_\mathrm{l} = \max_i J_\mathrm{l}(P_i)$
    For each $P_i \in \mathcal{P}$, (pruning step)
        If $J_\mathrm{u}(P_i) < \hat{J}_\mathrm{l}$, then remove $P_i$ from $\mathcal{P}$.
    Select a $P_i$ such that $J_\mathrm{u}(P_i) = \hat{J}_\mathrm{u}$), (branching step)
        Partition $P_i$ into $P_{i,1}$ and $P_{i,2}$ according to some method.
        Add $P_{i,1}$ and $P_{i,2}$ to $\mathcal{P}$.
        Remove $P_i$ from $\mathcal{P}$.
```

Table 5.2: Branch and bound algorithm.

between branch and bound and simply gridding the parameter space. Only those regions which may contain the worst signal are searched.

The final step is the branch. One polyhedron with $J_\mathrm{u}(P_i) = \hat{J}_\mathrm{u}$ is divided into two smaller polyhedra. These two smaller polyhedra are added to $\mathcal{P}$ and the original, larger polyhedron is removed. The process then repeats, until the stopping criterion is satisfied.

5.4 Examples

To demonstrate the nonlinear performance algorithms, a set of random systems was generated. Continuous fifth order linear systems with one input were randomly generated and discretized. A saturation nonlinearity was placed on the input and a state feedback LQR controller was then designed, ignoring the saturation. The resulting closed loop systems had five states and three PL regions. All simulations shared the same initial condition set, $||x[0]||_\infty \leq 1.3$. This class of systems is neither the hardest nor the easiest class of problems known. These systems demonstrate the algorithms and have properties that are typical of physically motivated problems. In particular, the state update law is continuous across region boundaries. While continuity across boundaries is not a generic property, the flows in each region near a boundary are typically similar.

All upper bound calculations were performed with $\gamma = 1$. Increasing γ

did not lead to large improvements in the upper bound. The lower bound calculations are a combination of Monte Carlo methods and gradient descent. Random points in the parameter space are selected, and gradient descent is applied until a local maxima is obtained. While computationally expensive, this lower bound technique generally gives good results. The main purpose of these examples is to demonstrate the upper bound algorithm, not to achieve tight bounds with little computation.

It is important to consider the purpose of the upper bound algorithm. Robust simulation is used to guarantee performance when the system exhibits good nominal performance. If the system is nominally unstable, then there is little value in calculating the worst case performance. For this set of systems, nominal instability was defined as $J_\text{l} \geq 10$.

The first example indirectly measures the conservatism in the robust simulation algorithm. A stable PL system was selected and the exact robust simulation algorithm was compared to the approximate solution with $\gamma = 1$. The relative error, defined as

$$\frac{||x_\text{simulated}||_\infty - ||x_\text{exact}||_\infty}{||x_\text{exact}||_\infty},$$

is shown in Figure 5.2. For this system, the difference between the exact solution and approximate robust simulation is negligible. This means that little additional volume is added at the extreme points of the robust simulation. Nothing can be said about regions of state space closer to the origin.

While Figure 5.2 demonstrates that robust simulation can perform well and tight performance bounds can be obtained, it does not demonstrate the average performance of the algorithm. As a second test, two hundred randomly generated systems were analyzed. Robust simulation with $\gamma = 1$ was used to calculate the upper bound and Monte Carlo techniques combined with gradient descent were used to find the lower bound. The performance measure was $||x[30]||_\infty$.

Figure 5.3 is a plot of the distribution of the ratio J_l/J_u. Ideally, the two bounds are identical and the ratio is always 1. As shown by the solid line, for 90% of the runs, the ratio was larger than 0.9.

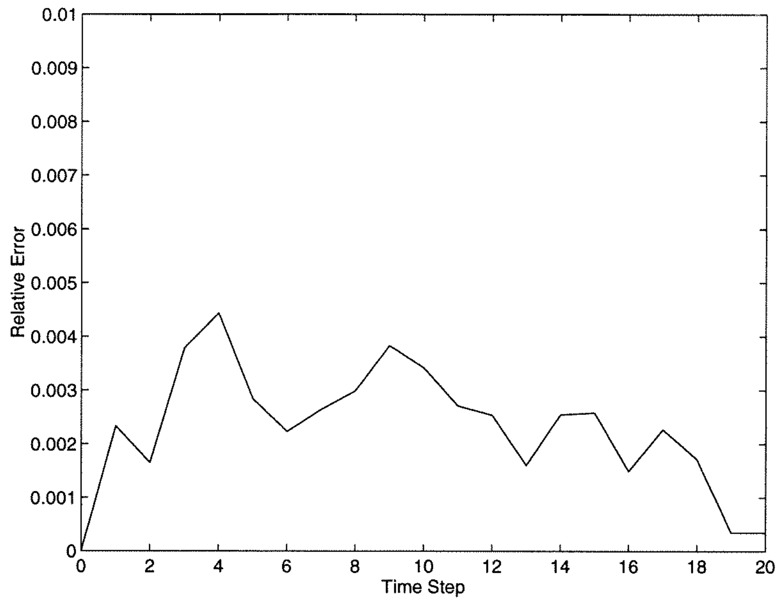


Figure 5.2: Relative error of robust simulation.

For the 20 systems where the ratio was less than 0.9, branch and bound was applied. The branch and bound algorithm terminated when either the ratio reached 0.9 or 50 branches were taken. As shown by the dashed line, branch and bound greatly improved the results when the initial gap was large. After applying branch and bound, only 12 runs had a ratio less than 0.9. Also, the worst ratio improved from 0.27 to 0.63.

To gain more insight into the different methods for reducing the gap between J_u and J_l, one system with a large gap was examined in more detail. Robust simulation with different values of γ was applied to this system, which has $J_l = 1.87$. The results, shown in Table 5.3, demonstrate that increasing γ improves the upper bound. However, increasing γ by one only yields a small reduction in J_u while roughly doubling the required computation time. Branch and bound with $\gamma = 1$ yields much better results with fewer computations. Computation times were obtained from simulations performed on a Sun UltraSparc 2 with a 170 MHz processor.

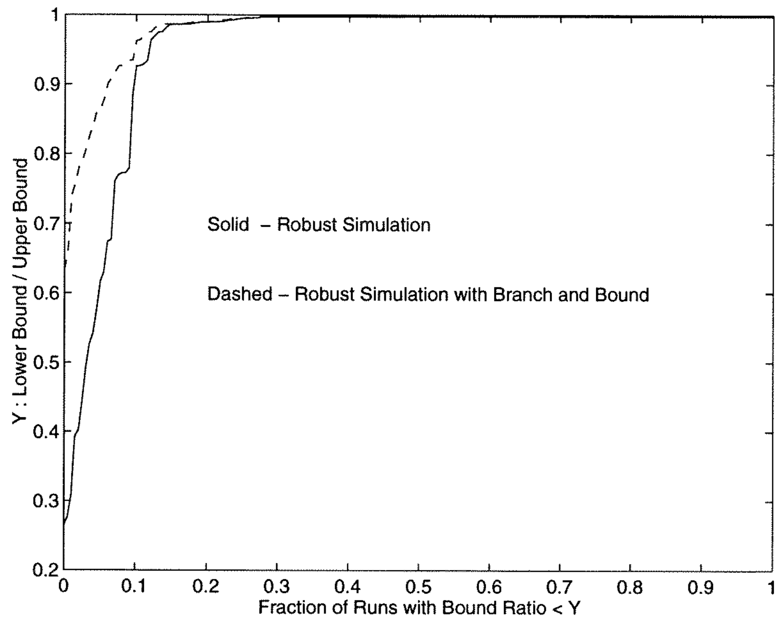


Figure 5.3: Nonlinear performance bound ratios.

algorithm	computation time	bound
$\gamma = 1$	15 seconds	5.38
$\gamma = 2$	2 minutes	4.41
$\gamma = 3$	5 minutes	4.09
$\gamma = 4$	10 minutes	3.83
$\gamma = 5$	18 minutes	3.64
branch and bound	13 minutes	2.07

Table 5.3: Computation times for robust simulation.

Chapter 6

Piecewise Linear Modeling

System modeling is often more art than science. There are a variety of frameworks available, and each has its own advantages. While piecewise linear modeling is not a new idea, it has seen limited use due to lack of computational tools. With the introduction of robust simulation and nonlinear performance guarantees, the case for PL modeling framework becomes much more compelling.

In addition to facilitating numerical analysis, PL models admit a broad set of nonlinear systems. As introduced in Chapter 3, saturation is exactly represented by a PL system. Many other common nonlinearities, ranging from hysteresis to dead zone, are also naturally PL. Additionally, there are no continuity requirements for PL systems. Discontinuous switching systems are easily described in the PL framework. Input constraints are handled exactly; no approximations are necessary.

PL systems also facilitate the blending of theoretical models with experimentally identified models. Each region of a PL system describes the system's local behavior independently of the other regions. If one has detailed knowledge of the behavior of the system in a region of state space, the information can be immediately incorporated by adding additional regions to the PL system.

However, most systems are not strictly PL. To account for this, a standard representation for a PL system is developed and a systematic technique for converting a general system to a PL system is presented. Various forms of uncertainty can also be cast in the PL setting. This PL approximation of a general, uncertain nonlinear system, called a PL cover, has one critical property. A performance guarantee for the PL cover is also a guarantee for the original system.

This chapter begins by defining the standard form for piecewise linear systems, the PL linear fractional transformation. Next, a method for converting general nonlinear systems to PL systems is presented. This method is robust in that guarantees from the PL approximation also hold for the original system. From the standard PL form, a measure of the degree of nonlinearity, called the nonlinear dimension, is developed. This nonlinear dimension determines the computational cost of analysis. Various forms of uncertainty, such as unmodeled dynamics and real parameter variations, are also cast as PL systems. The chapter closes with a list of common nonlinearities that are naturally PL.

6.1 Piecewise Linear LFT

All successful modeling frameworks have a standard building block for creating complex systems. Linear simulation packages generally use a transfer function and state space representation. In many robust control methodologies, the linear fractional transformation (LFT) is used. In the robust linear setting, the LFT allows the conversion of a variety of problems to a form where one analysis tool is applicable. In the piecewise linear setting, the LFT gives similar benefits.

The nonlinear LFT, shown is Figure 6.1, separates the linear portion of the system, including states, from the nonlinearities. It is defined as

$$\begin{aligned} \begin{bmatrix} \hat{y} \\ y \end{bmatrix} &= M(z) \begin{bmatrix} \hat{u} \\ u \end{bmatrix} \\ \hat{u} &= N(\hat{y}) \end{aligned} \tag{6.1}$$

where $M(z)$ is a linear system and N is a stateless nonlinearity. The only restriction on N is that it is solely a function of its current inputs. All state information is contained within the linear system $M(z)$.

The idea of separating the linear portion of the system from the nonlinear portion appears in a variety of nonlinear analysis techniques. Once this

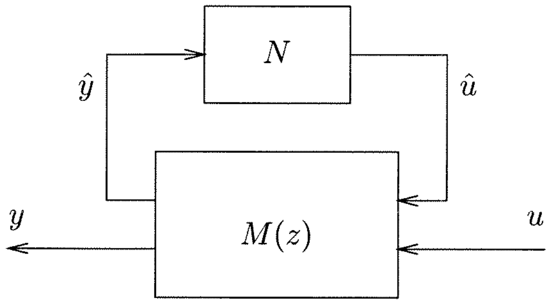


Figure 6.1: Nonlinear LFT.

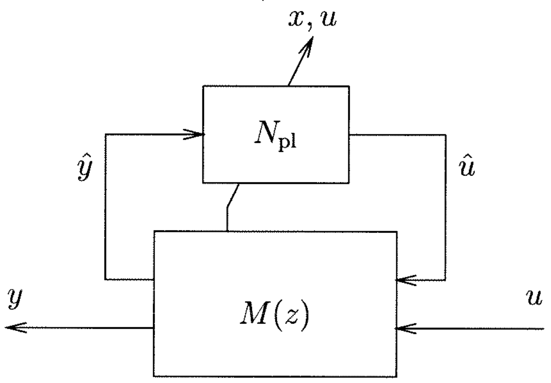


Figure 6.2: Piecewise Linear LFT.

separation is made, analysis methods can focus on the nonlinearity N and its interactions with the linear system $M(z)$. Typically, restrictions are placed on N to simplify analysis and $M(z)$ is left untouched.

When N is a piecewise linear function, a PL system is obtained and robust simulation can be applied. The piecewise linear LFT (PL-LFT), shown in Figure 6.2, restricts the nonlinearity to be of the form

$$N_{\mathrm{pl}}(\hat{y}) = \bar{C}_i^N + D_i^N \hat{y} \tag{6.2}$$

where $\bar{C}_i^N$ and D_i^N implicitly are functions of x and u. In each region R_i of the state and input space, the nonlinearity follows a different affine rule. Expanding (6.1) into its piecewise linear form yields

$$\begin{gathered}\begin{bmatrix} x[k+1] \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} x[k] \\ \hat{u} \\ u \end{bmatrix} \\ \hat{u} = \bar{C}_i^N + D_i^N \hat{y}\end{gathered} \tag{6.3}$$

and gives insight into the behavior of PL-LFTs. First, all of the standard LFT interconnection rules apply. The interconnection of two PL-LFTs is a PL-LFT. Thus, complex PL systems can be built from the interconnection of simpler PL systems. Second, the PL-LFT is well-posed when $I - D_{11}D_i^N$ is invertible for every D_i^N. For most systems, $D_{11} = 0$ since the nonlinearity generally does not feed into itself and the LFT is well-posed. While problems, such as algebraic loops, can occur when interconnecting PL systems, these are the same type of problems that occur with the interconnection of linear systems. Third, the PL-LFT yields a PL system in the form of (3.2). Assuming that the PL-LFT is well posed, (6.3) can be written as

$$\begin{bmatrix} x[k+1] \\ y \end{bmatrix} = \begin{bmatrix} A + B_1\hat{D}_i^N C_1 & B_1\bar{D}_I^N\bar{C}_i^N & B_2 + B_1\hat{D}_i^N D_{12} \\ C_2 + D_{21}\hat{D}_i^N C_1 & D_{21}\bar{D}_I^N\bar{C}_i^N & D_{22} + D_{21}\hat{D}_i^N D_{12} \end{bmatrix} \begin{bmatrix} x[k] \\ 1 \\ u \end{bmatrix}$$

$$\hat{D}_i^N = D_i^N(I - D_{11}D_i^N)^{-1}$$

$$\bar{D}_i^N = I + \hat{D}_i^N D_{11}$$

which has the expected PL form. Also, any PL system in the form (3.2) can be written as a PL-LFT in the form (6.3).

