DISSERTATION

STABILITY ANALYSIS OF RECURRENT NEURAL NETWORKS WITH APPLICATIONS

Submitted by

James N. Knight

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2008

ABSTRACT OF DISSERTATION

STABILITY ANALYSIS OF RECURRENT NEURAL NETWORKS WITH APPLICATIONS

Recurrent neural networks are an important tool in the analysis of data with temporal structure. The ability of recurrent networks to model temporal data and act as dynamic mappings makes them ideal for application to complex control problems. Because such networks are dynamic, however, application in control systems, where stability and safety are important, requires certain guarantees about the behavior of the network and its interaction with the controlled system. Both the performance of the system and its stability must be assured. Since the dynamics of controlled systems are never perfectly known, robust control requires that uncertainty in the knowledge of systems be explicitly addressed. Robust control synthesis approaches produce controllers that are stable in the presence of uncertainty. To guarantee robust stability, these controllers must often sacrifice performance on the actual physical system. The addition of adaptive recurrent neural network components to the controller can alleviate, to some extent, the loss of performance associated with robust design by allowing adaptation to observed system dynamics. The assurance of stability of the adaptive neural control system is prerequisite to the application of such techniques.

Work in [49, 2] points toward the use of modern stability analysis and robust control techniques in combination with reinforcement learning algorithms to provide adaptive neural controllers with the necessary guarantees of performance and stability. The algorithms developed in these works have a high computational burden due to the cost of the online stability analysis. Conservatism in the stability analysis of the adaptive neural components has a direct impact on the cost of the proposed system. This is due to an increase in the number of stability analysis computations that must be made. The work in [79, 82] provides more efficient tools for the analysis of time-varying recurrent neural network stability than those applied in [49, 2]. Recent results in the analysis

of systems with repeated nonlinearities [19, 52, 17] can reduce the conservatism of the analysis developed in [79] and give an overall improvement in the performance of the on-line stability analysis.

In this document, steps toward making the application of robust adaptive neural controllers practical are described. The analysis of recurrent neural network stability in [79] is not exact and reductions in the conservatism and computational cost of the analysis are presented. An algorithm is developed allowing the application of the stability analysis results to online adaptive control systems. The algorithm modifies the recurrent neural network updates with a bias away from the boundary between provably stable parameter settings and possibly unstable settings. This bias is derived from the results of the stability analysis, and its method of computation is applicable to a broad class of adaptive control systems not restricted to recurrent neural networks. The use of this bias term reduces the number of expensive stability analysis computations that must be made and thus reduces the computational complexity of the stable adaptive system. An application of the proposed algorithm to an uncertain, nonlinear, control system is provided and points toward future work on this problem that could further the practical application of robust adaptive neural control.

James N. Knight
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523
Summer 2008

ACKNOWLEDGMENTS

The process of producing a dissertation is an arduous one. It is mentally and even physically taxing. I am indebted to following people for there help in surviving this process:

to Dr. Charles Anderson, for getting me out of my rut and introducing me to the problems considered here, for the countless technical and grammatical suggestions incorporated into this document, and for putting together the grant that funded my travel to Europe;

to Peter Young, Michael Kirby and Bruce Draper for the advice on presenting my research in writing and putting up with the delays;

to José Millán at the IDIAP Research Institute in Martigny, Switzerland and to Barak Pearlmutter at the Hamilton Institute in Maynooth, Ireland for hosting my three month visits;

to the staff of the Computer Science department for answering the many questions I was too lazy or tired to answer myself;

to my fellow graduate students, in and out of the department, for the random doses of sanity and encouragement;

to Monte and Agnieszka for helping pull off the triple play defense day and keeping me going near the end;

to Michal Kocvara for the use of PENBMI, it proved invaluable;

to my parents for their constant support and encouragement;

and finally, to Dana, who probably wanted to file a missing persons report on me at various times during the writing process but supported me to the end.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Notation

$\mathbb{R}$	The set of real numbers.
$\mathbb{R}^n$	The set of $n \times 1$ real vectors.
$\mathbb{R}^{n \times m}$	The set of $n \times m$ real matrices.
$\mathbb{S}^n$	The set of real, symmetric matrices of size $n \times n$.
A, B, C, X, Y, Z, T, M	Elements of $\mathbb{R}^{n \times m}$ or $\mathbb{S}^n$.
$\lambda_i(A)$	The ith eigenvalue of A.
$\lambda_{\max}(A)$	The largest eigenvalue of A: $\max_i \lvert\lambda_i(A)\rvert$.
$A \succ 0$	A is positive definite: $A \in \mathbb{S}^n$, $\lambda_i(A) > 0 \quad \forall\, i$.
$A \succeq 0$	A is positive semideinite: $A \in \mathbb{S}^n$, $\lambda_i(A) \geq 0 \quad \forall\, i$.
$A \prec 0$	A is negative definite: $A \in \mathbb{S}^n$, $\lambda_i(A) < 0 \quad \forall\, i$.
$A \preceq 0$	A is negative semideinite: $A \in \mathbb{S}^n$, $\lambda_i(A) \leq 0 \quad \forall\, i$.
$A \succ (\prec)\ B$	$A - B \succ (\prec)\ 0$.
$A \succeq (\preceq)\ B$	$A - B \succeq (\preceq)\ 0$.
$[a_{ij}]$	A matrix with a_{ij} in the ith row and jth column.
$\mathcal{A}(x)$	A function of $x \in \mathbb{R}^n$: $\mathcal{A}(x) = \sum x_i A_i,\ A_i \in \mathbb{S}^n$.
$\operatorname{tr} A$	The trace of a matrix: $\sum_i A_{ii}$.
A^T	The transpose of A.
$A \bullet B$	A matrix inner product: $\operatorname{tr} A^T B$.
$\operatorname{vec} A$	A vector formed by stacking the columns of A.

Chapter 1

Introduction

Solutions to real world control problems often make simplifying assumptions to enable automated design techniques. To ensure that the resulting controllers behave as expected on the actual system, the simplifications made for analysis generally involve constructing a set of simpler systems which cover all the behaviors of the real world plant. These types of relaxations generally result in controllers that are suboptimal with respect to the desired performance metric and the plant of interest. Adaptive control techniques allow the initial design to be modified with the goal of improving performance based on observations of the plant's actual behavior. The application of adaptive control systems to real world control problems is, however, often impossible or too risky because of a lack of guarantees about the adaptive system's performance and stability.

Most existing research on stable adaptive control takes a specific algorithm and proves stability on some class of plants. For example, a Lyapunov function might be constructed showing that a certain gain scheduling procedure is stable for a class of linear, time-invariant plants [64]. A second approach is to ensure stability by restricting the updates an adaptive system makes to the controller parameters [49, 62]. This type of approach is not restricted to a given adaptive control or learning algorithm, but it requires online monitoring of the control parameter updates. Compared to the first approach, online update monitoring increases the computational cost of the adaption, sometimes considerably. On the other hand, this type of approach is more general.

This document focuses on the second approach for ensuring the stability of adaptive control systems. Specifically, the use of recurrent neural networks in adaptive control systems is considered. Recurrent neural networks are capable of representing a wide class of both static and dynamic mappings. In fact it can be shown that recurrent neural networks are capable of approximating a many dynamical systems arbitrarily well [32]. This makes them suitable for many different uses in control systems. For example, a recurrent neural network can be used as a controller or as an estimator of system parameters or unobservable states. Since recurrent neural networks are dynamic systems, they have their own stability properties which must be understood. The first part of this dissertation analyzes existing techniques for assessing the stability of recurrent neural networks and extends them in several ways to improve their performance. Recurrent neural network stability is analyzed in terms of the input-output relationship of the nonlinear, time-varying mapping defined by the network parameters. The main stability analysis tool that is applied is the theory of integral quadratic constraints [59]. Stability properties are framed in terms of the feasibility of certain matrix inequalities. Such formulations are addressed mathematically and computationally by semidefinite programming [92]. The second part of this dissertation proposes an algorithm for ensuring the stability of a recurrent neural network that is adapted as part of a larger control system. The algorithm is applicable to the adaption of recurrent neural networks in isolation or in

a loop with a controlled plant. Examples are presented of application in both of these situations.

1.1 A Motivating Example

Reinforcement learning is a term used to describe a large class of problems and algorithms that involve learning through observation of, and interaction with, an environment. While the theory and practice of reinforcement learning for control has improved dramatically over the last two decades, applications of this approach for online adaptation on real systems remain elusive. Two problems persist that impede the deployment of reinforcement learning systems: guarantees of performance and guarantees of safety. Much of the progress in reinforcement learning has come in the form of proofs of convergence and characterizations of solutions. Sufficient conditions for the convergence of reinforcement learning algorithms have emerged for increasingly large classes of algorithms and problems. This body of research gives results concerning the properties of the resulting controllers. More recently, literature addressing the dynamic behavior of these algorithms has appeared [49, 67]. Here, the research seeks to address the problem of providing guarantees of safety. The dynamic effects of reinforcement learning algorithms interacting with a controlled system must be understood.

Consider a class of problems in which the goal is to drive the state of a system to some operating point. This class of problems includes stabilization of unstable systems, reference tracking problems, and regulation problems. Controllers for these problems must meet some performance goal while also guaranteeing stability of the closed loop system. A large body of research exists that addresses the problem of designing and characterizing controllers for these types of problems. These design methods are generally restricted to time-invariant linear systems and controllers that are linear in their inputs. In most cases, however, the system that is to be controlled will exhibit nonlinear dynamics that change over time. To specify the discrepancy between the design assumptions and reality, linear, time-invariant (LTI) *models* can be derived with the unmodeled dynamics of the system — its time varying components and its nonlinearities — characterized as uncertainty in the basic LTI model. Robust control theory provides methods for designing controllers for these uncertain models and for characterizing the uncertainties.

Robust controllers, however, can exhibit suboptimal performance because of restrictions in their design. In this situation performance can be improved through the use of adaptation. Rather than adapting the given robust controller, an extra adaptive component can be added to the control loop. Retaining the fixed controller gives the system a guaranteed level of initial performance. The adaptive component is allowed to have a more general structure than the fixed controller. The actual control signal is generated by combining the output of the fixed controller and the adaptive controller. The stability of the closed loop system is given for the fixed controller case, but the addition of an adaptive component can drive the system to instability. This instability can result in damage to the physical plant or its environment and an associated decrease in performance. Thus, a method is needed to guarantee the stability of the system with the adaptive component. In Chapters 5 and 6 such a method is given using recurrent neural networks as the adaptive component.

Consider the simple spring-mass-damper system in Figure 1.1. This system can be described by a second order, nonlinear, differential equation in the variable y, giving the distance from the equilibrium point. The equation describing the system is

$$m\ddot{y} + (c + c_f)\dot{y} + ky + ka^2y^3 = u, \tag{1.1}$$

where c is the damping coefficient, c_f is the coefficient of friction, k is the spring constant and a is the spring hardening constant. A force, u, can be applied to the mass. A control system similar to this simple spring-mass-damper is described in Chapter 6.

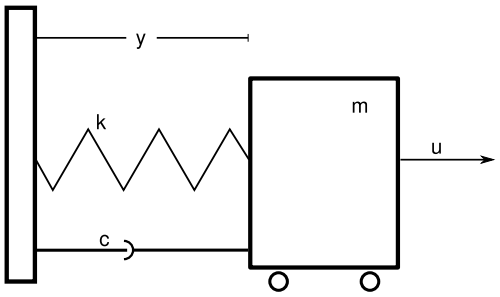

Figure 1.1: A simple spring-mass-damper.

A control problem for this system is defined by requiring the mass to be held at a certain position, $r(t)$, that can vary with time. For expository purposes a simple proportional integral (PI) controller is defined for this problem. The controller has the form

$$\begin{aligned} u(t) &= K_p e(t) + K_i \int_0^t e(\tau) d\tau \\ e(t) &= y(t) - r(t) \end{aligned}$$

where K_p and K_i are controller parameters. Values of the parameters giving good performance and stability can be derived using standard control techniques and tuning on model systems. An example trajectory of the controlled system is shown in Figure 1.2.

A simple, single node neural network is added to the system as the adaptive controller component. A schematic of the control system is shown in Figure 1.3. The neural controller is described by the equation

$$a(t) = w_o(t) \tanh(w_h(t)\ e(t))$$

and has two adjustable parameters, the hidden weight, w_h, and the output weight, w_o. In later chapters more neural networks with recurrent connections will be used, but here a simple forward network suffices. By adjusting these parameter values away from zero, the behavior of the system can be modified. It is then necessary to answer the following question: what setting of these parameters gives optimal performance under the stability constraint? In general, performance will be measured as some function of the tracking error, e, over time, and stability will require that the system's states do not grow without bound for bounded reference signals.

The stability of the system is analyzed, using the methods presented in Chapters 2 and 3, for parameter values $(w_h, w_o) \in [-4, 4]^2$. The region of the parameter space that can be proved to result in stable behavior is shown in Figure 1.4. The average tracking error over a fixed amount of time is estimated for all parameters settings of the neural controller in the same range. The estimated performance for stable parameter settings is shown in Figure 1.5. These results show that stable, performance improving parameter settings exist for this problem. Examples of stable and unstable behavior of the system are shown in Figure 1.2. The stability analysis methods developed in Chapters 2, 3, and 4 can, to a certain extent, distinguish the parameter settings that result in unstable behavior from those that result in stable behavior.

To guarantee stability during learning, transitions from one controller to another must be considered. The stability analysis presented in Chapter 4 considers a range of possible controllers and assesses whether or not the system will remain stable as the controller changes within this range. In Figure 1.6, for example, several regions are shown in which the control parameters can vary while not making the control loop unstable. Notice that the regions are smallest near the stability boundary. This means that the controller can vary only a little when its parameters are in this

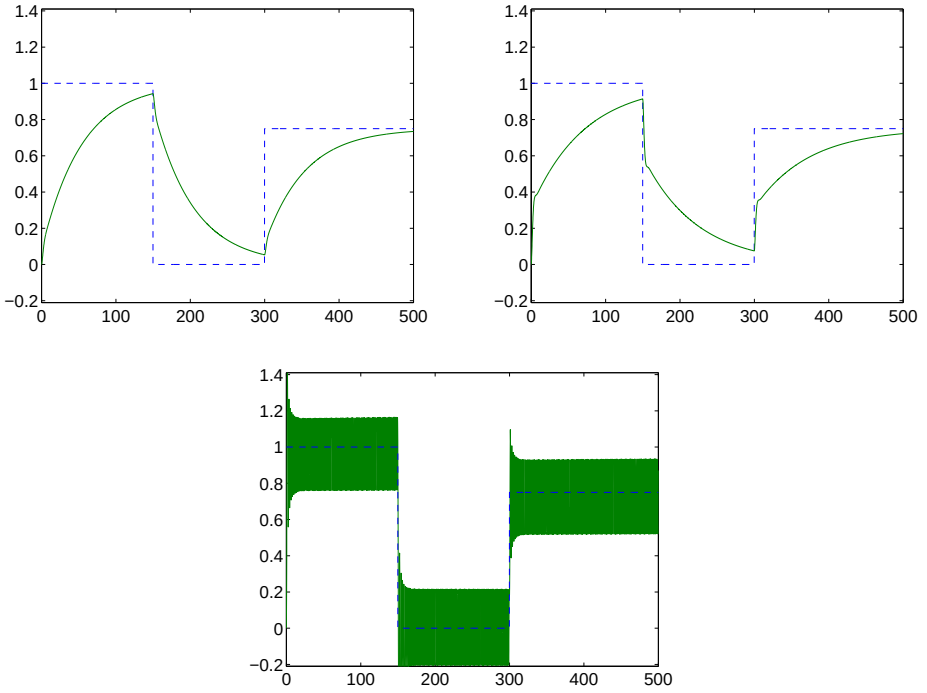

Figure 1.2: Behavior of the spring-mass-damper system with different controllers: PI controller (left-top), PI with stable NN controller (right-top), and PI with unstable NN controller (bottom). The dashed line is the reference point, and the solid line is the position of the mass. The parameters used were $m = 1.0$, $c = 1.2$, $c_f = 0.05$, $k = 0.5$, $a = 0.1$, $K_p = 1.0$ and $K_i = 0.2$.

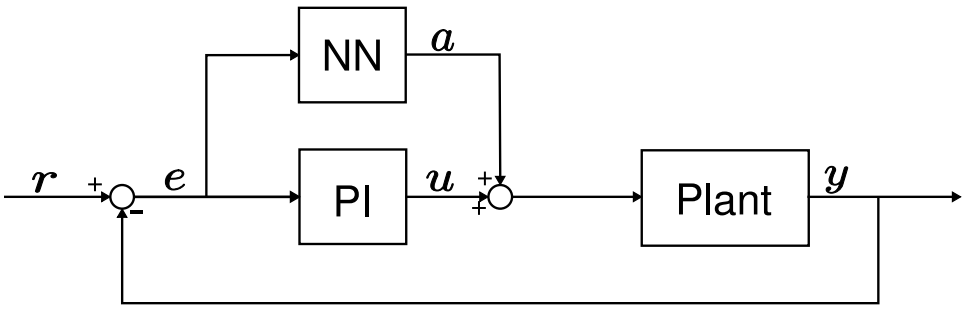

Figure 1.3: A control system with a neural network controller in parallel to a PI controller.

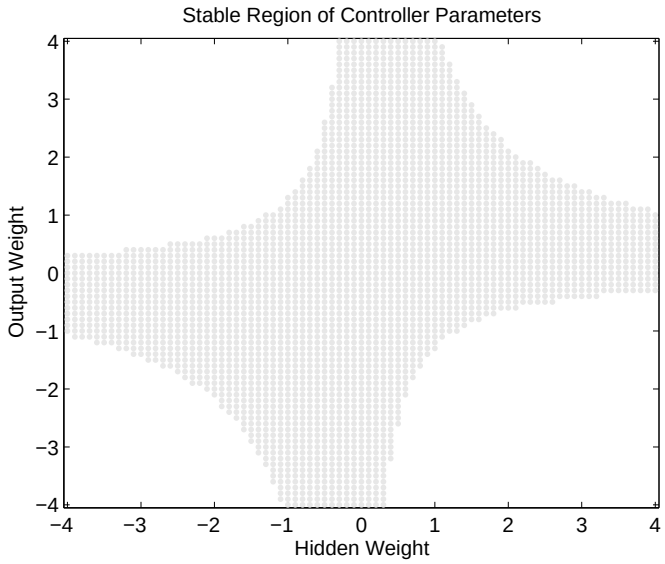

Figure 1.4: An analysis of a finite mesh of controller parameters reveals the structure of the region of static stability.

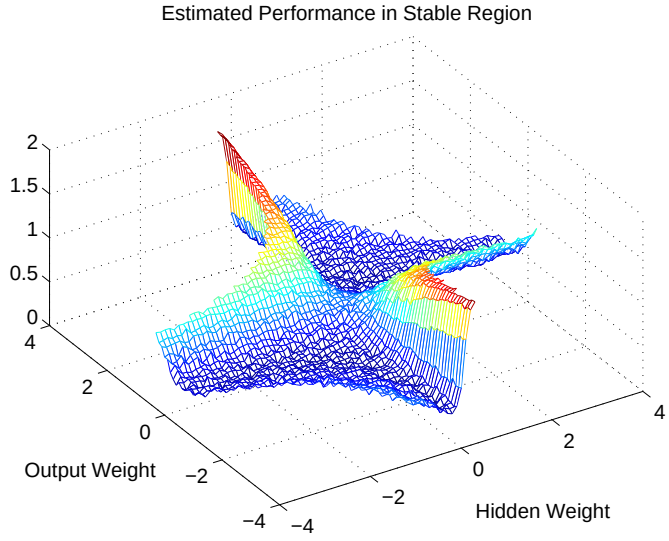

Figure 1.5: Tracking error is estimated over a range of NN parameter values. The estimated average tracking error is shown only for statically stable controllers. The plot shows that the error can be reduced for values that result in statically stable controllers.

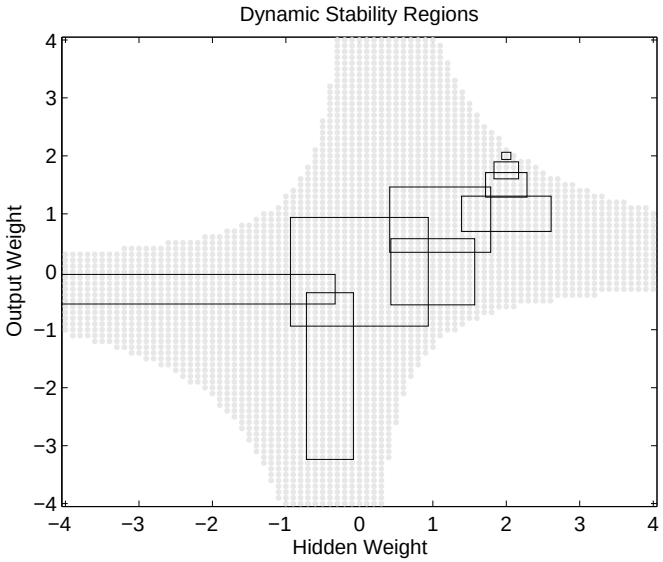

Figure 1.6: Regions of provably stable dynamic stability are shown for the example problem.

part of the space. This in turn can degrade learning performance by restricting the amount of change that can be made to the controller. In this example good parameter settings for the neural controller are not near the boundary between stable and unstable controllers, but it can be the case that the parameters resulting in the most improved performance lie near the boundary of instability. When this happens, this type of approach can become quite inefficient.

Because of this problem, it is useful to consider whether or not the stability analysis provides information that can be used to influence the updates made to the adaptive controller. The stability analysis that resulted in Figure 1.4 produces an upper bound on the gain around the feedback loop — the amount of amplification of the input signal — between the plant and controller. These upper bounds are shown in Figure 1.7. The gain increases rapidly near the stability boundary and is useful for biasing the learning algorithm away from this boundary. These ideas are explored more fully in Chapter 5, where a method of exploiting this information is developed.

1.2 Objectives

The immediate goals of this research are two fold: to improve the state of the art in the analysis of recurrent neural network stability and to provide a method for efficiently applying such stability analysis to the online adaptation of recurrent neural networks in control systems. Improvements in stability analysis techniques can be measured along two axes. The first is the conservativeness of the analysis. A conservative analysis provides only sufficient conditions for stability as opposed to conditions that are both sufficient and necessary. Less conservative methods provide conditions which are closer, in some sense, to the necessary conditions of stability. The second axis is computational complexity. The methods presented in the following chapters rely on testing the feasibility of certain matrix constraints. Reducing the size or number of such constraints allows the stability analysis to be applied to larger or more complex systems. Often, reduction in conservativeness and reduction in computational complexity are at odds. Understanding the relationship between the two allows better choices to be made in practical systems. Improvements along both axes are presented in Chapters 3 and 4.

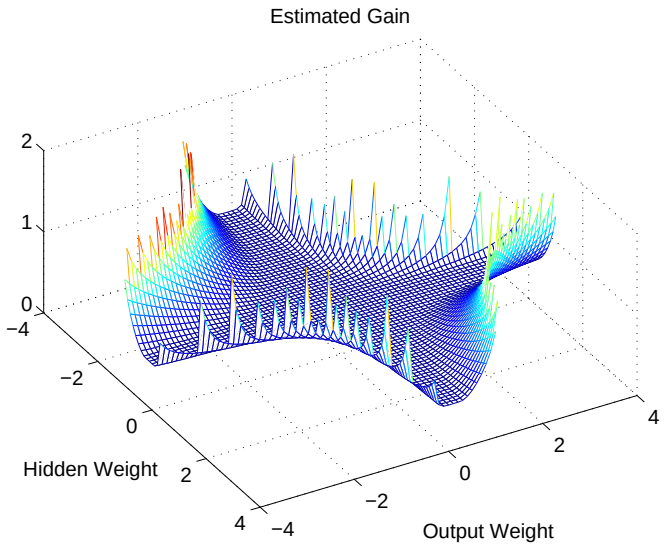

Figure 1.7: The estimated gain for points in the stable region of the parameter space.

Improvements in the stability analysis of recurrent neural networks are only applicable to adaptive control systems in a framework that considers the whole control system. The second goal of this research is to develop a method that allows recurrent neural networks to be adapted in control systems with guarantees of stability for the entire system. This type of system has been proposed in earlier work [49, 50, 2]. This research pointed to a problem with naively applying the proposed stability analysis techniques to an adaptive control system. Often the adaptation drives the system to the boundary in the parameter space between parameter settings for which stability could be proved and those settings for which it could not. When this occurs, an increasingly large number of stability analysis computations must be made. This, in turn, makes the method very expensive computationally. The objective in proposing a new algorithm here is to reduce the cost of guaranteeing stability of adaptive control systems with recurrent neural network components and make such methods more practically applicable.

A secondary goal of this document is to illustrate some practical aspects of working with the proposed stability analysis computations and stable adaptation algorithms. To this end computational comparisons are made throughout the document between different formulations of problems and different optimization algorithms. Rather than attempting to provide a comprehensive picture of the computational aspects of these problems, results are presented simply to give a sense of the class of computations involved.

1.3 Document Outline

The basic ideas presented in the motivating example will be developed more fully in the remaining chapters of this document. In Chapter 2 background material on recurrent neural networks and stability analysis is presented. A formal definition of stability is established, and the basics of integral quadratic constraints analysis are explained. The presentation assumes some knowledge of dynamical systems theory and analysis, but briefly covers all of the necessary control theoretic material used in the dissertation. Pointers to more thorough presentations of the material are provided. Previous work in this area is described at the end of the next chapter. Some limitations

of this previous work are described.

Chapter 3 is an in-depth study of the integral quadratic constraint approach to the stability analysis of recurrent neural networks. Theoretical as well as computational aspects of the approach are considered. Specifically, work in [19, 52] is applied to reduce the conservativeness of the stability analysis. Also, it is shown that for solving the resulting matrix constraint feasibility problems, the augmented Lagrangian method of [46, 47] is much more efficient than standard semidefinite programming algorithms. These results are important for practical application of the algorithm presented in Chapter 5.

When applied in an adaptive control context, the parameters of recurrent neural networks vary with time. Chapter 4 examines the stability analysis of recurrent neural networks in the time varying case. The application of integral quadratic constraints analysis to recurrent neural networks requires formulation of the networks as feedback systems between linear, time-invariant components, and the nonlinear, time-varying and uncertain parts of the system. A new formulation of time-varying recurrent neural networks as feedback systems is developed in Chapter 4. Experiments show that compared to the formulation developed in [79], this results in a less conservative stability analysis. Modifications of the basic stability problem that improve its numerical conditioning are also presented.

The example earlier in this chapter hinted at how certain problems might arise in the application of stability analysis techniques to time varying neural networks in a feedback loop with a plant. Naive application of these techniques requires a large number of stability analysis problems to be solved. Since these problems can be expensive, this limits the applicability of these results to extremely simple systems. In Chapter 5 a general algorithm for filtering parameter updates to ensure stability is described. A method of biasing the parameter trajectory away from the stability boundary is developed. This bias reduces the number of expensive stability analysis computations that must be performed. On the other hand, it requires the solution of many smaller, but non-trivial problems. A technique for reducing the cost of solving these problem is also presented.

In Chapter 6, the proposed algorithm is applied to a multiple spring-mass-damper system with nonlinear friction. The example demonstrates the capability of the algorithm to ensure stability of an adaptive controller for a non-trivial system. The example also exposes some remaining problems with the approach that are proposed as future research. Some general conclusions and a description of this future research are presented in the final chapter.

Chapter 2

Background

This chapter presents some background material useful for understanding the later chapters. Specifically, a class of dynamical systems known as *recurrent neural networks* is introduced. Then, after a brief review of some of the basic notions of stability for dynamical systems, a general framework for the analysis of feedback systems, known as integral quadratic constraints (IQC) analysis, is described. The IQC analysis method addresses stability analysis in terms of optimization problems with matrix constraints. The basics of numerical methods for solving this type of optimization problem are covered in Section 2.4. Two main classes of methods are considered: interior point methods and augmented Lagrangian methods. Both methods will be used throughout the remaining chapters. Finally, in Section 2.5, some previous work in the area of recurrent neural network stability is examined.

2.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a large class of both continuous and discrete time dynamical systems. RNN formulations range from simple ordinary differential equation (ODE) models to elaborate distributed and stochastic system models. The main focus of this work is on the application of RNNs to control problems. In this context, RNNs can be seen as input-output maps for modeling data or acting as controllers. For these types of tasks, it will be sufficient to restrict attention to continuous time RNN formulations, primarily of the form

$$\begin{aligned} \dot{x} &= -Cx + W\Phi(x) + u \\ y &= x. \end{aligned} \tag{2.1}$$

Here, x is the state of the RNN, u is a time varying input, y is the output of the network, C is a diagonal matrix of positive time constants, W is the RNN's *weight matrix* and Φ is a nonlinear function of the form

$$\Phi(x) = [\phi(x_1)\ \phi(x_2)\ \ldots\ \phi(x_n)]^T.$$

The function $\phi(x)$ is a continuous one dimensional map, and generally a sigmoid like function, such as $\tanh(x)$. Since the RNN will be applied as an input-output map, the output, denoted by y, is defined to be the state x. More general models allow the selection of certain states as outputs or an additional mapping to be applied at the output layer. These modifications do not affect the stability analysis of the RNNs dynamics, but will need to be considered when the network is used in a control system.

The stability analysis and stable learning methods constructed in this work are, in principle, applicable to other RNN structures such as discrete time RNN models or other continuous time

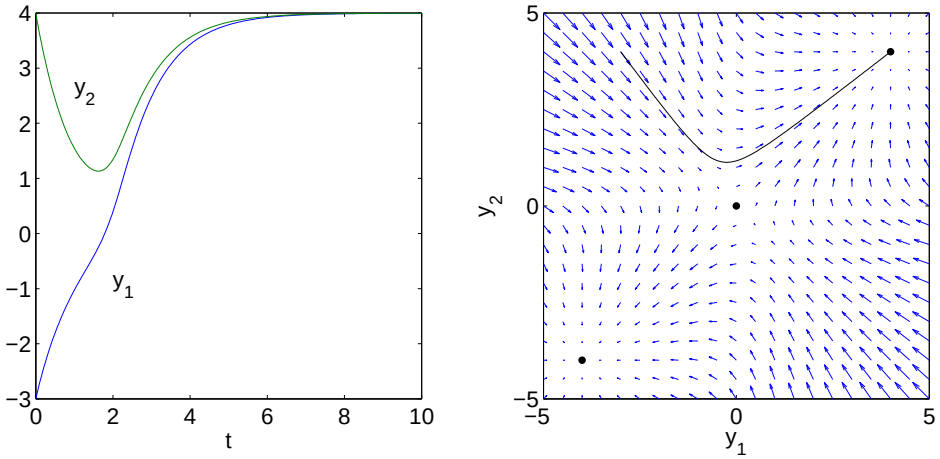

Figure 2.1: An example of a recurrent neural network with three fixed points. On the left is a sample trajectory over time and on the right is the same trajectory in state space.

models; an example is the echo state network [37]. Continuous time echo state networks have the form

$$\dot{x} = \frac{1}{c}\left(-ax + \Phi(Wx + W_{\text{in}}u + W_{\text{fb}}y)\right)$$
$$y = g\left(W_{\text{out}}\begin{bmatrix} x \\ u \end{bmatrix}\right).$$

While some of the specific details of the proposed methods must be changed for application to this type of network, its structure is similar enough to (2.1) that the general principles will be the same.

The dynamic system (2.1) can exhibit a wide variety of dynamics. Equation 2.1 can have a single, globally attractive fixed point whose value varies with different constant input signals. The ability of an RNN to have multiple fixed points allows it to be used as a type of associative memory since regions of the input space can be associated with the different fixed points. An example of an RNN with three fixed points is shown in Figure 2.1. The figure on the left shows the time evolution of the two states of the RNN. The figure on the right depicts the trajectory in state space with the arrows representing the direction of flow in the system.

RNNs can also possess limit cycles in the constant input case [73]. Figure 2.2 shows example dynamics of such a network. Time-varying input signals obviously can lead to even more complex dynamics, but even when the inputs are not time-varying (2.1) can exhibit chaotic dynamics [6]. The stability analysis that will be presented considers the case of time-varying input signals explicitly, but the results are intimately connected to these different dynamic situations in the constant input case.

2.2 Input-Output Stability of Dynamical Systems

Since the focus of this work is on the stability analysis of RNNs and control systems with RNN components, it is important to make clear what *stability* means in this context. This section provides an overview of important concepts from nonlinear systems analysis, and a working definition of stability is established. More thorough introductions can be found in [20, 42, 44, 91]. The presentation here follows along the lines of [44].

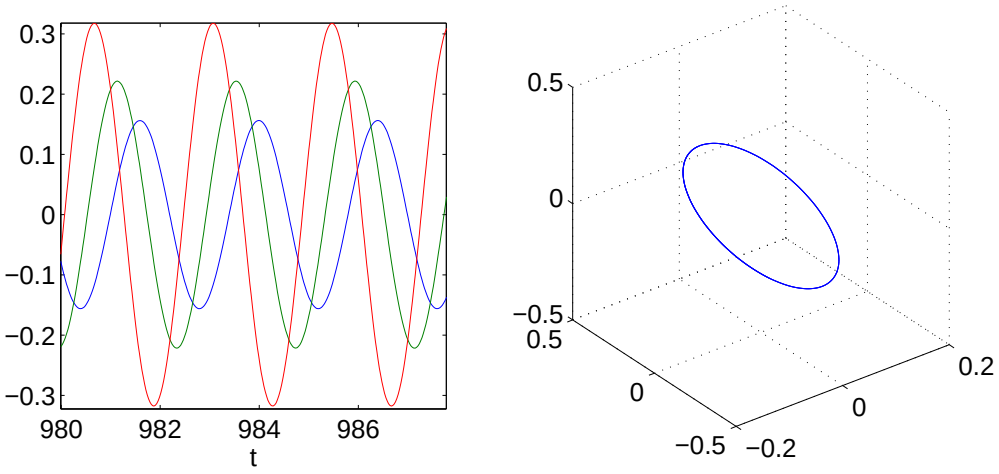

Figure 2.2: An example of a recurrent neural network with a stable limit cycle. On the left is a sample trajectory over time and on the right is the same trajectory in state space.

For the purposes of this document, an m-dimensional signal, u, will be defined as a mapping from the time interval $[0, \infty)$ to $\mathbb{R}^m$. The most important class of signals for the analysis that will be presented is the space of signals that are piecewise continuous and square integrable. This space of signals in combination with the norm defined by

$$||u||_{\mathcal{L}_2^m} = \sqrt{\int_0^\infty u^T(t)u(t)\ dt}, \quad u(t) \in \mathbb{R}^m,$$

forms a normed vector space denoted $\mathcal{L}_2^m$ or $\mathcal{L}_2^m[0, \infty)$. When the dimension of $u(t)$ is unimportant the superscript m will generally be dropped. Also, the restriction of u to the interval $[0, \infty)$ will be assumed and the simpler notation, $\mathcal{L}_2$ will be used.

The space $\mathcal{L}_2$ is of interest for stability analysis purposes since it describes the space of signals with finite energy. The following inner product can be associated with $\mathcal{L}_2$

$$\langle u, v \rangle = \int_0^\infty u^T(t)v(t)dt = \frac{1}{2\pi}\int_{-\infty}^\infty \hat{u}^*(j\omega)\hat{v}(j\omega)d\omega$$

where $\hat{u}(j\omega)$ and $\hat{v}(j\omega)$ are the Fourier transforms of the signals, u and v. Since $\mathcal{L}_2$ is also a complete space, the addition of the inner product defines a Hilbert space. Hilbert spaces have important properties that will be exploited in the development of the stability analysis results that follow. Other classes of signals can be defined in a similar manner. For instance, the space of piecewise continuous, bounded signals with the norm

$$||u||_{\mathcal{L}_\infty^m} = \sup_{t \geq 0}\ ||u(t)||, \quad u(t) \in \mathbb{R}^m,$$

is also a normed vector space and is denoted $\mathcal{L}_\infty^m$.

A nonlinear system such as (2.1) can be viewed as an operator mapping an input signal, u, in some signal space to an output signal, y, into another signal space. It is the properties of this operator that determine the stability of the system. While it is tempting to define an operator as a mapping from say, $\mathcal{L}_2^m \to \mathcal{L}_2^q$, a more general definition is necessary. It is possible that an operator

such as (2.1), call it H, can map a signal $u \in \mathcal{L}_2^m$ to a signal that is not bounded on the interval $[0, \infty)$, in other words, not in $\mathcal{L}_2^q$. To address this difficulty, the notion of extended spaces must be introduced. The extended space, $\mathcal{L}_{pe}^m$, is defined as

$$\mathcal{L}_{pe}^m = \{u \mid u_\tau \in \mathcal{L}_p^m, \ \forall \ \tau \in [0, \infty)\}$$

where u_τ represents the restriction of the signal u to the time interval $[0, \tau)$. Given this definition, the operator H is taken to be a mapping from $\mathcal{L}_{pe}^m$ to $\mathcal{L}_{pe}^q$. This definition allows poorly behaved operators, H, to be dealt with within the same framework. While an operator may map $u \in \mathcal{L}_2$ to some unbounded signal not in $\mathcal{L}_2$, the truncation of this unbounded signal, $(H(u))_\tau$, will often be in the extended space, $\mathcal{L}_{2e}$.

Before proceeding to define stability, one further definition is necessary. An operator, H, is said to be causal if $(H(u))_\tau = (H(u)_\tau)_\tau$. Causal operators depend only on signal values in the past and not on any value of signals in the future. Systems such as (2.1) that have state space representations are causal by definition. With the basics in place, a formal definition of stability can be given.

Definition 2.1 (Finite Gain $\mathcal{L}_p$ Stability [44])**.** *An operator $H : \mathcal{L}_{pe}^m \to \mathcal{L}_{pe}^q$ is finite gain $\mathcal{L}_p$ stable if there exists a non-negative constant γ such that*

$$||(H(u))_\tau||_{\mathcal{L}_p^q} \leq \gamma ||u_\tau||_{\mathcal{L}_p^m}$$

for all u_τ in $\mathcal{L}_p^m$ and τ in $[0, \infty)$. The constant γ is called the gain of the system.