6.2 Piecewise Linear Covers

When forming the PL-LFT from the nonlinear LFT, N is required to be piecewise linear. However, many nonlinearities, including all differentiable nonlinearities, are not PL. To account for this, a piecewise linear cover of the original nonlinearity is needed.

A piecewise linear cover of N, denoted $\hat{N}$, is a combination of a static piecewise linear nonlinearity and a convex set of noises $\mathcal{N}$ such that there exists $n \in \mathcal{N}$ satisfying

$$N(\hat{y}) = N_{\text{pl}}(\hat{y}) + D_i^{\mathcal{N}} n = \bar{C}_i^N + D_i^N\hat{y} + D_i^{\mathcal{N}} n, \ \forall \hat{y}.$$

The matrix D_i^n is a noise scaling and typically $\mathcal{N}$ is l_∞. Simply stated, for any possible input sequence $\hat{y}$, there exists a noise sequence n such that $\hat{N}(\hat{y}, n) = N(\hat{y})$.

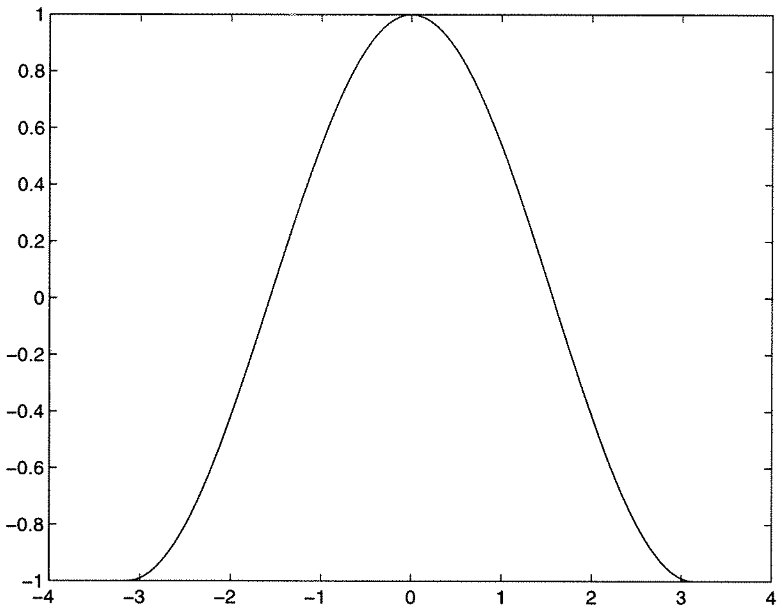


Figure 6.3: A nonlinearity.

Since the original nonlinearity N is contained in the set of nonlinearities defined by $\hat{N}$, any sufficient analysis condition for $\hat{N}$ is also sufficient for N. For example, robust simulation gives sufficient performance conditions for PL systems. Through the use of PL covers, it can give performance guarantees for any nonlinear system. This allows arbitrary nonlinear systems to be analyzed in the PL framework. The idea of covering the true nonlinearity by one that can be analyzed is used in several other nonlinear techniques, such as LPV.

As an example, consider the nonlinearity

$$N(\hat{y}) = \begin{cases} \cos\hat{y} & \text{if } -\pi \leq \hat{y} \leq \pi, \\ -1 & \text{otherwise,} \end{cases} \tag{6.4}$$

shown in Figure 6.3. While portions of this nonlinearity are PL, the region $-\pi \leq \hat{y} \leq \pi$ is not. A trivial cover,

$$\hat{N}(\hat{y}, n) = n, \; n \in l_\infty, \tag{6.5}$$

completely ignores the nonlinearity and exclusively uses noise to account for its behavior. By choosing the noise $n = N(\hat{y})$, the requirement $N(\hat{y}) = \hat{N}(\hat{y}, n)$

is satisfied. A slightly less trivial cover,

$$\hat{N}(\hat{y},n) = \begin{cases} -1, & \text{if } \hat{y} < -\pi \\ n, & \text{if } -\pi \le \hat{y} \le \pi \\ -1, & \text{if } \hat{y} > \pi \end{cases}, \quad n \in l_\infty, \tag{6.6}$$

has three PL regions and captures the piecewise linear regions exactly while ignoring the nonlinear cosine portion. The cover

$$\hat{N}(\hat{y},n) = \begin{cases} -1, & \text{if } \hat{y} < -\pi \\ 0.7246\hat{y} + 1.1382 + 0.1382n, & \text{if } -\pi \le \hat{y} \le 0 \\ -0.7246\hat{y} + 1.1382 + 0.1382n, & \text{if } 0 < \hat{y} \le \pi \\ -1, & \text{if } \hat{y} > \pi \end{cases}, \quad n \in l_\infty, \tag{6.7}$$

uses four PL regions and captures some of the cosine behavior. As shown in Figure 6.4, the regions $\hat{y} > \pi$ and $\hat{y} < -\pi$ are captured exactly by $\hat{N}$ while the cosine portion is covered by two PL regions. In the cosine regions, the nominal PL value is shown by a dashed line and the extreme values obtainable for any noise are shown by a dotted lines. As required, every value of the original nonlinearity lies within the dotted lines. Also note that the PL cover (6.7) has discontinuities at $\hat{y} = \pm\pi$. There are no continuity requirements on PL covers and continuous nonlinearities can have discontinuous covers.

The PL cover defines a set of nonlinearities that contains the original nonlinearity. Any sufficient analysis condition must hold for all nonlinearities in the set. If this set is large, then analysis results are likely to be conservative since the result holds for every nonlinear system in the set. One measure of the size of the set is the amount of noise needed to create the cover. More noise leads to larger sets of nonlinear systems contained in the cover. For example, any nonlinearity with output bounded by -1 and 1 is included in the cover (6.5). This cover has $D_i^{\mathcal{N}} = 1$. The cover (6.7) has $D_i^{\mathcal{N}} = 0$ in two regions and $D_i^{\mathcal{N}} = 0.1382$ in two regions. As shown in Figure 6.4, it covers a much smaller set of nonlinearities.

When forming the PL cover, the size of $D_i^{\mathcal{N}}$ can be reduced by increasing the number of regions. The benefit from adding one region can be enormous.

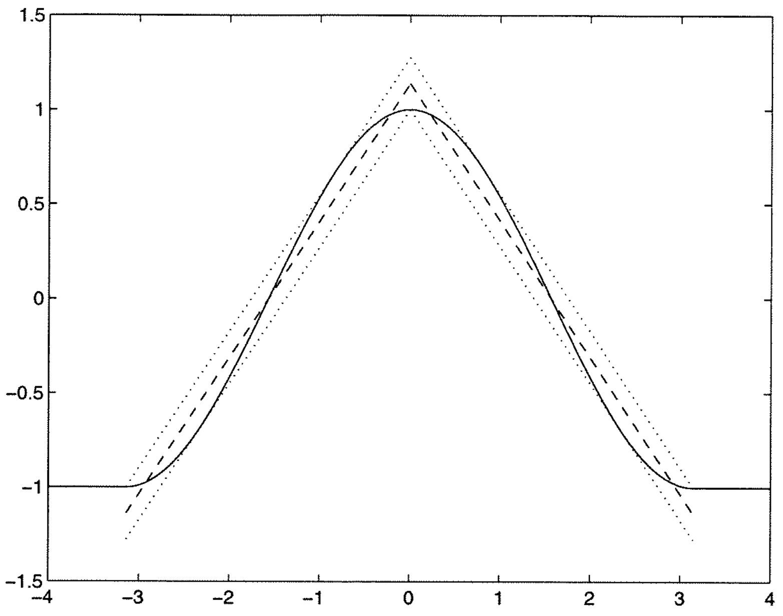


Figure 6.4: A nonlinearity and one possible cover.

Adding one region to (6.6) results in (6.7) and reduces the largest noise by a factor of seven. However, increasing the number of regions increases the computation time required for analysis.

There are always infinitely many possible covers for any nonlinearity. Those presented in this example demonstrate the idea of a cover; they do not have any special properties. Algorithms for creating PL covers, the effects of conservatism in the covers, and the use of sets of covers of a single nonlinearity are discussed in Chapter 7.

6.3 Nonlinear Dimension

The dimension of a system is a measure of its size. It is the key parameter that affects computational cost during numerical analysis. For linear systems, the number of states is the dimension of the system. While increasing the number of inputs or outputs has some affect on numerical analysis, changing the state dimension has the largest effect. For example, verifying the stability of a linear system requires finding the eigenvalues of an $n \times n$ matrix, which is $\mathcal{O}(n^3)$.

For nonlinear systems, the state dimension is not as useful. Small increases in the state dimension can have both large and small effects on computational cost. Even systems with small state dimension can be very difficult to analyze. A measure of the amount of nonlinearity in the system and its effect on computational requirements is desired. The PL-LFT indirectly gives one such measure. The nonlinear dimension, ν, is defined as the minimum number of linearly independent variables needed to recreate (6.2) exactly. This nonlinear dimension is analogous to the number of parameters needed when using LPV techniques.

Recreating (6.2) requires identifying the different PL regions R_i. As shown in Table 3.3, this requires repeated evaluation of $\langle d_j^i, z\rangle$ where d_j^i is the jth normal vector defining region R_i and $z \in \mathcal{R}^{n+m}$. The components of z orthogonal to all of the d_j^i never contribute to the region identification and can be ignored. Thus, finding ν is equivalent to finding the largest linear subspace $\mathcal{Z} \subseteq \mathcal{R}^{n+m}$ such that

$$\langle d_j^i, z\rangle = 0 \ \forall z \notin \mathcal{Z}.$$

The nonlinear dimension is the dimension of $\mathcal{Z}$. $\mathcal{Z}$ is simply the space spanned by the d_j^i. Using the notation from (3.3), another definition is

$$\nu = \operatorname{rank} \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_l \end{bmatrix}. \tag{6.8}$$

As an example, consider the system (3.13). To differentiate between PL regions, both states are required. However, (6.8) has rank of one for this example, so $\nu = 1$. Examining the region boundaries reveals that only the linear combination $[0.5\ 1.5]x[k]$ is evaluated when identifying regions.

For robust simulation, computations grow exponentially with ν. Though computations grow polynomially with the number of regions in a PL system, l increases exponentially as ν grows. For example, a system with a single saturation, like (3.13), has $\nu = 1$ and $l = 3$. A system with two independent saturations has $\nu = 2$ and $l = 9$. A generic system with m saturating inputs has $\nu = m$ and $l = 3^m$. PL covers exhibit identical exponential behavior. Forming a PL cover is equivalent to gridding the nonlinearities and approximating them by a finite number of PL regions. When covering a nonlinearity with dimension ν, dividing each dimension into k segments yields k^ν PL regions.

Any technique that requires gridding the nonlinearities also exhibits exponential growth as the number of nonlinear variables increases. To avoid gridding, some techniques approximate the nonlinearities by arbitrary operators. However, these coarse approximations can be very conservative. For example, traditional LPV analysis treats the nonlinear parameter variations as arbitrary operators. The analysis results are conservative, but require little computation. LPV analysis with parameter rate bounds requires gridding the parameter space. The results are less conservative, but computations grow exponentially with the number of parameters. In both cases, computations grow polynomially with state dimension [29].

Similar computational reductions can be achieved for robust simulation by choosing the complexity of the PL covers. The various covers of (6.4) demonstrate this process. The simplest cover (6.5) approximates the nonlinearity as a noise. With this cover, the original nonlinear system is reduced to a linear system with noise and $\nu = 0$. The less conservative covers (6.6) and (6.7) both have $\nu = 1$.

Two properties of ν merit mentioning. Linear systems have $\nu = 0$. Since a linear system has no bounding polyhedra, (6.8) reduces to the rank of an empty matrix. The interconnection of PL systems cannot create additional nonlinear variables. In other words, for the interconnection two systems I and J, $\nu_{I+J} \leq \nu_I + \nu_J$. This follows directly from the interconnection properties of LFTs.

6.4 Unknown Real Parameters

One common form of model uncertainty is an unknown real parameter. For example, due to temperature variations, a circuit's resistance may be known to lie within a bounded range. Consider an unknown gain which obeys the rule

$$y = ku, \;\; k = \bar{k} \pm \delta, \;\; |\delta| \leq k_\delta$$

where δ is time varying. Though this gain cannot be represented exactly as a PL system, it can be approximated to any desired accuracy by a PL cover.

The PL representation is developed by separating the gain into two parts: the constant gain and the uncertain term. The constant gain, $\bar{k}u$, is a linear system. The uncertain term, δu, accounts for the unknown behavior of the gain. A PL cover for the uncertain term is given by

$$\delta u = C(u) k_\delta n, \;\; |n| \leq 1 \tag{6.9}$$

where C is a piecewise constant nonlinearity with $|C(u)| \geq |u|$. One nonlinearity meeting this requirement is

$$C(u) = \begin{cases} \vdots & \vdots \\ 3 & \text{if } -3 \leq u < -2, \\ 2 & \text{if } -2 \leq u < 1, \\ 1 & \text{if } -1 \leq u \leq 1, \\ 2 & \text{if } 1 < u \leq 2, \\ 3 & \text{if } 2 < u \leq 3, \\ \vdots & \vdots \end{cases}$$

One feature of this cover is that it has an infinite number of regions. However, with bounded u, only a finite number of regions are required.

The entire uncertain gain can be written as the PL system

$$y = \bar{k}u + C(u) k_\delta n, \;\; |n| \leq 1.$$

The uncertain gain is the parallel interconnection of a linear system and a PL system with ν of 1. The interconnection is also as PL system with ν of 1. The accuracy of the cover is determined by the number of levels in Q. More regions in Q leads to a less conservative result.

This example is typical of uncertainty modeling in the PL setting. Uncertainty is represented by the addition of l_∞ noise with gains that scale according to a PL function.

6.5 Unmodeled Dynamics

Like unknown real parameters, unmodeled dynamics are also approximated by PL systems. Consider the causal norm bounded operator

$$y = \Delta u, \;\; \|\Delta\|_1 \leq 1.$$

This operator has the property that its output at any time has magnitude equal to or less than the largest input it has seen.

The PL representation cover is given by

$$\begin{aligned} x[k+1] &= \max(x[k], u[k]) \\ y[k] &= C(\max(x[k], u[k]))n, \;\; n \in l_\infty \end{aligned}$$

where C is as specified in (6.9). As shown later in this chapter, max is a PL nonlinearity. This cover has $\nu = 2$ and $l = 2l_C$ where l_C is the number of regions in C.

Unknown real parameters and unmodeled dynamics are very similar. In both cases, the output is a noise multiplied by C. However, unmodeled dynamics require the addition of a state which holds the size of the maximum input seen. This cover can also be thought of as the interconnection of a maximum element and an uncertain gain.

6.6 Piecewise Linear Catalog

Two common PL nonlinearities, saturation and maximum, have already been introduced. Many other common nonlinearities are also PL. The section presents simplified versions of several common nonlinearities. More complex forms, such as saturations with different slopes can be constructed by slightly modifying those presented here. In all cases, the PL-LFT representation is trivial to construct, and is omitted.

Saturation

The linear saturation

$$y = \text{sat}(u) = \begin{cases} -1 & \text{if } u < -1, \\ u & \text{if } -1 \leq u \leq 1, \\ 1 & \text{if } u > 1, \end{cases}$$

is exactly represented by a PL system with no states, three regions, and ν of one.

Rate saturation

A rate saturation requires an additional state that stores the previous value of the output. At any time step, the output can change by at most a fixed amount. One implementation of the rate saturation is the PL system

$$\begin{aligned} x[k+1] &= x[k] + \text{sat}(u[k] - x[k]) \\ y[k] &= x[k] + \text{sat}(u[k] - x[k]) \end{aligned}$$

which has one state, three regions, and ν of one.

Absolute value

The function $y = |u|$ is exactly represented by the PL system

$$y = \begin{cases} u & \text{if } u \geq 0, \\ -u & \text{otherwise}, \end{cases}$$

which has no states, two regions, and ν of one.

Dead zone

A dead zone, shown in Figure 6.5, is the interconnection of a saturation and a linear gain. It can be implemented as

$$y = u - u_c \text{sat}(\frac{u}{u_c}).$$

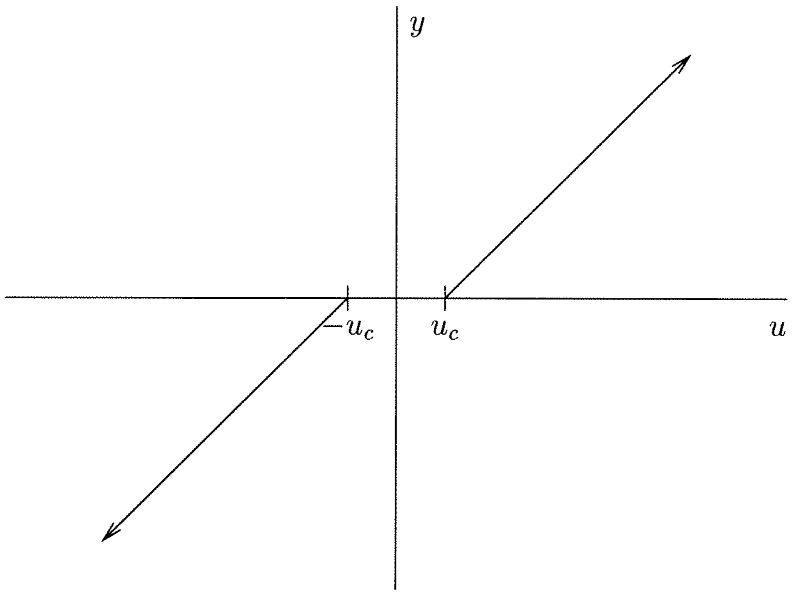


Figure 6.5: Dead zone.

Alternatively, it can be written as

$$y = \begin{cases} u + u_c & \text{if } u \leq -u_c, \\ 0 & \text{if } -u_c < u < u_c, \\ u - u_c & \text{if } u \geq u_c. \end{cases}$$

Both representations have no states, three regions, and ν of one.

Quantization

Quantization can also be captured exactly by a PL nonlinearity. For example, the simple quantization law

$$y = Q(u) = u - u \bmod 1$$

can be written as the PL system

$$y = \begin{cases} \vdots & \vdots \\ -1 & \text{if } -1 \leq u < 0, \\ 0 & \text{if } 0 \leq u < 1, \\ 1 & \text{if } 1 \leq u < 2, \\ \vdots & \vdots \end{cases}$$

which has no states, an infinite number of regions, and ν of one. The number of regions can be limited by restricting the size of the input. Alternatively, the portions of the quantization element can be covered by one region and additive noise. More details are presented in Section 7.2.

Maximum and minimum

The function $y = \max(u_1, u_2)$ is exactly represented by the PL system

$$y = \begin{cases} u_1 & \text{if } u_1 \geq u_2, \\ u_2 & \text{otherwise,} \end{cases}$$

which has no states, two regions, and ν of one. In general, the maximum or minimum of n inputs is a PL system with no states, $n-1$ regions and ν of $n-1$.

Maximum and minimum are unique in that the number of regions l grows linearly with ν. These functions do not require gridding the input space to represent the nonlinearity. Instead, the n-dimensional input space is partitioned into n regions. Due to this special structure, efficient analysis is possible even when ν is large.

Hysteresis

Hysteresis, shown in Figure 6.6, is another common PL nonlinearity. It is modeled using one binary state variable. This state, which only takes values of -1 and 1, determines the output curve. One possible PL representation is

$$x[k+1] = \begin{cases} -1 & \text{if } u \leq -u_s, \\ 1 & \text{if } u \geq u_s, \\ x[k] & \text{otherwise.} \end{cases}$$

$$y[k] = \begin{cases} y_o \mathrm{sat}(\frac{u-u_c}{2u_c}) & \text{if } x[k] \leq -1, \\ y_o \mathrm{sat}(\frac{u+u_c}{2u_c}) & \text{if } x[k] \geq 1, \\ y_o \mathrm{sat}(\frac{u}{2u_c}) & \text{otherwise.} \end{cases}$$

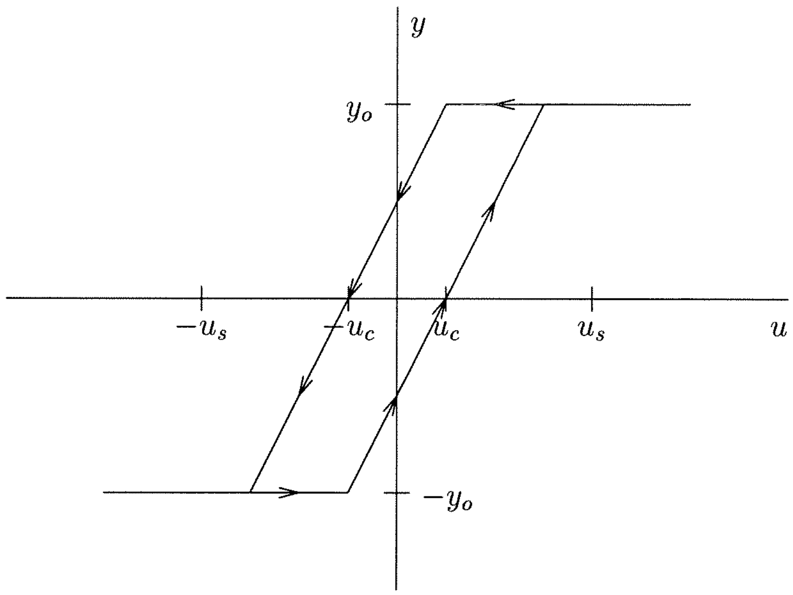


Figure 6.6: Hysteresis.

This definition also accounts for improper initialization of the state variable by selecting the most appropriate output for $x[k] \neq \pm 1$. When $-1 < x[k] < 1$, a standard saturation element is assumed.

Hysteresis, which has ν of two, demonstrates the gridding of the nonlinear space needed to account for complex nonlinearities. Each of the three regions of $x[k]$ is partitioned into five regions depending upon u, yielding a total of fifteen regions. However, with different assumptions on the behavior for $x[k] \neq \pm 1$ and the values of u_c and u_s, l can be reduced to six.

Multiplication

Though many common nonlinearities are naturally piecewise linear, multiplication is not. In order to account for nonlinearities such as $y = u^2$, PL covers are needed.

A PL cover for $y = u_1 u_2$ requires two quantization elements $\hat{Q}_1$ and $\hat{Q}_2$ that satisfy

$$u_1 = \hat{Q}_1(u_1) + e_1, \;\; |e_1| \leq E_1$$
$$u_2 = \hat{Q}_2(u_2) + e_2, \;\; |e_2| \leq E_2.$$

Using these quantizations and simple algebraic manipulations, one PL cover for the multiplication $y = u_1 u_2$ is given by

$$\hat{y} = \hat{Q}_1(u_1)u_2 + \hat{Q}_2(u_2)E_1 n_1 + E_1 E_2 n_2, \ n_1, n_2 \in l_\infty. \tag{6.10}$$

This cover has ν of two, requires two additional noise inputs, and l depends upon the number of regions in the $\hat{Q}_1$ and $\hat{Q}_2$.

This cover can be simplified by choosing $\hat{Q}_2$ such that $|u_2| \leq E_2$. With this requirement, $\hat{Q}(u_2) = 0 \ \forall u_2$ and (6.10) reduces to

$$\hat{y} = \hat{Q}(u_1)u_2 + E_1 E_2 n, \ n \in l_\infty$$

which has ν of one and requires only one additional noise input. This is another example of how ν can be affected by choosing a coarser PL cover.

Chapter 7

Hierarchical Modeling and Multiresolution Simulation

Analysis results are only as good as the model being analyzed. For most physical systems, any level of detail can be included in the model. Some simulation packages even provide sets of models for common components. For example, SPICE has several models of many common semiconductor components [37]. In the previous chapter, PL covers of general nonlinearities were introduced. Since the PL cover is not unique, many valid covers exist for every nonlinearity. These covers also form a set of models for a nonlinear system.

When two or more models for a given item are available, some method is required to choose the correct one. When building large systems from the interconnection of smaller parts, small changes in one component can have large effects on the overall system. There are many heuristics for choosing the level of detail to incorporate in a large model. In short, one wants to include the phenomena that greatly affect simulation results and ignore those that do not. With too much detail, simulations can take a long time to compute. With too little detail, the results are meaningless.

In some simulation methods, the complexity of the model is varied as the simulation progresses. These techniques, known as multiresolution simulation, attempt to use extra computation only where it gives a large benefit. Multigrid methods, one form of multiresolution simulation, change the spatial discretization of the original model dynamically. Multigrid methods have been particularly successful when simulating partial differential equations [11].

Both multiresolution simulation and heuristic based model selection can greatly reduce computation time. However, they fail to address the same underlying problem:

> When multiple models exist for a single system, which gives the correct simulation result?

There are other limitations to these techniques. When switching between models in a multiresolution simulation, initial values must be chosen for the new model.

Even if only one model exists for each component, building large models from simpler systems can introduce errors. The assumptions made when modeling one subsystem may be incompatible with those of another. For example, when modeling a flexible aircraft wing, one would like to mix structural dynamics with quasi-static aerodynamic modeling. However, the forces calculated from the quasi-static aerodynamic model induce oscillations in the wing, which violate the quasi-static assumption. To account for these types of problems, the assumptions and associated uncertainty in each model must be quantified and the simulation technique must use the additional information.

Robust simulation and the PL modeling framework address these issues. PL modeling explicitly quantifies uncertainty as noise. Robust simulation gives the set of all possible states, rather than a single value. By using these two techniques, one can build sets of models for a complex system and perform multiresolution simulation in a mathematically sound manner.

This chapter focuses on creating model hierarchies and interpreting their simulation results. It does not address issues related to building models of physical systems. Methods for selecting the appropriate model fidelity are addressed in [22]. First, model hierarchies are defined and two examples are presented. From these ideas, robust multiresolution is developed. Changing models during simulation fits naturally within the robust simulation framework. The chapter closes with two examples that demonstrate the problems encountered with sets of models and the potential benefits from multiresolution simulation.

7.1 Hierarchical Modeling

When creating a mathematical model of a physical system, any degree of complexity is possible. When developing models for interconnection with other systems, the degree of accuracy needed is not known in advance. Effects critical to one system may be superfluous to another. Thus, a set of models with varying degrees of accuracy is needed. These models must also be useful for computation.