Finite gain, stable operators that are also causal can be shown to satisfy the additional property [44]

$$||H(u)||_{\mathcal{L}_p^q} \leq \gamma ||u||_{\mathcal{L}_p^m}.$$

The choice of p and thus of signal space affects the meaning of the definition. When p is taken to be ∞, finite gain $\mathcal{L}_\infty$ stability of an operator H implies that it maps point-wise bounded signals to point-wise bounded signals. When the space of signals is taken to be $\mathcal{L}_2$ the definition implies that a stable operator H maps signals of finite energy to signals of finite energy. For complex systems, computational methods of assessing the stability of an operator are very useful. The choice of p determines the applicability and complexity of some of these computational approaches. In what follows, the space of signals is taken to be $\mathcal{L}_2$. This choice gives the resulting stability analysis an interpretation in terms of energy amplification in the system. The stability analysis that is developed for RNNs will determine if an RNN is finite gain $\mathcal{L}_2$ stable and can produce an upper bound on this gain.

2.3 Establishing Stability

For linear operators, $\mathcal{L}_2$ stability analysis is straightforward. Consider the linear system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du, \end{aligned} \tag{2.2}$$

that has the associated transfer function, $G(s) = C(sI - A)B + D$. A transfer function relates the input of an LTI system to the output in the space of the Laplace transform of the signals, $Y(s) = G(s)X(s)$. The Laplace transform is given by the equation

$$X(s) = \int_{-\infty}^{\infty} x(t)e^{-st}dt.$$

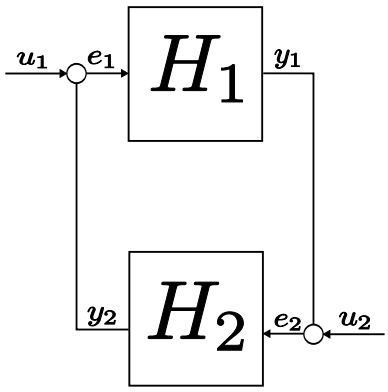

Figure 2.3: A feedback loop with interconnection noise.

A necessary and sufficient condition for the stability of (2.2) is that all eigenvalues of the matrix A have real part less than zero. Such a matrix is called a *Hurwitz* matrix and the associated transfer function is also called Hurwitz. The $\mathcal{L}_2$-gain, γ, of the system is given by its so-called H_∞ norm

$$||G(j\omega)||_{H_\infty} = \sup_{\omega \in \mathbb{R}} ||G(j\omega)||_2 = \sup_{\omega \in \mathbb{R}} \sigma_{\max}(G(j\omega)),$$

where $\sigma_{\max}(\cdot)$ denotes the largest singular value of a matrix argument.

Consider two systems, with operators H_1 and H_2, connected in a loop as in Figure 2.3 and given by the equations

$$\begin{aligned} e_1 &= u_1 - H_2(e_2) \\ e_2 &= u_2 - H_1(e_1) \\ y_1 &= H_1(e_1) \\ y_2 &= H_2(e_2). \end{aligned} \tag{2.3}$$

The system is said to be *well posed* if for all inputs $e_1, e_2 \in \mathcal{L}_2$ the system has a unique solution. In other words, u_1, u_2, y_1, and y_2 exist and are unique. A sufficient condition for stability of the feedback loop, that is, of the mapping from $u = [u_1 \ u_2]^T$ to $y = [y_1 \ y_2]^T$ is given by the small gain theorem.

Theorem 2.1 (Small Gain Theorem [44])**.** *If the feedback system given by* (2.3) *is well posed and the operators H_1 and H_2 are finite-gain $\mathcal{L}_2$-stable then the feedback loop* (2.3) *is finite-gain $\mathcal{L}_2$-stable if*

$$||H_1||\ ||H_2|| < 1.$$

A proof is given in [44].

The small gain theorem gives an efficient test for stability of interconnected components when the computation of the necessary operator norms is efficient. Notice, however, that the theorem only states a sufficient condition for stability, and it is thus possible for the condition to be conservative. In fact, this condition is known to be quite conservative in many cases and techniques for reducing this conservativeness have been developed [15].

Another characterization of stability can be made in terms of the passivity of operators. Passivity theory is derived from the theory of linear circuits, and passivity signifies a property that can be likened to the dissipation of energy in a circuit. The following definition is from [42].

Definition 2.2 (Passivity)**.** *A causal operator, $H : \mathcal{L}_{2e} \to \mathcal{L}_{2e}$, is said to be* passive *if*

$$\langle (H(u))_\tau, u \rangle \geq 0$$

for all $u \in \mathcal{L}_{2e}$ and all $\tau \geq 0$. The operator is said to be strictly output passive *if there exists an $\epsilon > 0$ such that*

$$\langle (H(u))_\tau, u \rangle \geq \epsilon \, ||(H(u))_\tau||^2$$

for all $u \in \mathcal{L}_{2e}$ and all $\tau \geq 0$.

A system that is strictly output passive has an $\mathcal{L}_2$-gain less than $1/\epsilon$ [44]. The following passivity theorem is proved in [42].

Theorem 2.2 (Passivity Theorem [42])**.** *Assume that the feedback system* (2.3) *is well posed and that $u_2 = 0$. If H_1 is strictly output passive and H_2 is passive then the system is stable in the sense that $||e_{2\tau}|| \leq \frac{1}{\epsilon}||u_{1\tau}||$ for all $\tau \geq 0$. If additionally, H_2 is bounded then $||e_{1\tau}|| \leq c||u_{1\tau}||$ for all $\tau \geq 0$ and some $c > 0$.*

Like the small gain theorem, the passivity theorem is only a sufficient condition for stability. The application of loop transforms and multipliers can significantly reduce the conservativeness of the stability condition. The use of loop transforms and multipliers will be introduced in the context of a slightly modified feedback loop. Many nonlinear and uncertain systems can be viewed as the connection of an LTI system with some nonlinear or uncertain operator in a feedback loop. A general feedback system of this type is depicted in Figure 2.4, where the LTI operator G has the block form

$$\begin{aligned} G(s) &= \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \\ G_{ij} &= C_i(sI - A)^{-1}B_j + D_{ij} \end{aligned}$$

with the matrices, A, B_i, C_i, D_{ij} derived from the corresponding state space realization

$$\begin{aligned} \dot{x} &= Ax + B_1 u + B_2 w \\ y &= C_1 x + D_{11} u + D_{12} w \\ v &= C_2 x + D_{21} u + D_{22} w \\ w &= \Delta(v). \end{aligned} \tag{2.4}$$

A common short hand for the operator, G, is

$$G = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]$$

Since the operator G is LTI its $\mathcal{L}_2$-gain, γ_1, and passivity coefficient can be computed explicitly. Often the gain, γ_2, of the nonlinear operator Δ can also be computed, or at least bounded. Nevertheless the small gain theorem may prove to be conservative. On the other hand, by transforming the feedback loop as in Figure 2.5, it is often possible to show that the conditions of the passivity theorem are satisfied for the modified system. Under certain conditions stability of the transformed system is a necessary and sufficient condition for stability of the original loop. The first transformation considered is the addition of the additive operators, H_1 and H_2, to the loop. If H_1 and H_2 are bounded, causal, linear operators and the transformed loop is well-posed, then stability

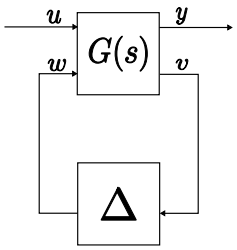

Figure 2.4: An extended system for performance analysis.

of the transformed loop is equivalent to stability of the original loop [42]. The input and output signals of the feedback loop are unaffected by the transformations. On the other hand, the passivity conditions are now applied to the operators $(G - H_2)(I + H_1 G)^{-1}$ and $(\Delta + H_1)(I - H_2\Delta)^{-1}$. If operators H_1 and H_2 can be found that make the modified feedback loop passive, then stability can be assured.

A second transformation that can be made is the introduction of an invertible operator M into the feedback loop. This operator is called a *multiplier*. The multiplier approach to stability analysis originated in the works of Popov [70], Zames [97] and Brockett and Willems [11]. A good summary of the approach can be found in [91]. Under the conditions of the following theorem, the stability of the multiplier transformed loop implies stability of the original system. A proof can be found in [20].

Theorem 2.3 (Multiplier Theorem [31])**.** *Take the system in Figure 2.4 with* $\mathbf{\Delta}$ *a set of* $\mathcal{L}_2 \to \mathcal{L}_2$ *operators. Given a multiplier of the form*

$$M(s) = M_1^*(s) M_2(s) \tag{2.5}$$

with $M_1(s)$ *and* $M_2(s)$ *Hurwitz, and a constant* $\epsilon > 0$ *such that*

$$\int_{-\infty}^{\infty} Re[v^*(j\omega) M(j\omega) w(j\omega)] \mathrm{d}\omega \geq 0, \quad v \in \mathcal{L}_2, \quad w = \Delta(v), \quad \Delta \in \mathbf{\Delta}, \tag{2.6}$$

and

$$M^*(j\omega) G(j\omega) + G^*(j\omega) M(j\omega) \preceq -\epsilon I, \quad \forall\ \omega \in \mathbb{R}, \tag{2.7}$$

then the system is $\mathcal{L}_2$ *stable for all* $\Delta \in \mathbf{\Delta}$*.*

The two conditions of the theorem ensure that the transformed nonlinear operator is passive and that the transformed LTI operator is strictly passive thus ensuring the stability of the feedback loop by the Passivity theorem.

Rather than deriving a multiplier for each particular problem of interest, the search for multipliers can be automated. For some class of systems, such as RNNs with a $\tanh(x)$ nonlinearity, a set of valid multipliers, $\mathcal{M}$, must be specified. The following problem, if solvable, finds a valid multiplier.

Problem 2.1 (Multiplier Opimization Problem)**.**

$$\begin{gathered} \max_{\epsilon, M} \epsilon \ \ s.t. \\ M(j\omega) G(j\omega) + G^*(j\omega) M(j\omega) \preceq -\epsilon I, \quad \forall\ \omega \in \mathbb{R}, \\ M \in \mathcal{M}, \\ \epsilon > 0. \end{gathered}$$

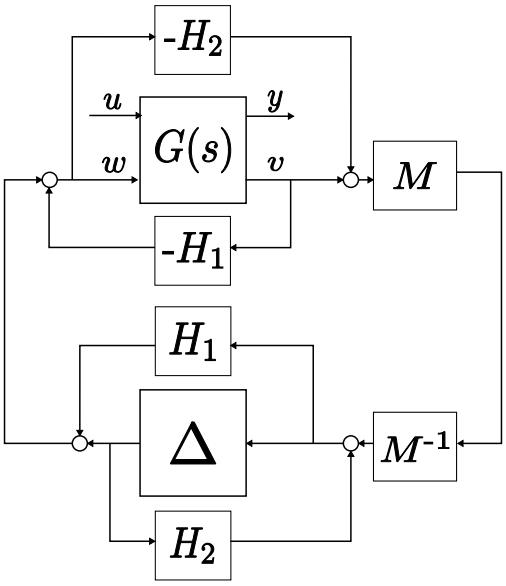

Figure 2.5: The extended system of Figure 2.4 transformed by various loop manipulations.

The constraint, $M \in \mathcal{M}$, poses both theoretical and computational problems. For Problem 2.1 to be efficiently solved the set $\mathcal{M}$ should be convex. Additionally, the representation of $\mathcal{M}$ by linear matrix inequalities and simple linear constraints allows standard LMI software to be applied to Problem 2.1. The determination of a valid set $\mathcal{M}$ and a computationally feasible representation is a difficult problem, but some examples will be seen in the next chapter.

Solving Problem 2.1 requires ensuring that the semi-infinite constraint (2.7) is satisfied. The problem is clearly computationally infeasible, but two approaches exist to circumvent the difficulty. The first approach tests the condition over a finite grid of frequencies, ω [1, 65]. While this is computationally attractive, the approach can not always guarantee stability. An alternative approach gives an exact solution but at a higher computational expense. Application of the following lemma converts the infinite dimensional constraint (2.7) into a finite dimensional constraint with the addition of another decision variable. A proof of the lemma can be found in [72].

Theorem 2.4 (Kalman-Yakubovich-Popov (KYP) [72])**.** *Given $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times p}, M = M^T \in \mathbb{R}^{(n+p)\times(n+p)}$, with $det(j\omega I - A) \neq 0$, for $\omega \in \mathbb{R}$, the statements*

$$\begin{bmatrix}(j\omega I - A)^{-1}B\\ I\end{bmatrix}^* M \begin{bmatrix}(j\omega I - A)^{-1}B)\\ I\end{bmatrix} \prec 0 \quad \forall\ \omega \in \mathbb{R} \cup \{\infty\} \tag{2.8}$$

and

$$M + \begin{bmatrix}A^TP + PA & PB\\ B^TP & 0\end{bmatrix} \prec 0 \text{ for some } P = P^T \in \mathbb{R}^{n\times n} \tag{2.9}$$

are equivalent.

When n is large, the cost of introducing the additional variable, P, can be prohibitively expensive for standard LMI software. These solvers generally have a complexity of $\Theta(m^3)$ per step where m is the number of variables. For KYP problems they have a complexity of $\Theta(n^6)$ per step since the number of variables, m, is of order n^2. Special purpose solvers have been developed for problems derived from the KYP lemma that can be much more efficient for large n [89].

Restricting the multipliers to be causal and decomposable in the multiplier stability theorem restricts the types of multipliers that can be applied. In [59] the method of integral quadratic constraint analysis (IQC) was introduced. The IQC approach to system analysis derives from much of the same literature and research as the multiplier approach and generalizes the multiplier theory by simplifying the conditions on valid multipliers. The main benefit of the IQC formalism is that it allows complex systems to be analyzed by considering descriptions of the system's various components. Also, the IQC paradigm allows general purpose software to be constructed for piecing together IQC models for complex systems and automating the derivation of the necessary optimization problems [43].

An integral quadratic constraint describes the relationship between two $\mathcal{L}_2$ signals, v and w in the following way

$$\int_{-\infty}^{\infty} \begin{bmatrix} \hat{v}(j\omega) \\ \hat{w}(j\omega) \end{bmatrix}^* \Pi(j\omega) \begin{bmatrix} \hat{v}(j\omega) \\ \hat{w}(j\omega) \end{bmatrix} \mathrm{d}\omega \geq 0, \tag{2.10}$$

where $\hat{v}$ and $\hat{w}$ are the Fourier transforms of the two signals [59]. Pairs of signals satisfying the constraint are said to satisfy the IQC given by Π. The IQC Π is taken to have the form

$$\Pi(j\omega) = G_f^*(j\omega)\tilde{M}G_f(j\omega), \tag{2.11}$$

with G_f a bounded, LTI operator and $\tilde{M}$ a constant, symmetric matrix. Some IQCs, called static IQCs, have no dynamic component and consist only of the static matrix $\tilde{M}$.

IQCs that are satisfied by the pair of signals, (v, w), in the feedback system in Figure 2.4 will be of particular interest due to the following theorem proved in [59].

Theorem 2.5 (IQC Stability [31])**.** *Take the system in Figure 2.4 with G a stable, linear, time-invariant operator mapping $\mathcal{L}_2 \to \mathcal{L}_2$ and Δ a causal $\mathcal{L}_2 \to \mathcal{L}_2$ operator with bounded gain. The system is stable if*

1. *for every $\tau \in [0, 1]$, the interconnection of G and $\tau\Delta$ is well-posed;*
2. *there is an IQC, Π, such that for every $\tau \in [0, 1]$, (2.10) is satisfied for $(v, \tau\Delta(v))$;*
3. *there exists an $\epsilon > 0$ such that*

$$\begin{bmatrix} G(j\omega) \\ I \end{bmatrix}^* \Pi(j\omega) \begin{bmatrix} G(j\omega) \\ I \end{bmatrix} \preceq -\epsilon I, \quad \forall\ \omega \in \mathbb{R}. \tag{2.12}$$

The three conditions of the theorem require further elaboration. As previously discussed, well posedness means that for each input the system has a unique solution. For the particular feedback configuration of interest well posedness is given by invertibility of the operator, $(I - \tau G\Delta)$, which can often be checked easily. In regard to the second condition, the following remark from [59] is important. An IQC, Π can generally be partitioned as follows

$$\Pi = \begin{bmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{12}^T & \Pi_{22} \end{bmatrix}.$$

If $\Pi_{11} \preceq 0$ and $\Pi_{22} \succeq 0$ then condition two of the theorem is satisfied for all τ if and only if it is satisfied for $\tau = 1$. Since most IQCs satisfy these inequalities, the second condition of the theorem is usually simple to verify. The third condition is a semi-infinite condition that can be addressed computationally by application of the previously stated KYP lemma. In addition, the search for a valid IQC can be formulated as an optimization problem much as the search for a valid multiplier

was formulated. In fact, the IQC approach generalizes the Multiplier approach. To see this, take the restricted set of IQCs

$$\Pi(j\omega) = \begin{bmatrix} 0 & M(j\omega) \\ M(j\omega)^* & 0 \end{bmatrix} \tag{2.13}$$

where $M(s)$ is a multiplier satisfying (2.6). Note that the IQC approach does not require the factorizability condition (2.5) of the multiplier $M(s)$. Both the multiplier and additive loop transformations can be treated by application of the IQC [42]

$$\Pi(j\omega) = \begin{bmatrix} I & -H_2(j\omega) \\ H_1(j\omega) & I \end{bmatrix}^* \begin{bmatrix} 0 & M(j\omega) \\ M(j\omega)^* & 0 \end{bmatrix} \begin{bmatrix} I & -H_2(j\omega) \\ H_1(j\omega)^* & I \end{bmatrix}.$$

In addition to analyzing stability, IQCs can be formulated for performance analysis [43]. In particular, an IQC can be constructed that bounds the $\mathcal{L}_2$-gain of a system. Consider the extended system depicted in Figure 2.4 and given, in general, by the equations

$$\begin{aligned} \dot{x} &= Ax + B_1 u + B_2 w \\ y &= C_1 x + D_{11} u + D_{12} w \\ v &= C_2 x + D_{21} u + D_{22} w \\ w &= \Delta(v). \end{aligned}$$

Assume the relation $v = \Delta(w)$ satisfies the IQC given, in the time domain, by

$$\sigma_\Delta(\lambda) = \int_0^\infty \begin{bmatrix} v \\ w \end{bmatrix}^T \Pi(\lambda) \begin{bmatrix} v \\ w \end{bmatrix}^T \mathrm{d}t \geq 0$$

where Π is linearly parameterized by λ. Also, define the *performance* IQC by

$$\sigma_p(\gamma^2) = \int_0^\infty \begin{bmatrix} y \\ u \end{bmatrix}^T \begin{bmatrix} I & 0 \\ 0 & -\gamma^2 I \end{bmatrix} \begin{bmatrix} y \\ u \end{bmatrix} \mathrm{d}t \leq 0.$$

If a solution to

$$\inf_{\gamma^2, \lambda} \gamma^2 \quad \text{s.t.} \quad \sigma_p(\gamma^2) + \sigma_\Delta(\lambda) < 0 \tag{2.14}$$

exists, then its square root is an upper bound on the $\mathcal{L}_2$-gain of the system. The σ_Δ condition ensures stability of the system, and the performance IQC ensures that

$$\begin{aligned} \int_0^\infty \begin{bmatrix} y \\ u \end{bmatrix}^T \begin{bmatrix} I & 0 \\ 0 & -\gamma^2 I \end{bmatrix} \begin{bmatrix} y \\ u \end{bmatrix} \mathrm{d}t &= \int_0^\infty y^T y - \gamma^2 u^T u \mathrm{d}t \leq 0 \\ &= ||y||_{\mathcal{L}_2}^2 \leq \gamma^2 ||u||_{\mathcal{L}_2}^2. \end{aligned}$$

Thus, γ is an upper bound on the $\mathcal{L}_2$-gain of the system.

Application of the KYP theorem to the frequency form of the constraints in (2.14) results in the following problem

Problem 2.2 (IQC Performance)**.**

$$\inf_{\gamma, \lambda, P} \gamma \quad s.t.$$
$$\begin{bmatrix} A^T P + PA & PB_1 & PB_2 \\ B_1^T P & 0 & 0 \\ B_2^T P & 0 & 0 \end{bmatrix} + \Sigma \prec 0,$$

where $\Sigma = \Sigma_1 + \Sigma_2$ *and*

$$\Sigma_1 = \begin{bmatrix} C_1 & D_{11} & D_{12} \\ 0 & I & 0 \end{bmatrix}^T \begin{bmatrix} I & 0 \\ 0 & -\gamma^2 I \end{bmatrix} \begin{bmatrix} C_1 & D_{11} & D_{12} \\ 0 & I & 0 \end{bmatrix},$$

$$\Sigma_2 = \begin{bmatrix} C_2 & D_{21} & D_{22} \\ 0 & 0 & I \end{bmatrix}^T \Pi(\lambda) \begin{bmatrix} C_2 & D_{21} & D_{22} \\ 0 & 0 & I \end{bmatrix}.$$

Problem 2.2 can be efficiently solved when Π is linearly parameterized by λ and λ is restricted to a convex set. The cost of directly estimating the $\mathcal{L}_2$-gain is an expansion of the main LMI constraint from a size of $(2n)^2$ to $(3n)^2$.

When the use of a single IQC is insufficient for proving stability or gives a poor estimate of the $\mathcal{L}_2$-gain, multiple IQCs can be applied. The application of multiple IQCs creates non-convex constraint sets which are difficult to optimize over. A relaxation known as the $\mathcal{S}$-procedure reduces the non-convex constraint set to a single convex constraint.

Let V be a linear vector space and $\sigma_k : V \to \mathcal{R}$ for $k = 0, \ldots, N$. The $\mathcal{S}$-procedure is essentially a Lagrangian relaxation technique for converting the generally non-convex condition

$$\mathcal{S}1 : \sigma_0(y) \geq 0 \ \forall \ y \in V \text{ such that } \sigma_k(y) \geq 0, \ k = 1, \ldots, N$$

into the convex sufficient condition

$$\mathcal{S}2 : \exists \ \tau_k \geq 0, \ k = 1, \ldots, N \text{ such that } \sigma_0(y) - \sum_{k=1}^{N} \tau_k \sigma_k(y) \geq 0 \ \forall \ y \in V.$$

Under certain conditions the convex condition, $\mathcal{S}2$, is both necessary and sufficient for condition $\mathcal{S}1$ [42]. Even when it is only sufficient, however, it is useful for constructing computationally tractable constraint conditions.

If the pair of signals, (v, w), can be described by multiple IQCs a less conservative analysis can often be achieved. The resulting set of conditions

$$\begin{bmatrix} G(j\omega) \\ I \end{bmatrix}^* \Pi_k(j\omega) \begin{bmatrix} G(j\omega) \\ I \end{bmatrix} \preceq -\epsilon I, \ k = 1, \ldots, N$$

are converted, via the $\mathcal{S}$-procedure, to the constraint

$$\sum_{i=1}^{N} \tau_k \begin{bmatrix} G(j\omega) \\ I \end{bmatrix}^* \Pi_k(j\omega) \begin{bmatrix} G(j\omega) \\ I \end{bmatrix} \preceq -\epsilon I. \tag{2.15}$$

Under the additional constraint that there exists a pair of signals, (v, w), such that $\sigma_{\Pi_k}(v, w) > 0$ for $k = 1, \ldots, N$, Theorem 2.5 with constraint (2.15) is, in fact, both necessary and sufficient. This is not to say that the conditions of the theorem are necessary for stability of a given system — since the IQCs may be satisfied by multiple operators, Δ — only that the use of the $\mathcal{S}$-procedure to incorporate multiple IQCs into the theorem's conditions does not introduce *further* conservativeness. If the IQCs are linearly parameterized by some λ_k's in a convex cone then the τ_k variables can be absorbed into the λ_k's and removed from the problem [42].

2.4 Solving LMI Problems

A brief introduction to linear matrix inequality problems of the type derived from the KYP lemma will be useful for understanding developments later in the paper. More complete discussions of LMIs and semidefinite programming can be found in [33, 9, 92].

A strict LMI is a constraint of the form

$$A(x) = A_0 + \sum_{i=1}^{n} x_i A_i \prec 0$$

with $x \in \mathbb{R}^n$ and $A_i \in \mathbb{S}^n$. A non-strict LMI relaxes the constraint to allow equality with zero. Optimizing a linear constraint subject to an LMI results in the following problem.

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t. } A(x) \prec 0 \end{aligned} \tag{2.16}$$

The LMI optimization problem (2.16) can be written as a standard semidefinite program (SDP) in *dual* form,

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t. } A(x) + S = 0 \\ &\quad S \succ 0, \end{aligned} \tag{2.17}$$

where $S \in \mathbb{S}^n$ is called a *slack* variable [92]. The associated *primal* form problem is an optimization problem over $\mathbb{S}^n$ given by

$$\begin{aligned} &\max\ A_0 \bullet X \\ &\text{s.t. } A^*(X) + c = 0. \\ &\quad X \succeq 0 \end{aligned} \tag{2.18}$$

where $X \bullet Y = \operatorname{tr} X^T Y$ and

$$A^*(X) = \begin{bmatrix} A_1 \bullet X \\ \vdots \\ A_n \bullet X \end{bmatrix}.$$

The SDPs that arise in this paper will all be written naturally in the dual form. The primal form of the problem is important, however, since the pair of problems (2.17) and (2.18) are related and under certain conditions have the same optimal value.

Linear SDP problems are convex and can be solved in polynomial time. Interior point methods are generally considered to be the most efficient and robust algorithms for solving SDPs. See [93] for a good reference. Primal-dual interior point methods attempt to solve problems (2.17) and (2.18) simultaneously by minimizing the gap between the objective functions subject to the constraints. A good description of the method can be found in the appendices of [89]. Since the number of steps the algorithm takes is independent of problem size, the asymptotic run-time complexity of the primal-dual interior point method for SDPs is determined by the cost of solving the set of linear equations that give the updates for the decision variables. Throughout this paper the Sedumi software is used to solve LMI problems [83].

A different approach that is directly applicable also to nonlinear SDPs is the augmented Lagrangian, or penalty-barrier, approach of [46, 47]. The augmented Lagrangian approach works solely with the dual problem (2.17). The dual problem is modified with a penalty function,

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t. } \Phi(A(x), p) \preceq 0. \end{aligned}$$

The penalty function is chosen to ensure, among other things, that when $p > 0$

$$A(x) \preceq 0 \Leftrightarrow \Phi(A(x), p) \preceq 0.$$

The Lagrangian of the modified problem is

$$F(x, U, p) = c^T x + U \bullet \Phi(A(x), p)$$

where U is the Lagrange multiplier and p is a penalty variable. The algorithm consists of repeated steps of minimizing $F(x, U, p)$ for fixed U and p, followed by updates to U and p. The unconstrained minimization of $F(x, U, p)$ is the most computationally expensive step of the algorithm. Much like the interior point methods, the computational complexity of this step is determined by the cost of solving a set of linear equations, specifically, Newton equations derived from the Hessian of the Lagrangian, $\nabla_x^2 F(x, U, p)$. Rather than solving the Newton equations via a standard Cholesky decomposition, preconditioned conjugate gradient methods can be used. For SDPs with many variables compared to the size of the constraints the conjugate gradient approach has much better time complexity. In addition, the Hessian need not be formed explicitly since it is only used in matrix-vector products. This drastically reduces the memory and run time requirements of solving large scale SDPs since actually forming the Hessian matrix can account for a large portion of the run time and requires a large amount of memory. In the next chapter the primal-dual interior point method and the augmented Lagrangian method will be compared on a set of problems derived from RNN stability conditions. The augmented Lagrangian approach of [47], because of the way it solves the Newton equations, has much better overall time and space complexity.

2.5 Previous work on Stability of RNNs

The stability of RNNs has been addressed from many different viewpoints. Much of the work in this area considers the RNN as a sort of dynamic memory device mapping constant input signals to different fixed values. Stability in this context means that the network states converge for constant inputs. Often the stability analysis allows some uncertainty in the knowledge of the nonlinearity, delays in the interconnections, or noise in the connection weights. This allows application of the stability results to hardware implementations of the networks where heat, defects, and other types of noise affect the behavior of the system. The literature is this area is quite vast but not immediately applicable to the problem at hand where time-varying inputs are the rule. A representative sample can be found in the references [5, 14, 94, 16].

The main point of origin for the research presented in this dissertation is the work of Steil in [79, 82, 80]. Of the most interest are LMI conditions developed in [79] for the stability of both time-invariant and time-varying RNNs with time-varying inputs. In addition, a method of approximating the maximal allowable bounds on weight variations is given in [82]. The conservativeness of these LMI conditions is analyzed and also reduced in Chapters 3 and 4 of the present document.

An LMI condition, identical to the one in [79], is presented in [9] for the analysis of the so called Lur'e system. The Lur'e system is given by the equations [9]

$$\begin{aligned} \dot{x} &= Ax + B_p p + B_w w \\ q &= C_q x \\ z &= C_z x \\ p_i(t) &= \phi_i(q_i(t)),\ \forall\ i = 1, \ldots, n_p, \end{aligned}$$

where $p(t) \in \mathbb{R}^{n_p}$ and the ϕ_i satisfy the conditions

$$0 \leq \sigma\phi_i(\sigma) \leq \sigma^2, \ \forall \ \sigma \in \mathbb{R}.$$

This type of system has been studied since the 1940's, beginning with the work of Lur'e and Postnikov in [56]. The class of RNNs considered in this dissertation are clearly a subclass of the Lur'e systems, so it is no surprise that similar stability analyses have resulted. The analysis in Chapter 3 applies recent results that reduce conservativeness in the case when all of the ϕ_i's are the same. In Chapter 4 the case when B_p is time-varying is considered. The work of Chu in [17, 18] considers the subset of Lur'e systems where the ϕ_i's are equal. Taking this fact into consideration, Chu derived an improved analysis for this specialized class of systems. Papers by D'Amato *et al.* [19] and Kulkrani and Safonov [52] present similar results to the work of Chu in the context of the multiplier and IQC theory. These results are considered in detail in the next chapter.

The class of recurrent networks known as echo state networks (ESN) was mentioned briefly early in this chapter. The defining property of ESNs is the *echo state* property which says that, in the long term, the dynamics of the network are independent of initial conditions and that similar inputs produce similar state trajectories [36]. This class of networks is popular because there exist efficient, well behaved learning algorithms for the weights of ESNs [38]. The efficiency comes from the fact that only a subset of the weights is adapted and that the resulting optimization problem is convex. More general adaptation like that used in standard RNN algorithms gives rise to nonconvex problems with many local minima. The well-behavedness of the learning algorithms comes from the fact that the networks satisfy the echo state property and only the output weights are adapted; the networks do not pass through bifurcations during adaptation [13]. Such bifurcations plague learning algorithms for standard recurrent network architectures by introducing discontinuities into the error surface [21].

Necessary and sufficient conditions can be derived for the echo state property based on algebraic properties of W [37, 12]. For certain classes of weight matrices, such as upper triangular W, these conditions are known to be both necessary and sufficient [12]. The echo state property is essentially an incremental stability property. A system that is incrementally stable maps input signals that are close to state trajectories that are close in the same sense. This type of behavior is analyzed in the work of V. Fromion, specifically [29] which gives conditions that ensure the incremental stability of RNNs. The results presented in the following chapters do not always guarantee incremental stability and so do not, for instance, always guarantee the echo state property. The results do, however, allow bifurcations in network dynamics to be avoided during adaptation. A more in depth study of the relation between the work presented here and the echo state network literature is left for future research.

Chapter 3

Analysis of Time-Invariant RNNs

Research on the stability of recurrent neural networks has yielded a large variety of results. Typically, these results are limited to a particular configuration of the RNN. For example, as discussed in the previous chapter, RNNs can be used as a type of associative memory when the inputs are not time varying. In this situation it is necessary to show that the RNN does indeed converge to a fixed point for some set of constant inputs. Many such results exist; see for example [5, 14]. For time-varying inputs, an RNN is generally seen as a nonlinear, input-output map, and the type of stability of interest is input-output stability. Several stability results are known for this situation. For instance, it is known that when W has a maximum singular value less than one, the RNN will have stable dynamics [36]. Often these criteria are too conservative and limit the optimization of some broader objective. Unfortunately, analytically characterizing the complete set of matrices that result in stable dynamics is difficult, and a computational approach must be taken. In this section an approach is developed that casts the stability analysis of RNNs as a convex optimization problem using the IQC method described in Chapter 2. The method presented here is similar to that in [79], but provides less conservative stability conditions. This will be useful in later chapters where the stability analysis is applied to RNNs in a feedback loop with a controlled system.

Throughout this chapter attention will be fixed on the RNN described by the equations

$$\begin{aligned} \dot{x} &= -Cx + W\Phi(x) + u \\ y &= x \end{aligned} \tag{3.1}$$

where u is the time-varying input signal and y is the output. The matrix C is a fixed, diagonal matrix of positive time constants, and the matrix W represents the time-invariant connection weights between the neurons of the network. In Chapter 4 the case of time-varying W is considered. The function $\Phi(x)$ will be taken as

$$\Phi(x) = [\tanh(x_1)\tanh(x_2)\dots\tanh(x_n)]^T$$

unless explicitly specified otherwise. Determining the stability of such a nonlinear dynamic system is difficult. The approach considered here relaxes the problem such that the stability analysis instead involves sets of linear systems. The problem is relaxed by replacing Φ with a set of operators, $\mathbf{\Delta}$, that contains Φ but has a more tractable representation. This can also be viewed as finding a description of the operator, Φ, that is more amenable to analysis. The methods considered below use descriptions of the nonlinear operator in terms of integral constraints on its behavior. For example, the constraint

$$\langle v, M\Phi(v)\rangle \geq 0, \quad \forall\ M \in \mathcal{M}$$

can convey useful information about Φ if $\mathcal{M}$ can, in turn, be described by a set of linear constraints. These types of integral constraints will often be satisfied for nonlinearities other than Φ; the set of all operators satisfying such constraints is denoted by $\mathbf{\Delta}$. Given descriptions of this type, feasible computational methods for stability analysis can be derived. These methods, however, will capture stability information about the *set* of systems derived from $\mathbf{\Delta}$, and as such, will generally be conservative for the particular nonlinearity of interest. More descriptive constraints on the nonlinear operator will reduce the conservatism of the stability results, but will generally increase the cost of the computational analysis.

The RNN defined by (3.1) can be viewed as a feedback loop between a linear, time-invariant (LTI) system and a nonlinear or uncertain operator. The LTI operator G and state space matrices are given by

$$G = \left[\begin{array}{c|cc} -C & I & W \\ \hline I & 0 & 0 \\ I & 0 & 0 \end{array}\right] \tag{3.2}$$

where $w = \Phi(v)$. Since the system has no feed-through terms, $D_{ij} = 0$, it is well posed for all inputs in $\mathcal{L}_2$. The sections that follow develop a number of different stability criteria for (3.1) ranging from simple small gain criteria to complex LMI based tests. These different results are compared in terms of conservativeness and computational complexity in Sections 3.5 and 3.6.

3.1 Stability by the Small Gain Theorem

Stability of (3.1) can be proved via the small gain theorem if

$$||G||\ ||\Phi|| < 1. \tag{3.3}$$

In general the norm of $\Phi(x)$ will be denoted by β. With $\phi(x) = \tanh(x)$, $\beta = 1$, and the small gain theorem simply requires that $||G|| < 1$. Since G is an LTI operator its norm is readily computed. First note that

$$G(j\omega) = (j\omega I + C)^{-1}W,$$

and that when $C = I$, as assumed for this Chapter, that $||G|| = ||W||$. So the small gain theorem gives a very simple test for stability of the RNN. This condition is equivalent to that developed in the echo state network literature for ensuring the so called echo state property [36]. The examples in Section 3.5 show that the small gain condition is quite conservative. This conservativeness can be reduced by applying a scaling to the operators [12]. Essentially, a multiplier is introduced into both paths in the feedback loop, resulting in

$$\tilde{G(j\omega)} = TG(j\omega)T^{-1}, \text{ and } \tilde{\Phi} = T^{-1}\Phi T.$$

Since the operator Φ is assumed to be diagonal, it commutes with a diagonal scaling T resulting in $\tilde{\Phi} = T^{-1}T\Phi = \Phi$. Applying the small gain theorem to the modified feedback loop results in the condition

$$||TWT^{-1}|| < 1.$$

To reduce the conservativeness of the condition the norm should be minimized over all positive, diagonal matrices, denoted $\mathcal{D}_+$. The resulting optimization problem is

$$\min_{\gamma,T} \gamma \quad \text{s.t. } W^TTW \prec \gamma^2 T,\ T \in \mathcal{D}_+$$

which is an instance of the generalized eigenvalue problem [9]. The problem is quasi-convex and can be solved efficiently using special interior point methods. If a T can be found such that $\gamma < 1$ then the RNN is shown to be stable.

3.2 Stability with Multipliers and Transformations

In [79] an LMI based method for analyzing the stability of RNNs is developed. The LMI is derived by introducing multipliers and loop transformations, and then applying the passivity theorem and KYP lemma. The nonlinearity $\phi(x)$ is assumed to lie in the sector $[0, \beta]$. A loop transformation, $H_2 = \beta^{-1}I$, is applied to normalize the sector and give the modified operator $\Phi(x)(I - \beta^{-1}\Phi(x))^{-1}$ a gain of one. The loop transformation operator, H_1 is taken to be zero. The resulting forward path operator is $G - \beta^{-1}I$.

To apply Theorem 2.3 to the modified RNN feedback system it is necessary to find a class of multipliers that satisfies the first two conditions of the theorem for the nonlinearity $\Phi(x)$. When the function $\phi(x)$ is bounded in a positive sector, that is, when it satisfies the condition

$$\begin{aligned} \alpha x^2 \leq x\phi(x) \leq \beta x^2, \\ 0 \leq \alpha < \beta \quad \alpha, \beta \in \mathbb{R}, \end{aligned}$$

it is easy to show that the set of diagonal matrices with positive entries, $\mathcal{D}_+$, is a valid set of multipliers. Taking $M(s) = T \in \mathcal{D}_+$, condition (2.6) becomes

$$\int_{-\infty}^{\infty} Re[z^*(j\omega)T\Phi(z(j\omega))]\mathrm{d}\omega \geq 0.$$

This condition holds for all $z \in \mathcal{L}_2$ since

$$xT\Phi(x) = \sum_{i=1}^{n} d_i x_i \phi(x_i) \geq \sum_{i=1}^{n} d_i x_i^2 \alpha \geq 0 \quad \forall\, x \in \mathcal{R}^n.$$

The set $\mathcal{D}_+$ is convex and has an efficient representation as a set of n linear constraints of the form $d_{ii} > 0$. Given this set of multipliers the stability of an RNN can be assessed by solving the LMI feasibility problem, Problem 2.1.