In order to choose between models in the set, a hierarchy is needed. Each model must have a measure of its fidelity and complexity. For LTI systems, the state dimension is often used to measure complexity. For uncertain linear systems, the hierarchy is not as obvious. The tradeoff between additional states and additional uncertainty is not clear. The hierarchy does not need to be, and rarely is, a complete ordering.

Model hierarchies are an area of active research in the computer graphics community. One example is the use of wavelets to create sets of models of three-dimensional objects for on-screen rendering. By varying the number of wavelet coefficients used during the reconstruction, the accuracy of the reproduction is varied. Typically, as an object moves toward the foreground of an image, more wavelet coefficients are used to display finer detail [32].

Developing model hierarchies for robust analysis and simulation has its own set of requirements. The model description must grow reasonably as problem size increases. In computer graphics, all problems are in 2 or 3 dimensions. The uncertainty in the model must also be explicitly quantified. For general nonlinear system, the PL modeling framework meets both of these requirements.

The third, and most important, requirement is consistency. Each model in the set must accurately describe the system. Specifically, given the true model of the system

$$\begin{aligned} x[k+1] &= f(x[k], u[k]) \\ y[k] &= g(x[k], u[k]) \end{aligned}, u \in U, x \in X$$

and the set of models

$$\begin{aligned} x_i[k+1] &= f_i(x_i[k], u[k], w_i[k]) \\ y_i[k] &= g_i(x_i[k], u[k], w_i[k]) \end{aligned}, u \in U, x_i \in X_i, w_i \in W_i$$

then for each i there must exist values of w_i that if $x_i[0] = z_i(x[0])$ then $y_i[k] = y[k]$ for all k and all possible inputs u. The function z_i is a mapping that allows for differing state dimension and coordinates frames. Simply stated, with the appropriate additional input signal w_i, each model in the set will yield the true output y when driven by the input u. However, in practice, one never knows the true model.

This is, in essence, the model validation problem. For PL systems, verifying consistency is NP-hard. However, sets of consistent models can be constructed by using PL covers. Given a nonlinear system in the LFT form (6.1), any PL cover, by definition, satisfies

$$\hat{N}(\hat{y}, n) = N(\hat{y}) \tag{7.1}$$

for some n. Letting z_i be the identity, $y_i = y$ trivially when n satisfies (7.1) and the PL cover is consistent with the original nonlinear system.

This method for constructing consistent sets of models does not allow for varying state dimension or coordinate frames. It requires that all models in the hierarchy have identical linear portions $M(z)$. Even though the state dimension is fixed, different PL covers can greatly affect computation time. As shown in Chapter 6, the nonlinear dimension is the key parameter affecting computations. The PL cover allows selection of ν.

The hierarchy for a set of PL covers is determined by ν, l, and the amount of noise needed to create the cover. However, none of these measures gives an absolute ordering. A model with $l = 3$ can yield much better simulation results than a model with $l = 200$ if it captures the important nonlinearities. Similarly, the regions where noise is added are as important as the amount of noise added.

7.2 Hierarchy Examples

As previously mentioned, verifying that two models are consistent is very hard, while constructing consistent models is not. To demonstrate this, two examples of model hierarchies for common nonlinearities are presented. The saturation example constructs a set of PL models that cover a continuous nonlinearity with varying amounts of additive noise. In the quantization example, all models have the same amount to noise, but different regions of state space have less noise added. While these examples do not address the issues of changing state dimension, they demonstrate the ideas of hierarchical modeling, including localizing where a model is accurate.

Saturation

Though the smooth saturation nonlinearity

$$y = \frac{2}{\pi} \arctan u, \tag{7.2}$$

shown by the solid line in Figure 7.1, is not piecewise linear over any region, it can be closely approximated by a PL system. Since the output of (7.2) is bounded by -1 and 1, the trivial cover

$$y = 0u + n, |n| \leq 1 \tag{7.3}$$

is a consistent model of the saturation. This cover, which uses noise to account for the entire nonlinear behavior, is identical to (6.5), the cover for the cosine bump from Section 6.2. While (7.3) does not describe the behavior of the nonlinearity, it does describe its bounds. This cover is useful when the saturation has little effect on the overall system.

To develop PL covers with multiple regions, assumptions on the covers' forms are needed. First, all regions of the cover will have the same amount of additive noise, and this amount will be minimized. Second, the cover will be symmetric about $u = 0$. Using these assumptions,

$$y = \begin{cases} -0.5 + 0.5n & u < 0 \\ 0.5 + 0.5n & u \geq 0 \end{cases}, |n| \leq 1 \tag{7.4}$$

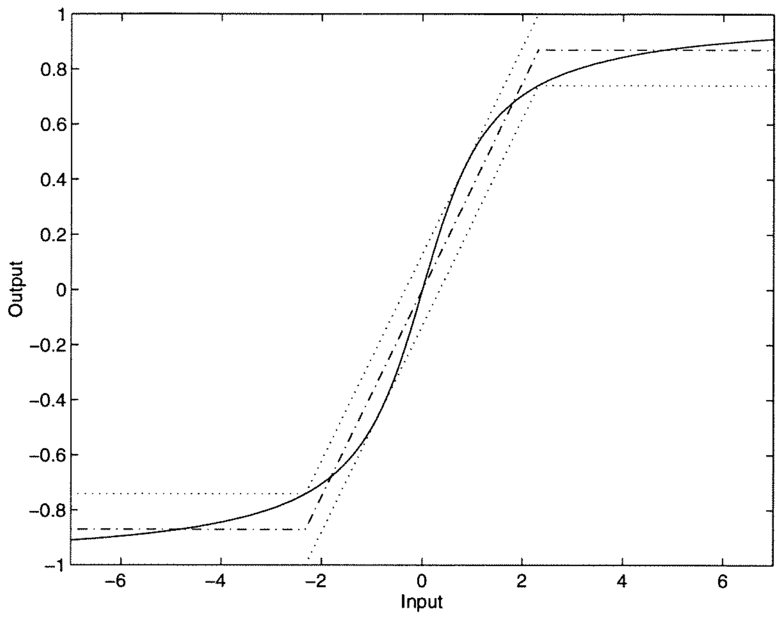


Figure 7.1: The arctan saturation and a 3-segment PL cover.

is the cover containing two regions. This cover gives only slightly more information than (7.3). The output of this cover depends only upon the sign of the input.

A more common approximation uses three segments to approximate the saturation. This cover, shown in Figure 7.1, is given by

$$y = \begin{cases} -0.87 + 0.13n & u < -2.32 \\ 0.38u + 0.13n & -2.32 \leq u \leq 2.32 \text{ , } |n| \leq 1. \\ 0.87 + 0.13n & u > 2.32 \end{cases} \tag{7.5}$$

The dotted lines indicate the extreme values obtainable for any value of n. As required by a PL cover, there is always a noise that lets the cover equal the output of the true system. Unlike (7.4), this cover is continuous and the nominal value is smooth in a neighborhood about $u = 0$. Note that the saturated regions, $|u| > 2.32$ are not nominally equal to the saturated values. They are offset to minimize the amount of noise needed to cover the true saturation (7.2).

While increasing the number of regions reduces the amount of noise needed, the reductions do not follow any pattern. Cover (7.5) has roughly half the noise

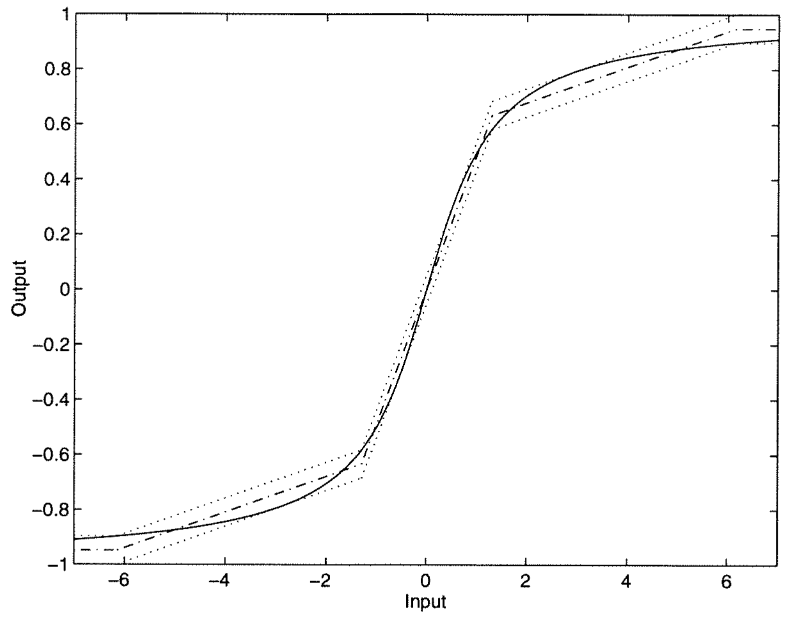


Figure 7.2: 5-segment saturation approximation.

of (7.4), which has half the noise of (7.3). However, the four segment cover

$$y = \begin{cases} -0.89 + 0.11n & u < -2.78 \\ 0.28u - 0.11 + 0.11n & -2.78 \leq u < 0 \\ 0.28u + 0.11 + 0.11n & 0 < u \leq 2.78 \\ 0.89 + 0.11n & u > 2.78 \end{cases}, |n| \leq 1 \tag{7.6}$$

only reduces the required noise by roughly 20%. The five segment cover, shown in Figure 7.2,

$$y = \begin{cases} -0.95 + 0.05n & u < -6.14 \\ 0.06u - 0.55 + 0.05n & -6.14 \leq u < -1.29 \\ 0.49u + 0.05n & -1.29 \leq u \leq 1.29 \\ 0.06u + 0.55 + 0.05n & 1.29 < u \leq 6.14 \\ 0.95 + 0.05n & u > 6.14 \end{cases}, |n| \leq 1 \tag{7.7}$$

requires 55% less noise than (7.6).

These five covers are consistent representations of the original nonlinearity (7.2). Covers with larger l require less noise, but may require more computation during robust simulation. For this example, a natural hierarchy is the number of regions or the amount of added noise. Since adding a region to each cover reduces the required additive noise over the entire input space, using more regions will generally yield a less conservative result. However, the improvement from using more complex covers cannot be predicted solely from the amount of noise added. It is also a function of the cover's interaction with the rest of the system being simulated.

Quantization

Due to the finite resolution of digital systems, quantization nonlinearities occur in almost every application. Usually, they are ignored since the errors introduced are generally small. Using the PL modeling framework, their exact behavior can be easily incorporated into system models when appropriate.

As shown in Section 6.6, quantization is represented exactly by a PL system with an infinite number of regions. A more general form of the quantization nonlinearity is

$$y = u - (u - q_c) \bmod q_r - q_c, \quad -q_r < q_c \leq 0 \tag{7.8}$$

where q_r is the difference between quantization levels and q_c is used to set the center of each quantized value. Even when u is bounded, the exact representation can have a large number of regions. For example, if $q_r = 0.1$, then a PL mapping valid for $|u| < 10$ requires 200 PL regions.

A much simpler model for the quantizer is

$$y = u - q_c - \frac{q_r}{2} + \frac{q_r}{2}n, \quad |n| \leq 1. \tag{7.9}$$

Like (7.3), this cover completely ignores the nonlinearity by covering it with noise. More accurate covers treat the nonlinearity exactly in some region, and cover the rest of the nonlinearity with noise. For example, the approximation

$$y = \begin{cases} u - q_c - \frac{q_r}{2} + \frac{q_r}{2}n, & x < q_c, |n| \leq 1 \\ u - (u - q_c) \bmod q_r - q_c, & q_c \leq x \leq q_r + q_c \text{ , } |n| \leq 1 \\ u - q_c - \frac{q_r}{2} + \frac{q_r}{2}n, & x > q_r + q_c \end{cases} \tag{7.10}$$

captures the nonlinearity exactly in the range $q_c \leq x \leq q_r + q_c$.

Other covers are formed by capturing the nonlinearity exactly in different regions of the input space. Each cover has the same maximum amount of additive noise, $q_r/2$. However, some regions in each cover require no noise. By carefully selecting the noise-free regions, simulation results can be improved without excessive additional computations.

7.3 Multiresolution Simulation

Ideally, additional detail is added to models only where it is needed. Generally, this added detail yields better simulation results at the expense of additional computations. However, modeling detail may be needed in different regions of the model at different times during the simulation. Including all details at every time step may give excellent simulation results, but may also require excessive computations. Multiresolution simulation is one way of addressing this problem.

Instead of using a single model for the entire simulation, the model is selected dynamically during the simulation process. Model details are included only where and when they are needed. While multiresolution simulation may reduce errors with reasonable computational cost, two problems are introduced. First, when switching models, the current state of one model must be used to initialize another model. If the models have different state dimensions, this is not trivial. Second, the results must be properly interpreted. One must understand how results change when models with less fidelity are simulated.

Robust simulation addresses both of these problems. When switching between models that share the same state variables, initialization is trivial. At each time step, robust simulation returns the set of all possible states. As long as each model in the hierarchy is a consistent cover, any model can used at any time step and the result is guaranteed to contain the true result. Typically, models with less additive noise give smaller regions of state space during robust simulation. Since the final result for simulation of any model in the hierarchy is a set of points reachable points, interpreting the results is straightforward.

The result is the set of all possible final conditions. More accurate models should return smaller possible regions.

While robust simulation greatly simplifies the implementation and interpretation of multiresolution simulation, many details are not trivial. When switching between covers with different state spaces, mappings between the state spaces are needed. If the state dimension increases, then the mapping must initialize new states. To maintain the analysis guarantees of robust simulation, this initialization must return the set of all possible values of the new states.

7.4 Examples

Hierarchical modeling and multiresolution simulation fit naturally within the PL modeling and robust simulation framework. To demonstrate these ideas, three examples are presented. The first two examples show how different models can yield acceptable results in one simulation and unacceptable results in another. The third example demonstrates multiresolution simulation. By switching to a more accurate model during the middle of the simulation, results are greatly improved with only minor added computational cost.

In Section 7.2, five covers were developed for (7.2). While some covers barely resembled the original saturation nonlinearity, they still may give acceptable results. Consider the stable system

$$x[k+1] = 0.4x[k] + 0.05\frac{2}{\pi}\arctan u[k] \tag{7.11}$$

with the feedback law $u[k] = -x[k]$. This system is open loop stable, and the input has very little control authority. All simulations are for 5 time steps starting from $x[0] = 100$.

The first simulation of (7.11), whose results are shown in Table 7.1, uses (7.3) to approximate to cover the saturation. For this system, covering the nonlinearity by noise gives reasonable results. The final result, $0.942 \leq x[5] \leq 1.107$, differs by only 5% from its center value and, as expected, contains the exact result of 0.956. Note that for the first few time steps, the lower bound

time step	lower bound	exact	upper bound
0	100	100	100
1	39.95	39.95	40.05
2	15.93	15.93	16.07
3	6.32	6.32	6.48
4	2.47	2.48	2.64
5	0.94	0.96	1.11

Table 7.1: Simulation results for cover (7.3).

cover	l	lower bound	upper bound
(7.3)	1	0.942	1.107
(7.4)	2	0.942	1.024
(7.5)	3	0.942	0.963
(7.6)	4	0.946	0.964
(7.7)	5	0.953	0.961
exact		0.956	

Table 7.2: Ranges for $x[5]$ for various covers.

is, after rounding, identical to the exact value. This is because the worst case noise, which yields the lower bound on the range, is also the value needed to make the approximation match the true model.

Table 7.2 summarizes the simulation results for each cover. Covers (7.3), (7.4), and (7.5) all yield the same lower bound. This is because the set of feasible states is always in the first region of these covers and all covers allow -1 as a possible output. The upper bound varies because the range of possible outputs in the regions varies. When simulating with (7.4), the saturation output lies in the range $-1 \leq y \leq 0$. When simulating with (7.5), the range is reduced to $-1 \leq y \leq -0.74$. By reducing the additive noise in the cover, the set of possible states is reduced.

The robust simulation results for (7.6) demonstrate two additional features of PL covers. First, the set of reachable states lies in two different regions of the cover during the simulation. This leads to the tighter lower bound. Second,

Saturation Model	$\min x[50]$	$\max x[50]$
Model (7.3)	-284	2170
Model (7.4)	-12.5	998
Model (7.5)	-0.464	137
Model (7.6)	-3.4 e-4	90.0
Model (7.7)	-0.128	0.128
Exact	5.97 e-11	6.50 e-8

Table 7.3: Simulation results for the unstable system.

the upper bound for this cover is larger than the upper bound for (7.5). The set of nonlinearities contained in (7.6) is not a subset of (7.5). For example, at $u = -2.5$, (7.6) allows outputs of $-0.92 \leq y \leq -0.70$ while (7.5) allows outputs of $-1 \leq y \leq -0.74$. However, since both simulations generate sets of all reachable states, the intersection of the results is also a set of all reachable states. Thus, the results from the simulations using (7.5) and (7.6) can be combined to yield the tighter result $0.946 \leq x[5] \leq 0.963$.

The five segment cover (7.7) has the tightest results. This is expected, since this cover adds much less noise than any of the other covers. For this simulation, all five covers give good results. This is because the nonlinearity does not play a large role in the output of the system.

In the second example, the nonlinearity is crucial and different covers yield dramatically different results. The nominally unstable system

$$x[k+1] = 1.1x[k] + \frac{2}{\pi} \arctan u[k] \tag{7.12}$$

is stabilized about the origin by the feedback law $u[k] = -x[k]$. What values can $x[50]$ attain if $7.5 \leq x[0] \leq 8.5$? Note that for $x > 9.32$, the system is unstable.

Table 7.3 shows the robust simulation for each of the five saturation covers. Though covers (7.5) and (7.6) appear to cover the nonlinearity without excessive noise, they are too conservative for this simulation. The true system maps the range $7.5 \leq x[0] \leq 8.5$ to a small region near the origin. These covers do not. Only (7.7) clearly demonstrates that the system is stable for all

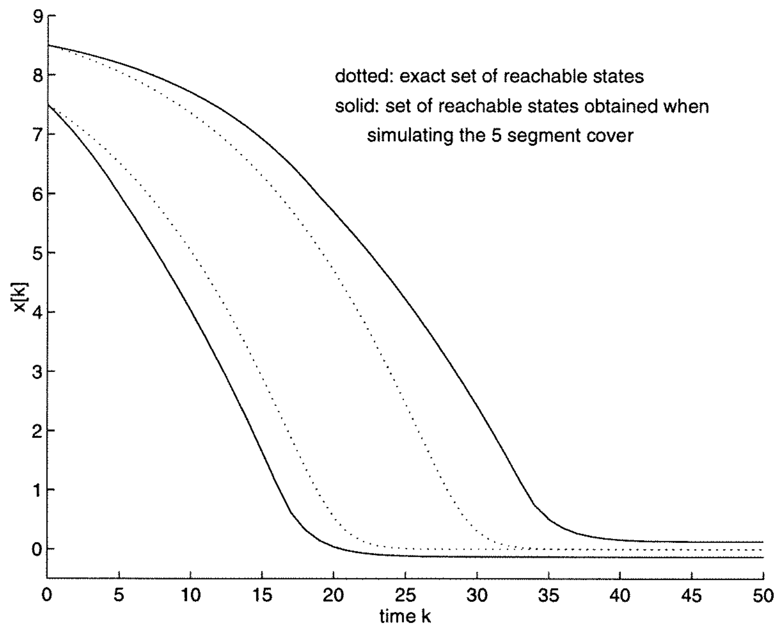


Figure 7.3: Comparison of exact simulation and cover (7.7).

initial conditions in the range. As shown in Figure 7.3, the robust simulation results for (7.7) are somewhat conservative. At every time step, the range of reachable states always contains the exact solution, as required.

Figure 7.3 also demonstrates one limitation of the modeling framework. Since the noise can be a constant input, the simulation never converges to a single equilibrium point. At best, the result is a ball about an equilibrium point. For this example, $x = 0$ is an equilibrium point and, as shown by Figure 7.3 and Table 7.3, simulation using (7.7) converges to the set $|x| \leq 0.128$. This value can also be derived by examining the behavior of the cover around the equilibrium point.

The set of all possible equilibrium points is obtained by setting $x[k+1] = x[k]$ and finding the set of x that satisfy this requirement for some value of n. For (7.12) approximated using (7.7), this is given by

$$x[k] = 1.1x[k] - 0.49x[k] + 0.05n, \quad |n| \leq 1.$$

Simple algebra yields $|x[k]| \leq 0.128$, which is the result obtained by robust simulation. It is impossible to find a tighter bound using this cover.

A third example demonstrates multiresolution simulation. Consider the nominally unstable system with a quantized input

$$x[k+1] = 1.9x[k] + q(u[k]) \tag{7.13}$$

where $q(x) = x - (x+0.05) \bmod 0.1 + 0.05$. While q can be exactly represented as a PL mapping, it has 10 PL regions for every unit of state space. A model valid over the range $|x| \leq 100$ has 2000 regions.

Using (7.9), a simple cover for the quantizer is

$$q(x) = x + 0.05n, \ |n| \leq 1. \tag{7.14}$$

Closing the loop with negative unity feedback and using the cover (7.9) for the quantization results in the stable system

$$x[k+1] = 0.9x[k] + 0.05n[k], \ |n[k]| \leq 1 \ \forall k. \tag{7.15}$$

Robust simulation of the initial condition set $|x[0]| \leq 100$ for 150 time steps using (7.15) yields $|x[150]| \leq 0.50$. This is the smallest result obtainable when robustly simulating (7.15). The exact solution, obtained by robustly simulating the 2000 segment exact PL model of the quantization, is $|x[150]| \leq 0.095$.

However, calculating the exact solution is incredibly expensive. Simulating the exact model for one time step requires more than ten times the computations used during the entire 150 time step simulation of (7.15). While the 2000 segment model is very accurate, the additional accuracy comes at great cost.

Since the coarse simulation converges to the region $|x| \leq 0.50$, it is likely that more detail is needed in this range. The thirteen segment PL cover, presented in Table 7.4, exactly captures the quantization over the range $-0.55 \leq x < 0.55$. Robust simulation using this cover yields $|x[150]| \leq 0.095$, the exact solution. This simulation requires roughly nine times as many computations as when simulating with (7.15).

Substantially fewer computations are required when the simulation changes the quantizer cover midway through the run. After simulating the first 100 steps using (7.9), the range $|x[100]| \leq 0.51$ is obtained. This region is then

$$q(x) = \begin{cases} x + 0.05n, & x \geq 0.55,\ |n| \leq 1 \\ 0.5, & 0.45x \leq x < 0.55 \\ 0.4, & 0.35x \leq x < 0.45 \\ 0.3, & 0.25x \leq x < 0.35 \\ 0.2, & 0.15x \leq x < 0.25 \\ 0.1, & 0.05x \leq x < 0.15 \\ 0, & -0.05x \leq x < .05 \\ -0.1, & -0.15x \leq x < -0.05 \\ -0.2, & -0.25x \leq x < -0.15 \\ -0.3, & -0.35x \leq x < -0.25 \\ -0.4, & -0.45x \leq x < -0.35 \\ -0.5, & -0.55x \leq x < -0.45 \\ x + 0.05n, & x < -0.55,\ |n| \leq 1 \end{cases}$$

Table 7.4: Thirteen segment quantization cover.

used to initialize a simulation using the thirteen segment cover. Simulating the last 50 time steps with the less conservative model gives the final result of $|x[150]| \leq 0.095$. This is less than one fifth the size of the result when only using (7.9) and is also the exact solution.

This multiresolution simulation requires less than half the computations used when simulating solely with the thirteen segment cover. Simulation using (7.9) reduces calculations by more than a factor of ten during the first 100 time steps. As shown by this example, the use of model hierarchies and multiresolution simulation can greatly reduce computation time without affecting results.

Chapter 8
Other Applications

Robust simulation and PL modeling can be applied to a variety of other problems. One view of PL analysis is that it is a way to generate bounds for non-convex optimizations. The first application, done in collaboration with James Primbs, uses this idea to generate lower bounds for model predictive control design. Robust simulation can also be used to examine nonlinear stability. Since all trajectories are calculated, unstable ones can be identified. This is the topic of the second application. The third application discusses the problem that motivated the development of the robust simulation algorithm, gain scheduling. Many other uses exist for robust simulation and PL modeling. This chapter introduces three of them.

8.1 Model Predictive Control Lower Bounds

Model predictive control (MPC), also known as receding horizon control, is a technique where an on-line, open-loop control problem is solved at each time step [14]. Using the current state, an input sequence is calculated to minimize a cost while satisfying specified constraints. Only the first element of the sequence is used, and then the algorithm is applied again, beginning at the new current state. For general nonlinear systems, this technique results in a constrained non-convex optimization [24]. At each time step, a local minimum of the cost is found, but no information about the global minimum is provided.

By restricting the class of systems considered, iterative robust simulation can be used to find a lower bound on the global minimum. If the MPC cost and the robust simulation bound are similar, then additional optimization will yield little benefit. If they differ greatly, then additional optimization may yield large improvements. Robust simulation results also generate initial conditions for additional optimization runs, which will generally converge to a better local minimum.

Problem statement

At each time step, MPC control algorithms attempt to solve

$$\begin{aligned} x[k+1] &= f(x[k], u[k]) \\ \min_u &\, J(x,u). \end{aligned} \tag{8.1}$$

In some special cases, this is a convex optimization that can be solved exactly. For most problems, including those with nonlinear f and constraints on x and u, a global minimum cannot be found.

The goal of this algorithm is to find a cost J_{lb} such that

$$0 \leq J_{lb} < J_{opt} \leq J_{mpc}$$

where J_{opt} is the global solution to (8.1) and J_{mpc} is a local solution to (8.1). A secondary goal of the algorithm is to reduce the gap between J_{lb} and J_{mpc}. This problem is very similar to the robust analysis problem solved in Chapter 5. The main difference is that for MPC, the cost is being minimized. J_{mpc} is the cost from a single, known feasible trajectory and is analogous to the robust analysis lower bound. J_{lb} is analogous to the robust analysis upper bound.

In order to apply robust simulation analysis techniques to the MPC problem, the problem must be cast in the PL setting. This setting has two main differences from the standard MPC form. First, polyhedral inputs constraints are required. Generic MPC allows arbitrary inputs. Second, the cost being minimized must be suitable for robust simulation. MPC typically minimizes a quadratic cost. For robust simulation, the tube cost (or l_∞ norm) is required and no input penalty is assessed.

Algorithm

By definition, there are no feasible trajectories with $J \leq J_{opt}$ and at least one trajectory exists that attains J_{opt}. If it can be shown that no feasible trajectories exist with cost less than or equal toJ_{lb}, then J_{lb} is a lower bound on J_{opt}. This is accomplished by iterative robust simulation.

The first step is to construct an MPC control sequence and calculate its cost, J_{mpc}. Since this is a feasible trajectory, it provides an upper bound on J_{opt}. Initially, $J_{lb} = 0$, since 0 is always a lower bound.