The full problem can be derived as follows. Call the modified feed-forward operator, $\tilde{G}$ and let $\tilde{G} = (G - \beta^{-1}I)$, also let $M(j\omega) = T \in \mathcal{D}_+$. Condition (2.7) of Theorem 2.3 can be written as follows

$$\begin{aligned} T\tilde{G}(j\omega) + \tilde{G}(j\omega)T &= TG(j\omega) - \beta^{-1}T + G(j\omega)T - \beta^{-1}T \\ &= \begin{bmatrix} G(j\omega) \\ I \end{bmatrix}^* \begin{bmatrix} 0 & T \\ T & -2\beta^{-1}T \end{bmatrix} \begin{bmatrix} G(j\omega) \\ I \end{bmatrix} \preceq -\epsilon I. \end{aligned}$$

At this point the KYP lemma can be applied. It yields the condition

$$\begin{bmatrix} -CP - PC & PW + T \\ W^T P + T & -2\beta^{-1}T \end{bmatrix} \preceq -\epsilon I \prec 0 \tag{3.4}$$

where $P = P^T > 0$. This result appears multiple times in the literature, see Chapter 6 of [9] for example. In [79], this condition is written as

$$\begin{bmatrix} CP + PC & PW + T \\ W^T P + T & 2\beta^{-1}T \end{bmatrix} \succeq \epsilon I \tag{3.5}$$

but the two formulations are equivalent due to the following result from [8].

Theorem 3.1 (Theorem 1.3.3 from [8]). *If $A, B \succeq 0$ then the block matrix*

$$\begin{bmatrix} A & X \\ X^* & B \end{bmatrix}$$

is positive if and only if $A \succeq XB^{-1}X^*$.

Applying the theorem to both (3.4) and (3.5) shows the equivalence.

3.3 IQC Stability Analysis

Integral quadratic constraint analysis [59] is a formalism for the analysis of complex, nonlinear, and uncertain systems made up of connections between different components. The IQC theory generalizes the multiplier theory by removing some of the constraints on multipliers. It also leads to automated analysis tools such as the IQCβ toolbox [43]. While this tool is not used for the work described here, it is a convenient way to automate the construction of the type of LMIs developed here. This is of particular importance, say, when an RNN is used in a feedback loop with a complicated nonlinear plant that must also be modeled with IQCs.

To formulate the RNN stability problem in the IQC framework it is necessary to find a set of IQCs describing the nonlinearity in the system. A simple IQC describing diagonal nonlinearities in the sector $[0, 1]$ is given by

$$\Pi(j\omega) = \begin{bmatrix} 0 & T \\ T & -2T \end{bmatrix} \tag{3.6}$$

where $T \in \mathcal{D}_+$. For this IQC, condition (2.10) is easily verified for $\Delta(z) = \Phi(z)$ with $\phi(z_i)$ bounded in the sector $[0, 1]$. To see this, consider

$$\begin{aligned} \begin{bmatrix} z(j\omega) \\ \Phi(z(j\omega)) \end{bmatrix}^* \Pi(j\omega) \begin{bmatrix} z(j\omega) \\ \Phi(z(j\omega)) \end{bmatrix} &= \Phi(z)'Tz + z'T\Phi(z) - 2\Phi(z)'T\Phi(z) \\ &= 2\sum_{i=1}^{n} t_i z_i \phi(z_i) - 2\sum_{i=1}^{n} t_i \phi^2(z_i) \\ &= 2\sum_{i=1}^{n} t_i (z_i\phi(z_i) - \phi(z_i)^2) \geq 0. \end{aligned}$$

The last step relies on the relation

$$(\phi(z_i) - \alpha z_i)(\phi(z_i) - \beta z_i) = \phi(z_i)^2 - z_i\phi(z_i) \leq 0$$

which follows from the definition of a sector bounded nonlinearity. Notice that this IQC results in the same stability conditions derived in the previous section by application of loop transformations, multipliers, and the passivity theorem. With this IQC in hand, stability can be proved by finding a particular $T \in \mathcal{D}_+$ for which condition (2.12) is satisfied.

The $\mathcal{L}_2$-gain of an RNN is the maximal amplification of energy in the input signal observed in the output signal. To estimate the $\mathcal{L}_2$-gain of an RNN, Problem 2.2 can be specialized for (3.1). Taking the IQC defined in (3.6), the specialized problem is as follows.

Problem 3.1 (RNN IQC Problem).

$$\inf_{\gamma,T,P} \gamma \quad s.t.$$

$$\begin{bmatrix} -CP - PC + I & P & PW + T \\ P & -\gamma I & 0 \\ W^TP + T & 0 & -2T \end{bmatrix} \prec 0, \quad P = P^T, \quad T \in \mathcal{D}_+.$$

3.4 Additional IQCs for the RNN Nonlinearity

In the previous section a test for the stability of an RNN was developed using the IQC formalism. A simple IQC describing nonlinearities restricted to a positive sector was described and applied to the $\Phi(x)$ function in the RNN system (3.1). While the test in Problem 3.1 is sufficient for proving stability of an RNN, more exact conditions can be constructed by the application of multiple IQCs. Additionally, more descriptive IQCs can be developed for the nonlinearity $\Phi(x)$. In this section several IQCs are described and applied to the RNN stability problem. The $\mathcal{S}$-procedure introduced in the previous chapter is applied to combine the multiple IQC constraints.

3.4.1 Popov IQC

The nonlinearity $\Phi(x)$ does not vary with time, but the simple diagonal multipliers and corresponding IQC (3.6) considered thus far do not account for this and are equally valid for time varying nonlinear operators. An IQC derived from the Popov criteria [70] is only valid for time invariant nonlinear operators and therefore provides a more exact description of the RNN nonlinearity. Results in [79] show that incorporating this constraint can reduce conservativeness in the stability analysis.

The Popov IQC is a dynamic IQC of the form

$$q \begin{bmatrix} 0 & j\omega \\ -j\omega & 0 \end{bmatrix}, \; q > 0.$$

This IQC is not bounded on the imaginary axis and must be used in combination with the loop multiplier, $M(s) = \frac{1}{s+1}$ [59], which yields the combined IQC

$$q \begin{bmatrix} 0 & \frac{j\omega}{1+j\omega} \\ \frac{j\omega}{1-j\omega} & 0 \end{bmatrix}.$$

The modified IQC is bounded and can be applied in combination with the previously presented IQC. The resulting LMI stability condition has the form

$$\begin{bmatrix} -CP - PC & PW + T - QC \\ W^T P + T - QC & QW + W^T Q - 2\beta^{-1} T \end{bmatrix} \preceq -\epsilon I \prec 0$$

where $Q \in \mathcal{D}_+$.

3.4.2 IQCs for Repeated Nonlinearities

In addition to using multiple IQCs to reduce conservatism in the stability analysis, better IQCs — those that account for more properties of the nonlinearity — can further improve the results. The IQC (3.6) is valid for nonlinearities in which $\phi_i(x)$ and $\phi_j(x)$ are different sector bounded functions. In [19] an IQC is developed for *repeated* nonlinearities which have the restricted form

$$\Phi(x) = [\phi(x_1) \; \phi(x_2) \; \ldots \; \phi(x_n)]^T. \tag{3.7}$$

A further restriction can be made to the case when $\phi(x)$ is an odd function, such as $\tanh(x)$. The following result is proved in [19],

Theorem 3.2 (IQC for Repeated, Odd Nonlinearity)**.** *Take $\Phi : \mathcal{L}_2 \to \mathcal{L}_2$ to be a static diagonal operator of the form* (3.7) *with ϕ an odd function. Let $T \in S^{n\times n}$ and satisfy*

$$T_{ii} \geq \sum_{j=1,\ j\neq i}^{n} |T_{ij}| \quad \forall\ i = 1 \dots n. \tag{3.8}$$

Then for any $z \in \mathcal{L}_2$ the following two conditions hold.

1. *If ϕ is monotonically non-decreasing and belongs to a finite sector $[0, k]$, then*

$$\langle \Phi(z), Tz \rangle \geq 0. \tag{3.9}$$

2. *If ϕ has slope in the interval (α, β) then*

$$\langle \Phi(z) - \alpha z, T(\beta z - \Phi(z)) \rangle \geq 0. \tag{3.10}$$

Matrices satisfying (3.8) are called *diagonally dominant* and denoted $\mathcal{S}_{dd}$. All diagonally dominant matrices are also positive definite, but the converse is not true. The set of positive diagonal matrices is a subset of the diagonally dominant matrices. Since $\tanh(x)$ is a monotonically non-decreasing, odd, function with slope in the interval $(0, 1 + \epsilon)$ the condition (3.10) holds. This condition is equivalent to (2.10) with

$$\Pi(j\omega) = \begin{bmatrix} 0 & (1+\epsilon)T \\ (1+\epsilon)T & -2T \end{bmatrix}.$$

Numerically, the $(1 + \epsilon)$ term is irrelevant since ϵ can be taken arbitrarily small. Compared to the IQC (3.6), less conservative results are achievable with this new IQC because the resulting set of valid IQCs is larger. This means that better estimates of the $\mathcal{L}_2$-gain bound of the system can be computed and that some systems that could not be proved stable using (3.6) can be proved stabled.

A result from [52] shows that T need not be diagonally dominant, or even symmetric, to satisfy the conditions of a valid multiplier; T must only satisfy

$$\begin{aligned} T_{ii} &\geq \sum_{j=1,\ j\neq i}^{n} |T_{ij}| \quad \forall\ i = 1 \dots n, \text{ and} \\ T_{ii} &\geq \sum_{j=1,\ j\neq i}^{n} |T_{ji}| \quad \forall\ i = 1 \dots n. \end{aligned}$$

Matrices satisfying this condition are called *doubly dominant*, and the set of such matrices is denoted $\mathcal{M}_{dd}$. The set contains the diagonally dominant matrices and also the positive diagonal matrices. The larger set of IQCs that results can further improve the gain estimates for RNNs and reduce the conservatism in the stability analysis. Since T is not symmetric, the resulting IQC is of the form

$$\Pi(j\omega) = \begin{bmatrix} 0 & T \\ T^T & -(T + T^T) \end{bmatrix}. \tag{3.11}$$

The second condition of the IQC theorem can be simplified as described in Chapter 2 because the lower right hand corner of Π satisfies $-(T + T^T) \succeq 0$.

Using these new IQCs in Problem 3.1 requires restricting T to be in $\mathcal{S}_{dd}$ or $\mathcal{M}_{dd}$. Following [19] the set of diagonally dominant matrices can be described by the equations

$$\begin{aligned}
&T = T^+ - T^-, \\
&T^+, T^- \in S^{n\times n}, \\
&T^+_{ij} \geq 0 \quad \forall\, i,j = 1,\ldots,n, \\
&T^-_{ij} \geq 0 \quad \forall\, i,j = 1,\ldots,n, \\
&T^-_{ii} = 0 \quad \forall\, i = 1,\ldots,n, \\
&T^+_{ii} \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} (T^+_{ij} + T^-_{ij}) \quad \forall\, i = 1,\ldots,n.
\end{aligned}$$

Implementing these constraints requires the introduction of $n(n-2)$ decision variables for T^+ and T^- and n^2 linear constraints. The constraints for T^-_{ii} are implemented implicitly — in the dual form problem — in the definition of T^-. For the IQC of [52], twice as many variables and constraints are necessary since T is not symmetric. Whether or not the additional computational cost is worthwhile depends on the application. In the next section several examples are developed that highlight the difference in cost and conservativeness of the different IQCs.

3.5 Experimental Evaluation

In this section the stability analysis described in the previous sections will be illustrated on a few example networks. The different IQCs are compared on various network sizes to provide insight into the trade-off between computational cost and conservativeness in the stability analysis. For all of the examples in this section $\phi(x)$ is the tanh function and time constants are all equal to one, $C = I$.

3.5.1 Analysis of Simple Network

An analysis of a simple RNN with $W \in \mathbb{R}^{n\times n}$ will illustrate some important properties of the different stability conditions. Define the matrix $W(a,b)$ as

$$W(a,b) = \begin{bmatrix} -.4326 & a \\ b & .2877 \end{bmatrix}$$

By allowing a and b to vary over the range $[-8,8]$ the conservativeness of different conditions can be compared visually. The first experiment compares the set of weight matrices that could be proven stable using the small gain theorem, the scaled small gain theorem, and the IQC method with IQC (3.6) (equivalently, the LMI (3.4) from [79]). The results are shown in Figure 3.1. The results show that the small gain theorems results in a more conservative analysis than the LMI approach, even with a simple IQC. On the other hand, the two small gain theorems — especially the simple condition (3.3) — are more computationally efficient.

Figure 3.2 shows the set of (a,b) pairs for which stability could be proven by the IQC approach in Problem 3.1 with different IQCs. Three cases were compared where T was restricted to be in $\mathcal{D}_+$, $\mathcal{S}_{dd}$, or $\mathcal{M}_{dd}$. In the 2×2 case the IQC from [19], with T diagonally dominant, does not produce much of a reduction in conservativeness. As shown in the next section, this is not the case for larger networks. The IQC of [52], with T doubly dominant, does reduce the conservativeness

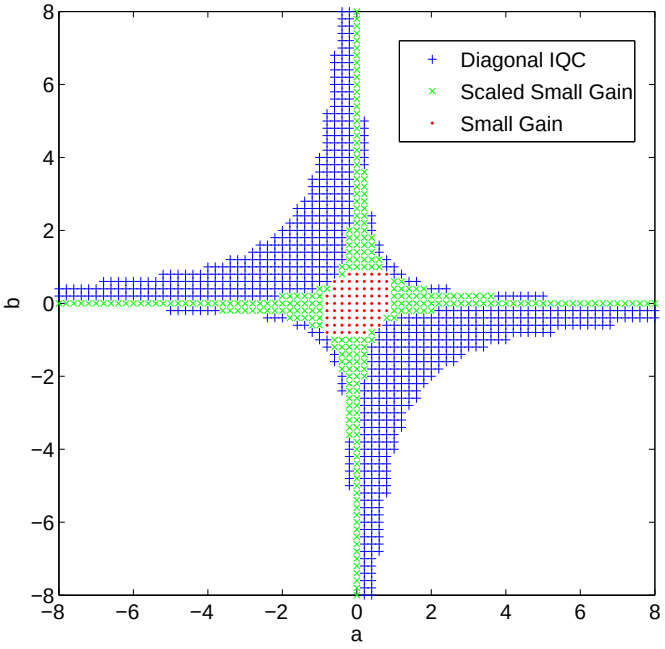

Figure 3.1: A comparison of the diagonal IQC approach with the small gain and scaled small gain theorems on a 2×2 RNN.

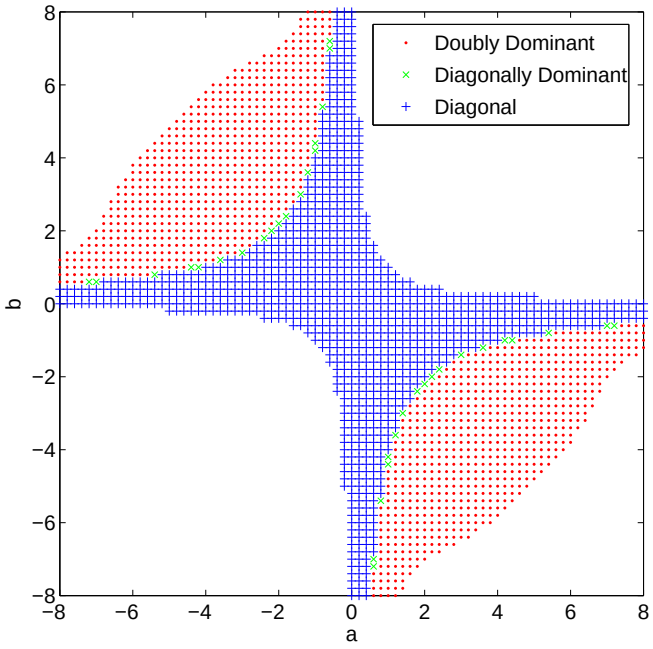

Figure 3.2: A comparison of different IQCs applied to a 2×2 RNN.

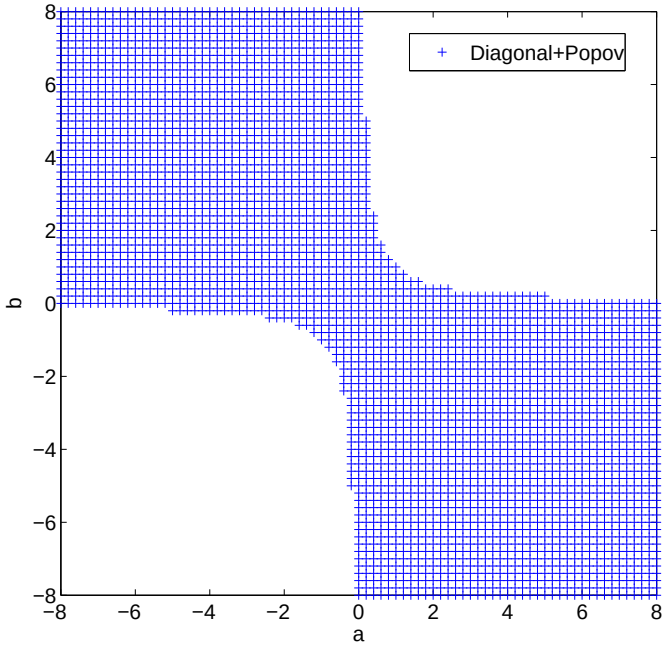

Figure 3.3: The set of matrices for which stability is given by the IQC 3.6 and the Popov IQC.

quite a bit, but as the next example will show, inclusion of the Popov IQC provides a much greater reduction in conservativeness for this small network.

Figure 3.3 shows the set of weight matrices proved stable by the use of the diagonal IQC and the Popov IQC. Addition of the Popov IQC expands the set of provably stable weight matrices and makes the stable sets for the three nonlinearity IQCs the same in this 2×2 example. Note, however, that the computed gains are not the same and more complex IQCs result in better upper bounds.

3.5.2 $\mathcal{L}_2$-gain Estimation Examples

Applying the IQC stability analysis allows an upper bound on the $\mathcal{L}_2$-gain of an RNN to be computed. A good estimate of this gain can allow the small gain theorem to be applied when the RNN in connected in a loop with other nonlinear systems. Better estimates of the gain allow a wider range of interconnected systems to be proven stable. To illustrate, upper bounds on the $\mathcal{L}_2$-gains for the RNN from the previous section are computed using the doubly dominant IQC. These gains are shown in Figure 3.4. The bound on the $\mathcal{L}_2$-gain increases exponentially as the weight matrix nears the stability boundary. This makes sense because the stability analysis declares stable only RNNs for which a finite upper bound on the gain can be computed.

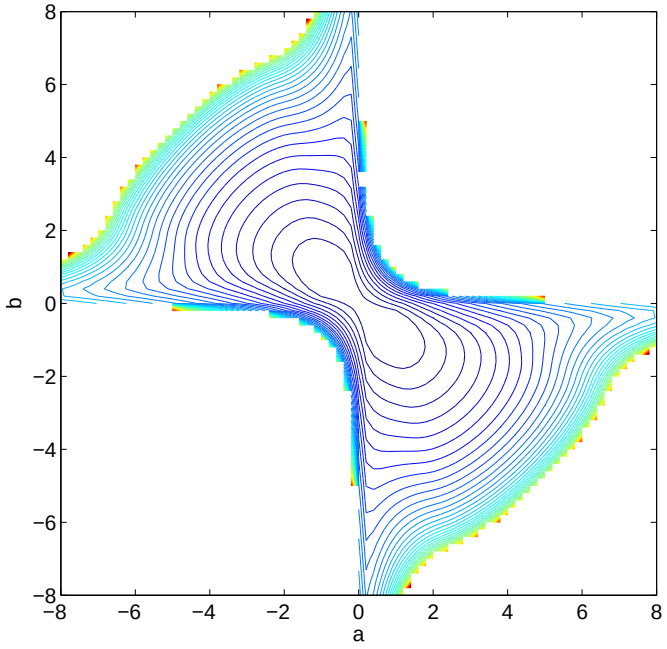

Figure 3.4: The estimated $\mathcal{L}_2$-gains for the stable weight matrices from application of the doubly dominant IQC. There are some edge effects in the figure due to the coarseness of sampling.

In the next experiment, gains are computed for RNNs with the following four weight matrices

$$W_1 = \begin{bmatrix} -0.4326 & 0.1253 \\ -1.6656 & 0.2877 \end{bmatrix}, \quad W_2 = 5W_1,$$

$$W_3 = \begin{bmatrix} 0.3274 & -0.3744 & -1.2173 & 0.9535 & -0.2624 \\ 0.2341 & -1.1859 & -0.0412 & 0.1286 & -1.2132 \\ 0.0215 & -1.0559 & -1.1283 & 0.6565 & -1.3194 \\ -1.0039 & 1.4725 & -1.3493 & -1.1678 & 0.9312 \\ -0.9471 & 0.0557 & -0.2611 & -0.4606 & 0.0112 \end{bmatrix},$$

$$W_4 \in \mathbb{R}^{30\times 30}.$$

The weight matrix W_4 has elements drawn independently from a standard normal distribution. It has the following properties: $\max \text{ Re } (\lambda(W)) = 0.8368$, $|\lambda_{\max}(W)| = 0.8805$ and $\|W\| = 1.8666$.

The following definitions are made for brevity:

- γ_{sg} is the gain estimated using the small gain theorem [79],

$$\gamma_{sg} = \frac{1}{1 - ||W||},$$

- γ_{ssg} is the gain estimated by the scaled small gain theorem [79],

$$\gamma_{ssg} = \frac{||T||||T^{-1}||}{1 - ||TWT^{-1}||}$$

	γ_{sg}	γ_{ssg}	γ_d	γ'_d	γ_{dd}	γ'_{dd}	γ_{ks}	γ'_{ks}
W_1	-	20.5633	1.8374	1.8304	1.8315	1.8304	1.8304	1.8304
W_2	-	-	-	-	27.8302	12.2278	21.3654	12.2278
W_3	-	-	77.3804	8.2504	2.7857	2.1354	1.9689	1.8552
W_4	-	-	-	-	17.7149	15.4765	13.6553	13.2346

Table 3.1: Results for estimating the gain of several RNNs using different techniques. A '-' signifies the inability of the given approach to prove stability.

- γ_d is the gain estimated using the IQC approach with IQC (3.6) and $T \in \mathcal{D}_+$,
- γ_{dd} is the gain estimated using the IQC approach with IQC (3.11) and $T \in \mathcal{S}_{dd}$,
- γ_{ks} is the gain estimated using the IQC approach with IQC (3.11) and $T \in \mathcal{M}_{dd}$.

To denote addition of the Popov IQC, the notations γ'_d, γ'_{dd}, and γ'_{ks} are used. The results of the different gain estimates are shown in Table 3.1. The results illustrate the decreasing conservativeness of the more computationally involved approaches. In the next section the question of conservativeness is addressed in more detail. In Chapter 5 the effect of conservativeness in the stability analysis on the ability of an adaptive system to optimize performance will be illustrated.

3.6 Discussion

The experiments in the previous section illustrated the general behavior of the proposed stability analysis method and showed the effects of different IQCs on the analysis. In this section three properties of the stability analysis are addressed in more detail. In the presence of constant inputs the stability analysis ensures that the RNN (3.1) is asymptotically stable. There is then a relationship between the space of weights matrices for which static stability can be proven and points in the weight space at which bifurcations of the RNN dynamics occur. This is discussed in more detail in the next section. Closely related is the issue of continuity in the network dynamics and the relationship of continuity to stability. The use of different IQCs affects the meaning of the stability results, in the sense that the use of some IQCs ensure, in addition to stability, continuity of the solutions. Finally, a number of computational issues related to solving Problem 3.1 are explored in the last section.

3.6.1 Stability and Bifurcations

The idea of input-output stability is related to the more common notions of stability, such as Lyapunov stability for unforced systems. If a system is input-output stable then the origin is an asymptotically stable fixed point for the zero input case [44]. The relationship ties the notion of input-output stability over a set of RNN weight matrices to the occurrence of bifurcations. In Figure 3.5 two phase diagrams are shown. The phase diagrams depict the RNN flow with a constant zero input. In the left plot, the weight matrix has $a = b = 1$ which is a point just inside the region of provable stability. There is a single globally attractive fixed point at the origin. In the right plot, the weight matrix has $a = b = 1.1$ which is a point just outside the stable region. The system has three fixed points: an unstable fixed point at the origin and two stable fixed points, symmetric about the line $y = -x$. This behavior is typical of a system passing through a *pitchfork* bifurcation [69]. The key feature of this example is that the IQC stability analysis is non-conservative with respect to this feature at this particular point in the weight space.

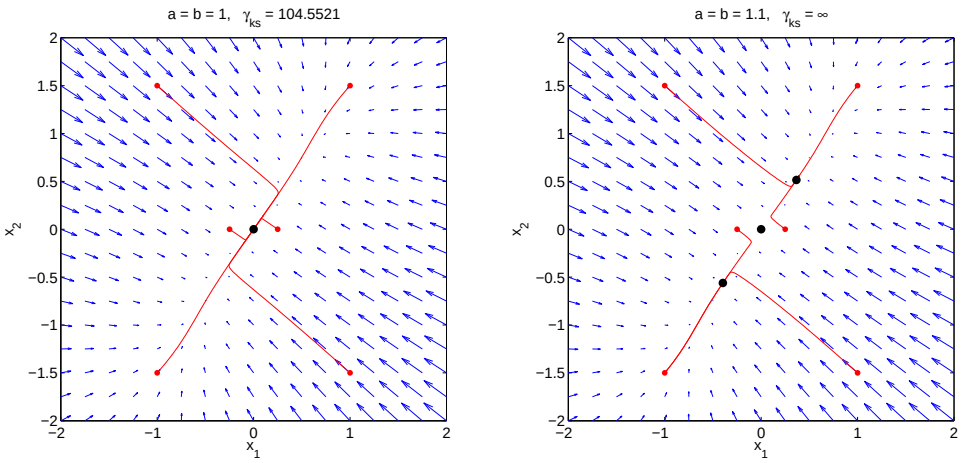

Figure 3.5: An example of a bifurcation occurring in the zero-input dynamics of an RNN. In the left plot, there is a single stable fixed point at the origin. In the right plot, there are three fixed points marked by large dots; the system has passed through a pitchfork bifurcation. The stability analysis with any IQC is non-conservative with respect to this transition. Sample trajectories are shown with the starting points marked by the smaller dots. The arrows represent the vector field induced by the zero input RNN equations.

Standard nonlinear systems analysis [69] shows that a fixed point is stable if all the eigenvalues of the system's Jacobian have negative real part at the fixed point. When evaluated at the origin, the Jacobian of the RNN equation (3.1) is given by

$$\left[\frac{\partial F_i}{\partial x_j}\right]_{ij} = W - I$$

where $F(x)$ is the right hand side of (3.1). The fixed point at the origin is stable if

$$\text{Re } \lambda_i(W - I) < 0, \quad \forall\, i \in 1, \ldots, n, \tag{3.12}$$

where λ_i is the ith eigenvalue of its argument. This condition is necessary for input-output stability as defined in Chapter 2. To see this consider the case where another point besides the origin is an asymptotically stable fixed point. For zero input, the state will converge to this nonzero fixed point for some initial conditions, and thus the output will have unbounded norm; there can be no finite gain bounding this signal amplification. While the condition is obviously necessary it is not necessarily sufficient.

For the 2×2 RNN example, the set of weight matrices that satisfy this necessary condition is shown in Figure 3.6. Clearly, the set is equivalent to the set of weight matrices proven stable by use of the diagonal and Popov IQCs. In this 2×2 case it appears that (3.12) may be both necessary and sufficient. The next example explores the issue further.

Take W to be the 3×3 matrix

$$W = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ a & -2 & 0 \end{bmatrix}. \tag{3.13}$$

Some example dynamics of this system were shown for $a = -4$ in Figure 2.2 of Chapter 2. The system has a stable limit cycle and an unstable fixed point at the origin. When $a > -4$ the origin is an attracting, stable, fixed point. So, as the system passes from $a > -4$ to $a \leq -4$ it passes through

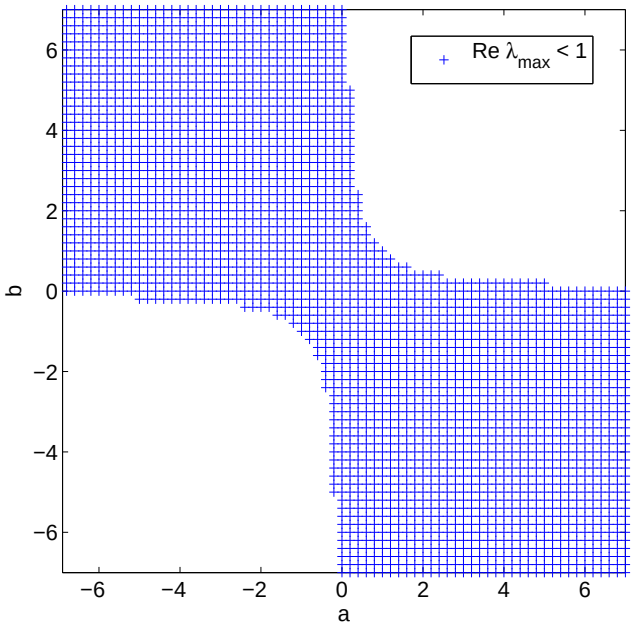

Figure 3.6: Weight matrices with Re $\lambda_i(W - I) < 0$.

a Hopf bifurcation [69]. In passing through this bifurcation, two of the eigenvalues get a positive real part. This explains why the origin is no longer an asymptotically stable fixed point. Passing through the bifurcation, makes $\mathcal{L}_2$-gain stability of the system impossible. Table 3.2 shows the minimum value of $a \in [-4, 0]$ at which different combinations of IQCs are able to prove stability.

It appears from these results that the scaled small gain theorem is not as conservative as the IQC approaches with respect to this transition. The use of doubly dominant IQCs with the Popov IQC, however, also transitions from stability to instability at $a = -4$ and so appears to be non-conservative in this case. These two examples suggest that there may exist conditions where the eigenvalue condition (3.12) is sufficient as well as necessary for stability. To understand the problem in more detail, the issue of continuity needs to be discussed.

3.6.2 Incremental Positivity and Continuity

The stability results presented thus far give sufficient conditions for an input-output operator, in particular (3.1), to map signals from $\mathcal{L}_2$ to $\mathcal{L}_2$. Following the definition of Zames [97], stability also requires that the input-output operator be continuous. To establish continuity in terms of the IQC or multiplier theory, it is necessary not only for the IQC or multiplier to preserve positivity (2.6), but also to preserve *incremental* positivity [20]. An operator, F, is incrementally positive if [53]

$$\langle x - y, F(x) - F(y) \rangle = \int_{-\infty}^{\infty} (x(j\omega) - y(j\omega))^* (F(x(j\omega)) - F(y(j\omega))) \mathrm{d}\omega \geq 0, \quad \forall x, y, F(x), F(y) \in \mathcal{L}_2.$$

For example, the operator Φ in the RNN is incrementally positive since it is monotone increasing.

For a multiplier, M, to preserve incremental positivity of an operator, F, it must satisfy

$$\langle x - y, M\left(F(x) - F(y)\right) \rangle \geq 0.$$

a_{sg}	a_{ssg}	a_d	a'_d	a_{dd}	a'_{dd}	a_{ks}	a'_{ks}
-	-3.998	-	-3.235	-	-3.420	-2.815	-3.999

Table 3.2: Minimum value of $a \in [-4, 0]$ at which different combinations of IQC give stability of an RNN with weight matrix 3.13. Recall that a' represents the addition of the Popov IQC.

The following example shows that the multipliers in $\mathcal{S}_{dd}$ and $\mathcal{M}_{dd}$ do not preserve incremental positivity. Consider the diagonally dominant matrix

$$G = \begin{bmatrix} 30.1273 & -4.3628 & 0.0000 & 5.5058 \\ -4.3628 & 46.9257 & -11.0083 & -0.0000 \\ 0.0000 & -11.0083 & 33.6067 & 0.0000 \\ 5.5058 & -0.0000 & 0.0000 & 93.3303 \end{bmatrix}.$$

Let the two signals, x and y, have the constant values

$$\begin{aligned} x &= \begin{bmatrix} 1.3683 & 16.1398 & 0.8666 & 1.4473 \end{bmatrix} \text{ and} \\ y &= \begin{bmatrix} 3.1623 & 1.9500 & -0.2566 & 1.4126 \end{bmatrix} \end{aligned}$$

for $t \in [0, 2]$ and to be zero elsewhere. Also, take $F(x)$ to be the $\tanh(\cdot)$ function. The signals are clearly in $\mathcal{L}_2$, yet

$$\langle x - y, F(x) - F(y) \rangle = 3.6862,$$

and

$$\begin{aligned} \langle x - y, G\left(F(x) - F(y)\right) \rangle &= \int_0^\infty (x(t) - y(t))^T G\left(F(x(t)) - F(y(t))\right) \mathrm{d}t \\ &= -18.5472 \int_0^2 \mathrm{d}t \\ &< 0. \end{aligned}$$

The multipliers in $\mathcal{S}_{dd}$ and $\mathcal{M}_{dd}$ do not in general preserve the incremental positivity of the nonlinearities of the RNN, and thus the multiplier theory does not in general give a sufficient condition for continuity of the RNN when these multipliers are used. The same result can be shown for the IQC conditions when T is in $\mathcal{S}_{dd}$ or $\mathcal{M}_{dd}$. So, while the output of the system is guaranteed to be bounded and in $\mathcal{L}_2$, it may be critically sensitive to the input. That is, a small perturbation of the input may produce a *change* in the output that is unbounded [30]. In [53] the Popov multiplier is also shown not to preserve incremental positivity. Additionally, it is proved that the inputs for which non-continuity can be observed are always time varying.

Having made these observations, it is then important to realize that even though the results of the previous section suggested the possibility that the eigenvalue condition on W might be necessary *and* sufficient for stability in certain cases, that the condition does not address sensitivity to inputs. The importance of this distinction is, to some extent, application dependent. For example the continuity property is essential to the definition of the echo state property [36], and the eigenvalue condition is not sufficient for this application. For other applications it may be sufficient to ensure that the output signal is in $\mathcal{L}_2$, and for these applications it might be possible to prove sufficiency of the eigenvalue condition for certain cases.

3.6.3 Computational Issues

There are a number of computational considerations that must be made when solving the IQC or multiplier optimization problem. In this section some run-time and complexity comparisons are made between an interior point solver, Sedumi [83], and an augmented Lagrangian solver, PENBMI [46, 47]. First, two results that reduce the complexity of the LMI problems are developed. Recall the main LMI stability constraint derived from the multiplier theory,

$$\begin{bmatrix} CP + PC & PW + T \\ W^T P + T & 2\beta^{-1} T \end{bmatrix} \succeq \epsilon I. \tag{3.14}$$

It was noted in [72] that if the upper left corner of M in the KYP lemma is positive semidefinite, the existence of a solution implies that P is positive definite. Thus it is not necessary to explicitly constrain P to be positive definite for (3.14), since for the constraint considered the upper left block of M, specifically 0, is positive semidefinite. Additionally, when T is taken to be a positive diagonal matrix an explicit constraint on the positivity of T is not necessary. The following relation from [98] is useful; if $A \succ 0$ then

$$\begin{bmatrix} A & X \\ X^* & B \end{bmatrix} \succ 0 \Rightarrow B \succ X^* A^{-1} X. \tag{3.15}$$

Since C is assumed to be a positive diagonal matrix and $P \succ 0$, the matrix $A = CP + PC$ is also positive definite. Let the matrix $B = 2\beta^{-1}T$ and $X = PW + T$. The relation (3.15) implies that

$$2\beta^{-1}T \succ (PW + T)^*(CP + PC)^{-1}(PW + T) \succ 0.$$

The last inequality follows from two basic properties of positive definite matrices. First, if $A \succ 0$ then $A^{-1} \succ 0$. Second, if $A \succ 0$ then $B^*AB \succ 0$ for any matrix B of conforming size [34]. So, by the previous arguments, T must be positive definite if the LMI (3.14) is satisfied. Applying these observations reduces the complexity of implementing the stability result by removing one order n^2 matrix constraint and n scalar constraints. Similar arguments can by applied to the IQC derived constraint in Problem 3.1.

Since the number of variables in the LMI conditions grows with the square of the dimension of the RNN and the complexity of an iteration in the SDP solvers is on the order of the cube of the number of variables, the overall complexity should grow as n^6. The PENBMI software, which implements an augmented Lagrangian approach to solving SDPs, avoids explicitly forming a Hessian matrix and uses a preconditioned conjugate gradient to solve the Newton equations at each iteration [47]. The method generally takes more iterations than interior point methods, but overall a significant time and space savings are achieved by this technique.

The PENBMI solver was compared to the interior point method solver, Sedumi [83], on a set of RNN stability problems of increasing size. The LMI condition (3.5) was used, and a comparison was made between $T \in \mathcal{D}_+$ and $T \in \mathcal{M}_{dd}$. In the diagonal case there is a single $2n \times 2n$ LMI constraint and $n(n+3)/2$ decision variables. In the doubly dominant case the problem has one $2n \times 2n$ LMI constraint plus $2n$ linear constraints for ensuring that $T \in \mathcal{M}_{dd}$. The problem has $5n(n-1)/2$ decision variables for representing P and T. Figure 3.7 shows the time spent in the SDP solver for different values of n averaged over ten random, but stable, RNNs.

The results clearly show both the added expense of taking $T \in \mathcal{M}_{dd}$ but also the benefit of not explicitly forming a Hessian matrix. Further analysis shows that the run time of Sedumi does indeed grow at a rate dominated by an n^6 term for both IQCs. For the PENBMI solver, the run time of the diagonal IQC problems grows at a rate of n^3, and the doubly dominant IQC problems grow at a rate of n^4. This difference is likely due to the effect of T being diagonal on the complexity of evaluating the Hessian-vector products necessary for solving the Newton equations.