The second step is to choose a cost J_{nom}, $J_{lb} \leq J_{nom} < J_{mpc}$, and determine if any feasible trajectories meeting that cost exist. This is accomplished using a modified robust simulation algorithm that calculates the set of states that can be reached at each time step while satisfying the cost constraint.

Since level sets of the tube cost are polyhedra, the set of reachable states meeting the cost constraint is also a set of polyhedra. Denoting the set of all states meeting the cost constraint by $\mathcal{S}_{J_{lb}[k]}$, at each time time step the set of states reachable at time $k+1$ is found by mapping the set $\widehat{\mathcal{S}_k} \cap \mathcal{S}_{J_{lb}[k]}$. If this set is empty, no feasible trajectory satisfying $J \leq J_{nom}$ exists and the robust simulation algorithm step is terminated.

If no trajectory satisfying $J \leq J_{nom}$ exists, then J_{nom} is a lower bound on J_{opt}. If robust simulation generates potential trajectories that meet the cost constraint, then two options exist. Either the results from the robust simulation can be used to find a better MPC sequence (reduce J_{mpc}) or the robust simulation step can be repeated with a smaller J_{nom}. The algorithm terminates when one is satisfied with J_{lb}. Typically, this is when $J_{mpc} - J_{lb}$ is small.

Example

Consider the piecewise linear system

$$x[k+1] = \begin{cases} x[k] + 0.15u[k], & x[k] < 1 \\ x[k] + u[k], & x[k] \geq 1 \end{cases}$$

with input constraints $|u[k]| \leq 1$ and the MPC problem of minimizing the cost

$$J = \max\{0.1x[1], x[2]\}$$

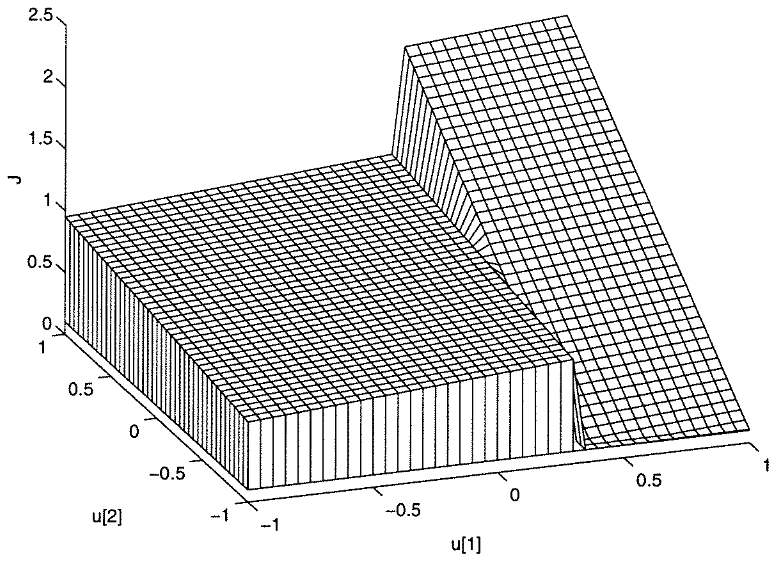


Figure 8.1: Cost J as a function of $u[1]$ and $u[2]$.

subject to $x[0] = 0.95$. The cost, J, as a function of the control inputs $u[1]$ and $u[2]$ is shown in Figure 8.1. Any optimization scheme using a gradient descent algorithm and starting from $u[1] < 1/3$ will fall into the local minimum at $u[1] = u[2] = -1$. MPC based control schemes are vulnerable to failures of this sort in their optimization step [27].

The local minimum at $u[1] = u[2] = -1$ has a cost $J = 0.65$. This is more than a factor of 6 greater than the global minimum cost of 0.1, which occurs at $u[1] = 1/3, -1 \leq u[2] \leq -0.9$.

Starting the MPC optimization from $u[1] = u[2] = 0$ converges to the local minimum, resulting in $J_{mpc} = 0.65$. Iterative robust simulation gives a lower bound of $J_{lb} = 0.099$. The gap between J_{mpc} and J_{lb} indicates that the MPC solution may only be a local minimum.

To improve J_{mpc}, branch and bound is used. A single branch on $u[1]$ yields $J_{lb} = 0.5$ for $u[1] < 0$ and $J_{lb} = 0.09$ for $u[1] \geq 0$. During the branch and bound step, no attempt is made to find the largest J_{lb}. The purpose is to find good initial conditions for MPC optimization. From these results, it is likely that the optimal solution has $u[0] \geq 0$.

A point in the middle of the likely region is used to initialize the MPC optimization. Starting a gradient descent algorithm from $u[1] = 0.5, u[2] = 0$ quickly converges to the global minimum, giving a new cost $J_{mpc} = 0.1$. Since $J_{lb} \approx J_{mpc}$, further optimizations will yield little benefit.

While this is a simple optimization problem that is easily solved by gridding the parameter space, it demonstrates the functionality of the algorithm. Robust simulation indicated that the MPC solution might be caught at a local minimum. Through branch and bound, a better initial condition was obtained and the MPC solution was greatly improved.

8.2 PL Stability

In [35], Sontag conjectured that verifying the stability of PL systems was undecidable. Though exact stability conditions cannot be computed, separate necessary and sufficient conditions have been developed. Two fundamentally different approaches have been taken. One set of conditions is based upon LMIs. The second set of conditions is based upon robust simulation.

The LMI approach requires finding a single, quadratic Lyapunov function for the PL system. However, for many stable systems, no quadratic Lyapunov functions exist and this technique is inconclusive [18]. Recently, the existence of a piecewise quadratic Lyapunov function has been cast as the solution to an LMI [17]. While piecewise quadratic Lyapunov functions are a large improvement, they still suffer from the same limitations. Many stable PL systems do not admit piecewise quadratic Lyapunov functions. These algorithms are also single step methods. When the results are inconclusive, nothing can be done to improve them.

The robust simulation based method does not have these limitations. When results are inconclusive, less conservative robust simulation algorithms can be used. This method examines all feasible trajectories over a finite time horizon. Though infinite time horizon problems may be undecidable, finite time horizon problems are generally decidable. If the trajectories satisfy certain conditions, stability (or instability) is inferred.

Robust simulation based algorithm

The robust simulation based algorithm examines generalized finite time horizon stability, specifically:

> Do all states in a set of convex regions $\mathcal{S}_0$ map into a set of convex regions $\mathcal{S}_f$ in t time steps?

With a careful choice of $\mathcal{S}_f$, stronger notions of stability can be inferred. If $\mathcal{S}_f$ is known to be invariant, then Lyapunov stability is implied. If $\mathcal{S}_f$ is known to be invariant and a quadratic Lyapunov function exists for $\mathcal{S}_f$, then quadratic stability is implied. The latter case occurs frequently, often when $0 \subset \mathcal{S}_f \subset R_i$ for a fixed i. Answering this question is a direct application of robust simulation, which calculates all possible trajectories for a set of initial conditions.

A sufficient condition for generalized stability is obtained by robustly simulating $\mathcal{S}_0$ for t time steps, yielding the set of all reachable states, $\mathcal{S}_t$. If $\mathcal{S}_t \subseteq \mathcal{S}_f$, then the stability condition is satisfied. Since robust simulation is conservative, $\mathcal{S}_t$ also contains states that cannot be reached from $\mathcal{S}_0$. This condition only guarantees stability; if it is not met, nothing can be inferred.

A necessary condition for generalized stability is found by robustly simulating $\mathcal{S}_f$ for t time steps backwards in time, yielding the set $\mathcal{S}_{f-t}$. If $\mathcal{S}_0 \not\subseteq \mathcal{S}_{f-t}$, then the system is unstable. When $\mathcal{S}_0 \not\subseteq \mathcal{S}_{f-t}$, some point in $\mathcal{S}_0$ does not map into $\mathcal{S}_f$ after t time steps. Since $\mathcal{S}_{f-t}$ contains states that do not map into $\mathcal{S}_f$, this is only a necessary condition.

If the sufficient condition fails and the necessary condition is satisfied, no conclusion can immediately be drawn. In this case, more accurate robust simulation is needed. Since robust simulation gives the exact answer after a finite number of refinements, stability can always be verified in finite time. However, computations grow exponentially as the robust simulation algorithm is refined and calculating the exact solution can be prohibitively expensive.

Example

Bounded noise cannot destabilize a stable linear system. For a nonlinear system, the amount of noise can affect stability. This example demonstrates that a PL system can be destabilized by the addition of bounded noise. While the result is not surprising, this example provides a test of the stability algorithms and illustrates the difference between nominal stability and robust stability of PL systems.

Consider the PL system

$$x[k+1] = \begin{cases} A_0 x[k] & \text{if } |x_1| \leq 0.1, \\ A_1 x[k] + \bar{A}_1 + B_1 n[k] & \text{if } x_1 < -0.1, \\ A_2 x[k] + \bar{A}_2 & \text{if } x_1 > 0.1, \end{cases}$$

where

$$A_0 = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix}, \; A_1 = A_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; \bar{A}_1 = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}, \; \bar{A}_2 = \begin{bmatrix} -0.1 \\ 0.1 \end{bmatrix}.$$

For the stability question, the initial region, $\mathcal{S}_0$, is the rectangle $\|x\|_\infty \leq 10$ and the final region, $\mathcal{S}_t$, is the rectangle $\|x\|_\infty \leq 0.1$. The number of time steps, t, is 100.

With no noise ($B_1 = [0 \; 0]^t$), the system satisfies the robust simulation based sufficient condition and is stable. As shown in Figure 8.2, flows starting with $|x_1| > 0.1$ move towards $x_1 = 0$, and then converge to the origin. In fact, once $|x_1| \leq 0.1$, the state converges to the origin quadratically.

When bounded noise is added as $B_1 = [0.1 \; 0]^t$, the system is destabilized. By choosing an initial condition with $x_1 < -0.1$ and a noise signal $n[k] = -1 \; \forall k$, the system diverges, as shown in Figure 8.3. This system fails both the sufficient condition and the necessary condition.

8.3 Gain Scheduling

Success with gain scheduled controllers motivated the initial development of robust simulation. On the Caltech ducted fan experiment, gain scheduled

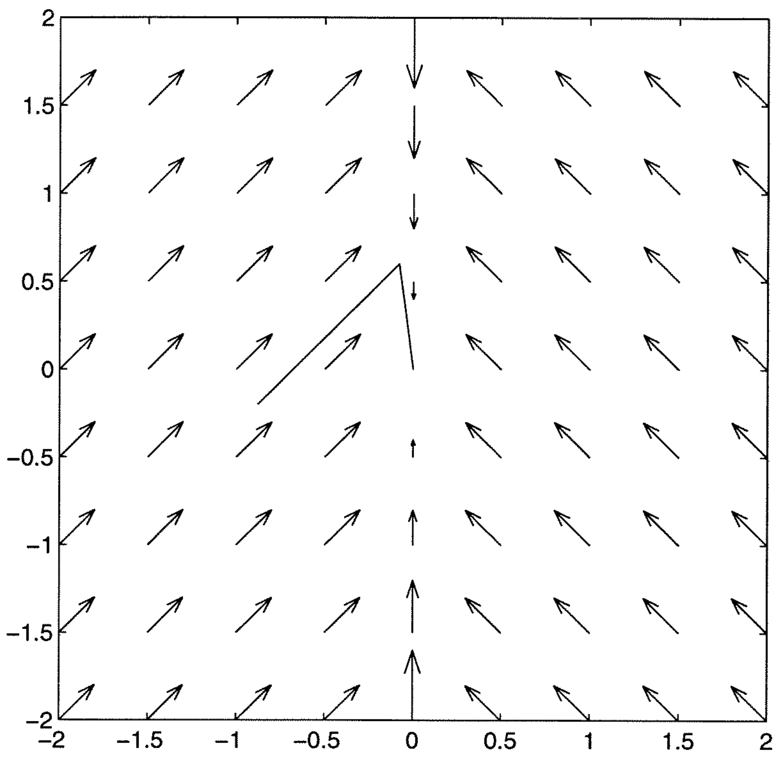


Figure 8.2: PL system trajectory without noise.

controllers were both easy to design and had exceptional performance [19]. Repeated experiments demonstrated that results were insensitive to the type of switching used. Both smooth transitions between controllers and abrupt changes yielded similar results. When using abrupt switching, simplified models of the closed loop system are piecewise linear.

While robust simulation of the simplified PL systems gave insight into the nonlinear performance, it did not give guarantees. The development of PL covers allowed for exact analysis of gain scheduled systems. When the gain scheduled controller switches abruptly, the controller is a PL system. The interconnection of this controller with a PL cover of the nonlinear plant results in a PL system, which can be analyzed through robust simulation. If smooth switching is desired, the controller can also be approximated using a PL cover.

This approach to analyzing gain scheduled systems differs greatly from other existing methods. Robust simulation explicitly considers the nonlinearities' dependence upon the state variables. LPV techniques ignore the state

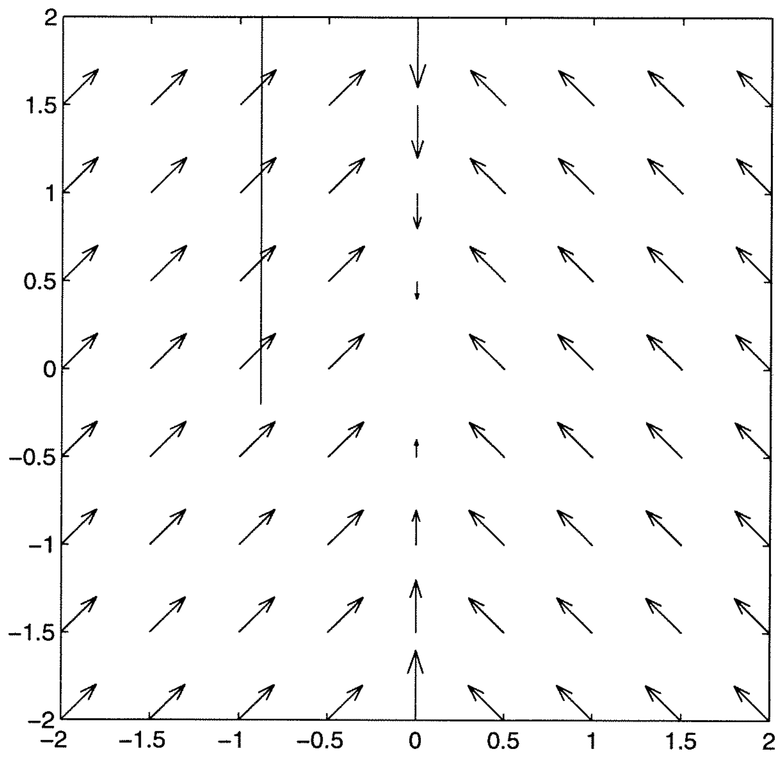


Figure 8.3: PL system trajectory with noise.

dependence by allowing the scheduling variables to change arbitrarily quickly. LPV techniques that bound the parameters' rates of variation exhibit exponential computational growth in the number of parameters. Other techniques require that the scheduling variable vary slowly [31].

The PL modeling framework also greatly simplifies the modeling process. Both slowly varying and LPV techniques place requirements on the form of the system. Generally, the model must be differentiable. PL models are allowed to be discontinuous. This facilitates the combination of experimentally identified models and theoretically derived models. When better information is obtained, one simply adds additional regions to the model. No curve fitting or parameter tuning is needed.

Though robust simulation has not been used to analyze any large, physically motivated gain scheduled systems, it provides a solid foundation. The key limitation, especially when compared with LPV, is that it does not provide a method for control synthesis.

8.4 The Caltech Ducted Fan

Thus far, robust simulation has only been applied to relatively simple systems. This example applies these techniques to the simplified dynamics of the Caltech Ducted Fan experiment, which is described in more detail in [19]. By examining this more complex, physically motivated example, insight into the algorithm's practical value is obtained. In addition, some limitations of the current approach are demonstrated.

Simplified continuous time dynamics are given by

$$\begin{aligned}
\ddot{x}_1 &= -d\dot{x}_1 + \frac{u_1 \cos x_3 - u_2 \sin x_3}{m} \\
\ddot{x}_2 &= -g + \frac{u_1 \sin x_3 + u_2 \cos x_3}{m} \\
\ddot{x}_3 &= -\frac{mgl}{J} \sin x_3 + \frac{r}{J} u_1
\end{aligned} \tag{8.2}$$

where g is the acceleration due to gravity and d, l, m, r, and J are physical parameters determined by the experiment's mass and geometry. Full state output is assumed. Though not explicitly shown, inputs u_1 and u_2 have magnitude saturations.

The first limitation encountered is that robust simulation cannot be applied to continuous time systems. Some method is needed to convert (8.2) to a discrete time PL system. To achieve this, a time step of 0.05 seconds is chosen and Euler discretization is applied. In addition, cosine and sine are replaced by piecewise linear approximations. These dynamics are given by

$$\begin{aligned}
x_1[k+1] &= x_1[k] + 0.05 x_4[k] \\
x_2[k+1] &= x_2[k] + 0.05 x_5[k] \\
x_3[k+1] &= x_3[k] + 0.05 x_6[k] \\
x_4[k+1] &= x_4[k] + 0.05 \left(-d x_4[k] + \frac{u_1[k] \operatorname{c_{PL}} - u_2[k] \operatorname{s_{PL}}}{m} \right) \\
x_5[k+1] &= x_5[k] + 0.05 \left(-g + \frac{u_1[k] \operatorname{s_{PL}} + u_2[k] \operatorname{c_{PL}}}{m} \right) \\
x_6[k+1] &= x_6[k] + 0.05 \left(-\frac{mgl}{J} \operatorname{s_{PL}} + \frac{r}{J} u_1 \right)
\end{aligned} \tag{8.3}$$

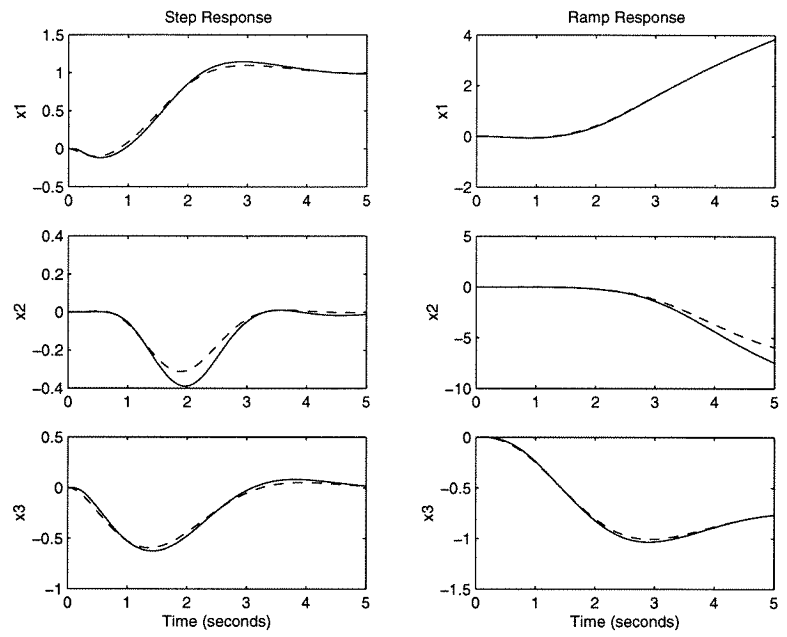


Figure 8.4: Comparison of continuous and discrete models.

where c_{PL} and s_{PL} are 16-segment PL approximations of $\cos x_3[k]$ and $\sin x_3[k]$ defined over the range $-1 \leq x_3[k] \leq 1$. The resulting model has 144 regions and $\nu = 3$. The saturations on u_1 and u_2 each add one nonlinear direction. Since both c_{PL} and s_{PL} are functions of x_3, only one additional nonlinear direction is required.

It should be noted that (8.3) is an approximation of (8.2); it is not a cover. To verify that both models have similar behavior, a single linear controller was designed and several trajectories were simulated. Two such trajectories, a step and a ramp, are shown in Figure 8.4. For both trajectories, the discrete time simulation results, shown by solid lines, nearly match those of the original model, shown by dashed lines. The behavior for one model qualitatively describes the behavior of the other. However, guarantees for (8.3) do not give guarantees for (8.2). To make the results more meaningful, a discretization technique that preserves guarantees is needed.

Given (8.3), robust simulation is used to predict the worst possible step performance when starting in the region

$$\left\| \begin{bmatrix} 10 & 10 & 20 & 20 & 20 & 50 \end{bmatrix} x[0] \right\|_\infty \leq 1.$$

Time horizons of 1 second (20 time steps) and 5 seconds (100 time steps) are considered. In both cases, $||x_1[t_f]||_\infty$ is the performance measure.

After roughly 15 time steps, the robust simulation algorithm diverged. Due to limitations of the algorithm implementation, described below, the degree of approximation γ could not be increased beyond 1 for this model. Branch and bound also did not converge within a reasonable number of branch steps. To better understand the limitations, a simpler model was studied.

One key limitation of the algorithm implementation is its memory management. The implementation allocates memory for all $l^{\gamma+1}$ possible volumes in advance. It assumes that one region can map into every other region of the system. While this is true for generic PL systems, discretizations of continuous time systems typically map only into adjacent regions. If the regions are created by gridding the nonlinear dimension, as is the case with this example, each region has $3^\nu - 1$ neighbors and robust simulation requires at most $3^{\nu\gamma}l$ volumes. Since most volumes are generally empty, memory requirements can be further reduced by dynamically allocating the volumes when needed. For this example, the current implementation uses 20 MB of memory for $\gamma = 1$. For $\gamma = 2$, 3 GB of memory are required. An implementation designed for discretizations of continuous systems would require less than 4 MB for $\gamma = 1$ and less than 100 MB for $\gamma = 2$. Experiments indicate that dynamic allocation of volumes will further reduce these requirements by a factor of 2^γ.

Eliminating the saturations reduces the number of regions to 16 and allows more experimentation with the algorithm. For this simplified model, which has ν of 1, robust simulation over a 1 second time horizon converged. The robust simulation results along with a lower bound obtained by Monte Carlo methods and gradient descent are shown in Figure 8.5. The lower bound is shown by a dashed line, and the remaining lines are upper bounds obtained for γ of 1, 2, 3, and 4. As expected, larger γ give tighter bounds.

Though robust simulation gave good results for 20 time steps, the algorithm diverged before completing 100 time steps. The results for the 5 second time horizon are shown in Figure 8.6. Though none of the runs converged, increasing γ delayed divergence substantially. Due to the previously described memory

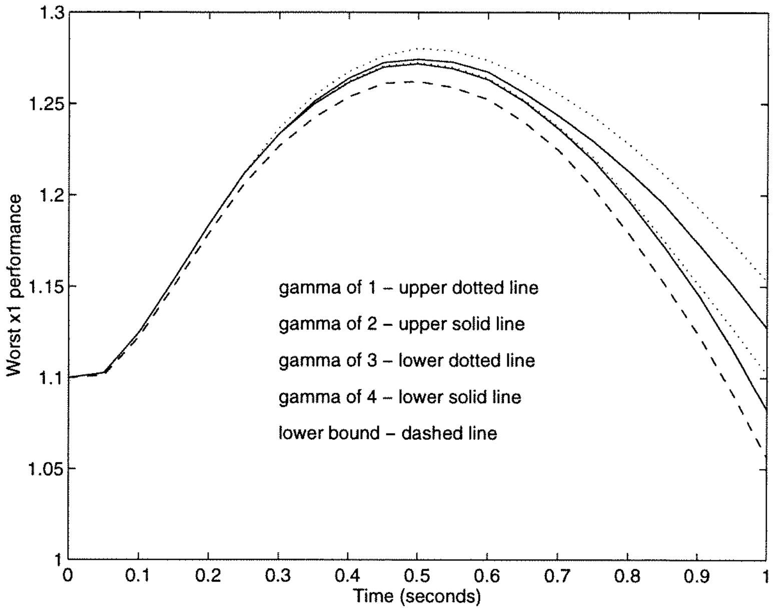


Figure 8.5: Robust simulation of the simplified model for 1 second.

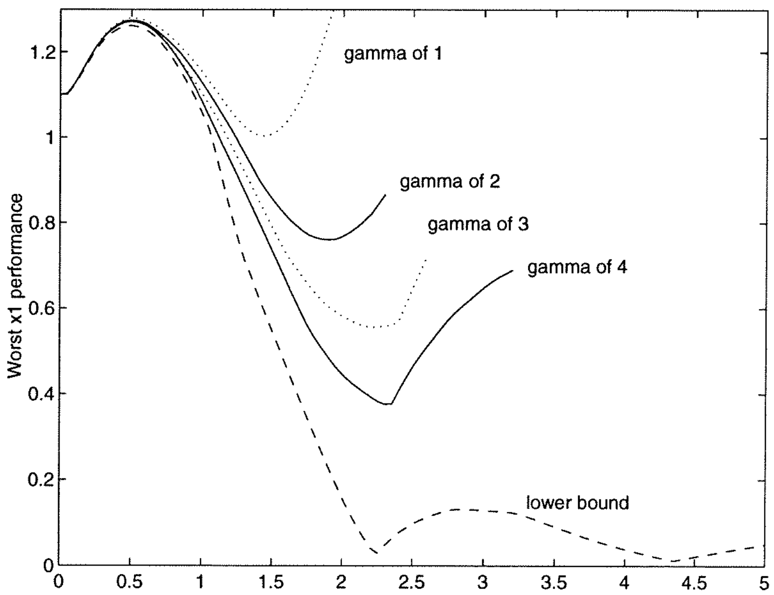


Figure 8.6: Robust simulation of the simplified model for 5 seconds.

constraints, γ was limited at 4. Computation time was not a limiting factor. For $\gamma = 3$, robust simulation for the 53 time steps shown required 18 minutes on a Sun UltraSparc 2 with a 170 MHz processor. For $\gamma = 4$, 26 minutes were needed to simulate 65 time steps. Due to the relatively short simulation horizon and the high cost of algorithm initialization, computation times for smaller values of γ are not meaningful.

The effects of branch and bound were also examined. When robust simulation converged, branch and bound improved the results. However, branch and bound had no effect on algorithm convergence. This experience is similar to that seen for other bounding algorithms. Branch and bound is beneficial only when the underlying bounds are good. Branch and bound has little effect when the bounding algorithms fail.

The main conclusion from this example is that increasing the number of polyhedra allowed when forming the approximation has a large effect on algorithm convergence. However, once the algorithm converges, additional increases give limited benefit. As previously shown in Table 5.3, branch and bound may be a more computationally efficient method to reduce conservatism for simulations that converge.

A second conclusion is that efficient algorithm implementations are needed. Discretizations of continuous time systems do not exhibit the worst case algorithm behavior and the implementation should take advantage of this. More efficient memory management will increase the size of the problems that can be considered. Computation times remained reasonable for all simulations.

Chapter 9 Conclusions

The preceding chapters presented robust simulation and described its uses. This final chapter summarizes these results and suggests future research directions.

9.1 Summary

Simulation has historically been the most common method for analyzing nonlinear systems. When no other analysis technique was applicable, one could always use simulation. However, traditional simulation methods give inherently local results. Many interesting analysis questions require global results, something traditional simulation cannot provide.