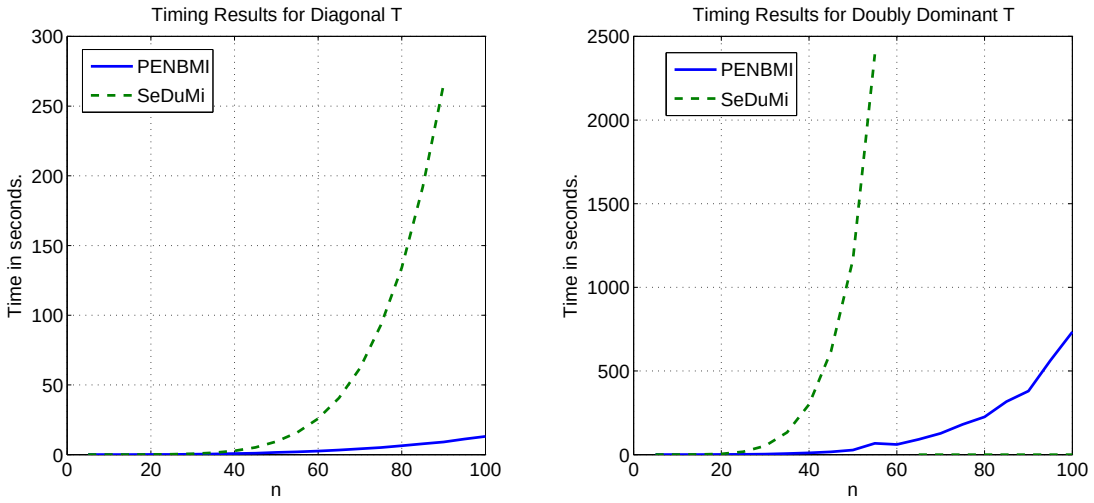

Figure 3.7: Average running times for LMIs of increasing sizes with PENBMI and Sedumi. The left figure shows the average run times when using the simple diagonal IQC. The figure on the right shows the run times for using the doubly dominant IQC. Note that in the case of $T \in \mathcal{D}_+$ Sedumi fails for $n > 90$ and for $T \in \mathcal{M}_{dd}$ Sedumi fails for $n > 55$. These failures are due to excessive memory consumption.

Estimating the $\mathcal{L}_2$-gain of a network directly, as in Problem 3.1, creates a computational difficulty when RNNs are almost unstable. For these RNNs the computed upper bound on the $\mathcal{L}_2$-gain is very large. This leads to ill-conditioning of the resulting LMI problem as the γ variable is much larger than the other decision variables. In Figure 3.8 run times for solving the 2×2 RNN stability problems from the previous section are shown. For this example T is restricted to be doubly dominant. The run times trend higher as the $\mathcal{L}_2$-gain increases for both solvers. For the PENBMI solver the run times for detecting infeasibility of the LMI problem are on average higher than the run times for feasible problems. For Sedumi the opposite is true, but the difference is less pronounced.

The results suggest that the PENBMI solver is a good choice due to the low complexity of the iterations. On the other hand, the PENBMI solver does exhibit increased run times for infeasible problems that the Sedumi solver does not suffer from. Other general purpose SDP solvers were applied to the LMI problems developed in the chapter and all performed similar to Sedumi. This suggests that the cost of general purpose interior point methods for SDPs is prohibitive for the analysis of large RNNs. The special purpose solver in [89] is efficient for SDPs derived from the KYP lemma, but only in the case where B has fewer columns than A (see Theorem 2.4). For the LMI derived for RNN stability B has at least as many columns as A and more if γ is estimated as part of the optimization problem. Whether or not a suitable modification exists that makes the method efficient for these types of problems is left as a question for future research.

3.7 Conclusions

In this chapter a method for assessing the stability of recurrent neural networks with time-invariant weights was developed in the framework of IQC analysis. The method was compared to an LMI approach developed in [79] and shown to be the same when simple diagonal multipliers are used. Better IQCs were incorporated into the analysis and shown to reduce its conservatism. In the next chapter when the maximal amount of allowable variation in the RNN weights is sought, this reduced conservatism is helpful. The computational cost of the LMI stability conditions was reduced by

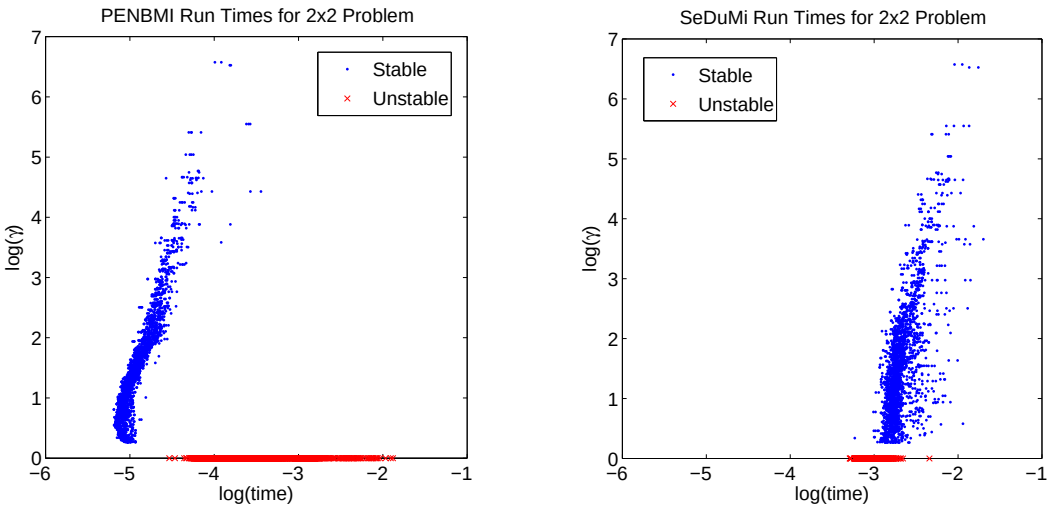

Figure 3.8: Estimating the $\mathcal{L}_2$-gain directly as part of the LMI problem makes the problem ill-conditioned near the stability boundary. The ill-conditioning increases the run time of the solvers.

exploiting some of the properties of the main IQC LMI constraint. Additionally, the PENBMI solver was shown to drastically reduce the run times for the analysis of large networks because it does not explicitly form a Hessian of the problem's Lagrangian. In Chapter 5 a large number of these LMI problems must be solved as part of a stable adaptive control framework. The ability to solve this type of LMI problem quickly is very important. Unfortunately, the conservatism and computational complexity of the stability analysis are clearly at odds even when an efficient LMI solver is available. Further reduction in the complexity of the analysis is important and may for example require the use of multipliers that lie somewhere in between the diagonal multipliers and the doubly dominant multipliers. For example, a block diagonal matrix T consisting of doubly dominant blocks could be constructed for use in the IQCs. This type of construction would allow a more fine grain control of the trade-off between complexity and conservativeness.

The use of the IQC (3.11) from [52] in combination with the Popov IQC was shown to result in a non-conservative analysis with respect to certain bifurcations in two different examples. The exact relationship between the necessary stability condition of $\lambda_{\max}(W) < 1$ and the sufficient conditions given by the LMI problems is, however, still unclear. The examples provided suggest that further exploration of this relationship is warranted and may produce simplified stability conditions under certain assumptions. The relation of the stability conditions presented in this chapter and the so-called echo state property is also not completely clear. At least for the case when T is positive diagonal and the Popov IQC is not used, the LMI conditions may provide a less conservative test for the echo state property.

Chapter 4

Stability of Time-Varying RNNs

The stability analysis presented in the previous chapter is applicable to time-invariant recurrent neural networks. Often, however, the weights of an RNN are tuned online to improve some performance measure. It is important in this situation to assure more than the stability of each individual RNN visited during the adaption. The transitions between different weight matrices must be considered explicitly. This chapter addresses the problem of determining the stability of time-varying RNNs. In Section 4.1, a sufficient condition for stability is formulated as a feasibility problem with matrix constraints. This result is applicable to RNNs whose weights are known to vary within fixed ranges. For application to adaptive control systems it is useful to compute the largest ranges of variation under which stability can be assured. In Section 4.2, a method for solving this problem is described. Computationally, this problem is more difficult than those of the previous chapter and the problem in Section 4.1. Several different solution methods are described and compared on example problems.

4.1 Analysis of RNNs with Dynamic Weights

Establishing the stability of an RNN with time-varying weights is not as simple as establishing the stability of the RNN at each value of W through which it passes. To see why this is not sufficient consider the pair of linear systems, $\dot{x} = A_i x$, with

$$A_1 = \begin{bmatrix} -0.9 & 0 \\ 10 & -0.9 \end{bmatrix}, \quad A_2 = \begin{bmatrix} -0.9 & 10 \\ 0 & -0.9 \end{bmatrix}.$$

Each system is stable — the system $\dot{x} = A_i x$ converges for all $x(0)$ — since $\lambda_{\max}(A_i) = -0.9 < 0$. Nevertheless, the time varying system

$$\dot{x} = \begin{cases} A_1 x & \lfloor t \rfloor \text{ is even}, \\ A_2 x & \lfloor t \rfloor \text{ is odd}, \end{cases}$$

is unstable. The system has the solution [77]

$$x(t) = [e^{A(\lfloor t \rfloor)(t - \lfloor t \rfloor)} e^{(\lfloor t \rfloor / 2)(A_1 + A_2)}] x(0)$$

which grows without bound for $x(0) \neq 0$ since $\lambda_{\max}(A_1 + A_2) > 0$. From the example it is obvious that considering only the stability of the individual values of W will not be sufficient for guaranteeing stability of the time varying system.

Since the variation of the RNN weights must be explicitly accounted for in the stability analysis, the RNN equations from Chapter 3 must be modified. The variation in the RNN weights is written as an additive perturbation to some fixed weight matrix, W, with the equations

$$\begin{aligned} \dot{x} &= -Cx + (W + \Delta W)\Phi(x) + u, \\ y &= x. \end{aligned} \tag{4.1}$$

Other formulations, such as writing the variation as a multiplicative perturbation are also possible, but the additive perturbation is more easily analyzed. The elements of ΔW, ΔW_{ij}, vary independently with time in the ranges $[-\underline{\Delta}_{ij}, \overline{\Delta}_{ij}]$. For convenience the following definitions are made

$$\begin{aligned} \overline{\Delta} &= \text{diag}\{\overline{\Delta}_{11}, \overline{\Delta}_{12}, \ldots, \overline{\Delta}_{nn}\}, \text{ and} \\ \underline{\Delta} &= \text{diag}\{\underline{\Delta}_{11}, \underline{\Delta}_{12}, \ldots, \underline{\Delta}_{nn}\}. \end{aligned}$$

The stability analysis of (4.1) can be framed in two ways. First, is (4.1) stable for a given set of variation bounds, $\underline{\Delta}$ and $\overline{\Delta}$. Second, for a given W and C, what is the maximum amount of variation under which stability of (4.1) can be guaranteed. A more concrete definition of maximum amount of variation will be given in Section 4.2, where the problem is addressed in detail.

The stability of a time-varying system such as (4.1) can be addressed in a number of ways. In this chapter the IQC theory is applied to the problem and results in a characterization of stability involving matrix constraints. To apply the IQC method the time varying parameters and nonlinearity must be separated from the LTI part of the system into a feedback formulation. With the appropriate IQCs defined the IQC theorem from Chapter 2 can then be applied.

4.1.1 IQC Analysis of Time-Varying RNNs

IQC models of the RNN nonlinearity were discussed in the previous chapter. All of the IQCs defined there are applicable in the time-varying case as well. There are several approaches to modeling the time varying parameters with IQCs. If ΔW is known to vary within a polytope, $\mathcal{P} = \bar{co}\{\Delta_1, \ldots, \Delta_N\}$ then the relation $w(t) = \Delta W(t)v(t)$ satisfies the IQC [59, 43]

$$\int_{-\infty}^{\infty} \begin{bmatrix} \hat{v}(j\omega) \\ \hat{w}(j\omega) \end{bmatrix}^* \begin{bmatrix} Z & Y \\ Y^T & -X \end{bmatrix} \begin{bmatrix} \hat{v}(j\omega) \\ \hat{w}(j\omega) \end{bmatrix} d\omega \geq 0$$

if $X = X^T \succeq 0$, $Z = Z^T$, and

$$\begin{bmatrix} I \\ \Delta_i \end{bmatrix}^T \begin{bmatrix} Z & Y \\ Y^T & -X \end{bmatrix} \begin{bmatrix} I \\ \Delta_i \end{bmatrix} \geq 0 \quad \forall\ i \in 1, \ldots, N. \tag{4.2}$$

When nothing is known about the variation of ΔW, a full rank polytope must be used. In other words, ΔW is allowed to vary in a hypercube defined by 2^{n^2} vertices. This results in an exponential number of constraints of the form (4.2). Obviously, this is an unreasonable computational burden when n is large.

Rather than modeling the time-varying parameters as a group, [79] proposed to model each parameter individually. An IQC for such time-varying scalars can be derived from the polytopic IQC by considering the one dimensional polytope $[a, b]$. The IQC has the form

$$\int_{-\infty}^{\infty} \begin{bmatrix} \hat{v}(j\omega) \\ \hat{w}(j\omega) \end{bmatrix}^* \begin{bmatrix} z & y \\ y & -x \end{bmatrix} \begin{bmatrix} \hat{v}(j\omega) \\ \hat{w}(j\omega) \end{bmatrix} d\omega \geq 0 \tag{4.3}$$

if $x \geq 0$,

$$\begin{bmatrix}1\\a\end{bmatrix}^T \begin{bmatrix}z & y\\y & -x\end{bmatrix} \begin{bmatrix}1\\a\end{bmatrix} \geq 0, \text{ and } \begin{bmatrix}1\\b\end{bmatrix}^T \begin{bmatrix}z & y\\y & -x\end{bmatrix} \begin{bmatrix}1\\b\end{bmatrix} \geq 0.$$

For a time varying scalar, $\delta(t)$, that varies in the range $[0, 1]$ the conditions simplify to $x \leq 0$, $z \geq 0$, and $z + 2y - x \geq 0$. Under these conditions, the IQC is easily shown to be true since

$$\begin{aligned}\begin{bmatrix}v(t)\\\delta(t)v(t)\end{bmatrix}^T \begin{bmatrix}z & y\\y & -x\end{bmatrix} \begin{bmatrix}v(t)\\\delta(t)v(t)\end{bmatrix} &= zv^2(t) + 2y\delta(t)v^2(t) - x\delta^2(t)v^2(t)\\ &= v^2(t)\left(z + 2y\delta(t) - x\delta^2(t)\right) \geq 0\end{aligned}$$

because $\delta^2(t) \leq \delta(t)$. An IQC for time varying scalars can also be derived from the multiplier approach. Recall the multiplier condition (2.6)

$$\int_{-\infty}^{\infty} Re[v^*(j\omega)M(j\omega)w(j\omega)]\mathrm{d}\omega \geq 0, \quad v \in \mathcal{L}_2, \quad w = \Delta(v), \quad \Delta \in \mathbf{\Delta}.$$

When $w(t) = \delta(t)v(t)$, $\delta(t) \in [0, \beta]$, and $M(j\omega) = y \in \mathbb{R}_+$

$$\int_{-\infty}^{\infty} Re[v^*(j\omega)M(j\omega)w(j\omega)]d\omega = \int_0^{\infty} Re[v(t)y\delta(t)v(t)]dt = \int_0^{\infty} yv^2(t)\delta(t)dt \geq 0.$$

From this simple positive multiplier, the IQC

$$\Pi(y) = \begin{bmatrix}0 & y\\y & 0\end{bmatrix}$$

can be defined. Clearly this parameterized IQC is contained in the more general set of IQCs described above. This positive multiplier approach is used in [79] to model the time varying scalar RNN weights. Finally, note that a time-varying scalar in the sector $[0, \beta]$ is essentially a nonlinear, sector bounded function, and the IQC

$$\Pi(y) = \begin{bmatrix}0 & y\\y & -2y\end{bmatrix} \tag{4.4}$$

is also applicable. In fact, this IQC is also contained in the set defined by (4.3). Preliminary results revealed that when the IQC (4.3) is used the resulting instance is always of the form (4.4). Thus, the more general IQC did not provide any reduction in conservativeness but did increase the cost of the optimization. For this reason the simpler IQC (4.4) is used throughout the remainder of this chapter and those that follow.

4.1.2 Time-Varying RNNs as Feedback Systems

To apply the IQCs of the previous section to time-varying RNNs, the weight variations must be written in terms of multiplication by positive, time-varying scalars. To facilitate such a representation, [79] suggested writing the variation as

$$\Delta W_{ij} = \overline{\delta}_{ij}(t)\overline{\Delta}_{ij} - \underline{\delta}_{ij}(t)\underline{\Delta}_{ij}, \quad \overline{\delta}_{ij}(t),\ \underline{\delta}_{ij}(t) \in [0, 1].$$

Additionally, the system (4.1) can be rewritten as

$$\dot{x}(t) = -Cx(t) + \widetilde{W}\Delta(x(t), t)KRx(t) + u(t) \tag{4.5}$$

where

$$\begin{aligned}
\widetilde{W} &= \begin{bmatrix} W & \widehat{W} & -\widehat{W} \end{bmatrix} \in \mathbb{R}^{n\times(n+2n^2)},\\
\widehat{W} &= \begin{bmatrix} e_1 & \overset{\times n}{\cdots} & e_1 & e_2 & \dots & e_n \end{bmatrix} \in \mathbb{R}^{(n\times n^2)},\\
K &= \mathrm{diag}\{I, \overline{\Delta}, \underline{\Delta}\},\\
R &= \begin{bmatrix} I & I & \overset{\times n}{\cdots} & I \end{bmatrix}^T \in \mathbb{R}^{(n+2n^2)\times n},\\
\Delta(x(t),t) &= \mathrm{diag}\{\delta_i(x_i(t)), \overline{\delta}_{ij}(t)\delta_i(x_i(t)), \underline{\delta}_{ij}(t)\delta_i(x_i(t))\} \in \mathbb{R}^{(n+2n^2)\times(n+2n^2)},\\
&\qquad 0 \le \delta_i(x_i(t)) \le 1,\ 0 \le \overline{\delta}_{ij}(t) \le 1,\ 0 \le \underline{\delta}_{ij}(t) \le 1.
\end{aligned}$$

Note that since $\phi(x)$ is in the sector $[0,1]$, it can be written as $\delta_i(x_i(t))x_i(t)$. Then,

$$\Delta W_{ij}(t)\phi(x_i(t)) = \overline{\delta}_{ij}(t)\delta_i(x_i(t))x_i(t) - \underline{\delta}_{ij}(t)\delta_i(x_i(t))x_i(t) = \overline{\delta}_{ij}(t)x_i(t) - \underline{\delta}_{ij}(t)x_i(t),$$

since multiplication by $\delta_i(x_i(t))$ does not change the sectors of the time varying parameters.

The formulation can be simplified when only certain elements of W are time-varying. For example, consider an RNN with $W \in \mathbb{R}^{n\times n}$ and the W_{11} and W_{12} elements time-varying in the range $[-3,2]$. The model parameters are

$$\begin{aligned}
\widetilde{W} &= \begin{bmatrix} W & \widehat{W} & -\widehat{W} \end{bmatrix} \in \mathbb{R}^{(2\times 6)},\\
\widehat{W} &= \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{(2\times 2)},\\
K &= \mathrm{diag}\{1,1,2,2,3,3\},\\
R &= \begin{bmatrix} I & I & I \end{bmatrix}^T \in \mathbb{R}^{(6\times 2)}.
\end{aligned}$$

The modified RNN system (4.5) can be written in the form of a feedback system with

$$\begin{aligned}
G &= \left[\begin{array}{c|cc} -C & I & \widetilde{W} \\ \hline I & 0 & 0 \\ R & 0 & 0 \end{array}\right],\\
\Delta &= K \begin{bmatrix} \Phi & & \\ & \overline{\delta} & \\ & & \underline{\delta} \end{bmatrix}.
\end{aligned} \tag{4.6}$$

Keeping with the assumption that ϕ is bounded in the sector $[0,1]$, a loop transformation, $H_2 = -K^{-1}$, is introduced to normalize the gain of the nonlinear operator Δ. The modified feedforward operator is $G(s) = R(C+sI)^{-1}\widetilde{W} - K^{-1}$. The multiplier and IQC theorems can be applied to construct LMI conditions whose feasibility is sufficient for stability of the time varying network. Using the IQC from the previous section to model the time varying scalars and one of the IQCs from the previous chapter for the nonlinearity ϕ, an IQC for Δ is given by

$$\Pi = \left[\begin{array}{ccc|ccc} & & & T & & \\ & & & & \overline{Y} & \\ & & & & & \underline{Y} \\ \hline T^T & & & & & \\ & \overline{Y} & & & & \\ & & \underline{Y} & & & \end{array}\right] \tag{4.7}$$

where T is either positive diagonal, diagonally dominant, or doubly dominant. The variables $\overline{Y}$ and $\underline{Y}$ are positive, diagonal matrices. Let the matrix $\hat{T}$ be defined as $\hat{T} = \text{diag}\{T, \overline{Y}, \underline{Y}\}$. Applying the loop transformation, Problem 2.2 results in the main LMI condition,

$$\begin{bmatrix} -CP - PC & P\widetilde{W} + R^T\hat{T} \\ \widetilde{W}^T P + \hat{T}^T R & -(\hat{T} + \hat{T}^T)K^{-1} \end{bmatrix} < 0. \tag{4.8}$$

The details of the derivation can be found in Appendix A. Appendix A also shows how to include gain estimation terms and the Popov IQC in the problem. Equation (4.8) is an LMI in T, Y, and P and can be solved by standard SDP software. When T is taken to be positive diagonal, this condition reduces to an LMI presented in [79]. When K is not fixed, but is instead intended to be part of the optimization problem, condition (4.8) is no longer an LMI constraint. The difficulties this presents will be discussed in Section 4.2

The formulation of time-varying RNNs in (4.5) does not fully exploit the power of the multipliers in $\mathcal{S}_{dd}$ and $\mathcal{M}_{dd}$. The terms $\Delta W_{ij}\phi(x_i)$ are modeled as purely unstructured time-varying scalars. This ignores the fact that the nonlinearity $\phi(x_i) = \delta_i(x_i)x_i$ and the time varying components $\delta_{ij}(t)$ are fundamentally different. The IQCs for the time varying components do not account for the repeated structure of $\Phi(x)$. This will introduce conservativeness into the analysis so a new approach is proposed here.

An alternative way to write (4.1) is

$$\begin{aligned} \dot{x}(t) &= -Cx(t) + \widetilde{W}\begin{bmatrix} I & 0 \\ 0 & \widetilde{\Delta}(t) \end{bmatrix} KR\widehat{\Delta}(x(t))x(t) + u(t) \\ \widehat{\Delta}(x(t)) &= \text{diag}\{\delta_i(x_i(t))\} \\ \widetilde{\Delta}(t) &= \text{diag}\{\overline{\delta}(t), \underline{\delta}(t)\} \end{aligned} \tag{4.9}$$

or equivalently as a feedback system with forward operator

$$G = \left[\begin{array}{c|cc} -C & I & \widetilde{W} \\ \hline I & 0 & 0 \\ \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} & 0 & \begin{bmatrix} 0 & 0 \\ \widetilde{R} & 0 \end{bmatrix} \end{array}\right],$$
$$\widetilde{R} = \begin{bmatrix} I & \overset{\times 2n}{\cdots} & I \end{bmatrix}^T$$

and the nonlinear, time-varying, feedback operator (4.6). Since the feedback operator has the same structure as the formulation (4.5), IQC (4.7) can be applied to this formulation as well. An LMI stability condition can be derived from Problem 2.2 and is given in Appendix A. In the next section the two LMIs developed thus far are compared on some example problems.

4.1.3 Examples

In the first example, all of the weights of the 2×2 RNN with weight matrix W_1 are allowed to vary in the range $[-\delta, \delta]$ for three values of δ: 0.25, 0.30, and 0.32. An upper bound on the $\mathcal{L}_2$-gain is computed using the six combinations of IQCs used to model the nonlinearity in the previous chapter. The computations were repeated for both formulations of the time varying RNN: (4.5) and (4.9). The computed bounds on the gain are shown in Table 4.1. There are several trends visible in the data. The first and most obvious is that the computed upper bounds on the $\mathcal{L}_2$-gain increase as the amount of variation allowed in the weights increases. Much like the gain

	γ_d	γ'_d	γ_{dd}	γ'_{dd}	γ_{ks}	γ'_{ks}
$W_1, \delta = 0.25$	9.5010	8.2631	8.6793	8.0019	7.9741	7.5941
$W_1, \delta = 0.30$	61.2402	29.4420	35.3116	25.3251	24.9618	20.7134
$W_1, \delta = 0.32$	-	-	-	190.3413	170.3540	67.4899
$W_1, \delta = 0.25$	7.6177	7.4700	7.4700	7.4700	7.4700	7.4700
$W_1, \delta = 0.30$	20.0857	18.7144	18.7144	18.7144	18.7144	18.7144
$W_1, \delta = 0.32$	57.7000	46.4455	46.4455	46.4455	46.4455	46.4455

Table 4.1: Results for estimating the gain of an RNN with weight matrix W_1 using different IQCs. The weights are allowed to vary in the range $[-\delta, \delta]$ with $\delta = .25, .30, .32$. The results in the top half of the table were computed using formulation (4.5). The results in the bottom half were computed using the formulation in (4.9).

bounds computed for the time-invariant RNNs, the bounds computed here increase rapidly near the boundary between an amount of variation that can be tolerated and an amount that can not. The second trend to notice is the decrease in the gain bounds as more complex IQCs are applied to model the nonlinearity. This is not so much of a trend in the case of formulation (4.9) as a phase shift, but for larger networks this transition is smooth and the trend is obvious. Finally, note that in all cases the formulation in (4.9) results in smaller $\mathcal{L}_2$-gain bounds. More accurate modeling due to the complete separation of the nonlinearity and time-varying parameters results in a less conservative analysis.

The second example repeats the first experiment on a larger network with weight matrix W_3 and takes $\delta \in \{0.25, 0.35, 0.45\}$. The trends observed in the first experiment are more clear in this example. The use of formulation (4.9) results in a larger reduction in conservativeness than for the smaller network. Also, the benefit of using more complex IQC models of the nonlinearity is more apparent. The results suggest that the least conservative analysis is achieved by application of the Popov IQC in combination with a doubly dominant T and formulation (4.9).

4.1.4 Computational Considerations

The LMI problems presented so far in this chapter are more expensive to solve than those in Chapter 3. When all of the weights of an RNN are allowed to vary, the cost can be much greater because an additional $2n^2$ variables are added to the problem. Before proceeding to examine the cost of working with these LMIs, a few arguments from the previous chapter should be recalled. No explicit constraint on the positive definiteness of P is needed. Since the IQCs used here are essentially the same as those applied in the previous chapter, the arguments applied there are still valid. In the same manner, when T and Y are taken to be positive diagonal, no explicit constraints on the positiveness of T and Y are needed. Since Y can be of order n^2, this can drastically reduce the number of constraints in the optimization problem. Problems with ill-conditioning due to large values of γ exist in these problems as well. The problem is somewhat more pervasive since the amount of variation allowed in the system is often taken to be large resulting in large γ.

For problems with a number of variables n that is large relative to the size of the LMI constraints, the run time of SDP solvers is dominated by the n^3 cost of solving a set of linear equations. For a fixed size RNN, allowing m weights to vary should increase the cost on the order of m^3 since each new weight that is allowed to vary adds two decision variables to the problem. Analysis of the run time data in Figure 4.1 shows that the cost grows at a slower rate for both the Sedumi and PENBMI solvers. The data was generated by testing feasibility of the LMI derived from either (4.5) or (4.9)

	γ_d	γ'_d	γ_{dd}	γ'_{dd}	γ_{ks}	γ'_{ks}
$W_3, \delta = 0.25$	-	-	7.5793	4.8872	4.8980	4.4094
$W_3, \delta = 0.35$	-	-	22.9458	10.4151	10.6142	9.1089
$W_3, \delta = 0.45$	-	-	-	-	-	458.3935
$W_3, \delta = 0.25$	-	683.6985	4.7750	3.1480	3.0335	2.8275
$W_3, \delta = 0.35$	-	-	6.8962	4.3408	4.3382	4.1482
$W_3, \delta = 0.45$	-	-	14.1467	8.0225	8.2289	7.9151

Table 4.2: Results for estimating the gain of an RNN with weight matrix W_3 using different IQCs. The weights $\{W_{1,2}, W_{2,1}, W_{1,3}, W_{1,4}, W_{1,5}\}$ are allowed to vary in the range $[-\delta, \delta]$ with $\delta = .25, .35, .45$. The results in the top half of the table were computed using formulation (4.5). The results in the bottom half were computed using the formulation in (4.9).

for an increasing number of time varying weights. The simple IQC with T positive diagonal is used, but results were similar for the other IQCs. The RNN tested had a 10×10 weight matrix, and all instances of the resulting LMI were feasible. Results in the previous chapter showed that the PENBMI solver is considerably slower on infeasible problems than on feasible problems. Ensuring that all problem instances were feasible gave a more meaningful comparison. Growth rates for the cost of increasing the number of varying weights were estimated to be between m^2 and m^3 for Sedumi and $m^{\frac{3}{2}}$ for PENBMI. Two factors may be contributing to the less than expected growth rates. First, the n^3 cost model assumes that the size of the constraints stays constant as the number of variables grow. Here, however, the size of the constraint is growing at a rate of $2m$. Second, the lower right hand corner of the LMIs is extremely sparse; it is a diagonal block. This may improve the cost of constructing the Newton equations, in the case of PENBMI, and the linear update equations in Sedumi.

The experiment also shows that the cost of using the new time-varying RNN formulation (4.9) and the cost of using the formulation (4.5) are nearly the same. This seems reasonable since comparison of the different LMIs reveals relatively minor structural differences. Since it is no more expensive and produces less conservative results, use of the formulation (4.9) appears to be the best choice.

4.2 Maximizing the Allowable Variation

The ability to prove stability and estimate the $\mathcal{L}_2$-gain of an RNN with time-varying weights is useful in certain situations where some fixed amount of variation needs to be accommodated. In other problems, it is more useful to compute the largest $\overline{\Delta}_{ij}$ and $\underline{\Delta}_{ij}$ values under which stability can be proved. A simple approach to the problem is to apply a bisection algorithm and repeatedly evaluate the LMI conditions of the previous chapter. This was done, for example, in [49], but is very expensive because of the large number of LMI problems which must be solved. In [79] it was proposed to maximize the sum of the $\overline{\Delta}_{ij}$'s and $\underline{\Delta}_{ij}$'s directly as part of an optimization problem subject to the stability constraints. Such a problem is computationally difficult, and some of the computational problems it poses are explored in this section. For expository purposes, the constraint (4.8), will be used throughout the development presented here. The results are equally applicable to the conditions derived from formulation (4.9) and to versions of the constraints with gain estimation terms.

When the bounds on the variations of the individual weights, $\underline{\Delta}$ and $\overline{\Delta}$, are fixed, testing the

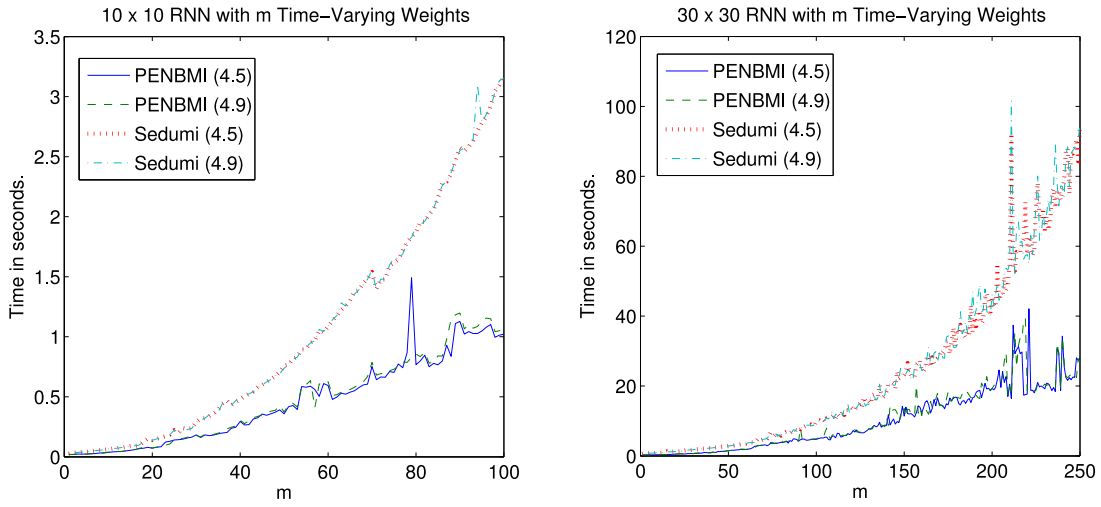

Figure 4.1: Run time results for an increasing number of time-varying weights in a 10×10 RNN and a 30×30 RNN.

feasibility of the stability constraints is computationally tractable. To compute maximum variation bounds as part of an optimization problem, $\underline{\Delta}$ and $\overline{\Delta}$ must be made decision variables. Beyond simply increasing the number of decision variables, this presents a serious problem for the following reason. The modified constraint is no longer linear in the decision variables since it contains terms of the form $\overline{Y}_{ij}\overline{\Delta}_{ij}^{-1}$. Such a constraint is known as a bilinear matrix inequality (BMI) [74]. BMI problems are not generally convex, and there is no known polynomial time algorithm for verifying the existence of feasible solutions for such constraints [87]. Despite the general difficulty of such problems, several relatively efficient algorithms exist for finding locally optimal solutions to optimization problems involving BMIs.

To formulate a concrete optimization problem for maximizing the allowable variation, this quantity needs a formal definition. Several objective functions can be said to describe maximum allowable variation. In [79] the problem is defined as maximizing the sum of the bounds,

$$\max \sum_{ij} \left(\overline{\Delta}_{ij} + \underline{\Delta}_{ij}\right). \tag{4.10}$$

The objective function has the advantage of being linear and is also a reasonable definition for the goal of the optimization problem. An alternative is to maximize the volume of the space within which the weights can vary. Such an objective function is given by the product of the length of the individual ranges

$$\begin{aligned} \max \prod_{ij} \left(\overline{\Delta}_{ij} + \underline{\Delta}_{ij}\right) &= \max \log\left(\prod_{ij} \left(\overline{\Delta}_{ij} + \underline{\Delta}_{ij}\right)\right) \\ &= \max \sum_{ij} \log\left(\overline{\Delta}_{ij} + \underline{\Delta}_{ij}\right). \end{aligned} \tag{4.11}$$

While not linear, the function is convex, and a log transformation of the objective results in a form that is easy to handle using epi-graph formulations [10]. Initial results for problems using this objective function, however, proved to be difficult to solve and resulted in poor solutions. That is, the volume of the resulting bounding box was smaller than that computed using the simpler additive

objective function (4.10). For this reason, the additive objective function is used throughout the remainder of the section.

4.2.1 Methods

Different approaches exist for optimizing (4.10) subject to the BMI constraint (4.8). A convex approximation developed in [79] is described first. To help assess the quality of the convex approximation, three approaches to directly solving BMI problems are then discussed. In Section 4.2.3 the methods presented here are applied to some sample problems to illustrate the differences in behavior between the algorithms.

4.2.1.1 A Convex Approximation

To remove the bilinear terms from the LMI constraint a set of auxiliary variables, $\overline{H}_{ij} = \overline{Y}_{ij}\overline{\Delta}_{ij}^{-1}$ and $\underline{H}_{ij} = \underline{Y}_{ij}\underline{\Delta}_{ij}^{-1}$ can be substituted into the matrix constraint. Since the Δ_{ij} variables only exist in the objective function of the modified problem, the objective function is unbounded and essentially meaningless. An approximate objective function involving the auxiliary variables must be used. In [79] the objective function is defined as

$$\max \sum \left(\overline{Y}_i - \overline{H}_{ij} + \underline{Y}_i - \underline{H}_{ij}\right) = \max \sum \left(\overline{Y}_i \left(1 - \frac{1}{\overline{\Delta}_{ij}}\right) + \underline{Y}_i \left(1 - \frac{1}{\underline{\Delta}_{ij}}\right)\right). \tag{4.12}$$

Maximizing such an objective function requires making the Y_{ij}'s large but also making the H_{ij}'s small. This implies that optimization of the objective will result in large values for the Δ_{ij}'s. These modifications result in a standard LMI problem that can be solved with the SDP software previously described.

4.2.1.2 Alternating Minimization Method

The alternating minimization method for finding approximate solutions to BMIs solves a sequence of LMIs generated by fixing alternating groups of decision variables. For example, in the RNN stability problem the Y_{ij} variables could be held constant while the objective function is optimized over the Δ_{ij} variables. Fixing the Y_{ij} variables makes the constraint (4.8) convex and the optimization problem solvable by standard software. The second step is to then fix the Δ_{ij} variables while the objective function is optimized over the Y_{ij} variables. Unfortunately, the Y_{ij} variables do not have a direct effect on the objective function, but an auxiliary objective function, such as minimizing the trace of Y can be used. The sequence of steps is repeated until some halting criteria is met. Unless the BMI is convex, the algorithm will not generally converge to the optimal solution and may fail to converge at all. The algorithm was implemented using Sedumi to solve the convex subproblems. When applied to the examples later in the section, the method always produced solutions that were nearly the same as those produced by the augmented Lagrangian method but at a much greater cost. For this reason and for conciseness, the results for this algorithm are not given in the examples that follow.

4.2.1.3 Sequential Semidefinite Programming

The sequential semidefinite programming (SSDP) method is an extension of the sequential quadratic programming algorithm used to solve nonlinear programming problems [25, 26]. Like the alternating minimization method, SSDP proceeds by solving a sequence of LMI subproblems, but SSDP

generates them differently. The SSDP method, as given in [26], is designed to solve a very general class of problems of the form

$$\begin{aligned} \min \quad & b^T x \quad \text{s.t. } x \in \mathbb{R}^n, \\ & \mathcal{B}(x) \preceq 0, \\ & c(x) \leq 0, \\ & d(x) = 0, \end{aligned} \tag{4.13}$$

where $b \in \mathbb{R}^n$, $\mathcal{B} : \mathbb{R}^n \to \mathcal{S}^m$, $c : \mathbb{R}^n \to \mathbb{R}^p$, and $d : \mathbb{R}^n \to \mathbb{R}^q$. The function $\mathcal{B}(x)$ is a nonlinear, matrix valued function such as a BMI constraint. The function $c(x)$ and $d(x)$ describe element-wise nonlinear constraints on the decision variables. The Lagrangian of the problem is given by

$$\mathcal{L}(x, Y, u, v) = b^T x + \mathcal{B}(x) \bullet Y + u^T c(x) + v^T d(x)$$

where Y, u and v are Lagrangian multiplier variables [26].

The SSDP algorithm solves a sequence of subproblems of the form

$$\begin{aligned} \min \quad & b^T \Delta x + \frac{1}{2}(\Delta x)^T H^K \Delta x \quad \text{s.t.} \qquad \Delta x \in \mathbb{R}^n \\ & \mathcal{B}(x^k) + D_x \mathcal{B}(x^k)[\Delta x] \preceq 0, \\ & c(x^k) + D_x c(x^k) \Delta x \leq 0, \\ & d(x^k) + D_x d(x^k) \Delta x = 0, \end{aligned}$$

where H^k is the Hessian of the Lagrangian at step k. If $H^k \in \mathcal{S}_+$ then the problem can be converted to an SDP and solved efficiently [90]. Unless the original problem is convex, however, this will not be the case, and a positive semidefinite approximation to H^k must be computed. For example, the projection of H^k onto the cone of positive semidefinite matrices can be used [39]. Updates for the Lagrange multipliers are easily computed from the dual variables of the SDP subproblem [26].

The generality of the optimization problem (4.13) allows some alternate formulations of the BMI constraint to be constructed. Obviously, the BMI formulation can be used directly. On the other hand, since the number of nonlinearities is small, it may be more computationally efficient to introduce auxiliary variables, $\overline{H}_{ij} = \overline{Y}_{ij}/\overline{\Delta}_{ij}$ and $\underline{H}_{ij} = \underline{Y}_{ij}/\underline{\Delta}_{ij}$. By introducing these variables into the BMI it becomes a standard LMI constraint. Rather than using an alternative approximate objective function as suggested in [79], the auxiliary variables can be constrained using element-wise nonlinear constraints of the form $\overline{H}_{ij}\overline{\Delta}_{ij} = \overline{Y}_{ij}$. These constraints are not convex, so the lack of convexity has simply been shifted out of the matrix constraint. On the other hand, the Hessian of the Lagrangian is simplified considerably in this formulation since the Hessian of the term $\mathcal{B}(x) \bullet Y$ is now zero. Unfortunately, analysis of the SSDP method in [90] shows that while the algorithm exhibits good global convergence properties it converges linearly near the solution. Any gain in the efficiency of solving the LMI subproblems is trivial compared to the cost of this linear convergence. The SSDP algorithm was implemented using Sedumi to solve the LMI subproblems. The method was applied to the examples in the next section, but the linear convergence was clearly evident. This resulted in the necessity of solving a large number of LMI subproblems which made the algorithm tremendously slow. The SSDP method does not appear to be appropriate for solving the BMI problem under consideration.

4.2.1.4 Augmented Lagrangian Approach

The augmented Lagrangian approach to solving linear SDPs that was described in Chapter 2 can also be applied to nonlinear SDPs. Certain modifications must be made to the algorithm to solve

non-convex problems. The algorithm is modified to add a small multiple of the identity to the Hessian of the augmented Lagrangian to ensure its positive definiteness. Its similarity with the other BMI methods is that it solves a sequence of simpler subproblems. In this case, however, the subproblems are unconstrained, nonlinear, optimization problems solved by preconditioned conjugate gradient methods. Also like the other BMI methods, the augmented Lagrangian algorithm is not guaranteed to converge to the globally optimal solution. The PENBMI software applied earlier to linear SDPs includes the necessary modifications for application to nonconvex problems. As the examples that follow will show, the PENBMI software produces very good solutions compared to the LMI approximation problem, but it has longer run times for large problems.

4.2.2 Problem Simplifications and Modifications

The BMI optimization problem for determining the maximum amount of allowable variation is not appropriate for all applications. One modification, designed mostly to decrease the cost of solving the problem, is to restrict all of the $\overline{\Delta}_{ij}$s and $\underline{\Delta}_{ij}$s to be the same. The restriction has two benefits: a reduction in the number of decision variables and a reduction in the problem difficulty. The simplification can be implemented simply by replacing all of the Δ_{ij} decision variables with a single decision variable δ. For example, the BMI constraint (4.8) becomes the constraint

$$\begin{bmatrix} -CP - PC & PW + R^T \begin{bmatrix} T & 0 \\ 0 & Y \end{bmatrix} \\ * & -\begin{bmatrix} T^T + T & 0 \\ 0 & -\frac{1}{\delta} 2Y \end{bmatrix} \end{bmatrix} \prec 0.$$

While the constrain is clearly not an LMI, it is a special type of constraint called a linear-fractional constraint. Optimization problems with linear-fractional constraints can be solved efficiently as generalized eigenvalue problems [9]. PENBMI does not explicitly exploit this problem structure, but it solves this simpler problem more quickly than the general case and is used in the examples below. Certain *a priori* knowledge about the expected weight variation can be incorporated into this problem formulation. For example, the IQC variables $\overline{Y}$ and $\underline{Y}$ can be multiplied by a constant, positive, diagonal matrix expressing the relative amount of expected variation in the weights. If it is known that, say, weight W_{11} varies more than weight W_{12}, then $\underline{Y}_{11}$ and $\overline{Y}_{11}$ can be multiplied by two everywhere these variables occur in the constraints. Such a modification keeps the simpler problem structure, but allows different bounds on the variation of the different weights.

Another modification that can be made to the problem is the imposition of minimum values for the $\overline{\Delta}_{ij}$s and $\underline{\Delta}_{ij}$s. These types of constraints are easily handled by standard SDP software and do not seriously impact the cost of solving the optimization problem. For the convex LMI approximation, however, it is unclear how such constraints can be enforced by themselves without constraints on other variables since the Δ_{ij} variables have been removed from the problem. Such a constraint can be enforced indirectly by bounding both the Y variables and the H variables. In addition to imposing minimum values on the Δ_{ij} variables, it can be useful to place an upper bound on the $\mathcal{L}_2$-gain. Using the BMI condition in Appendix A, such a constraint can easily be enforced by placing an upper bound on γ. This constraint can be used to ensure stability of a feedback loop between an RNN and some other system. The small gain theorem states that the feedback loop will be stable if the gain of the RNN is less than $1/\gamma_p$ where γ_p is the gain of the other system. In this way a certain amount of variation in the RNN weights can be allowed while still ensuring stability of the entire feedback loop.

Formulation	γ_d	γ'_d	γ_{dd}	γ'_{dd}	γ_{ks}	γ'_{ks}
(4.5)	0.0112	0.0751	0.4001	0.4426	0.4400	0.4521
(4.9)	0.0248	0.2525	0.5321	0.5595	0.5595	0.5604

Table 4.3: The results of maximizing δ for weight matrix W_3 using the different IQCs and both formulations of the time-varying RNN equations. The PENBMI solver is used to generate the results.

4.2.3 Examples

The first example in this section uses the simplified optimization problem in which all of the $\overline{\Delta}_{ij}$s and $\underline{\Delta}_{ij}$s are constrained to be equal to some value $\delta > 0$. Since the Δ_{ij} variables do not show up directly in the convex LMI approximation, it is unclear how to adapt the approximation to this particular problem. So, for this example only the PENBMI solver is used. The RNN had weight matrix W_3, and only the weights $\{W_{1,2}, W_{2,1}, W_{1,3}, W_{1,4}, W_{1,5}\}$ were allowed to vary. The six different combinations of IQCs applied in previous experiments were used here, and both formulations of the time-varying RNN equations were compared. The results are shown in Table 4.3. The results show that, as expected, the use of better IQCs results in reduced conservativeness and thus an increase in the value of δ. Clearly, separately modeling the nonlinearity and the time-varying components as in (4.9) also reduces conservativeness.

In the second example, the first experiment is repeated, but the $\overline{\Delta}_{ij}$s and $\underline{\Delta}_{ij}$s are not constrained to be equal. For this example the convex LMI approximation of the stability constraint can be applied, and it is compared with the PENBMI solution of the BMI version of the constraint. The trends obvious in the first example are clear in this example as well. Better IQCs result in less conservative analysis, and the problem formulation in (4.9) also reduces conservativeness. Additionally, in this problem it is clear that directly solving the BMI problem gives better solutions than solving the LMI approximation. In two cases the BMI solution is twice as large as the solution to the approximate problem. These results suggest that there is some slackness in the LMI approximation, which is to be expected. The relative computational costs of these two approaches are compared in the next section.

Solutions to the convex approximation problem and the BMI problem have a qualitative as well as quantitative difference. The approximate problem tends to produce solutions where at least some of the Δ_{ij}s are zero. Solutions to the BMI problem on the other hand tend to have all of the Δ_{ij}s with positive values. For example, solving the two optimization problems with a combination of diagonal T the Popov IQC for $W = W_3$ results in the solutions

$$\Delta_{\text{LMI}} = \begin{bmatrix} 0.023 & 0.581 & 0.000 & 0.123 & 0.297 & 0.284 & 0.190 & 0.475 & 0.526 & 0.000 \end{bmatrix},$$
$$\Delta_{\text{BMI}} = \begin{bmatrix} 0.234 & 0.492 & 0.281 & 0.585 & 0.367 & 0.362 & 0.310 & 0.427 & 0.759 & 0.143 \end{bmatrix},$$

with objective function values of 2.4967 and 3.9496, respectively. Bounding the Y_{ij} and H_{ij} variables in the approximate problem improves the conditioning and the solution to some extent. For example, using the bounds, $10^{-3} < Y_{ij} < 10^3$ and $10^2 < H_{ij} < 10^3$ results in the solution

$$\Delta_{\text{LMI}} = \begin{bmatrix} 0.107 & 0.885 & 0.012 & 0.091 & 0.552 & 0.777 & 0.448 & 0.690 & 0.250 & 0.000 \end{bmatrix}$$

with an objective function value of 3.8104. The last value in the solution is 10^{-5}. Note that bounding Y_{ij} and H_{ij} implies the bounds $10^{-6} < \Delta_{ij} < 10$. Imposing the same constraints on the BMI problem does not changes the solution. To avoid the problem with the LMI solutions it makes sense to bound the minimum values of the Δ variables from below by some minimum acceptable

Formulation	γ_d	γ'_d	γ_{dd}	γ'_{dd}	γ_{ks}	γ'_{ks}
(4.5)	0.0257	1.0846	2.9710	3.8680	3.1947	4.2511
(4.9)	0.4832	2.4976	3.7931	6.1525	4.9405	7.8575
(4.5)	0.1777	2.3244	4.7165	5.2476	4.8831	5.3251
(4.9)	0.7211	3.9142	5.7217	8.1077	6.8020	8.2610

Table 4.4: The results of maximizing (4.10) for weight matrix W_3 using the different IQCs and both formulations of the time-varying RNN equations. The γ in this column titles simply refers to the combination of IQCs used. The numbers in the table are sum of variation bounds and not $\mathcal{L}_2$-gains. The top two rows of results were generated using the convex LMI stability constraint and the approximate objective function (4.12). The numbers in parentheses were computed using a modified objective function described in the next section. The bottom two rows were generated using the BMI constraint and the PENBMI solver.

value. This is rather arbitrary, however, and it can not be known *a priori* whether or not such constraints can be satisfied.

Returning to the 2×2 RNN example from the previous chapter, Figure 4.2 shows three examples of computed bounds on weight variation. For these computations the combined Popov and doubly dominant nonlinearity IQCs were used. The two different time-varying RNN formulations were compared along with the two different computational approaches: the BMI problem and the LMI approximation. Like the previous examples, the results here show that the formulation (4.9) and the direct BMI problem generally produce better results. There is, however, an exception in this example. At the origin, the LMI approximation results in a larger sum of variation bounds, 4.29, than the BMI problem, 4.04. Neither method, however, produces the solution that maximizes the sum of the Δ's. This solution is achieved by making one pair of $\overline{\Delta}$ and $\underline{\Delta}$ vanishingly small and the other increasingly large. Such solutions can be found and do satisfy the stability constraints. On the other hand, the solutions actually returned by the software, while not optimal, are, in a practical sense, better. This suggests that in some cases actually optimizing the given optimization criterion might produce *bad* solutions. Placing bounds on the maximum and minimum Δ values or on the maximum difference in magnitude between any two Δ's would improve the objective function. These types of constraint are easily included in the BMI problem, and can be included implicitly in the LMI approximation. Many such constraints can be imagined, but the simple objective function works well enough in most cases and is used throughout the remainder of the document.

4.2.4 Computational and Numerical Considerations

Ill-conditioning can occur for a number of reasons in the optimization of (4.10) subject to the BMI stability constraint (4.8). Inclusion of gain estimation terms easily leads to ill-conditioning unless a bound is placed on the $\mathcal{L}_2$ gain, γ. For both computational and numerical reasons these terms should not be included unless such an explicit bound is required. It is possible for the difference in magnitudes among the Δ_{ij} variables to be quite large, and for this difference in scale to result in ill-conditioning. For instance, the amount of negative variation allowed for a self feedback connection, a weight W_{ii}, is unbounded. Placing a bound on the maximum magnitude of the $\underline{\Delta}_{ij}$ variables can improve conditioning in cases like this without affecting the usefulness of the computed solution. In other words, since there is often little point in allowing some weight to vary on a scale which is orders of magnitude different from the other weights, it is preferable to limit the variation to some

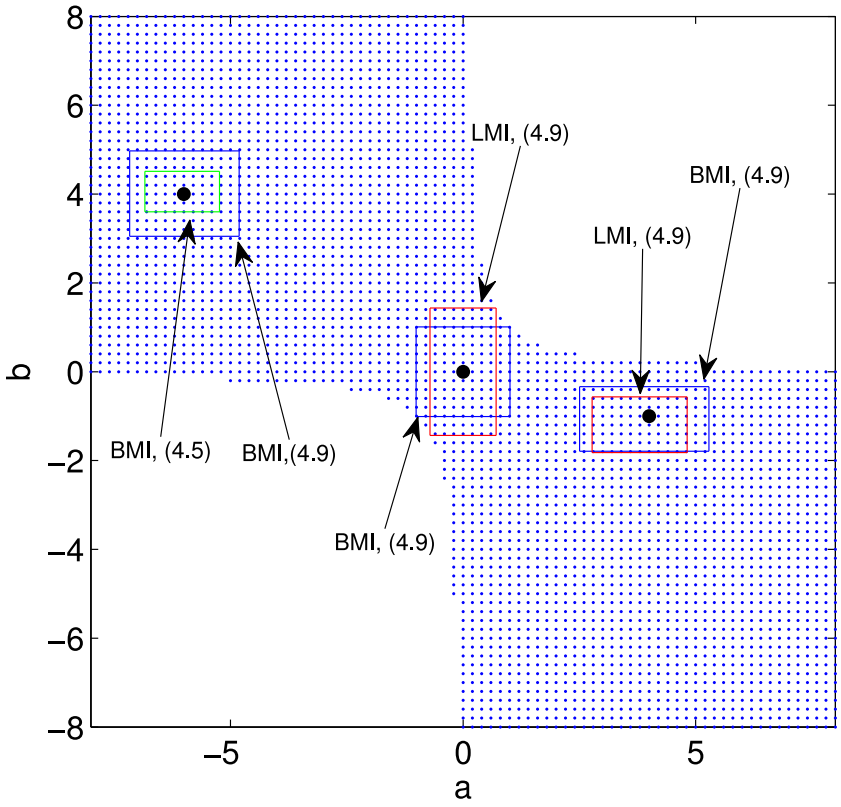

Figure 4.2: Examples of stable variation ranges for different formulations and solution techniques.

extent to improve the problem conditioning.

In Figures 4.3 and 4.4 timing and performance results are given for experiments on a 5×5 weight matrix and a 10×10 weight matrix. Three variation maximization methods are used: the BMI formulation, the LMI approximation and the simplified problem where all Δ_{ij}'s are restricted to be the same. The two time-varying RNN formulations (4.5) and (4.9) are compared. In this experiment the Popov IQC is not used and T is taken to be positive diagonal. The top left plots show the objective function values for an increasing number of time-varying weights. The top right plots show the average Δ_{ij} value for the same increasing set of time varying weights. The bottom plots show the run time of each of the methods. PENBMI is a local solver, and the solution can differ in different runs. The reported results for the BMI solutions are an average of five trials. In both experiments it is clear that formulation (4.9) generally produces the best results regardless of the solution method without much cost in terms of run time. The BMI solution is generally better than the LMI solution and always better than the restricted problem solution. The BMI solution, however, has a large computation cost that grows rapidly with the number of time varying weights. The LMI approximation solution for formulation 4.9 is also quite good and has a run time cost near to and sometimes less than the restricted problem. A final observation that can be made from these results is that while the cost of computing the variation bounds grows quickly with the number of weights that are allowed to vary, the average amount of variation allowed in the solution decreases rapidly. There is clearly a trade-off between flexibility in the adaptation of an RNN's weights and the amount of variation that can be tolerated with known stability. This suggests that limiting the number of weights that are allowed to vary may be wise in terms of a cost benefit trade-off. There are examples of algorithms, such as ESN approaches, where only the weights for a subset of neurons are adapted and good performance can still be achieved [81, 36].

4.3 Conclusions

The results in this chapter give sufficient conditions for the stability of RNNs with time-varying connection weights. Starting from a formulation of the problem in [79], a novel formulation of the time-varying RNN equations was developed. This new formulation reduced the conservativeness of the stability analysis. The problem of finding the maximal amount of weight variation under which stability can be assured was developed as a BMI problem. Through several example problems, it was shown that directly solving the BMI problem produced better results than those obtained with the convex approximation developed in [79]. On the other hand, it was shown that as the problem size increases, the convex approximation is more computationally tractable.

The analysis in this chapter used a simple IQC model of the time-varying RNN parameters and assumed that all the parameters varied independently. A more general IQC for the time-varying parameters was introduced, but did improve performance in preliminary experiments. Further exploration of this problem will provide insight into the reasons for this. The IQC presented in this chapter bounded only the values of the time-varying parameters and not their derivatives. Additional reduction in conservativeness can be achieved if the rates at which the parameters vary are bounded. Often, however, the parameter variation is dependent on some error signal and the rates of change are not naturally bounded. Even when such bounds are available the cost of exploiting this information in an IQC is a factor of three increase in the number of decision variables. For these reasons it might be better to accept some conservatism in the analysis for the sake of computational tractability.

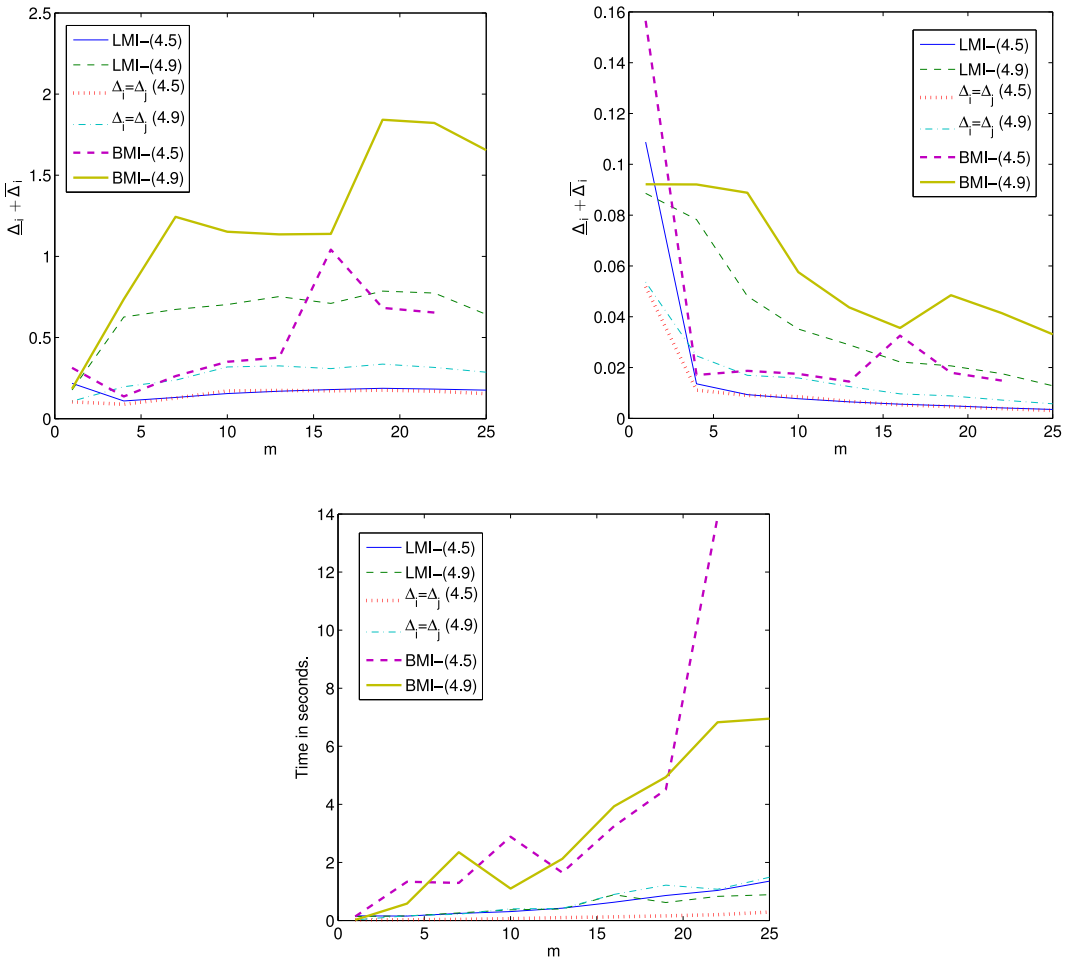

Figure 4.3: The allowable variation under stability constraints is computed using three approaches: the BMI problem, the LMI approximation, a restricted problem where all Δ_{ij}'s are equal. The RNN weight matrix is of size 5×5 and the number of time-varying weights varied from 1 to 25. The Popov IQC is not used and the matrix $T \in \mathcal{D}_+$. Both time varying RNN formulations, (4.5) and (4.9), are compared. The top left plot shows the objective values attained, the top right plot shows the mean Δ_{ij} value, and the bottom plot shows the run times.

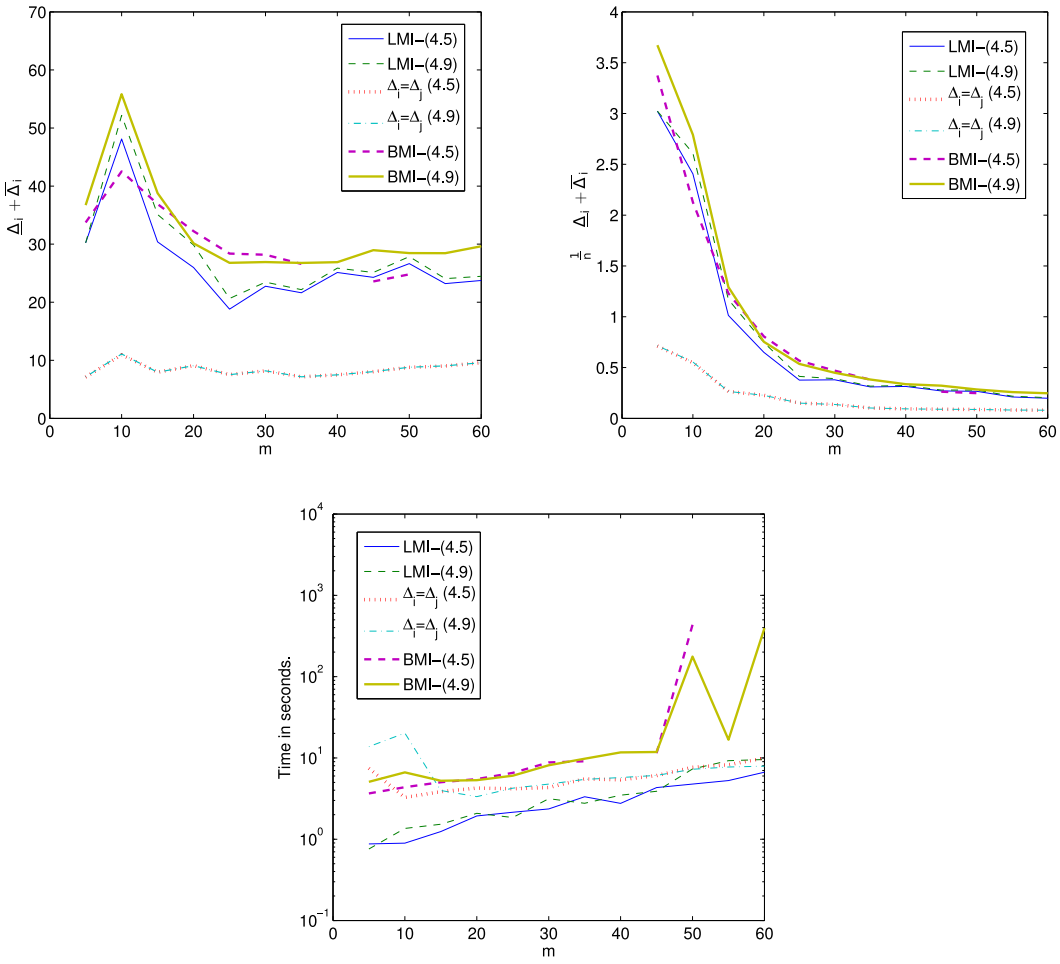

Figure 4.4: The allowable variation under stability constraints is computed using three approaches: the BMI problem, the LMI approximation, a restricted problem where all Δ_{ij}'s are equal. The RNN weight matrix is of size 10×10 and the number of time varying weights varied from 5 to 60. The Popov IQC is not used and the matrix $T \in \mathcal{D}_+$. Both time varying RNN formulations, (4.5) and (4.9), are compared. The top left plot shows the objective values attained, the top right plot shows the mean Δ_{ij} value, and the bottom plot shows the run times.

Chapter 5

Stable Learning with RNNs

Adapting RNNs in an off-line setting, whether as controllers or models, generally requires no stability analysis. In a control setting, off-line adaptation of recurrent neural networks uses interaction with model systems that may not accurately reflect reality. Online adaptation allows RNN components observe the actual properties of the controlled plant and track the changes of that system over time. Using RNNs in online adaptive systems, however, requires that their stability be guaranteed during adaptation. In this chapter an algorithm is presented which ensures the stability of an adaptive RNN. The algorithm is applicable to many systems that can be modeled and analyzed using the approach described in the previous chapters. For instance, the algorithm is easily extended to cover an RNN in a control loop with an uncertain plant model. In the next chapter, such an extension is developed more fully. Here, however, the exposition is focused on simply maintaining stability of an RNN under adaption from some arbitrary algorithm. This removes some distractions and keeps the focus on the basic properties of the algorithm.

In Chapters 3 and 4 a stability analysis was developed for RNNs with fixed and time-varying weights, respectively. Both types of analysis are necessary for enabling the algorithm proposed in this chapter. Because the stability analysis computations are expensive and generally have worse asymptotic complexity than the computation of weight updates, they dominate the run time of the proposed algorithm. Steps are taken to minimize the number of stability analysis computations thereby reducing the cost of ensuring the stability of adaptive RNNs. As shown in the previous two chapters, there is a direct trade-off between the conservativeness of the stability analysis and its computational cost. Conservativeness in the stability analysis impacts the online stable learning algorithm in two ways. Increased conservativeness reduces the amount of variation allowable in the weights of an RNN and will tend to increase the number of necessary stability computations. Also, conservativeness can limit optimization of the overall control objective by over zealous restriction of the RNN weight settings. Whether or not the increased cost of a single analysis computation is worthwhile in terms of its effect on the total number of stability computations is dependent both on the size of the network and the problem under consideration.

In the next section, a simple example problem is introduced. It is used throughout the chapter to illustrate the properties of the proposed stable learning algorithm. The later sections will develop the necessary components of the algorithm, beginning with initialization of RNNs to stable weight settings, through the basics of the algorithm and ending with some techniques to improve its performance in certain situations.

Before proceeding to the example problem some notation is introduced. Define the set $\mathcal{W}_{ss}^n$ as all $n \times n$ weight matrices that result in stable RNN dynamics. The analysis of Chapter 3 produces an inner approximation of this set whose quality is dependent on the choice of IQCs. The set $\mathcal{W}_{ss}^n$

is not generally known explicitly and to limit the notation somewhat, the symbol $\mathcal{W}_{ss}^n$ is also used to represent approximations of it. The specific set of IQCs used to determine the approximation is not indicated in the notation but is made clear when it is important.

5.1 An Example Adaptive System

To illustrate the behavior or the proposed algorithm, a simple RNN learning problem with stability constraints is formulated in this section. The problem will be to train a recurrent neural network to model a given dynamic system subject to constraints on the stability of the RNN. The problem simulates the type of situation in which the proposed stable learning algorithm will be applied and helps to illustrate many of the underlying issues which must be considered.

5.1.1 Problem Definition

The dynamic system that the RNN is trained to model is itself another RNN. The problem can be constructed to have an optimal solution with weights in the set of stable weights or outside of it. Also the solution can be made to lie near to or far from the boundary of $\mathcal{W}_{ss}^n$. The problem has the following form,

$$\min_{W \in \mathcal{W}_{ss}^2} \sum_{i=1}^{N} (\bar{x}(t_i) - x(t_i))^2 \quad \text{where}$$

$$0 < t_1 < t_2 < \cdots < t_n \in \mathbb{R},$$

$$\bar{x}(t) = \int_0^t -C\bar{x}(\tau) + \bar{W}\phi(\bar{x}(\tau)) + u(\tau)\ \mathrm{d}\tau, \quad \bar{x}(0) = 0,$$

$$x(t) = \int_0^t -Cx(\tau) + W\phi(x(\tau)) + u(\tau)\ \mathrm{d}\tau, \quad x(0) = 0, \text{ and}$$

$$u(t) = \begin{bmatrix} \sin(t) \\ 2\cos(t/2) \end{bmatrix}$$

The input to the adaptive system will be the instantaneous error, $E(t) = \hat{x}(t) - x(t)$, generated by continuously running versions of the optimal network and the adapted network. The times, t_i specified in the problem definition are used only for evaluating the performance of the adaptive network. The task is to find an RNN with a 2×2 weight matrix that reproduces the behavior of the optimal RNN on a given input sequence using only the available error signal. The optimal RNN has a weight matrix $\bar{W}$ that is chosen to illustrate different properties of the stable learning algorithm. For illustrative purposes the diagonal elements of W will be fixed to the corresponding values in $\bar{W}$. The resulting problem has a two dimensional parameter space and allows the dynamics to be easily visualized. The matrix C is taken to be the identity and $\phi(x)$ is taken to be the $\tanh(\cdot)$ function.

5.1.2 Learning Algorithm

Many algorithms exist for adapting the weights of recurrent neural networks. A survey of gradient based approaches can be found in [66]. To solve the example learning problem in this chapter, the Real Time Recurrent Learning (RTRL) algorithm is applied. RTRL is a simple stochastic gradient algorithm for minimizing an error function over the parameters of a dynamic system. It is used

here because it is an online algorithm applicable in adaptive control systems. Computing the gradient of an error function with respect to the parameters of a dynamic system is difficult because the system dynamics introduce temporal dependencies between the parameters and the error function. Computation of the error gradient requires explicitly accounting for these dependencies, and RTRL provides one way of doing this. A brief description of the algorithm follows, and a sample implementation is given in Figure 5.1.

Recall the RNN equations

$$\begin{aligned}\dot{x} &= F(x, u; C, W) = -Cx + W\Phi(x) + u \\ y &= x\end{aligned}$$

The gradient of the error function $E(||x(t) - \bar{x}(t)||_2^2)$ with respect to the weight matrix, W, is given in RTRL by the equations

$$\begin{aligned}\frac{\partial E}{\partial W} &= \int_{t_0}^{t_1} \gamma \frac{\partial E}{\partial y} dt \\ \frac{\partial \gamma}{\partial t} &= \frac{\partial F(x, u; C, W)}{\partial W} + \frac{\partial F(x, u; C, W)}{\partial x}\gamma\end{aligned}$$

where $\gamma(t_0) = 0$. The variable γ is a rank three tensor with elements γ_{ij}^l corresponding to the sensitivity of x_l to changes in the weight W_{ij}. RTRL requires simulation of the γ variables forward in time along with the dynamics of the RNN. The gradient, however, need not necessarily be integrated. The weights can instead be updated by

$$W \leftarrow W - \eta\gamma(t)\frac{\partial E(t)}{\partial y(t)}$$

for the update time t. The parameter η is a learning rate that determines how fast the weights change over time. The stochastic gradient algorithm requires that this parameter decrease to zero over time, but often it is simply fixed to a small value. The listing in Figure 5.1 shows a discrete time simulation of the RTRL algorithm using an Euler method, but the simulation can also be done using other numerical integration techniques. The algorithm has an asymptotic cost of $\Theta(n^4)$ for each update to γ when all the RNN weights are adapted. For large networks the cost is impractical, but improvements have been given. For example an exact update with complexity $\Theta(n^3)$ is given in [84] and an approximate update with an $\Theta(n^2)$ cost is given in [3]. The basic algorithm is practical for small networks and is sufficient for illustrating the properties of the proposed stable learning algorithm. For this reason it is used throughout this chapter and the next.

5.1.3 Application of RTRL to the Sample Problem

For the example problem, $\bar{W}$ is taken to be the matrix

$$\bar{W} = \begin{bmatrix} 0.0756 & 1.0663 \\ 1.2691 & -0.4792 \end{bmatrix}.$$

Using the least conservative stability analysis described in Chapter 3, the Popov IQC in combination with T doubly dominant, the $\mathcal{L}_2$-gain is bounded above by 149.1788. Using these IQCs an approximation to $\mathcal{W}_{ss}^n$ is computed over the weights W_{12} and W_{21}. In Figure 5.2 a sample of weight trajectories generated by RTRL are displayed along with the approximation to $\mathcal{W}_{ss}^n$. The optimal solution, given by $\bar{W}$, is near the boundary of $\mathcal{W}_{ss}^n$. Two weight trajectories are shown

REAL-TIME-RECURRENT-LEARNING(Δ, η)

Initialize $W(0)$, $x(0) \leftarrow 0$, $\gamma(0) \leftarrow 0$, $k \leftarrow 0$

repeat

$$k \leftarrow k+1$$

$$x(k) = x(k-1) + \Delta t\left(-Cx(k-1) + W\phi(x(k-1))\right)$$

$$\gamma_{ij}^{l}(k) = \gamma_{ij}^{l}(k-1) + \Delta t\Big(-\gamma_{ij}^{l}(k-1) + \delta_{li}\phi(x_j(k)) + \textstyle\sum_m w_{lm}(k-1)\phi'(x_m(k))\gamma_{ij}^{m}(k-1)\Big)$$

$$w_{ij}(k) = w_{ij}(k-1) - \eta\textstyle\sum_i (x_i(k) - \bar{x}_i(k))\gamma_{ij}^{l}(k)$$

Figure 5.1: The RTRL Algorithm

for each of two starting points. The different trajectories were generated using different learning rates. For higher learning rates the trajectories tend to be more erratic and can leave $\mathcal{W}_{ss}^{n}$. This is to be expected since RTRL has no knowledge of the stability constraints. Higher learning rates also allow more rapid optimization of the objective function. The purpose of the stable learning algorithm proposed in this chapter is to constrain the trajectories to the set $\mathcal{W}_{ss}^{n}$ and to ensure that the weight matrix transitions do not lead to instability. Forcing the trajectory to remain within $\mathcal{W}_{ss}^{n}$ can allow larger learning rates to be used safely and thus provide an improvement in the rate at which performance is improved.

5.2 Generating a Stable Initial Network

Any algorithm designed to ensure the stability of adaptive systems needs to ensure the stability of the system at the initial parameter settings. The algorithm presented in the next section requires that the initial weight matrix, W, of the RNN be in the set $\mathcal{W}_{ss}^{n}$. Many heuristic and algorithmic approaches for initializing RNNs based on *a priori* information about a problem or problem domain exist. For example an RNN can be trained in a simulation environment before being used on a real system. Literature on the echo state network (ESN), a special type of RNN, contains many examples of heuristics for constructing RNN weight matrices [36]. The focus of this section is the development of methods that take some arbitrary matrix, W, and modify it to ensure satisfaction of the initial stability condition. Two optimization problems are developed for finding a matrix $\hat{W}$ that is in $\mathcal{W}_{ss}^{n}$ and is, in some appropriate sense, *close* to W.

5.2.1 Scaling W into $\mathcal{W}_{ss}^{n}$

As discussed in Chapter 3, a sufficient condition for $W \in \mathcal{W}_{ss}^{n}$ is that $\|W\|_2 < 1$. If this condition is not satisfied, a simple procedure for constructing a stable approximation to W is to scale it by α where $\alpha^{-1} = (1+\epsilon)\|W\|_2$ and $\epsilon > 0$. In other words, make $\hat{W} \leftarrow \alpha W$. This approach is used throughout the ESN literature and is sufficient for constructing an RNN with the so called *echo state* property, a strong form of stability.

The norm condition on W is, however, very conservative compared to the LMI conditions developed in Chapter 3. To reduce the effect of this conservatism on the initialization procedure — and hopefully to retain more *a priori* information encoded in $\hat{W}$ — an optimization problem is

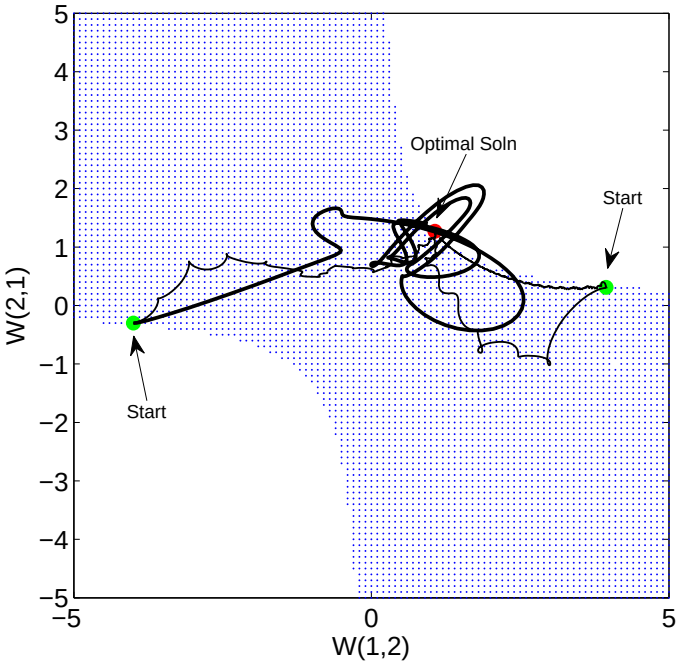

Figure 5.2: Examples of weight trajectories induced by training an RNN to reproduce the output of dynamic system using RTRL. The space of RNNs with fixed weights that can be proved stable is shown in blue. Only the off diagonal weights of the 2×2 RNN weight matrix are modified. The diagonal weights are set to the optimal values. Some trajectories leave the provably stable region.

formulated to find a maximal α such that $\hat{W} \in \mathcal{W}_{ss}^n$. The scaling, α, is the solution to the problem

$$\begin{aligned} \max_{\alpha,T,P} \alpha \quad & \text{s.t.} \\ \begin{bmatrix} -CP - PC & \alpha PW + T \\ \alpha W^T P + T^T & -(T + T^T) \end{bmatrix} \prec 0, \quad & P = P^T, \\ T \in \mathcal{D}_+,\ \mathcal{S}_{dd},\ \text{or } \mathcal{M}_{dd}, & \\ 0 < \alpha \leq 1. & \end{aligned} \tag{5.1}$$

This problem is not linear in the decision variables, but the introduction of an auxiliary variable allows it to be written as a generalized eigenvalue problem. Recall that GEVPs can be solved efficiently using special interior point methods. To formulate the problem as a GEVP, first define the matrix $\hat{P} = \alpha P$ and rewrite the constraint as

$$\begin{bmatrix} \frac{1}{\alpha}(-C\hat{P} - \hat{P}C) & \hat{P}W + T \\ W^T\hat{P} + T^T & -(T + T^T) \end{bmatrix} \prec 0.$$

Second, introduce the auxiliary variable $X \in \mathbb{S}^n$, and rewrite the optimization problem as

$$\begin{aligned} \min_{\lambda,T,\hat{P},X} \lambda \quad & \text{s.t.} \\ \begin{bmatrix} -X & \hat{P}W + T \\ W^T\hat{P} + T^T & -(T + T^T) \end{bmatrix} \prec 0, \quad & P = P^T,\ X = X^T, \\ X \prec \lambda \left(C\hat{P} + \hat{P}C \right), & \\ C\hat{P} + \hat{P}C \succ 0, & \\ T \in \mathcal{D}_+,\ \mathcal{S}_{dd},\ \text{or } \mathcal{M}_{dd}. & \end{aligned} \tag{5.2}$$

The optimal scaling is given by $\alpha = \min\{1/\lambda, 1\}$. Despite the fact that such problems can be solved in guaranteed polynomial time [9], the introduction of the auxiliary variable, X, can make the problem expensive.