Robust simulation addresses this problem. Instead of calculating individual trajectories, sets of trajectories are simultaneously found. In the linear setting, robust analysis makes guarantees about sets of systems. Robust simulation allows these guarantees to be made for sets of nonlinear systems.

Simulating sets also has intuitive justification. If a system is uncertain, traditional simulation generates one of many possible outputs. Robust simulation generates all possible outputs. When simulating a set of systems, the natural result is a set of trajectories.

As with all analysis techniques, the system representation is chosen to facilitate computation. For robust simulation, the discrete time piecewise linear model representation is chosen. Chapter 3 describes these systems and develops basic algorithms for their manipulation. As shown in Chapter 6, this class of system admits a very rich set of nonlinearities. Subject to a minor technical condition on continuity, any nonlinear system can be approximated to any desired accuracy by a PL system.

In Chapter 4, efficient robust simulation algorithms are developed. While calculating the exact solution requires an exponential number of computations, approximations can be found in polynomial time. These approximations maintain the property that the result contains all possible trajectories. If the approximations are too conservative, the simulation can be systematically refined until the exact solution is obtained.

Chapter 5 uses these algorithms to solve the nonlinear robust performance problem. By applying a measure to the set of all possible trajectories, a performance guarantee is obtained. The measure used is the tube cost, an extension of the l_∞ norm to the PL setting. Though not a true norm, the tube cost is a very flexible tool for evaluating performance.

Robust simulation also provides a foundation for interpreting simulation results when multiple models exist for a single system. In traditional simulation, each model returns a different result. In robust simulation, each model returns a set of possible trajectories. If both models describe the same system, then intersection of the sets of possible trajectories is not empty. Generally, more accurate models return smaller sets of trajectories. These ideas are formalized in Chapter 7.

Chapter 7 also presents multiresolution simulation. The ability to dynamically change models during simulation is a direct result of simulating sets. The result at any time step can be used as the initial condition for simulating any model one time step forward. The technique does not provide guidelines for model selection; it only gives the ability to change models. As long as all models correctly describe the same system, the results are guaranteed, by construction of the algorithm, to be meaningful.

9.2 Future Directions

Robust simulation has been successfully demonstrated on a large number of small problems. Mixed results were obtained on larger, physically motivated problems. While the technique shows promise, additional software development is needed.

Further study of the various heuristics is warranted. Increasing the number regions allowed in the approximation $\widehat{\mathcal{S}_k}$ greatly improves convergence. However, the only heuristic studied has been the systematic increase of γ. Other methods could yield similar benefit with less cost.

To gain acceptance, the modeling framework must both be powerful and easy to use. Currently, complex PL systems must be created by hand. This tedious process would be greatly simplified by a graphical system building package. Other modeling issues include exploiting nonlinear structure and finding guidelines for choosing the number of regions and level of detail needed in PL covers.

Perhaps the largest limitation of robust simulation is that it only applies to discrete time systems. The ideas extend to continuous time systems, but the modeling framework does not. For practical robust simulation, efficient set mapping is required. In continuous time, PL systems do not map a polyhedron to a set of polyhedra.

To develop a robust simulation algorithm for continuous time systems, a different system representation is needed. Over any time interval, continuous affine systems map a polyhedra to a polyhedra. This behavior can be exploited using a locally affine representation. Polyhedra are simulated using an affine law with additive noise for a chosen time interval. After each time interval, a new affine flow, amount of noise, and time interval are chosen for each polyhedron and the step repeats. If too much noise is needed, the polyhedron is divided into two smaller polyhedra.

This takes advantage of the locally affine nature of any continuous nonlinear system. However, it is not a convenient representation. The amount of noise added depends both upon the size of the polyhedra and the time for which the flow must hold. For a short time around a small neighborhood of a point, a nonlinear system behaves like an affine system. As the time duration and size of the volume increase, the nonlinearities become more prominent. Representing this information efficiently is difficult at best.

There are two halves to the robust analysis problem: the upper bound and the lower bound. This work has examined the upper bound. While an efficient upper bound algorithm is essential, without a good lower bound the robust analysis question cannot be answered. The lower bounds in this thesis were found through a mix of Monte Carlo simulation and gradient descent. While better methods, such as power iteration techniques, exist, they are not compatible with the tube cost or l_∞-induced norm. Though the Monte Carlo technique was sufficient for the small problems presented, larger problems will need more efficient algorithms.

Appendix A
Robust Simulation Software

This appendix briefly describes the robust simulation software used throughout this thesis. Each of the algorithms presented in Chapters 3 and 4 is implemented, though the implementation may differ slightly from the presentation in this thesis. In all cases, the implementations are provably correct. Information about the most recent version of the software may be obtained by contacting the author via e-mail at `kantner@alumni.caltech.edu`.

This appendix begins with a brief description of the software implementation. This section also gives insight into the issues encountered during software design and testing. The second section describes the representation of PL systems. The appendix closes with a description of the analysis programs.

A.1 Design and Implementation

The robust simulation software package can be divided into four main sections: matrix tools, linear program solvers, polyhedra tools, and robust simulation algorithms. Each of these sections is roughly independent of the others. As long as the interface remains unchanged, the underlying software may change. To reduce the coding requirements, existing software packages were used whenever possible.

Since robust simulation uses only real arithmetic, matrix tool development was greatly simplified. Simple operations, such as multiplication and transpose, were implemented directly. The freely available matrix library LAPACK [2] was used for more complex operations, such as inverse. File input and output routines support both an ASCII format and MATLAB's binary format.

Linear programs were solved used the convex optimization package LOQO [38]. LOQO uses an infeasible primal-dual interior-point method and is capable of solving problems with hundreds of variables in seconds. The robust simulation software accessed LOQO directly through its subroutine library. LOQO is freely available to universities; commercial users must purchase a license.

The polyhedra tools were built using the matrix tools and the linear programming subroutine library. To ensure independence from an LOQO's interface, an additional layer was inserted between the polyhedra tools and LOQO. This layer standardizes the interface to the linear program solver. Any change in LOQO only requires a change to the additional layer; the polyhedra tools are unaffected. This also simplifies the use of other linear program solvers.

The robust simulation algorithms were designed to facilitate experimentation with different heuristics. Each heuristic was encapsulated in its own subroutine, and switching between them was accomplished through command line arguments. The software also provided extensive logging capabilities so results can be monitored during individual simulations.

The robust simulation software consists of over 10000 lines of ANSI C code, excluding LAPACK and LOQO. Since exact solutions to robust simulation cannot generally be computed, testing and debugging posed additional challenges. The robust simulation software provides guarantees on system performance. In order for the guarantees to hold, the software must also be guaranteed to function correctly. Three approaches were taken to ensure software correctness.

First, algorithm requirements were generated and assertions were used to verify that the requirements were satisfied. It was assumed that LAPACK and LOQO both gave correct answers. While the specification process is time consuming, it guarantees software correctness. In general, problems resulted from improper specification, not from coding errors.

Second, examples for which the exact solution could be tested were created. For the matrix tools, polyhedra tools, and linear program solver, extensive testing was performed. For the robust simulation algorithm, only small problems could be solved exactly. These tests verified that each component of the software worked properly.

Finally, numerical tests were performed on larger problems. LOQO results were verified against results from a different linear program solver. The robust simulation code was tested by verifying that the upper bound was greater than the lower bound for all problems.

A.2 Model Representation

A PL system is stored as a set of files. The main file is contains a list of the files that define each region. Each region is defined by a single file that contains the bounding polyhedra and state update law stored as matrices. These files are loaded using the matrix tools and can be either in ASCII or binary format. More details are included with the software documentation.

Model creation is one of the most difficult tasks encountered when using the robust simulation software. No tools exist to automate the creation of complex systems. Since the software can read MATLAB data files, small amounts of automation are possible. For example, the 200 random tests performed in Chapter 4 were generated using a MATLAB script.

A.3 The Executables

Four main commands comprise the robust simulation software package.

- `simulate` performs traditional simulation of the system for a specified initial condition and input sequence.
- `lowerbound` uses a combination of Monte Carlo and gradient descent methods to find a lower bound on system performance.
- `upperbound` uses robust simulation to find an upper bound on system performance. The degree of accuracy γ may be specified as an argument to the command.
- `branchbound` uses robust simulation and branch and bound to find an improved upper bound.

Each command takes the main file of the PL system, the number of time steps, and a file containing the initial conditions as arguments. Each command also has additional arguments, described in the software documentation, that control other algorithm parameters.

Bibliography

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide.* SIAM, second edition, 1995.

[3] S. Bennett. A brief history of automatic control. *IEEE Control Systems*, 16(3):17–25, June 1996.

[4] H. Bode. *Network Analysis and Feedback Amplifier Design.* Van Nostrand, 1945.

[5] B. Bodenheimer. *The Whirling Blade and The Steaming Cauldron.* Ph.D. thesis, California Institute of Technology, 1996.

[6] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *Studies in Applied Mathematics.* SIAM, Philadelphia, PA, 1994.

[7] R. Burden and J. Faires. *Numerical Analysis.* PWS-Kent Publishing Company, fifth edition, 1993.

[8] W. Cheney and D. Kincaid. *Numerical Mathematics and Computing.* Brooks / Cole Publishing Company, second edition, 1985.

[9] L. Chua and A. Deng. Canonical piecewise-linear representation. *IEEE Transactions on Circuits and Systems*, 35(1):101–111, January 1988.

[10] L. Chua and R. Ying. Canonical piecewise-linear analysis. *IEEE Transactions on Circuits and Systems*, CAS-30(3):125–140, March 1983.

[11] C. Douglas. Multigrid methods in science and engineering. *IEEE Computational Science and Engineering*, 3(4):55–86, Winter 1996.

[12] J. C. Doyle, B. Francis, and A. Tannenbaum. *Feedback Control Theory.* Macmillan, 1992.

[13] P. Gahinet, A. Nemiroskii, A. Laub, and M. Chilali. *The LMI Control Toolbox.* The MathWorks, Inc., 1994.

[14] C. E. García, D. M. Prett, and M. Morari. Model predictive control: Theory and practice — A survey. *Automatica*, 25(3):335–348, May 1989.

[15] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[16] G. Golub and C. Van Loan. *Matrix Computations.* The Johns Hopkins Universtiy Press, second edition, 1989.

[17] M. Johansson and A. Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. Technical Report TFRT–7549, Department of Automatic Control, Lund Institute of Technology, June 1996.

[18] M. Kantner. Robust stability of piecewise linear discrete time systems. To appear in *Proceedings of the American Control Conference*, 1997.

[19] M. Kantner, B. Bodenheimer, P. Bendotti, and R. M. Murray. An experiemental comparison of controllers for a vectored thrust, ducted fan engine. In *Proceedings of the American Control Conference*, pages 1956–1961, 1995.

[20] N. K. Karmarkar. A new polynomial–time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[21] H. Khalil. *Nonlinear Systems.* Macmillan, 1992.

[22] S. Khatri, R. D'Andrea, and J. C. Doyle. Uncertain hierarchical modeling. In *Proceedings of the 35th IEEE Conference on Decision and Control*, pages 412–417, December 1996.

[23] W. Lu and J. C. Doyle. A state space approach to robustness analysis and synthesis for nonlinear uncertain systems. Technical Report CIT/CDS 94-010, California Institute of Technology, 1994.

[24] D. Mayne and H. Michalska. Receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 35:814–824, 1990.

[25] C. Moler and C. Van Loan. Nineteen dubious ways to compute the matrix exponential. *SIAM Review*, 20:801–36, 1978.

[26] Y. Nesterov and A. Nemeroskii. *Interior-Point Polynomial Algorithms in Convex Programming*, volume 13 of *Studies in Applied Mathematics.* SIAM, 1994.

[27] V. Nevistić and J. Primbs. Constrained nonlinear optimal control: A converse HJB approach. CDS Technical Memo CIT-CDS 96-021, California Institute of Technology, Pasadena, CA 91125, 1996.

[28] M. P. Newlin and P. M. Young. Mixed μ problems and branch and bound techniques. In *Proceedings of the 31st IEEE Conference on Decision and Control*, pages 3175–3180, 1992.

[29] A. Packard and M. Kantner. Gain scheduling the LPV way. In *Proceedings of the 35th IEEE Conference on Decision and Control*, pages 3938–3941, December 1996.

[30] N. Pettit and P. Wellstead. A graphical analysis method for piecewise linear systems. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 1122–1127, December 1994.

[31] W. Rugh. Analytical framework for gain scheduling. In *Proceedings of the American Control Conference*, pages 1688–1694, 1990.

[32] P. Schröder. Wavelets in computer graphics. *Proceedings of the IEEE*, 84(4):615–625, 1996.

[33] E. Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Transactions on Automatic Control*, 26(2):346–357, April 1981.

[34] E. Sontag. Remarks on a picewise-linear algebra. *Pacific Journal of Mathematics*, 98(1):183–201, 1982.

[35] E. Sontag. From linear to nonlinear: Some complexity comparisons. In *Proceedings of the 34th IEEE Conference on Decision and Control*, pages 2916–2920, December 1995.

[36] J. E. Tierno. *A Computational Approach to Nonlinear Systems Analysis.* Ph.D. thesis, California Institute of Technology, 1996.

[37] P. Tuinenga. *SPICE: A Guide to Circuit Simulation and Analysis Using PSpice.* Prentice Hall, 1988.

[38] R. Vanderbei. Loqo user's manual – version 2.27. Technical Report SOR-96-07, Princeton University, 1997.

[39] S. Wright. *Primal-Dual Interior-Point Methods.* SIAM, 1997.

[40] G. Zames. Input-output feedback stability and robustness, 1959-85. *IEEE Control Systems*, 16(3):61–66, June 1996.

[41] K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control.* Prentice Hall, 1996.