5.2.2 Projecting W onto $\mathcal{W}_{ss}^n$

Considering only scalings of W restricts what it means for an approximation $\hat{W}$ to be close to W. A more general definition of closeness is to measure the magnitude of the difference in some matrix norm, $||W - \hat{W}||$. The resulting optimization problem must find a $W \in \mathcal{W}_{ss}^n$ that minimizes this value. The solution of this problem is a projection of W onto the feasible set $\mathcal{W}_{ss}^n$. The problem is formulated as follows

$$\begin{aligned} \min_{\hat{W},T,P} ||W - \hat{W}|| \quad & \text{s.t.} \\ \begin{bmatrix} -CP - PC & P\hat{W} + T \\ \hat{W}^T P + T^T & -(T + T^T) \end{bmatrix} \prec 0, \quad & P = P^T, \\ T \in \mathcal{D}_+,\ \mathcal{S}_{dd},\ \text{or } \mathcal{M}_{dd}, & \end{aligned}$$

where clearly the matrix inequality is now bilinear in the decision variables $\hat{W}$ and P. In addition to the bilinear term, the optimization problem has a convex quadratic objective function. An epi-graph representation [10], can be used to convert this into a linear objective at the cost of an additional variable and additional constraints.

To transform the objective function, first introduce the auxiliary variable $t > 0$ and change the objective of the optimization problem to

$$\min_{t,\hat{W},T,P} \; t.$$

Second, add the constraint, $||W - \hat{W}|| < t$ to the problem. It is clear that minimizing t results in minimizing the norm of $W - \hat{W}$. For both the two norm and the Frobenius norm it is possible to transform the constraint into an LMI. The procedure differs depending on the choice of norm. In the case of the 2-norm, the constraint can be written as an equivalent LMI via the following transformations [10]

$$\begin{aligned} ||W - \hat{W}||_2 \leq t^2 \quad &\Rightarrow \quad (W - \hat{W})^T (W - \hat{W}) \preceq t^2 I \\ &\Rightarrow \quad \begin{bmatrix} tI & W - \hat{W} \\ (W - \hat{W})^T & tI \end{bmatrix} \succeq 0. \end{aligned}$$

The last step is a consequence of Schur's complement theorem [98, 9]. The resulting LMI is of size $2n \times 2n$. In the case of the Frobenius norm the transformation results in an LMI of size $(n+1) \times (n+1)$. The transformation is [9]

$$\begin{aligned} ||W - \hat{W}||_F \leq t \quad &\Rightarrow \quad \operatorname{tr}\Big((W - \hat{W})^T (W - \hat{W})\Big) \leq t \\ &\Rightarrow \quad \sum_i \sum_j (W_{ij} - \hat{W}_{ij})^2 \leq t \\ &\Rightarrow \quad ||\operatorname{vec}\,(W - \hat{W})||^2 \leq t \\ &\Rightarrow \quad \begin{bmatrix} I & \operatorname{vec}\,(W - \hat{W}) \\ (\operatorname{vec}\,(W - \hat{W}))^T & t \end{bmatrix} \succeq 0. \end{aligned}$$

5.2.3 Examples

Figure 5.3 shows an example of four different initialization methods for a weight matrix $W \notin \mathcal{W}_{ss}^n$. The two scaling methods produce weight matrices on the line between W and the origin. It is clear that scaling by the norm of W is more conservative than scaling based on the LMI stability conditions. Also, the two projections are much closer to W than the two rescalings. It is not exactly clear how to interpret closeness in terms of the 2-norm in this context. The projection based on the Frobenius norm has a lower computational cost and a more intuitive meaning. Ideally, the projection would change the dynamics of the network as little as possible. It may be impossible however, to preserve the dynamics, since projecting the network weights might cause them to cross a bifurcation boundary of some kind. Using the Frobenius norm, causes the projection to minimize the changes made to the weights. Because of this, the Frobenius norm projection will tend to preserve the weight values of parts of the network that are not contributing to instability and may preserve more of the dynamics.

When compared in terms of computational cost, however, the Frobenius projection method is much less appealing. In Figure 5.4 performance and run time results are given for the different initialization approaches. Initial weight matrices were generated with normally distributed elements. The results are averages over ten different random weight matrices for each value of $n \in 2, \ldots, 11$. The results produced by the Frobenius projection method are much better than the other approaches, but the run time of this approach grows rapidly with n. The Frobenius projection method introduces n^2 variables into the problem to represent W. Along with the non-convexity of the matrix inequality condition, this causes the rapidly increasing cost.

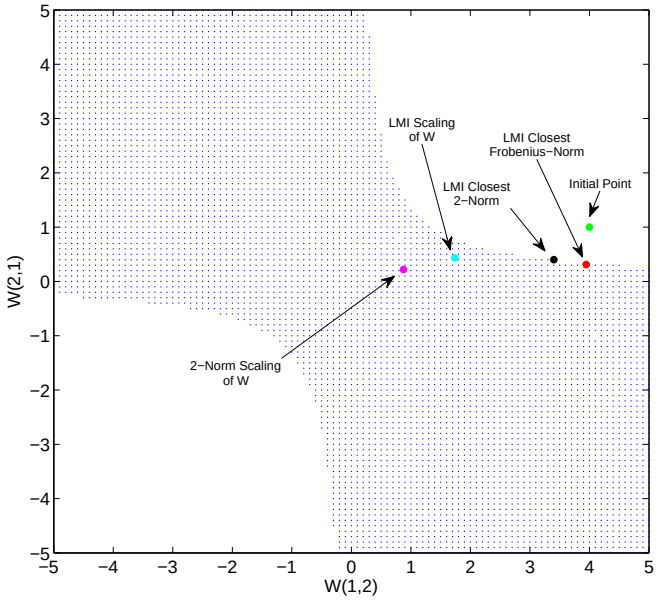

Figure 5.3: An example of the differences between several scaling methods for generating stable networks from initial weight matrices.

Two methods were used to compute a scaling based on the LMI stability conditions. The first method solved the BMI problem (5.1) directly using PENBMI. The second approach solved the GEVP version of the problem in (5.2) using the GEVP solver from LMILAB [4]. The PENBMI solver was slightly faster, but both methods produce very similar scalings. Further experimentation, however, revealed that as n grows larger the PENBMI method becomes much more efficient due to the cost of the additional variables in the GEVP version.

The initialization methods described find stable weight matrices that are close to some existing weight matrix that, itself, may or may not result in a stable RNN. The cost of these initialization methods is only worthwhile if there is useful information encoded in the initial $\hat{W}$. If $\hat{W}$ is generated randomly, for example, there is no reason to expect one method to be better than the other, and a simple scaling of W by its norm is sufficient to generate a stable initial system. If, on the other hand, an RNN is adapted to some model system, it can be useful to find the closest approximation to it that is guaranteed to be stable.

5.3 Maintaining Stability of an Adaptive RNN

Given a stable initial RNN it is now possible to consider adapting the weights of the network to optimize an objective function. As discussed in the introduction, it is necessary to guarantee the stability of the system as it changes through time. Consider a simple approach to this problem. Compute a set of bounds on the variation of the RNN weights within which stability is guaranteed, and then filter out any weight updates that put the weights outside of the computed bounds. Such an approach is at one end of a trade-off between computational cost and conservativeness. Only a single stability analysis is required, and the cost of rejecting weight updates that can not be proved stable is trivial. On the other hand, the initial weights may not be close to the optimal weights and the bounds may limit optimization of the problem objective. In [79] this approach was applied

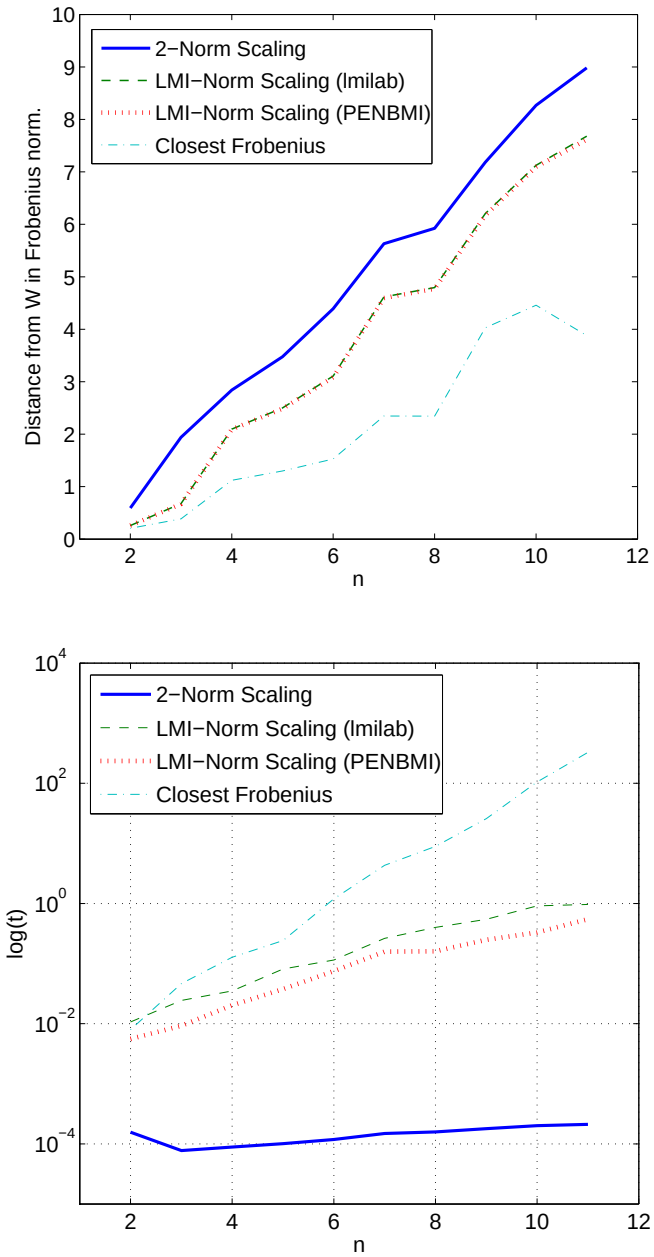

Figure 5.4: A comparison of the performance and run times of the different initialization methods. On the left is a plot of the distances of $\hat{W}$ from W. The distances tend to grow with n because W was with normally distributed elements and on average is further from the $\mathcal{W}_{ss}^n$ as n grows. On the right is the log of the run times. The Frobenius projection method has a rapidly growing computational cost.

to training an RNN to model a chaotic system. Given a *good* initial weight matrix, the learning algorithm was able to improve the model within the specified stability bounds. In general, however, it can not be expected that the optimal weight matrix for a problem, $\bar{W}$, will be reachable from W while still respecting the initial stability constraints. The relative inexpensiveness of this approach has as its price a reduction in the achievable performance. At the other end of the spectrum is an algorithm that recomputes the bounds on weight variations at every update to the weights. The algorithm does not ensure that every update is accepted, but it does, in theory, result in the acceptance of many more updates than the simple approach. It also allows, again, in theory, better performance to be achieved. The computational cost of the algorithm is, however, prohibitively expensive because of the large number of stability analysis computations required. The algorithm proposed in this section falls somewhere between these two extremes allowing better optimization of the objective than the first approach with less computational expense than the second.

An algorithm, called the Stability-Constrained-Learning algorithm, is listed in Figure 5.5. The algorithm assumes that changes to the weight matrix are proposed by an external agent, such as RTRL, at discrete time steps indexed by the variable k. The algorithm ensures the stability of an adaptive RNN by filtering weight updates that can not be guaranteed stable. A constraint set, $\mathcal{C}_j(W)$ is a set of bounds, $\{\underline{\Delta}, \overline{\Delta}\}$, on the variation in the RNN weight matrix centered on the fixed matrix W. Variations in $W(k)$ that stay within these bounds can be assured not to result in instability. The constraint set, $\mathcal{C}_j(W)$ is a hypercube centered on W with each element of $W(k)$ constrained by

$$W_{ij} - \underline{\Delta}_{ij} \leq W_{ij}(k) \leq W_{ij} + \overline{\Delta}_{ij}.$$

When an update causes $W(k)$ to lie outside of the constraint set, a new set of constraints is computed if updates to $W(k)$ have occurred since the last set of constraints was constructed. Otherwise, the update is rejected. Given this new set of constraints centered on the most recently seen stable W, the current update $W(k-1) + \Delta W$ is again checked for validity. If the update fails to satisfy the new constraints it is then rejected. Rather than rejecting the update outright, the proposed procedure makes better use of the available weight update suggestions from the adaptation algorithm. Figure 5.6 illustrates the behavior of the proposed algorithm.

In Figure 5.7 the result of applying the stability constrained learning algorithm to a weight trajectory generated by RTRL for the example problem described in Section 5.1 is shown. The weight matrix W was initialized using a scaling by its 2-norm. The constraint sets were generated using the BMI method described in the previous chapter applied to time-varying RNN formulation 4.9. The Popov IQC was used in conjunction with the repeated nonlinearity IQC, taking T to be doubly dominant. The example is ideal in that very few sets of stability constraints were computed and the optimal weight matrix was in $\mathcal{W}_{ss}^n$. In later examples it will be seen that this is not always the case.

Figure 5.2 shows an example of starting from an initial point generated by the Frobenius projection method and adapting the weights using the RTRL algorithm. Without constraints on the weight updates it is clear that the learning algorithm does not respect the stability constraints. This is not to say that the algorithm produces unstable behavior, only that stability can not be guaranteed along the trajectory induced by the unconstrained RTRL algorithm. Such excursions from the stability region could have more damaging effects if the RNN was in a control loop with some actual physical system.

Applying the proposed algorithm forces the trajectory to remain in $\mathcal{W}_{ss}^n$. The example shown in Figure 5.8 and Figure 5.9 illustrates an unfortunate problem with the approach. In some instances, weight trajectories can remain near the boundary of $\mathcal{W}_{ss}^n$. This results in the computation of a large number of constraints sets. Since this computation is expensive, its occurrence over the course of

STABILITY-CONSTRAINED-LEARNING

$j \leftarrow 0$, $k \leftarrow 0$
update $\leftarrow$ **false**
Initialize $W(0)$
Compute constraint set $\mathcal{C}_0\,(W(0))$
repeat
    $k \leftarrow k + 1$
    Compute ΔW
    if $W(k-1) + \Delta W \in \mathcal{C}_j$
        $W(k) \leftarrow W(k-1) + \Delta W$
        update $\leftarrow$ **true**
    else
        if update
            $j \leftarrow j + 1$
            Compute constraint set $\mathcal{C}_j(W(k-1))$
            update $\leftarrow$ **false**
            if $W(k-1) + \Delta W \in \mathcal{C}_j$
                $W(k) \leftarrow W(k-1) + \Delta W$
                update $\leftarrow$ **true**

Figure 5.5: The stability constrained learning algorithm.

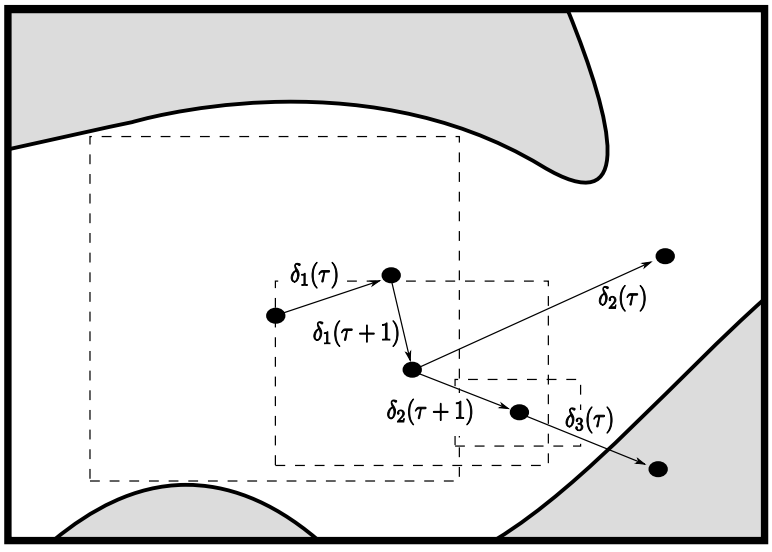

Figure 5.6: Given an initial stable point, the update $\delta_1(\tau)$ is proposed and accepted since it satisfies the stability constraints. The next update is also accepted. Update $\delta_2(\tau)$ violates the stability constraints, so the constraints are updated. The proposed update violates the new constraints as well, and so is rejected. Update $\delta_2(\tau+1)$ is accepted since it satisfies the new constraints. Update $\delta_3(\tau)$ violates the constraints, and causes the constraints to be updated again. Since it still violates the new constraints the update is rejected.

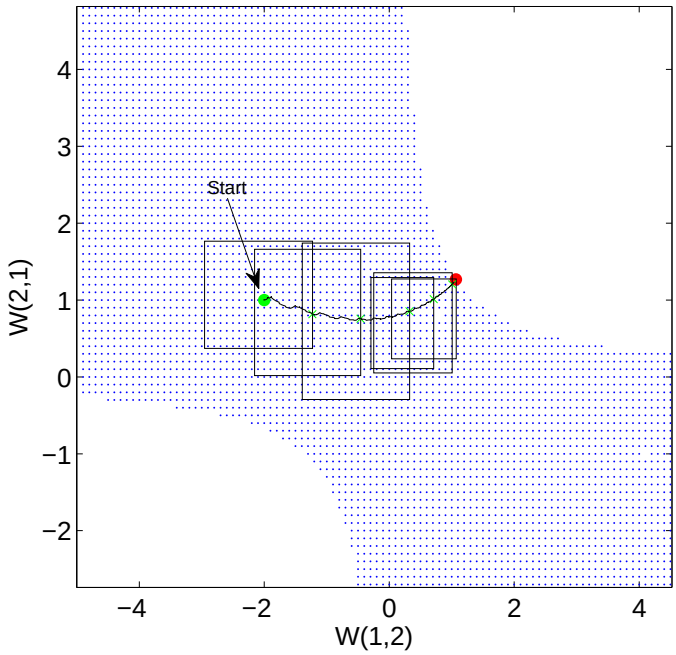

Figure 5.7: Shown here are the results of applying the stability constrained learning algorithm to a sample trajectory. This is an ideal case where very few stability constraint sets are generated.

the weight adaption should be minimized. Reducing the conservativeness of the stability analysis is not much help in this situation. If the error gradient points toward the stability boundary and the weight trajectory moves toward the boundary, allowing more variation in the interior of $\mathcal{W}_{ss}^n$ does not help. To alleviate this problem the error gradient should be modified with a term that captures stability information. In the next section, a method is introduced that explicitly biases the trajectory away from the boundary of $\mathcal{W}_{ss}^n$.

5.4 A Stability Bias

To bias learning trajectories away from the boundary of $\mathcal{W}_{ss}^n$ a measure of closeness to this boundary is needed. Fortunately, as is obvious from its definition and illustrated in Figure 3.4, the $\mathcal{L}_2$-gain of the RNN with static weights grows rapidly near the boundary of $\mathcal{W}_{ss}^n$. So the $\mathcal{L}_2$-gain acts as the inverse of distance to the boundary. The magnitude of the $\mathcal{L}_2$-gain is not immediately useful as a bias, however, since it contains no information about the direction of the boundary from a weight matrix. On the other hand, the gradient of this value with respect to W carries information about closeness to the boundary and of its direction from W. In this section it is shown how to compute this derivative. Its use as a bias for weight trajectories is illustrated using the sample problem.

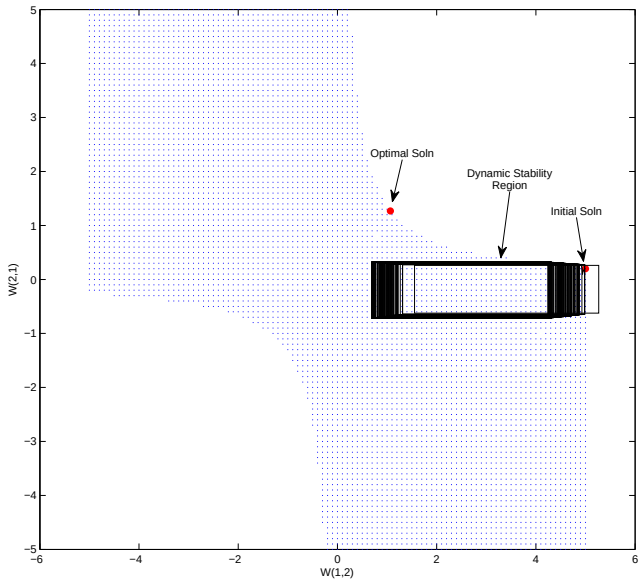

Figure 5.8: Bounds on the variation of the RNN weights within which stability can be proved are shown in black rectangles. If the learning methods attempts to update the RNN weights outside the current region a new region is computed, centered at the last stable weight values. If the update falls outside this new region it is dropped and the algorithm proceeds to the next iteration of learning. Notice how RTRL causes the weight trajectory to proceed along the stability boundary and forces the continual recomputation of the stable variation region.

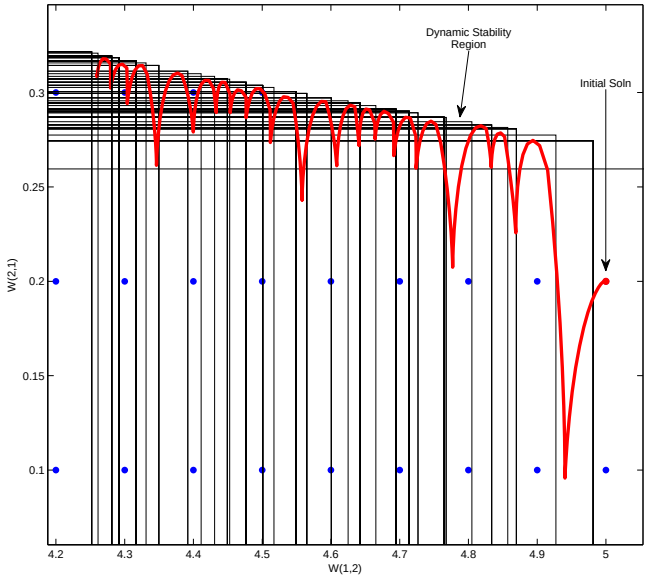

Figure 5.9: A close up of Figure 5.8 showing the weight trajectory.

5.4.1 Optimality and SDPs

Consider a general, nonlinear, semidefinite program given in [27]

$$\begin{aligned} p^* = \min b^T x \quad \text{s.t.} \quad & x \in \mathbb{R}^n, \\ & \mathcal{B}(x) \preceq 0, \\ & c(x) \leq 0, \\ & d(x) = 0. \end{aligned} \tag{5.3}$$

The optimization problem encompasses all of the stability problems constructed in the earlier chapters. The Lagrangian of this problem $\mathcal{L} : \mathbb{R}^n \times \mathbb{S}^m \times \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$, is defined by [27]

$$\mathcal{L}(x, Y, u, v) = b^T x + \mathcal{B}(x) \bullet Y + u^T c(x) + v^T d(x) \tag{5.4}$$

where $Y \in \mathbb{S}^m$, $u \in \mathbb{R}^p$, and $v \in \mathbb{R}^q$ are the Lagrange multiplier variables. The Lagrangian dual function, defined as, [10]

$$g(Y, u, v) = \inf_x \mathcal{L}(x, Y, u, v) \tag{5.5}$$

is a lower bound on the optimal value of (5.3) for all values of the multiplier variables. When the best lower bound given by (5.5), that is,

$$d^* = \max_{Y,u,v} g(Y, u, v) \quad \text{s.t. } Y \succeq 0, u \succeq 0, \tag{5.6}$$

is equal to, p^*, the optimal value of (5.3), the problem is said to satisfy a *strong duality* condition. For convex optimization problems a sufficient condition, known as Slater's condition, for strong duality is the existence of a strictly feasible point. So, if $\mathcal{B}(x)$, $c(x)$, and $d(x)$ are convex functions of x, and there exists an x satisfying, $\mathcal{B}(x) \preceq 0$ and $c(x) < 0$ then $d^* = p^*$.

Often, rather than considering a single SDP a set of related SDPs parameterized by some data θ is of interest. For example, the LMI stability condition for RNNs with time invariant weights forms a set of SDPs parameterized by W. For parameterized SDPs, the Lagrangian, p^*, and d^* are functions of the problem data and are written $\mathcal{L}(x, Y, u, v; \theta)$, $p^*(\theta)$, and $d^*(\theta)$. Of specific interest is the set of θ for which the SDP satisfies the strong duality condition. Over this set, the affect of perturbing the data, θ, on the optimal solution, $p^*(\theta)$, can be estimated with a Taylor series expansion using the gradient defined by

$$\nabla_\theta p^*(\theta) = \left[\frac{\partial}{\partial \theta_i} p^*(\theta)\right] = \left[\frac{\partial}{\partial \theta_i} d^*(\theta)\right] = \left[\frac{\partial}{\partial \theta_i} \mathcal{L}(\bar{x}, \bar{Y}, \bar{u}, \bar{v}; \theta)\right].$$

This gradient is well defined when the Lagrangian is differentiable with respect to the data which is always the case when it is linear in the parameters of interest. The gradient is a first order approximation to the function $p^*(\theta)$ and gives the direction in the parameter space in which $p^*(\theta)$ increases the most in this approximation.

5.4.2 Application to RNN Stability Conditions

Consider the optimization problem from Chapter 3,

$$\bar{\gamma} = \inf_{\gamma,T,P} \gamma \quad \text{s.t.} \tag{5.7}$$

$$\begin{bmatrix} -CP - PC + I & P & PW + T \\ P & -\gamma I & 0 \\ W^T P + T & 0 & -2T \end{bmatrix} < 0, \quad P = P^T, \quad T \in \mathcal{D}_+, \tag{5.8}$$

associated with proving the stability of a time invariant RNN. Here, the decision variables are $x = (\gamma, T, P)$. An upper bound on the gain of the RNN with weight matrices C and W is given by $\sqrt{\bar{\gamma}}$. The LMI constraint in the problem has an associated Lagrange multiplier denoted Y. Take the Lagrangian to be a function of the weight matrix, W, and write $\mathcal{L}(x, Y; W)$. The problem is convex and by the definition of $\mathcal{W}_{ss}^n$ satisfies Slater's condition for all W in $\mathcal{W}_{ss}^n$. The Lagrangian takes the value $\bar{\gamma}$ at the solution to (5.7). Since the Lagrangian is linear in W, and thus differentiable, the gradient of $\bar{\gamma}$ with respect to W can be computed by the formula

$$\nabla_W \bar{\gamma} = \left[\frac{\partial}{\partial W_{ij}} \mathcal{L}\left(\bar{x}, \bar{Y}; W\right) \right]. \tag{5.9}$$

For conciseness $\nabla_W \bar{\gamma}$ will be denoted by ∇_s throughout the remainder of the chapter. The Lagrangian in (5.4) specializes to

$$\mathcal{L}(x, Y; W) = \gamma + \mathcal{B}(x) \bullet Y$$

in this case. The function $\mathcal{B}(x)$ corresponds to the left hand side of the LMI constraint in (5.8). It is a function of the problem data W given by

$$\mathcal{B}(x; W) = \sum_i x_i B^{(i)}(W)$$

where the $B^{(i)}$s are linear functions taking W into $\mathbb{S}^m$. Since this is the only point at which the problem data enters the Lagrangian, its gradient is easily computed as

$$\left[\frac{\partial}{\partial W_{ij}} \mathcal{L}\left(\bar{x}, \bar{Y}; W\right) \right] = \left(\sum_{i=1}^{n} \bar{x}_i \frac{\partial}{\partial W_{ij}} B^{(i)}(W) \right) \bullet \bar{Y}. \tag{5.10}$$

Since $\mathcal{B}(x; W)$ is linear in W, the partial derivatives $\frac{\partial}{\partial W_{ij}} B^{(i)}(W)$ are constant and can be computed without the knowledge of a specific W. Evaluation of the gradient is a simple operation, but requires the optimal values $\bar{x}$ and $\bar{Y}$ associated with a given W. This in turn requires that the optimization problem (5.7) be solved for each evaluation of the stability bias. Solving the optimization problem (5.7) dominates the computational cost of computing ∇_s. As discussed in Chapter 3, this computation has a run time on the order of $\Theta(n^3)$ to $\Theta(n^6)$ depending on the IQCs used and the choice of SDP solver.

Figure 5.10 shows the computed gradients of the $\mathcal{L}_2$-gain at a sample of weight matrices from the example problem. It is clear from the example that ∇_s contains the desired information. That is, the gradient points away from the stability boundary toward the interior of $\mathcal{W}_{ss}^n$ and has a greater magnitude when W is nearer to the boundary. It appears that biasing weight trajectories using the gradient ∇_s will successfully push the weight trajectories away from the stability boundary. Nevertheless, the effect of the bias on the optimization of the overall problem objective still needs to be considered.

Ideally, the stability bias would affect only the path the parameters take to an optimal solution and not the final point to which they converge. To achieve this ideal behavior the effect of the stability bias on the weight trajectories should be small relative to the effect of the updates given by the learning algorithm and should diminish over time. For example, the update rule might scale the magnitude of ∇_s with respect to the magnitude of the learning update, $\eta \nabla_l$, where, for example, $\nabla_l = \gamma(t) \frac{\partial E(t)}{\partial y(t)}$ in the case of RTRL. Such an update would look like

$$W \leftarrow W - \left(\eta \nabla_l + \eta_2 \frac{\|\eta \nabla_l\|_2}{\|\nabla_s\|_2} \nabla_s \right) \tag{5.11}$$

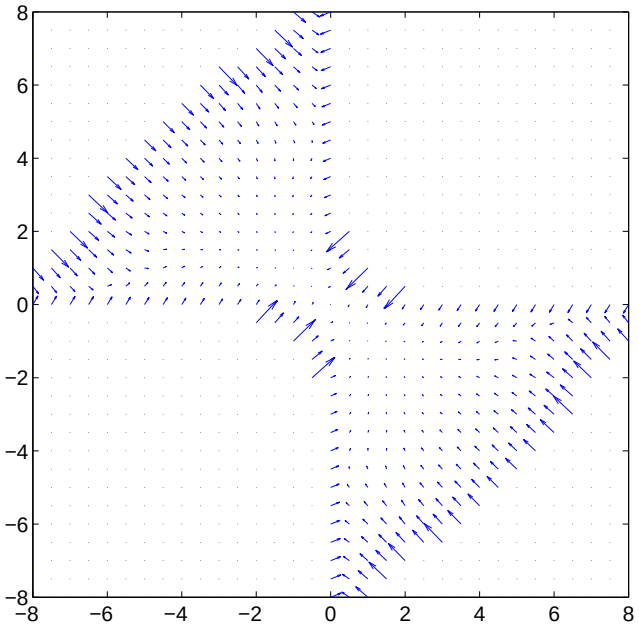

Figure 5.10: Examples of $\nabla_W \bar{\gamma}$ with respect to the two off diagonal weights of W_1. See Figure 3.4 in Chapter 3 for the actual $\mathcal{L}_2$-gain values.

with $\eta_2 < 1$ and decreasing with time. A potential problem with this approach is that the effect of the stability bias is small regardless of closeness to the boundary of $\mathcal{W}_{ss}^n$. An alternative updating scheme varies the scale of the stability bias with the magnitude of the $\mathcal{L}_2$-gain bound, γ. The update is given by

$$\begin{aligned} \eta_3 &\leftarrow \eta_2 \frac{3}{4}(\tanh(\gamma - \hat{\gamma}) + 1) \\ W &\leftarrow W - \left(\eta \nabla_l + \eta_3 \frac{\|\eta \nabla_l\|_2}{\|\nabla_s\|_2} \nabla_s \right) \end{aligned} \tag{5.12}$$

where $\eta_2 \leq 1$ and decreases with time as before. The scaling, η_3 is a monotonically increasing function of γ with a transition from low to high centered at $\hat{\gamma}$. This approach has the benefit of not affecting the weight trajectory when it is far from the boundary of $\mathcal{W}_{ss}^n$, but requires the transition point $\hat{\gamma}$ to be chosen. The scaling η_2 must decrease to zero with time to ensure that asymptotically the stability bias has no effect. If a certain maximum gain value is desired for the system, allowing η_3 to remain positive and setting $\hat{\gamma}$ to the intended gain bound helps to bias the system toward satisfying this condition. Methods for choosing a decay schedule for the parameter η_2 are rather ad-hoc, but the learning rate η for the RTRL updates has the same requirements and associated problems.

5.4.3 Example

The example from Figure 5.8 is repeated in Figure 5.11 but with a stability bias added to the weight updates. The update formula (5.11) is used and $\eta_2 = 0.1$ throughout the run. The BMI approach was used to generated the constraint sets as in the previous example. The bias clearly improves the performance of the algorithm by reducing the number of constraint sets that must

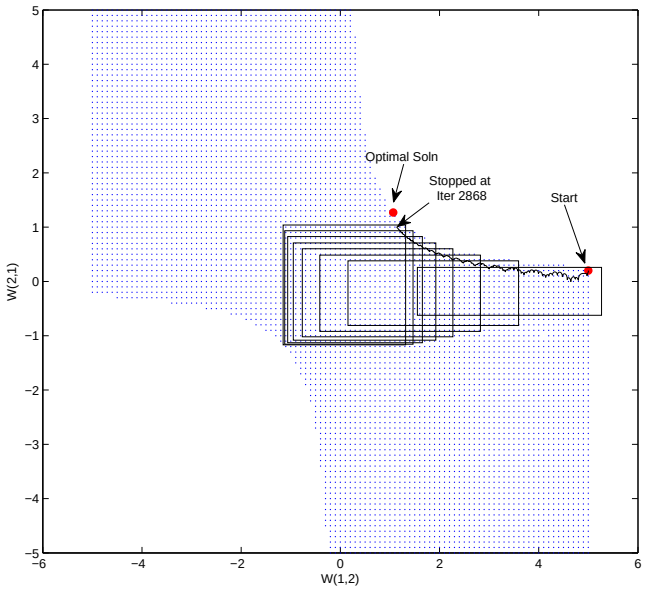

Figure 5.11: The same problem, but with the stability gradient incorporated into the weight update. Notice how the trajectory is further from the stability boundary and fewer recomputations of the stable variation region are needed.

be computed. In the example shown in Figure 5.8 a constraint set was computed at almost every update. Replacing these expensive computations with the less costly stability bias computations reduces the cost of the STABLE-LEARNING-ALGORITHM.

To explore the effect of the stability bias further, four trials of 1000 steps were run from different starting points in $\mathcal{W}_{ss}^n$. The starting points are $(3.95, 0.3)$, $(-0.4, -3.0)$, $(0.3, 4.0)$, and $(-4.0, 0.0)$. During the trials the number of rejected updates and constraint set computations were counted. Also, the mean amount of variation allowed in a weight was measured, and the distance of the final solution from the optimal solution was recorded. Three configurations of the stable learning algorithm are compared. The first uses no stability bias. The second uses a stability bias with the update rule (5.11) and $\eta_2 = 0.001$. The third configuration uses the update rule (5.12) with $\eta_2 = 1.0$. The results for this experiment are shown in Table 5.1. The four rows in each box correspond to the results for the four different starting points. The matrix W_f is the final weight matrix reached during the 1000 steps of learning.

The results contain several interesting features. For the first and third starting point the use of the stability bias update (5.12) significantly reduces the number of constraint sets that are computed, the number of rejected updates, and the distance from the optimal solution after 1000 steps. Also, the average amount of weight variation allowed is larger when using (5.12) due to the trajectories begin further from the stability boundary. The differences on the second and fourth starting points are minimal, but the stability bias does not seem to have a negative impact. Increasing η_2 to 0.1 for use with (5.11) improves its performance to about the level of (5.12) in the reported results. Nonetheless, the update (5.12) seems to be preferable because it has zero impact when the trajectory is far from the stability boundary. This simple experiment illustrates the benefit of using the proposed stability bias. The cost of computing the bias is still prohibitive, since it is done every step. In the next section a method for reducing the cost of the bias computation is developed.

	Rejected Steps	Constraint Sets	Mean Variation	$\|W - W_f\|_F$
No Bias	38	220	0.0794	1.9957
	0	5	1.8751	2.0885
	24	145	0.2538	1.5252
	0	2	4.2041	2.5307
(5.11)	38	174	0.0836	2.2487
	0	5	1.4834	2.0700
	26	171	0.1451	1.5538
	0	2	4.2116	2.5377
(5.12)	0	8	1.0029	1.0968
	0	4	2.0527	2.0393
	0	10	1.3200	1.0911
	0	2	4.2041	2.5314

Table 5.1: Results of applying the stable learning algorithm with and without the stability bias.

5.5 Solving Perturbed SDPs

The computation of the proposed stability bias requires that the semidefinite program (5.7) be solved for each W at which the gradient, ∇_s is evaluated. While this SDP is smaller and less computationally burdensome than the SDP associated with computing a new set of variation constraints, computing its solution at each update to W is still costly. Updates to the RNN weights, however, are generally small and give rise to SDPs that are mild perturbation of the SDP associated with the previous weights. If the perturbation is small enough, it is reasonable to assume that the solutions of the original and perturbed problems will be similar. Analysis in [26, 28] shows that this is indeed the case under certain conditions on the problem. The details of this analysis are discussed below. In this section the cost of evaluating the stability bias over a sequence of weight matrices is significantly reduced by making use of similarity between the solutions of (5.7) for W and $W + \Delta W$.

5.5.1 A Warm-Start Method for SDPs

Optimization methods that use the solutions of previous problems to speed up the optimization of new problems are known variously as *warm-start* or *hot-start* methods. These methods make the most sense and have the greatest impact when the problems being solved are related in some way. For example, consider a sequence of linear programs where a constraint is added in each new problem. In the context of computing the proposed stability bias, the problems are related through perturbations of the problem data. A successful warm-start method reduces the cost of solving the set of problems in sequence relative to the cost of solving them all individually. Unfortunately, simply initializing the problem variables to the values of the previous problem's solution does not generally make a successful warm-start method. In fact, using such an initialization can sometimes increase the cost of solving the sequence of problems. This slow down effect has been observed in the case of interior point methods for linear programming [95].

While interior point methods (IPMs) are generally the most efficient methods for solving semidefinite programs and other convex optimization problems [47], it has proved quite difficult to successfully benefit from warm-start information in IPMs [95, 95, 41]. Because of this deficiency in IPMs, methods that are less generally efficient, but better able to leverage warm-start information

may preferable in the context of solving sequences of related problems. In the context of semidefinite programming, there is very little published research on warm-start techniques for solving perturbed problems. A method was suggested specifically for application to the Max-Cut graph partitioning problem where constraints are added sequentially to a sequence of SDP problems [51]. More general purpose approaches are lacking in the current literature.

Completely addressing the issue of warm-start for semidefinite programming is beyond the scope of this work. Nonetheless, the basic outline of a warm-start procedure is developed and successfully applied to the problem of computing the stability bias at a sequence of weight matrices. The findings reported in this section indicate that the augmented Lagrangian algorithm described in Section 2.4 and developed in [46, 47] is capable of capitalizing on available warm-start information. When the solution to a nearby problem is known, the number of iterations of the augmented Lagrangian algorithm needed to solve a problem can be significantly reduced. Furthermore, the results in [27, 28], characterize the effect of data perturbations on the decision variables and Lagrange multipliers of an SDP as the solution to a linear program. At little additional cost, this characterization can be used to modify a given warm-start solution in the direction of the perturbed problem's solution. Improving the warm-start information in this way further reduces the number of iterations necessary in the augmented Lagrangian algorithm.

5.5.2 Warm-Start and the Augmented Lagrangian Method

In the light of the known problems with warm-starting IPMs and given the results in Chapter 3 demonstrating the efficiency of the augmented Lagrangian method [47] on the SDPs of interest, a simple warm-start procedure for the augmented Lagrangian method is presented. Before describing the warm-start approach, some details of the algorithm in [47] are discussed. Recall that the augmented Lagrangian approach involves solving a sequence of minimization problems over the augmented Lagrangian function

$$F(x, Y, p) = b^T x + \Phi(\mathcal{B}(x), p) \bullet Y$$

where $p > 0$ is a penalty parameter and $\Phi(\cdot)$ is a penalty function satisfying the condition

$$\mathcal{B}(x) \preceq 0 \;\Leftrightarrow\; \Phi(\mathcal{B}(x), p) \preceq 0.$$

The algorithm alternates between finding the minimum of $F(x, Y, p)$ for fixed Y and p and updating Y and p. The solutions of the minimization problem, $\hat{x}$, are required to satisfy the penalty barrier constraint, $\Phi(\mathcal{B}(\hat{x}), p) \preceq 0$. By decreasing p feasible solutions can be approached if they exist. The problem becomes ill-conditioned as p approaches zero, but it can be shown that if a solution exists it will be found with $p > 0$. The basic algorithm is given in Figure 5.12 and details about $\Phi(\cdot)$ and the updates to the Lagrange multiplier variables and the penalty parameter can be found in [46, 47].

The naive approach to warm-starting the augmented Lagrangian algorithm is to simply initialize x and Y to the solution of the previous problem. The experiments later in the chapter show that, unlike IPMs, the augmented Lagrangian algorithm is able to use this warm start information successfully. When the perturbations are small the cost of solving the perturbed problem is significantly reduced by initializing the solver with the warm start solution.

5.5.3 Improving the Warm-Start Data

The experiments in the next section show that the augmented Lagrangian method can successfully use warm-start data to decrease the cost of solving a perturbed SDP. The cost can be further

PENSDP

Initialize x_0, $Y_0 \succeq 0$, and $p_0 > 0$, $k \leftarrow 0$.
repeat
$x_{k+1} \leftarrow \operatorname{argmin}_{x \in \mathbb{R}^n} F(x, Y_k, p_k)$
$Y_{k+1} \leftarrow D\Phi(\mathcal{B}(x_{k+1}), p)[Y_k]$
$p_{k+1} \leftarrow f(p_k)$ where $f(p_k) < p_k$.

Figure 5.12: The PENSDP algorithm. The notation $D\Phi(\mathcal{B}(x_{k+1}), p)[Y_k]$ denotes the derivative of the augmented Lagrangian with respect to the Lagrange multiplier evaluated at Y_k.

reduced by applying some SDP analysis results from [26, 28]. In this section it is shown how to improve the warm-start data at the cost of solving a linear program. The solution of the linear program gives a perturbation of the warm-start data that is closer to the true solution of the perturbed SDP problem. The augmented Lagrangian solver starting at this improved initial point requires fewer iterations to solve the perturbed SDP.

The following discussion is a simplification of results in [26] to the case of linear SDPs. Consider an SDP of the form

$$\begin{array}{lll} \min b^T x & \text{s.t.} & x \in \mathbb{R}^n \\ & & \mathcal{A}(x) + C \preceq 0 \end{array} \tag{5.13}$$

where $\mathcal{A}(x) : \mathbb{R}^n \to \mathbb{S}^m$ is an affine function. The problem (5.13) can be viewed as being parameterized by the data $\mathcal{D} = [\mathcal{A}, b, C]$. Given a perturbation to this problem data, $\Delta\mathcal{D} = [\Delta\mathcal{A}, \Delta b, \Delta C]$, related perturbation of the problem solution, (x, S, Y), can be computed. Here, x is the decision variable, S is the slack variable, and Y is the dual variable associated with the matrix constraint. A solution, $\left(\bar{x}, \bar{S}, \bar{Y}\right)$ of (5.13) is called a *stationary point* if it satisfies the conditions

$$\begin{aligned} \mathcal{A}(\bar{x}) + C + \bar{S} &= 0, \\ b + \mathcal{A}^*(\bar{Y}) &= 0, \\ \bar{Y}\bar{S} + \bar{S}\bar{Y} &= 0, \text{ and} \\ \bar{Y}, \bar{S} &\succeq 0. \end{aligned} \tag{5.14}$$

The point is called a *strictly complementary* stationary point if it is a stationary point and satisfies the condition

$$\bar{Y} + \bar{S} \succ 0. \tag{5.15}$$

The following theorem concerning the effect of the perturbation, $\Delta\mathcal{D}$, on the solution, $(\bar{x}, \bar{S}, \bar{Y})$, is proved in [26].

Theorem 5.1. *Assume that* (5.13) *satisfies Slater's condition. That is, assume there is at least one feasible $x \in \mathbb{R}^n$. Let the point, $\left(\bar{x}, \bar{S}, \bar{Y}\right)$, be a locally unique, and strictly complementary stationary point of* (5.13)*. For sufficiently small perturbations, $\Delta\mathcal{D}$, there exists a locally unique stationary point, $\left(\bar{x}(\mathcal{D} + \Delta\mathcal{D}), \bar{S}(\mathcal{D} + \Delta\mathcal{D}), \bar{Y}(\mathcal{D} + \Delta\mathcal{D})\right)$ of the perturbed program* (5.13) *with data $\mathcal{D} + \Delta\mathcal{D}$ that is a differentiable function of the perturbation. The derivative $D_{\mathcal{D}}\left(\bar{x}(\mathcal{D}), \bar{S}(\mathcal{D}), \bar{Y}(\mathcal{D})\right)$ of $\left(\bar{x}(\mathcal{D}), \bar{S}(\mathcal{D}), \bar{Y}(\mathcal{D})\right)$ with respect to $\mathcal{D}$ evaluated at $\left(\bar{x}, \bar{S}, \bar{Y}\right)$ is characterized by the directional derivatives*

$$\left(\dot{x}, \dot{S}, \dot{Y}\right) = D_{\mathcal{D}}\left(\bar{x}(\mathcal{D}), \bar{S}(\mathcal{D}), \bar{Y}(\mathcal{D})\right)[\Delta\mathcal{D}] \tag{5.16}$$

for any $\Delta\mathcal{D}$. *The point* $\left(\dot{x}, \dot{S}, \dot{Y}\right)$ *is the unique solution of the system of linear equations,*

$$\begin{aligned} \mathcal{A}(\dot{x}) + \dot{S} + \Delta C + \Delta\mathcal{A}(\bar{x}) &= 0, \\ \mathcal{A}^*(\dot{Y}) + \Delta b + \Delta\mathcal{A}^*(\bar{Y}) &= 0, \\ \bar{Y}\dot{S} + \dot{Y}\bar{S} + \dot{S}\bar{Y} + \bar{S}\dot{Y} &= 0, \end{aligned} \tag{5.17}$$

for the unknowns $\dot{x} \in \mathbb{R}^n$ *and* $\dot{S}, \dot{Y} \in \mathbb{S}^m$.

The system of linear equations describing the directional derivative, (5.16), has $m^2 + m + n$ equations, (5.17), and $m^2 + m + n$ unknowns, the entries of $(\dot{x}, \dot{S}, \dot{Y})$. The LP has many more variables than the original SDP problem, but if it can be solved more quickly than the original SDP and if the perturbed warm-start data significantly improves the run time of the SDP solver an overall speedup may be achieved.

The application of Theorem 5.1 to the warm-start problem is straightforward. Given the solution to an SDP, (x, Y, S), and a perturbation of the problem data, $\Delta\mathcal{D}$, solve the linear program (5.17). Update, (x, Y, S), by adding to each the associated part of the computed solution, $(\dot{x}, \dot{S}, \dot{Y})$. The point, $(x + \dot{x}, S + \dot{S}, Y + \dot{Y})$, will not generally be the exact solution of the perturbed SDP. The quality of the approximation given by the LP will degrade as the magnitude of the perturbation to the problem data increases. The modified data can, however, be used to initialize the augmented Lagrangian solver which can compute the exact solution to the perturbed SDP.

The RNN stability analysis problem (5.7) is easily written in the form of (5.13). Multiple LMI constraints and scalar linear constraints can be treated as a single LMI by diagonal augmentation of the constraints into a single matrix. Modifications of the RNN weight matrix affect the $\mathcal{A}(x)$ data but do not affect b and C. This simplifies the constraints (5.17) slightly. The perturbation to $\mathcal{A}(x)$ will affect all of the problem variables, (x, Y, S). As previously discussed, for stable RNNs, the stability analysis problem has a strictly complementary solution that will be unique because of the problem's structure. Perturbations to the weight matrix may not be *sufficiently small*, and the LP will only give an approximation to the solution of the perturbed problem. Modifications to the weight matrix that are too large can cause the approximation to be very poor. Using these poor approximations to update the warm-start solution may actually increase the solution time of the augmented Lagrangian solver. Another problem that may be encountered is that the computed solution to the previous SDP may not be exact or of high enough accuracy. The inaccuracy leads to inconsistencies in the linear program and bad approximations to the solution of the perturbed SDP. Both of these problems occurred in the experiments in the next section on occasion. These problems can generally be detected by analyzing the magnitude of the LP solution. Since the weight perturbations are typically small, solutions of the LP with magnitudes much larger than the magnitude of the perturbation most likely signify that one of these problems has occurred. In these cases, no update to the warm-start data should be made. The next section analyzes the proposed warm-start procedure experimentally and shows that the correction given by the solution of (5.17) further reduces the cost of solving the perturbed SDPs relative to the naive warm-start approach.

5.5.4 Experimental Evaluation

The PENBMI software used in the previous chapters for examining the augmented Lagrangian approach does not support the initialization of the Lagrange multiplier variables needed to test the proposed warm-start method. In the evaluation here a Matlab implementation based on the publications [47, 46] is used. The implementation is not as efficient as the software available from the authors, so the evaluation is not performed in terms of actual run times. Instead, the evaluation

	Main Iteration Count	Minimization Problems	CG Steps
No Warm Start	32	69	271
Warm Start	41	12	126
Augmented Warm Start	57	6	34

Table 5.2: Iteration counts for the different warm start approaches applied to a single 5×5 weight matrix and perturbation.

is presented in terms of three counts: the number of iterations of the main algorithm, the total number of iterations in the minimization of the Lagrangian, and the total number of conjugate gradient steps in the same minimization problem. When the modified Newton's method is used to minimize the augmented Lagrangian, the run time of the algorithm is governed by the cost of forming and solving the Newton's equations. The total number of steps in the minimization then acts as a surrogate for total run time. If the conjugate gradient approach is used, the run time is governed by both the cost of preconditioning and the cost of the conjugate gradient steps. In this case, both the number of steps in the minimization and the total number of conjugate gradient steps should be considered. The number of steps in the main algorithm captures the the total cost of updating the Lagrange multipliers. These updates are relatively cheap compared to solving the minimization problem but are not trivial. The updates require a matrix inverse and the computation of a quadratic matrix product, ABA^T. A large increase in the number of main steps can slow the algorithm down.

A simple first experiment is constructed using the matrix

$$W = \begin{bmatrix} -0.1153 & 0.3176 & -0.0498 & 0.0304 & 0.0785 \\ -0.4442 & 0.3171 & 0.1935 & 0.2845 & -0.3563 \\ 0.0334 & -0.0100 & -0.1569 & 0.0158 & 0.1905 \\ 0.0767 & 0.0873 & 0.5822 & -0.0255 & 0.4329 \\ -0.3057 & 0.0466 & -0.0364 & -0.2220 & -0.1845 \end{bmatrix}$$

and the perturbation

$$\Delta W = \begin{bmatrix} 0.0009 & -0.0004 & 0.0007 & -0.0016 & 0.0005 \\ 0.0013 & 0.0007 & 0.0012 & 0.0003 & 0.0002 \\ -0.0016 & 0.0008 & -0.0012 & -0.0011 & -0.0009 \\ -0.0014 & 0.0007 & -0.0000 & 0.0014 & -0.0022 \\ 0.0006 & 0.0013 & -0.0002 & -0.0008 & -0.0001 \end{bmatrix}.$$

Problem (5.7) is solved for W to derive the warm start information. The minimization subproblems in the augmented Lagrangian algorithm are solved by the conjugate gradient algorithm. Since the augmented Lagrangian method implemented for this experiment is rather basic, the multiplier T was taken to be positive diagonal, and the Popov multipliers were not used in the stability analysis and bias computations. The iteration counts for using no warm start information, using the naive warm start approach, and using the augmented warm start data to solve the stability problem for $W + \Delta W$ are shown in Table 5.2. A clear trend is visible in the data: using the warm start methods decreases the number of inner iterations and conjugate gradient steps at the cost of more multiplier updates. The use of the augmented warm-start data reduces the number of conjugate gradient steps of the naive warm-start approach by a factor of four. The cost of solving the LP needed to produce the augmented warm-start data is discussed at the end of this section.

The naive warm-start approach and the LP augmented approach are compared using the conjugate gradient method to solve the minimization subproblems. The methods were evaluated on

	Main Iterations	Minimization Problems	CG Steps	Bad LPs
No Warm Start	2235.66	4787.66	9432.66	0
WS, p def	2941	803.66	4231.33	0
WS, $p = 1$	1331.40	387.60	1890.40	0
WS Aug, p def	2476.00	684.00	3554.00	0.33
WS Aug, $p = 1$	2673.33	744.00	3828.33	0

Table 5.3: Average iteration counts for solving three sequences of 100 slightly perturbed SDP problems. The sequences of problems are solved with three different methods: no warm-start, naive warm-start, and augmented warm-start. The naive warm-start method with a fixed initial $p = 1$ performs the best.

sequences of weight matrices generated by applying the stability biased RTRL algorithm to the test problem in Section 5.1. Three sequences of 100 weight matrices were generated starting from different stable starting points. The actual stability bias used in the updates was computed by solving problem (5.7) to high accuracy using Sedumi. This ensured that the sequences of weight matrices in different runs did not differ due to slight differences in the solutions computed by the methods under comparison. The solution computed by Sedumi were not used in the augmented Lagrangian computations in any way.

When warm-starting was not used, the penalty parameter p was initialized using the standard approach from [46]. In the warm-start case two different settings of p were tested: the standard initialization and a fixed initialization of $p = 1$. In Table 5.3, average iteration counts over the three weight sequences are shown for the various algorithm configurations.

It is clear from the data that using warm-start information provides a drastic reduction in the cost of solving the sequence of perturbed SDPs. Furthermore, initializing the penalty parameter p to one can more than double the performance improvement. For the augmented warm-start data, however, fixing p to one decreases the performance. In fact, the naive warm start solution with p initialized to one provides the most efficient solution of the sequence of problems. This suggests that while the augmented warm-start data can improve the solution cost in some cases, the necessary LP computation may not be worthwhile. Additionally, solution of the LP with freely available software generally took longer than solving the perturbed SDP from scratch. While this may not be the case with better performing, commercial LP solvers, the results suggest that the approach may not be cost effective.

5.6 Conclusions and Contributions

In this chapter an algorithm has been proposed that ensures the stability of an adaptive recurrent neural network under the adaption of an arbitrary agent. This so called Stable-Learning-Algorithm filters updates proposed by the adaptation mechanism to ensure that the RNN remains within the space of stable RNNs and that the variations in the weights do not lead to instability. The cost of this algorithm is the computation of bounds on the allowable variation in the weight of the neural network. Under certain circumstances these bounds must be computed very often. This often occurs when the weight trajectory is near the boundary of the set of stable RNNs, $\mathcal{W}_{ss}^n$. A method for biasing the trajectory away from this boundary was developed and shown to reduce the number of constraint sets generated by the stable learning algorithm. The computation of this stability bias is not cheap, however, and requires the solution of an SDP. When the weight matrix changes in small steps, the cost of computing the stability bias can be reduced by applying

warm-start methods. An approach to solving a sequence of perturbed SDPs was developed using a perturbation analysis of SDPs and the augmented Lagrangian method. Experiments in the previous section showed that the warm-start method could significantly reduce the cost of computing the stability bias.

Maintaining the stability of an RNN under adaption is not often useful or necessary on its own. On the other hand, when the RNN is introduced into a control system, stability becomes very important. Stability of the RNN is a first step toward ensuring the stability of the entire closed loop control system, but it is not sufficient. In the next chapter the stable learning algorithm and stability bias techniques developed here are applied in the context of a robust, adaptive, neural control system.

Chapter 6

Robust Adaptive Neural Control

The control of physical systems requires dealing with uncertainty. Uncertainty enters the control problem in at least three ways: unmeasured states, unknown dynamics, and uncertain parameters. Robust control is the problem of ensuring stability and performance in uncertain control systems. Modern robust control theory is based on explicit mathematical models of uncertainty [23]. If it is possible to describe what is unknown about a system, stronger assurances can be made about its stability and performance. The automation of robust controller design relies on the tractable representation of the uncertainty in a system. For example, some types of uncertainty can be described by IQCs and lead to representations of uncertainty as convex sets of operators. Linear systems are a particularly tractable type of model, and the design of feedback controllers for linear systems is a well understood problem. Thus, linear models of physical systems are particularly attractive. Most physical systems, however, exhibit some nonlinear dynamics and linear models are generally insufficient for accurately describing them. Unmodeled nonlinear dynamics can often be treated within the same framework as uncertainty. The recurrent neural network stability analysis presented in Chapters 3 and 4 relies on this approach. Because robust controllers must be insensitive to inaccuracies and uncertainties in system models, performance is often sacrificed on the actual system to which the controller is applied. Additional loss in performance is introduced by restricting controllers to be linear and of low order. These properties are generally desirable because low order, linear controllers can be easily analyzed and understood. Performance can often be improved by the use of nonlinear and adaptive control techniques, but guaranteeing stability and performance is more difficult in this environment. In this chapter the use of adaptive, recurrent neural networks in control systems is examined within a framework of robust stability requirements.

Recurrent neural networks are, in some respects, ideal for applications in control. The nonlinearity in neural networks allows for the compensation of nonlinearties in system dynamics that is not generally possible with low order, linear controllers. The dynamics of recurrent neural networks allow internal models of unmeasured states to be produced and used for control. The histories of the measured variables can often be used to internally model the behavior of the hidden dynamics. The difficulty in applying recurrent neural networks in control systems is in the analysis and prediction of the system's behavior for the purpose of stability analysis.

The control of a nonlinear, uncertain, multiple spring-mass-damper system is considered in this chapter. The system is a simple instance of a larger class of models representing physical systems such as flexible manipulators and active suspension systems. The goal of this chapter is to construct a neural controller for the system with guaranteed stability during operation and adaption. The stable learning algorithm presented in the previous chapter is applied to the problem, but suffers from conservativeness in the stability analysis of the closed feedback loop. Several modifications

to the basic algorithm are considered. An alternative algorithm that ensures that the feedback loop is stable for each static setting of the RNN weights visited during adaption of the controller is developed. The algorithm gives up the guarantee of dynamic stability given by bounding the variation in the RNN weights. This compromise reduces the computational cost of the algorithm and allows the RNN to successfully adapt a stable control strategy for the system. The stability bias described in Chapter 5 is used to keep the weight trajectory from getting stuck near the stability boundary. Without the bias, the adaption algorithm produces many updates that must be rejected, stalling the progress of learning. A comparison is made to a controller adapted without stability constraints. The unconstrained controller exhibits instability during its adaption that decreases its performance.

The next section introduces the control system under consideration and the associated models used for stability analysis and simulation. Also, an IQC analysis of the uncertain plant model is developed. Section 6.2 describes the control configuration used and develops an IQC analysis of the closed loop, control system. Also, details of the reinforcement learning algorithm used to train the adaptive controller are given. An experimental evaluation of the stable adaptive control system is reported in Section 6.3. The example illustrates the ability of the simplified stability algorithm to improve control performance by removing instability from the control loop. The stability bias proposed in the previous chapter is integral to the success of the approach. The final section gives a summary of the results and discusses some of the remaining deficiencies in the algorithm.

6.1 Two Degree of Freedom Spring Mass Damper

In this chapter, the stable learning algorithm of the previous chapter is used to adapt a controller for an uncertain, nonlinear spring mass system. The model is adapted from the work in [45]. To simplify the experimentation, the algorithm is applied to a simulated model of the actual system. Certain features of the simulated model are considered unknowns and not used explicitly in the adaption of controllers or stability analysis. The nonlinear model and simulation details are provided in this section. Additionally, an uncertain model of the plant is developed, and an IQC analysis of the closed loop control system is described.

6.1.1 The Simulated System

A diagram of the simulated plant is shown in Figure 6.1. Two masses are connected by nonlinear springs and linear dampers. The first mass is attached via a spring and damper to a stationary point. A control force is be applied to the first mass which is also acted upon by a nonlinear, static, friction force. A position sensor is attached to the second mass. The goal of the control problem is for the second mass to track a time-varying reference signal given by an external agent.

The plant dynamics are governed by the ordinary differential equations

$$\begin{aligned} &m_1\ddot{x}_1 + c_1\dot{x}_1 + c_2(\dot{x}_1 - \dot{x}_2) + k_1x_1 + k_2(x_1 - x_2) + h_1x_1^3 + h_2(x_1 - x_2)^3 = u - f(\dot{x}_1) \\ &m_2\ddot{x}_2 + c_2(\dot{x}_2 - \dot{x}_1) + k_2(x_2 - x_1) + h_2(x_2 - x_1)^3 = 0 \end{aligned} \tag{6.1}$$

where u is the control force applied to the first mass. Actuator dynamics are ignored for the purpose of these experiments, and the control enters the system linearly. The parameters c_1 and c_2 govern the damping force of the two dampers. The spring dynamics are governed by the spring constants k_1 and k_2 and the spring hardening constants $h_1 = h^2k_1$ and $h_2 = h^2k_2$ with $h \geq 0$. The spring hardening constant was set to $h = 0.1$ for all simulations of the system. The friction force is

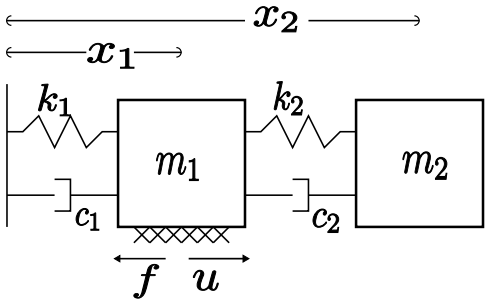

Figure 6.1: A multiple spring-mass-damper system.

modeled by the equation

$$f(\dot{x}_1) = g_7(g_1(\tanh(g_2\dot{x}_1) - \tanh(g_3\dot{x}_1)) + g_4\tanh(g_5\dot{x}_1) + g_6\dot{x}_1)$$

with

$$(g_1, g_2, g_3, g_4, g_5, g_6, g_7) = (12.5, 50, 1, 11, 50, 9, 0.05).$$

The magnitude of the friction force is shown in Figure 6.2 for $\dot{x}_1 \in [-5, 5]$. The friction equation and parameters are taken from [58] and model both stiction and Coulomb type friction.

All simulations of the system were performed using a variable step size, Dormand-Prince algorithm with an absolute error tolerance of 10^{-5}. Changes in the control signal occur at a rate of 10Hz, and observations are sampled at the same rate. For the purposes of simulation, the parameters were set to $m_1 = 2$, $m_2 = 2.01$, $c_1 = 1.05$, $c_2 = 0.97$, $k_1 = 5.3$, and $k_2 = 4.8$.

6.1.2 An Uncertain Linear Plant Model

For the purposes of controller design and stability analysis the parameters of the system are measured as $m_1 = m_2 = 2$, $c_1 = c_2 = 1$, $k_1 = k_2 = 5$ with an uncertainty of 2%, 10%, and 10%, respectively. The parameters are thus assumed to lie in the ranges $m_i \in [1.96, 2.04]$, $c_i \in [.91.1]$, and $k_i \in [4.5, 5.5]$. The simulated plant's parameters are within the assumed measurement errors, and uncertainty models based on these error estimates will be valid.

A linear model of the plant is easily constructed for use in control design and analysis. Ignoring the hardened spring and friction effects in (6.1) yields the linear model

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \dot{y}_4 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & m_1 & \\ & & & m_2 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -(k_1+k_2) & k_2 & -(c_1+c_2) & c_2 \\ k_2 & -k_2 & c_2 & -c_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}.$$

The set of linear models derived by taking the parameters in the given uncertainty ranges does not completely capture all of the possible dynamics of the system that is to be controlled. The uncertainty model should also account for the nonlinearity in the spring response and the friction force. It is also possible that the system exhibits other unmodeled dynamics. To account for the effect of the unmodeled dynamics, a multiplicative input uncertainty is added to the model. If $G(s)$ is the transfer function of the nominal linear plant model, then the uncertainty model looks like

$$G_{\text{unc}}(s) = G(s)(1 + 0.2\Delta_{\text{lti}})$$

where Δ_{lti} is an unknown, linear, time invariant system with $||\Delta_{\text{lti}}||_\infty < 1$. The unknown LTI system can model up to a 20% deviation in the plant input.

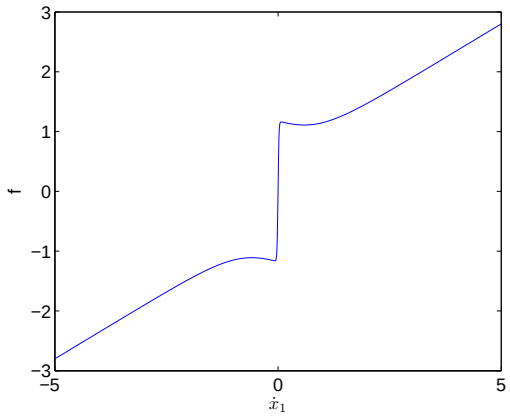

Figure 6.2: A continuously differentiable friction model.

An uncertain real scalar, such as any of the parameters of the linear plant model, can be modeled with the representation $p = (p_n + \delta_p m_p)$ where p_n is the nominal value of the parameter, δ_p is an unknown constant satisfying $|\delta_p| \leq 1$ and m_p is a scaling factor on the uncertainty [57]. To represent the uncertain parameter, $k_1 \in [4.5, 5.5]$, for example, take $p_n = 5$ and $m_p = 0.5$. The uncertain parameters in the model are assumed to be constant. The representation of time-varying parameters is essentially the same, but δ_p is allowed to vary with time. Time varying parameters can have a more varied impact on a system's behavior than constant uncertain parameters. The stability analysis presented below takes advantage of the fact that the parameters are constant to reduce the conservativeness of the analysis.

To perform robustness analysis and control design for the uncertain plant, the uncertainties must be factored out of the plant to form a feedback loop between the known LTI plant and the uncertainties: Δ_{lti} and the δ_p's for the uncertain parameters. The details of this procedure can be found in standard robust control references such as [57, 23]. The factored system is of the form depicted in Figure 2.4. Standard robustness analysis shows that the linear plant model is stable for all possible values of the uncertain parameters and all possible input perturbations allowed by the model [57]. The IQC model developed in the next section is used to show the same result.

6.1.3 IQC Analysis of the Plant

Recall that the uncertain parameters in the linear model are represented by $p = (p_n + \delta_p m_p)$ where δ_p is unknown but satisfies $|\delta_p| \leq 1$. If $w, v \in \mathcal{L}_2$, the relation $w = \delta_p v$ satisfies IQCs, with $\Pi(s)$ of the form

$$\Pi(s) = \begin{bmatrix} x(s) & z(s) \\ z^*(s) & -x(s) \end{bmatrix}$$
$$x(s) \geq 0$$

where $x(s)$ and $z(s)$ are bounded, measurable functions [59, 43]. This type of IQC is known as a dynamic IQC, as opposed to the static IQCs used in the earlier chapters. To work with the IQC, the functions $x(s)$ and $z(s)$ need computationally tractable representations. Generally, the functions are described as a linear combination of a finite set of simple basis functions. Increasing the number or complexity of the basis functions enlarges the set of IQCs but also increases the size of the resulting LMI constraints. For the analysis done in this section and the next, the function

$x(s)$ is represented as the combination of a constant function and a simple scaled transfer function

$$x(s) = x_0 + x_1 \frac{1}{s+1}.$$

The function $z(s)$ has the representation

$$z(s) = z_1 \frac{1}{s^2 - 1} = -\frac{1}{2} z_1 \left(\left(\frac{1}{s+1} \right)^* + \frac{1}{s+1} \right).$$

The representation of the resulting IQCs as LMIs is described in detail in Appendix B. Essentially, the representation requires the extension of the plant state with states representing $\frac{1}{s+1}v$ and $\frac{1}{s+1}w$. This extension increases the dimension of the system by two and adds $2n + 3$ decision variables in the KYP matrix P. Also, three decision variables are added for x_0, x_1, and z_1. Enforcing the constraint $x(s) \geq 0$ requires application of the KYP lemma and adds a further decision variable and a 2×2 matrix constraint to the resulting problem. Modeling uncertain parameters can quickly become expensive, even when $x(s)$ and $z(s)$ are restricted to be linear combinations of only two basis functions.

Unmodeled LTI dynamics, of the kind used in the uncertain plant model developed above, can also be described with IQCs. If Δ_{lti} is an LTI operator with norm less than one, then the relation $w(s) = \Delta(s)v(s)$ satisfies all IQCs of the form

$$\Pi(s) = \begin{bmatrix} x(s) & 0 \\ 0 & -x(s) \end{bmatrix}$$
$$x(s) \geq 0$$

with $x(s)$ a bounded, measurable function of the form used in the previous IQC [59]. The unmodeled dynamics IQC increases the number of decision variables by $2n + 6$ and adds an additional 2×2 LMI constraint to the resulting problem.

A stability analysis of the uncertain plant model can be constructed using the IQC stability theorem from Chapter 2. Application of the theorem requires that the uncertainties be factored out of the model into a feedback formulation. This type of representation is known as a linear fractional representation and was used to describe recurrent neural networks in the earlier chapters. The details of constructing such a representation can be found in [57]. A depiction of the nominal plant with uncertainty feedback is shown in Figure 2.4. The uncertain operator in the feedback model is structured as

$$\Delta(s) = \text{diag}\{\Delta_{\text{lti}}(s), \delta_{m_1}, \delta_{m_2}, \delta_{k_1}, \delta_{k_2}, \delta_{c_1}, \delta_{c_2}\}.$$

A bound on the $\mathcal{L}_2$-gain from $u \to y$, computed using the IQC stability theorem, has a value of $\gamma = 1.4755$. The finite gain is a proof of stability for all plants covered by the uncertainty model.

6.2 Robust Adaptive Control of the Multiple Spring-Mass-Damper

The closed loop control system under investigation is depicted in Figure 6.3. A reference signal enters the system from an external source and specifies the desired value of the plant output. In this case, the position of the second mass in the spring-mass-damper system is to be controlled. For the experiments that follow, the reference signal takes values in the range $[-2, 2]$ and can change every 50 seconds. The reference signal might represent, for example, the desired position of a read head in a hard drive or the desired location of the end point of a flexible manipulator.

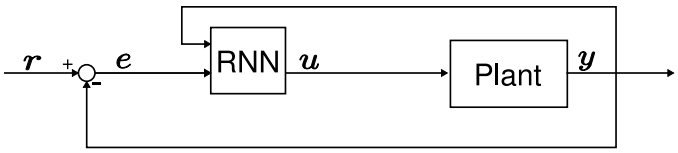

Figure 6.3: The closed loop control system.

Standard robust control designs for reference tracking problems generally use the error signal, $e = r - y$, as the input to the controller. This particular representation is invariant with respect to the position of the observed mass in the multiple spring-mass-damper system. It does not allow the controller to adequately compensate for nonlinearity in the spring, since the nonlinearity in (6.1) is a function of x_1 and $x_1 - x_2$. Even though the position of the first mass is unobserved, the trajectory of x_2 contains information about the true state of the system and thus x_1. It is possible for a recurrent neural network to use this inferred information to improve control performance. For this reason y is also included as an input to the RNN controller. The output of the RNN is the control action, u. The control signal is fed directly into the plant since actuator dynamics are being ignored.

6.2.1 Recurrent Neural Network Control Structure

The basic RNN equations must be adapted to fit the desired control structure. The simple RNN model considered in earlier chapters had the same number of inputs, outputs, and states. In the desired control configuration the network has two inputs, a single output, and a number of states set to determine the learning complexity. Input and output weights are added to the RNN in the following way

$$\begin{aligned} \dot{x_r} &= -Cx_r + W\Phi(x_r) + W^{\mathrm{i}} \begin{bmatrix} e & y \end{bmatrix}^T \\ u &= W^{\mathrm{o}} x_r. \end{aligned} \tag{6.2}$$

Different configurations of W^{i} and W^{o} affect the behavior of the network. A common configuration, see for instance [75], feeds each input into a different node and reads the output from another node. This leads to

$$W^{\mathrm{i}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ \vdots & \\ 0 & 0 \end{bmatrix} \text{ and } W^{\mathrm{o}} = \begin{bmatrix} 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}.$$

A similar configuration removes feedback from the output node by fixing certain weights in W, in this case $w_{3,i} \forall\ i \in 1, \dots, n$, to zero [24]. In echo state models, W^{i} and W are often assigned fixed, random values while W^{o} is fully adjustable [36]. For the experiments that follow the first configuration was used throughout.

The RNN stability analysis presented in Chapters 3 and 4 can easily be adapted to include the input and output weights. These weights do not affect analysis of the internal stability of the RNN, but they do affect the gain measured from the input to output signals. For instance, if the output weights are all zero then the gain from input to output is zero regardless, even, of the internal stability of the network. It is important, then, that these weights be included in the analysis since the measured gain affects the stability analysis of the closed loop control system. The weights, W^{i} and W^{o} only enter the analysis through the main KYP condition. As an example, consider

the LMI constraint derived from (6.2) by adapting the analysis presented in Chapter 3. The IQC stability theorem can be applying by modifying G in (3.2) to be

$$G = \left[\begin{array}{c|cc} -C & W^{\mathrm{i}} & W \\ \hline W^{\mathrm{o}} & 0 & 0 \\ I & 0 & 0 \end{array}\right]$$

and solving Problem 2.2.

6.2.2 IQC Analysis of the Closed Control Loop

The analysis of robust control systems focuses on two properties of an uncertain system model: robust stability and robust performance. A robustly stable system is stable for all systems in the given uncertainty set. A system with robust performance, on the other hand, is guaranteed to meet certain performance requirements for all systems in the uncertainty set. Robust stability is necessary for robust performance, but it is not sufficient. Robust stability of a set of uncertain systems implies stability of an actual physical system, *if* the physical system is adequately described by the uncertain model. Increasing the size of the uncertainty set — that is, increasing the range of system dynamics it covers — can allow better assurances to be made about the stability of an actual system. At the same time, increasing the size of the uncertainty set can make it more difficult to prove robust performance.

Performance of a robust control system is measured by the $\mathcal{H}_\infty$ norm of the system. In other words, performance is measured in terms of the $\mathcal{L}_2$-gain from the system's input to its output. Performance objectives in robust control systems are specified by augmenting a given control system with weighted outputs that specify the desired behavior. For example, the weighted outputs might be designed to penalize large actions or low frequency deviations of the plant output from the reference signal. Robust performance of a system is given if the gain from the input to the weighted outputs is less than one. Robust controller synthesis methods, such as the D-K iteration [23], are computational approaches to designing linear controllers that have robust performance for a given uncertain system model.

The synthesis of neural controllers with robust performance guarantees is a difficult problem. The main difficulty arises from treating the RNN nonlinearity as uncertainty in the analysis. The uncertainty descriptions of recurrent neural networks given in Chapter 3 in terms of IQCs have the serious drawback that they do not explicitly model the *boundedness* of the nonlinearities. The resulting uncertainty model and analysis can not distinguish between say, $\phi(x) = x$ and $\phi(x) = \tanh(x)$. Obviously, the resulting dynamics of the two RNNs will be very different. Recent work in [35] on generalized sector conditions may provide a way to incorporate the boundedness of $\phi(x)$ into the analysis, but at present this work is in its infancy. Until such improvements can be made, it is necessary to restrict attention to the analysis of robust stability in neural control systems. The inaccuracy in the analysis is slightly less troublesome here because the question being addressed — stability of the uncertain, closed loop system — is less specific. The analysis requires only that the gain from reference input to plant output is finite and not that it meet a prescribed bound.

While a robust synthesis method for neural controllers would be useful in generating the initial neural control design for a system, adaptation is necessary to specialize the controller to a given plant. In the context of adaptive control, robust performance is forgone in the name of performance on a specific plant. This is convenient in the case of neural control because of the problems discussed above with analyzing the robust performance of recurrent neural network controllers. Robust stability, however, is still desirable. Given that the plant is not completely known, and that even models adapted to the plant online can not be completely accurate, assuring stability with

respect to a set of plant models makes sense. The robust stability of the closed control loop can be addressed using the IQC descriptions of the plant and recurrent neural network derived earlier.

A linear fractional representation of the closed loop control system is needed to apply the IQC analysis results. Representations of the control loop containing an RNN with time-invariant weights and an RNN with time-varying weights are both needed. The details of constructing LFRs can be found in [57]. The uncertain, non-linear operator in the resulting model has the structure

$$\Delta(s) = \text{diag}\{\Delta_{\text{lti}}(s), \delta_{m_1}, \delta_{m_2}, \delta_{k_1}, \delta_{k_2}, \delta_{c_1}, \delta_{c_2}, \Phi(\cdot)\},$$

when the RNN weights are static. The operator is augmented with the time varying coefficients for the full, time-varying control loop model,

$$\Delta(s) = \text{diag}\{\Delta_{\text{lti}}(s), \delta_{m_1}, \delta_{m_2}, \delta_{k_1}, \delta_{k_2}, \delta_{c_1}, \delta_{c_2}, \Phi(\cdot), \overline{\delta}_{ij}, \underline{\delta}_{ij}\}.$$

Using IQCs for the different uncertainties and nonlinearities that have been previously described, an LMI problem can be constructed to assess the stability of the closed loop system and compute a bound on its gain. If the resulting LMI is feasible, the control loop is proved stable for all plants in the uncertainty set and additionally, for all controllers satisfying the IQC description of the RNN.

An an example, consider the RNN with $n = 3$, $C = I$, and

$$W = \begin{bmatrix} -1.4992 & 0.5848 & 0.5417 \\ 0.3425 & 0.4623 & 0.4551 \\ 0.7165 & 0.0323 & -0.2045 \end{bmatrix}.$$

The gain across the RNN is bounded from above by $\gamma_{\text{r}} = 0.9751$. The bound was computed using the Popov IQC and taking $T \in \mathcal{M}_{dd}$, the doubly dominant nonlinearity IQC. Since the gain of the uncertain plant model is bounded by $\gamma_{\text{p}} = 1.4755$ and $\gamma_{\text{r}} > 1/\gamma_{\text{p}} \approx 0.6777$, the small gain theorem fails to prove stability. The full IQC analysis, on the other hand provides a gain bound of $\gamma_{\text{cl}} = 1.7938$ proving the stability of the closed loop for all systems in the uncertainty set. A sense of the effect the plant uncertainty has on the analysis of the closed loop can be gained by measuring the gain of the RNN in a loop with just the nominal plant. The gain of this system is bounded above by $\gamma_{\text{n}} = 1.4830$. In this particular case the plant uncertainty has only a mild effect on the estimated loop gain, but this is not always the case.

Bounds on the allowable variation in the RNN parameters can be computed using a similar approach to the one used in Chapter 4, but using the full closed loop system model in the analysis. For the weight matrix given above variation bounds are computed as

$$\overline{\Delta} = \begin{bmatrix} 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 \end{bmatrix} \quad \text{and} \quad \underline{\Delta} = \begin{bmatrix} 0.3409 & 0.1210 & 0.2290 \\ 0.8891 & 0.4266 & 0.3296 \\ 1.8523 & 0.2090 & 0.4622 \end{bmatrix}$$

using the LMI approximation approach from Chapter 4 and restricting the bounds to be greater than or equal to 0.1. The uncertainty in the plant has a large effect on this particular computation. Computing the variation bounds using just the nominal plant model leads to an increase in the sum of the bounds by 1.8220 and for some weights doubles the amount of variation that can be tolerated. Inaccuracies in the uncertain plant model can thus negatively impact the performance of the stable learning algorithm presented in the previous chapter by allowing less variation in the weights than can be safely tolerated by the actual system. Another feature of this example is that the lower bound constraint is active for all of the positive variation bounds. As discussed in Chapter 4 this is an artifact of the LMI approximation approach and a price that is paid for its relatively low computational cost.

6.2.3 Reinforcement Learning for Adaptive Control

In the previous chapter, the weights of an RNN were adapted using a supervised approach that trained the RNN to reproduce a given temporal sequence. For adaptive control an alternative approach must be taken since the desired output of the RNN is not explicitly known. In the experiments that follow the RNN weights are adapted using a reinforcement learning approach that is described in this section. Reinforcement learning is an unsupervised learning approach characterized by its focus on learning through interaction. When applied to control problems, reinforcement learning can be viewed as a class of direct, adaptive, optimal control algorithms [85, 78].

A reinforcement learning formulation of the reference tracking problem considered in this chapter can be formulated following [22]. Since the algorithms are generally applied in discrete time to sampled data, the following presentation uses a discrete time representation of certain parts of the problem. Given a deterministic, dynamical system

$$\dot{x} = f(x, u)$$

where $x \in \mathbb{R}^n$ is the system state and $u \in \mathbb{R}^m$ is the control input, find a control law, or *policy*, $\mu(x) : \mathbb{R}^n \to \mathbb{R}^m$, mapping x to u that minimizes

$$V^{\mu}(x(t)) = \int_t^{\infty} e^{-\frac{s-t}{\tau}} c(x(s), u(s))\ ds, \quad u(t) = \mu(x(t)). \tag{6.3}$$

The function $c(x, u) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ describes the cost of being in a particular system state and taking a particular control action. For example, in the reference tracking problem, a simple cost function is $c(x(t), u(t)) = \|r(t) - x(t)\|$ where $r(t)$ is the current reference signal. The design of the cost function is one of the most important parts of the problem specification since it determines the behavior of the optimal solution [55]. The parameter, τ, is a discount factor that determines the time horizon over which the algorithm attempts to minimize the cost. For small values of τ the optimal policy will be rather myopic; only the near term effects of actions are considered. As τ increases the optimal control policy considers more of the long term impact of actions. Selecting an appropriate value for τ is another important design decision in the construction of the reinforcement learning problem, but it is not always straightforward [55].

The function $V^{\mu}(x)$ is called a value function and captures the long term cost of following a particular policy starting from a state x. When working with sampled data the value function is written

$$V^{\mu}(x(t_0)) = \sum_{i=0}^{\infty} e^{-\frac{t_i - t_0}{\tau}} c(x(t_i), u(t_i)), \quad u(t_i) = \mu(x(t_i)),$$

where the t_i's are the sample times and t_0 is just the point of reference in time for some given sample. If the sample time is a constant, Δt, let $\gamma = 1 - \frac{\Delta t}{\tau}$ and write

$$V^{\mu}(x(t_0)) = \sum_{i=0}^{\infty} \gamma^i c(x(t_i), u(t_i)), \quad u(t_i) = \mu(x(t_i)).$$

Another value function, often called a Q-function or state-action value function, is defined as

$$\begin{aligned} Q^{\mu}(x(t_0), u(t_0)) &= c(x(t_0), u(t_0)) + \sum_{i=1}^{\infty} \gamma^i c(x(t_i), \mu(x(t_i))) \\ &= c(x(t_0), u(t_0)) + \gamma V^{\mu}(x(t_1)) \end{aligned} \tag{6.4}$$

and captures the cost of taking a particular action in a given state and from then on following the policy μ.

These value functions satisfy several useful relations that lead to a wide variety of algorithms for finding — or more often, approximating — μ. The *optimal* value functions are evaluated at the optimal policy, which is denoted μ^*. They are defined by

$$V^{\mu^*}(x(t_0)) = V^*(x(t_0)) = \min_{\mu} V^{\mu}(x(t_0))$$
$$Q^{\mu^*}(x(t_0), u(t_0)) = Q^*(x(t_0), u(t_0)) = \min_{\mu} Q^{\mu}(x(t_0), u(t_0)).$$

The minimization over policies requires an ordering relation which is defined by

$$V^{\mu_1}(x) \leq V^{\mu_2}(x) \ \forall \ x \in \mathbb{R}^n \Rightarrow \mu_1 \leq \mu_2.$$

The optimal value functions satisfy Bellman's equation, or in continuous time the Hamilton-Jacobi-Bellman equations. Bellman's equation is a recurrence given by

$$V^*(x(t_0)) = \min_u Q^{\mu^*}(x(t_0), u) = \min_u \ c(x(t_0), u) + \gamma V^*(x(t_1))$$

and

$$Q^*(x(t_0), u(t_0)) = c(x(t_0), u(t_0)) + \gamma \min_u Q^*(x(t_1), u).$$

If exact representations of V or Q are available for every policy, then the reinforcement learning problem, finding the optimal policy, can be solved using policy iteration. Policy iteration is a simple algorithm that iterates between two steps: policy evaluation and policy improvement. In policy evaluation, a value function for the current policy is determined. In the policy improvement step, the policy is made greedy with respect to the current value function. In other words

$$\mu(x) \leftarrow \operatorname*{argmin}_u Q^{\mu}(x, u), \ \forall \ x.$$

If only V is available, and not Q, then a model of the system dynamics is necessary to perform policy improvement since the results of taking a particular action at a given state must be known. Also, an explicit representation of μ is not strictly necessary since it can be computed implicitly from Q or V.

Obviously, when x and u are in continuous spaces, exact representation of the value functions is not generally possible. Exact evaluation of the policy improvement step is even less tractable. To proceed, it is necessary to find computationally tractable representations of one of the value functions. Parametric representations of V and Q are commonly used, but non-parametric representations have also been explored [63]. Parametric representations of V and Q are denoted $\tilde{V}_\theta$ and $\tilde{Q}_\theta$ where θ is the current set of parameters, for example, a set of neural network weights. Many algorithms for approximating the value function for a given policy are based on minimizing the temporal difference error of the approximation. A temporal difference error is an error of the form

$$\delta_{td} = c(x(t_0), u(t_0)) + \gamma \max_u \tilde{Q}_\theta(x(t_1), u) - \tilde{Q}_\theta(x(t_0), u(t_0))$$

If the parametric architecture is capable of representing the policy's value function exactly then at the optimal θ the architecture will produce no temporal difference errors. A general procedure for producing an approximation of a value function is to modify the parameters of the representation to minimize the temporal difference errors observed over some sample trajectories generated by following the policy. When an exact representation of the value function is not possible minimizing

the temporal difference errors of an approximation can still produce a value function that is *close* to true value function in some appropriate sense [60, 76]. The convergence of such temporal difference learning schemes is not guaranteed and has been explored extensively [71, 86, 96, 7, 88]. Given an approximate representation of one of the value functions, an approximate form of policy improvement can be performed. When these approximations are involved, convergence of the policy iteration algorithm is not necessarily assured [68, 54, 60]. Despite the lack of formal guarantees of convergence great success has been reported with temporal difference approaches and approximate policy iteration.

To use a reinforcement learning approach within the context of the proposed stable learning algorithm, an explicit representation of the policy is needed. When an explicit representation of the current policy is used in the policy iteration framework, the resulting algorithms are called actor-critic algorithms. In actor-critic methods, policy evaluation and policy improvement correspond to updating the *critic*, a Q function representation, and the actor, μ, respectively. Since the Q function and policy can not be represented exactly, function approximators are used in both cases [48]. The parameters of the critic are denoted θ_c, and the parameters of the actor are denoted θ_a. Stochastic gradient descent on temporal difference errors is used to update the parameters of the critic. The parameters of the actor are updated using the derivative of the critic, $Q(x, u)$, with respect to the control input. This update directs the actor to choose actions that minimize the Q-value for a particular state. In actor-critic methods policy evaluation and policy iteration are often interleaved. That is the actor is updated from the critic before the critic has converged. The dynamics of the learning system are quite complex in this case and care is needed to ensure reasonable behavior [55].

The specific details of the reinforcement learning algorithm used in Section 6.3 are now described. Two of the basic assumptions of the reinforcement learning approach are that the state of the environment is fully observable and that the dynamics satisfy the Markov property. The Markov property requires that the observed effect of a control action at time t is a function of only of the state at the current time and does not have any dependence on the past state trajectory. The two assumptions of state observability and Markov dynamics are closely related. In the control problem under consideration, the state of the plant is not fully observable. Because of this, the observed dynamics do not satisfy the Markov property. One approach to solving this type of problem requires modeling the control problem as a type of partially observable Markov decision process or POMDP. Another approach requires that a representation of the system that satisfies the Markov property be developed. Recurrent neural networks are useful for this purpose because of their ability to model temporal sequences and model hidden dynamics [13]. In this approach a recurrent neural network is used to model the value function. The internal dynamics of the network are used to construct, implicitly, a model of the system satisfying the Markov property. Because the focus of this chapter is on the proposed stable learning algorithm and not reinforcement learning *per se*, an unrealistic approach is used here to simplify the dynamics of the learning system. A standard feedforward neural network is used to model the value function and is given the full state of the plant as part of its input. The other inputs are the tracking error and the control signal. The output of the critic network is the value of the Q-function at the given state and control inputs. The network has 50 hidden nodes and uses the sigmoid nonlinearity, $\sigma(x) = \frac{1}{1+e^{-x}}$. A constant bias input is provided to all of the neurons. The network equations are

$$Q(x, u) = W^o \left[\sigma \left(W^i \begin{bmatrix} x & u & 1 \end{bmatrix}^T \right)^T 1 \right]^T$$

The critic is trained by standard backpropagation of the temporal difference errors using the equa-

tions

$$W^o \leftarrow W^o - \eta_o \frac{\partial \delta_{td}}{\partial W^o}$$
$$W^i \leftarrow W^i - \eta_i \frac{\partial \delta_{td}}{\partial W^i}.$$

The learning rates, η_o and η_i, have the values, 0.001 and 0.01, respectively, throughout the experiments.

The actor is a recurrent neural network of the design described in Section 6.2.1. The RNN used for the actor has three states, thus $W \in \mathbb{R}^{3\times 3}$. Updates to the actor weights are made using stochastic gradient descent on the gradient of the Q-function with respect to the control inputs,

$$W \leftarrow W - \eta_a \frac{\partial Q(x,u)}{\partial u} \frac{\partial u}{\partial W}$$

The update can be computed by using $-\frac{\partial Q(x,u)}{\partial u}$ as the error function in the RTRL algorithm. The learning rate is varied for some experiments, but as a default $\eta = 0.001$. The updates to the actor weights proposed by this algorithm are the update monitored by the stable learning algorithm to ensure stability. The updates to the critic need not be considered in the stability analysis because the critic is not directly part of the control loop.

A few other specifics of the actor-critic method should be discussed. In order to find the optimal policy, or even a good policy for that matter, the critic requires observations of all of the actions in all of the states. In continuous spaces the condition is slightly different, essentially the critic needs to experience a large variety of state action combinations. This is generally referred to as exploration. Often reasonable exploration can be achieved by causing the actor to deviate slightly from its preferred output. In the simulations that follow random perturbations are occasionally added to the output of the controller. These random perturbation are generally small, and can be adequately addressed in the stability analysis by the input-multiplicative uncertainty that already exists in the plant model. For the actor-critic model to converge to a reasonable solution the amount of noise injected into the actions should decrease over time. In the experiments below the noisy actions are taken with a probability that depends on time and is given by the schedule

$$\begin{cases} (0.09, 2.0) & : 0 \leq t \leq 500 \\ (0.08, 1.6) & : 500 \leq t \leq 1000 \\ (0.05, 1.0) & : 1000 \leq t \leq 4000 \\ (0.01, 0.2) & : 4000 \leq t \leq 8000 \end{cases}.$$

The first value of the pair is the probability of corrupting the action and the second value is the magnitude of the random perturbation added to the actor output.

6.3 Experimental Evaluation

In this section the stable learning algorithm developed in Chapter 5 is applied to the control of the multiple spring-mass-damper system using the actor-critic method discussed in the previous section. The experiments are designed to illustrate the properties of the stable learning algorithm more than to simulate the actual application of these techniques in practice. For instance, in practice, a robust controller would be designed for the plant and the actor-critic model would simply modify the output of the robust controller in some way. This was done in [2] for example.

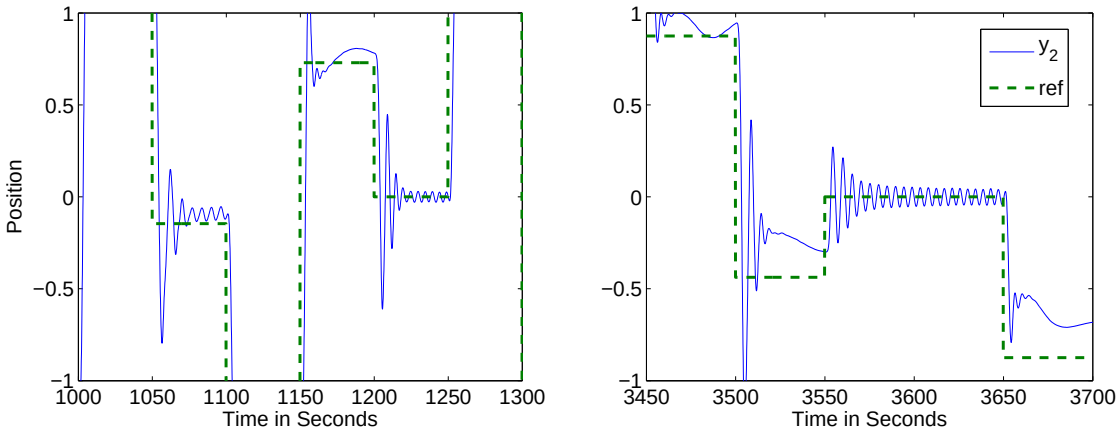

Figure 6.4: An example of unstable behavior exhibited during the training of the actor-critic system.

The combination of robust controller and actor-critic system allows a certain amount of performance to be guaranteed at the deployment of the system. In the experiments that follow less attention is payed to the quality of the learned control law than to the stability properties of the adaption.

The actor and critic models were initialized by simulating them for 5000 seconds on the nominal, linear, plant model described in Section 6.1.2. The simulation was done without regard for stability since the actor and critic adaption was performed on a model and not the real system. Initializing the actor and critic in this way allowed them to be hooked into the real system with some *a priori* knowledge. All of the experiments that follow begin with these initialized actor and critic parameters.

6.3.1 Actor-Critic Learning without Stability Analysis

Beginning with the actor and critic trained on the nominal, linear plant model, the control system was simulated for 1000 seconds on the actual plant without any constraints on the stability of the system. At a sampling rate of 10Hz, the actor and critic weights were updated 10000 times. In Figure 6.4 a portion of the recorded system behavior is shown. Clearly, the system exhibits instability for reference signals near zero.

In Figure 6.5 the results of simulating the actor with the weights from the 3600 second mark and a reference input of $r = 0.2$ are shown. The actor weight matrix has the values

$$W = \begin{bmatrix} 1.5808 & -0.2169 & -0.5501 \\ 4.1411 & 0.1407 & 0.5892 \\ 2.9354 & 0.8355 & -0.1111 \end{bmatrix}.$$

The closed loop system can not be proved stable by the IQC analysis developed in Section 6.2.2. This type of instability should be avoided during the training of the actor and critic. The algorithm from Chapter 5 is applied in the next section to prevent this type of behavior.

6.3.2 Stable Actor-Critic Learning

To avoid the instability seen in the previous example the stable learning algorithm from Chapter 5 is used to filter the actor updates. The stability bias is used with two slight modifications. First, to make better use of the available information, updates that are rejected by the original algorithm are

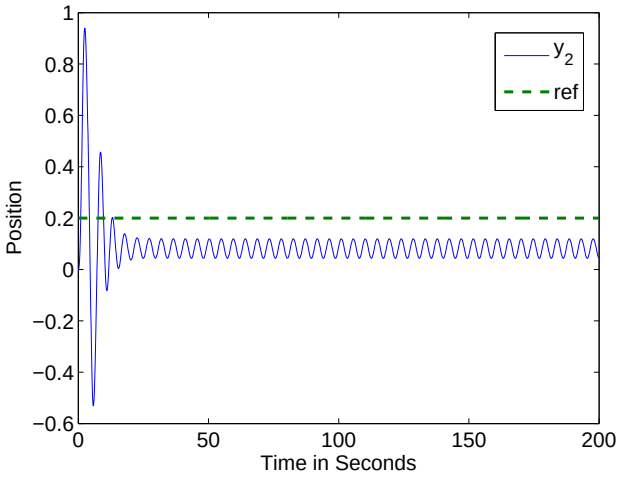

Figure 6.5: An example of an unstable RNN controller visited by the actor-critic learning algorithm during training. The controller is unstable for small reference inputs.

instead rescaled such that they satisfy the current constraint set. This allows progress to be made at every step. Second, rather than compute the stability bias at every step, the bias computation is turned on and off based on whether or not the previous steps were accepted without modification. Initially, the stability bias is not used, but after a sequence of three updates in a row have been rescaled due to violation of the constraints the stability bias computation is turned on. The bias computation remains on until a sequence of five steps in a row are accepted without scaling.

The initial actor weights, generated by training on the linear plant, can not be proved stable. To initialize the stable learning algorithm the weights are scaled down to satisfy the stability constraints. The resulting initial network weights produced a gain of $\gamma_{cl} = 0.600$. The initial variation bounds were

$$\overline{\Delta} = \begin{bmatrix} 0.1000 & 0.1000 & 0.3313 \\ 0.1013 & 0.1000 & 0.2957 \\ 0.1000 & 0.1183 & 0.1095 \end{bmatrix} \text{ and } \underline{\Delta} = \begin{bmatrix} 0.3474 & 0.3970 & 0.1141 \\ 0.1775 & 0.3067 & 0.1497 \\ 0.2674 & 0.1949 & 0.3359 \end{bmatrix}.$$

A minimum value of 0.1 was enforced on the variation bounds, but like in Chapter 5 this was allowed to decrease if a set of bounds could not be found that satisfied this constraint.

The system was simulated for 1100 seconds and the following results were observed. Unlike the previous example, no instability was observed in the controlled system during the adaptation. Out of the 11,000 updates generated by the actor-critic algorithm, 2,541 — roughly one in four — required rescaling to satisfy the stability constraints. The stability bias was computed 4200 times, or on almost half of the steps. The stability bounds were recomputed 6540 times. The mean variation allowed per weight over these constraint sets was 0.028. Because of repeated failures to find bounds that satisfied the minimum constraint, the lower bound was decreased repeatedly. The amount of variation allowed in the weights is, on average, very little. This causes the number of constraint computations to rise and increases the cost of the algorithm drastically. The gains computed in the stability bias computations remained relatively small, $\gamma < 5$, during the simulation. This suggests that the small amounts of allowable variation are not due to closeness of the controller to the boundary of the stable weight set, $\mathcal{W}_{ss}^n$. It appears that conservativeness in the analysis of the time-varying RNN is hindering the application of the stable learning algorithm. The algorithm

succeeds in keeping the control system stable, but has an excessive cost. In the next section, the stability constraints are loosened somewhat and a modified version of the algorithm is applied.

6.3.3 Step-wise Stable Actor-Critic Learning

Because the conservativeness in the analysis of the time-varying RNN limits the amount of variation that can be tolerated under the stability constraints, it seems worthwhile to consider what benefit might be had from accepting weaker stability guarantees. Rather than constraining the variation in the weights using the analysis developed in Chapter 4, the weights of the actor are simply constrained to remain in $\mathcal{W}_{ss}^n$ at all times using the analysis of Chapter 3. Weight updates that push the weights out of $\mathcal{W}_{ss}^n$ are rejected. This guarantees that stopping the adaptation at any point will always result in a stable controller. This type of stability guarantee has been called step-wise stability [55]. The weaker stability algorithm does not protect against the type of instability due to switching problems like the one described in Chapter 4. Such problems might occur due to cycles in the weight settings cause by a non-decaying step size in the actor updates. This type of problem was never encountered during simulation, however.

Starting from the scaled actor used in the previous section, this new algorithm was simulated for 5,000 seconds on the actual plant. Of the 50,000 updates generated, only 10,364 were accepted. The updates were rejected when the weights approached the boundary between provably stable and possibly unstable weight matrices. The behavior during the last 1000 seconds is shown in Figure 6.6. No instability was seen over the entire 5000 second history. The weight trajectories over the entire adaptation period are also shown.

6.3.4 Actor-Critic Learning with a Stability Bias

Only one in four updates to the actor was accepted using the simple step-wise stability algorithm. In the last chapter the addition of a stability bias to the weight updates was shown to increase the number of updates accepted by the stable learning algorithm. To test whether this result carries over to the modified step-wise stability algorithm, the previous experiment is repeated with the addition of a stability bias term to the weight updates. The weight update in (5.12) was applied over two 5000 second adaption trials. The weight update is parameterized by $\hat{\gamma}$ which determines at what $\mathcal{L}_2$-gain the bias begins to have a major effect on the updates. The first trial used $\hat{\gamma} = 5$ and the second used $\hat{\gamma} = 50$. The weighting parameters, η_2 followed the decay schedule $\eta_2 = \frac{8000-t}{8000}$. Figure 6.7 shows the last 1000 seconds of the episode and the weight trajectories for $\hat{\gamma} = 5$. Figure 6.8 shows the same data, but from the trial with $\hat{\gamma} = 50$. Several observations can be made about the data. Compared to the previous experiment where no stability bias was used, both trials of this experiment perform better at tracking the reference signal. Also, the weights grow larger in these two trials. The lack of stability bias in the previous experiment caused the actor to suffer from a lack of progress due to discarded updates.

In the case of $\hat{\gamma} = 5$ all 50000 updates were accepted. When $\hat{\gamma} = 50$ nearly 82% of the updates were accepted. The larger value of $\hat{\gamma}$ in this trial allowed the network weights to approach the stability boundary more closely. This increases the likelihood that weight updates will be rejected. The dynamics of the plant output in the low $\hat{\gamma}$ experiment are smoother than those of the high $\hat{\gamma}$ trial. This is due, again, to the soft bound on the network gain that results from the update functions dependence on γ and $\hat{\gamma}$.

The performance of the two actor-critic systems was compared by running the systems for an additional 5000 seconds and comparing the mean squared tracking error and mean squared control output over the 5000 second window. The trial with $\hat{\gamma} = 5$ had a mean squared tracking error of

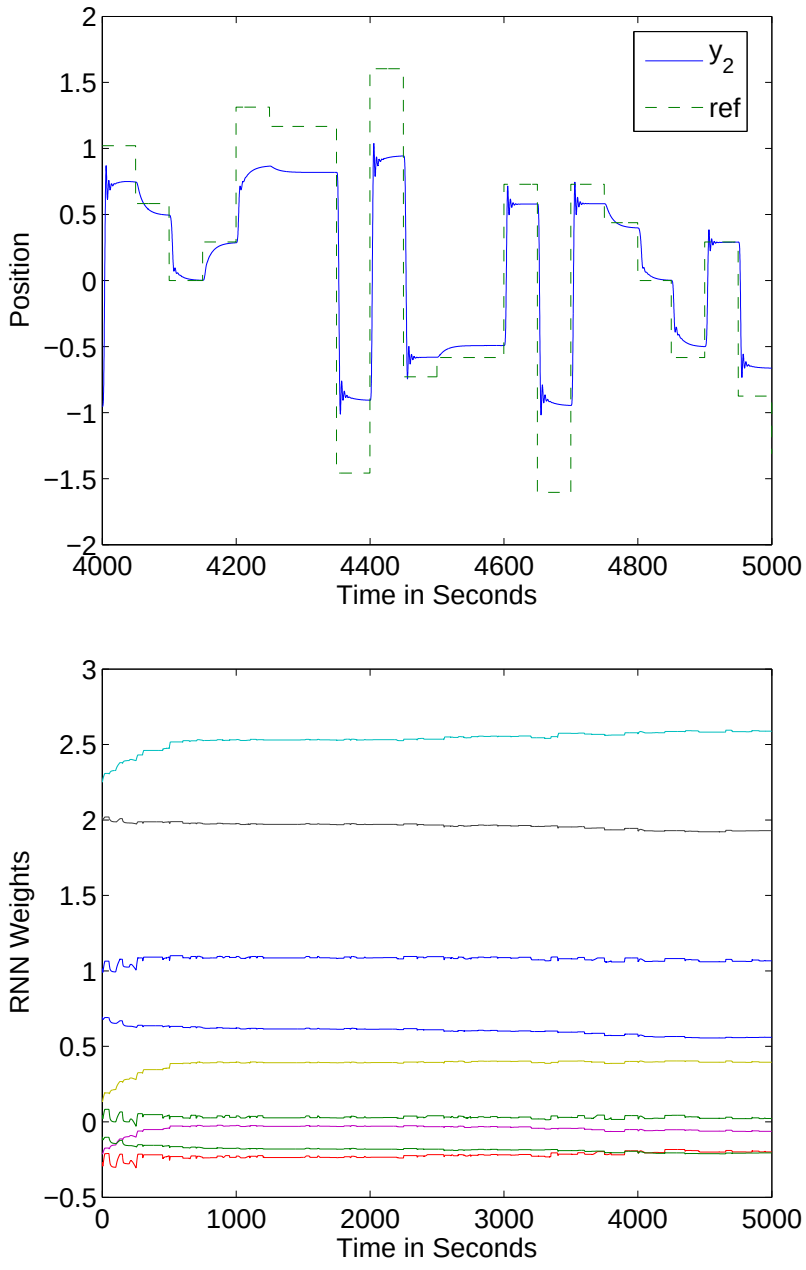

Figure 6.6: Example of the behavior and weight trajectories of the step-wise stable adaption.

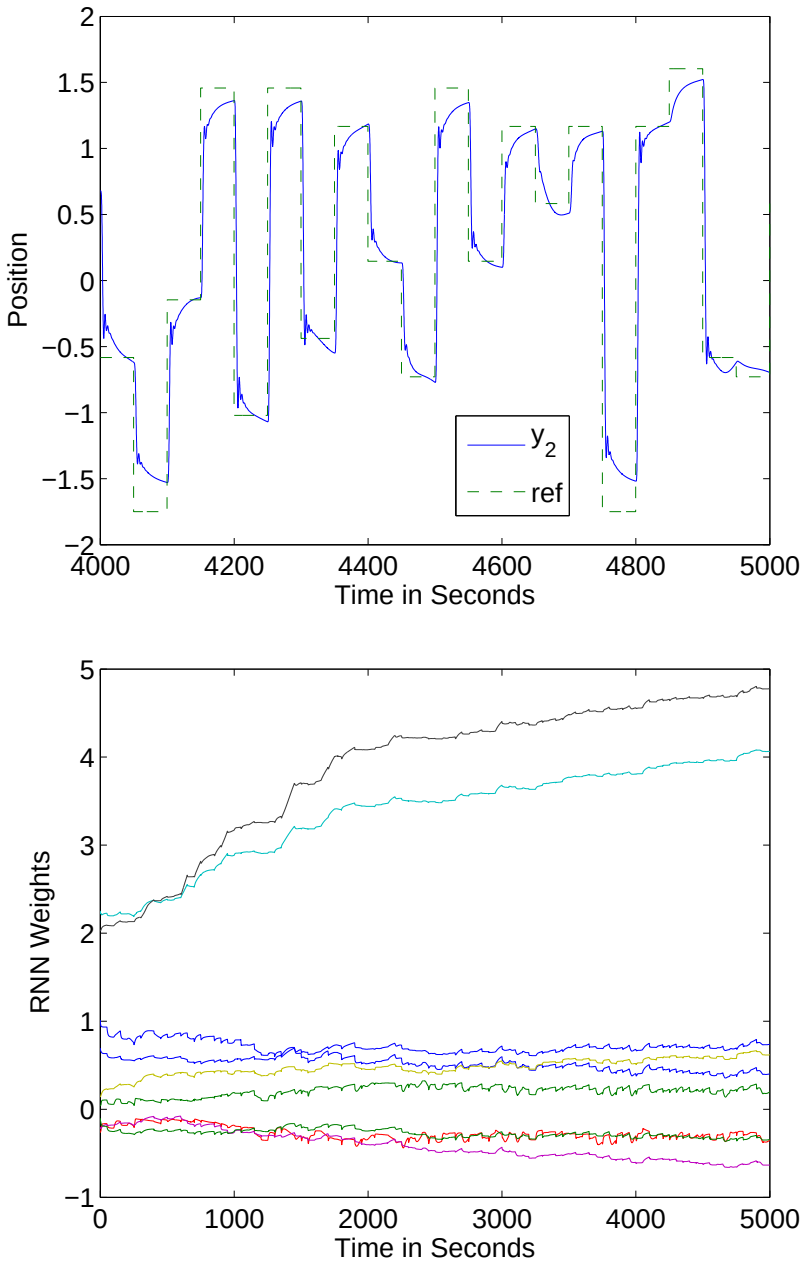

Figure 6.7: Example of the behavior and weight trajectories of the step-wise stable adaption and a stability bias.

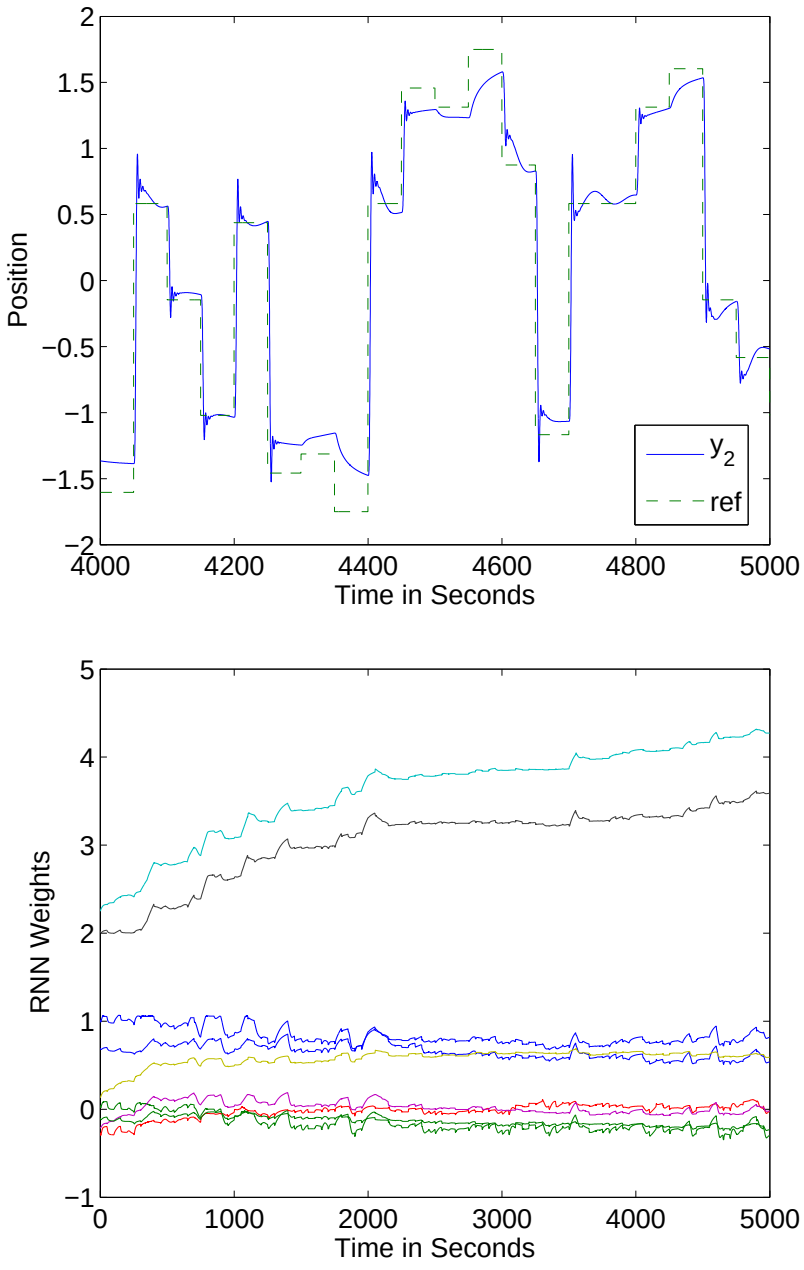

Figure 6.8: Example of the behavior and weight trajectories of the step-wise stable adaption and a stability bias.

	MSTE	MSC	γ
Init	0.3493	6.3065	13.623
NS	0.1084	24.9356	∞
SS	0.2478	8.9198	643.933
SSB	0.1004	17.1258	3.999

Table 6.1: After 5000 seconds of training the learned controllers were tested on a fixed sequence of reference changes. Three cases were compared: no stability analysis (NS), step-wise stability (SS), step-wise stability with the stability bias (SSB).

0.1858 and a mean squared control output of 23.1557. For the $\hat{\gamma} = 50$ case these values were 0.2050 and 26.1874 respectively. Neither of the controllers perform extremely well, but the system with the lower $\hat{\gamma}$ value performs slightly better. This may be due to the fact that the smaller gain constraint results in smoother dynamics which may in turn help to regularize the learning dynamics.

To complete the analysis, the controllers learned after the 5000 step trials of this and the previous section were compared on a fixed set of reference changes over 1000 steps. No adaptation was performed during these test trials. The mean-squared tracking error and mean-squared control action were recorded. These results are reported in Table 6.1 along with the gain of the controllers. The results reported for the stability bias trial are for $\hat{\gamma} = 5.0$. The results show that the step-wise stability approach with the stability bias has the best overall mean-squared tracking error and a lower mean-squared control output than the unconstrained case. The gain of the controller in the step-wise stability constrained trial where no stability bias was used is very large. This is further evidence of the controller parameters getting stuck near the stability boundary.

6.4 Conclusions

Application of the stable learning algorithm from the previous chapter proved to be to restrictive and computationally expensive for application to the multiple spring mass damper system. By relaxing the stability constraints to consider a weaker *step-wise* stability, the computational cost was considerably reduced and better performance was achieved. The stability bias developed in Chapter 5 improved the ability of the actor-critic system to learn under the step-wise stability constraints. Instability was observed in an actor-critic control system when no stability constraints were explicitly enforced. Application of even the relaxed step-wise stability constraint kept the actor from instability.

Chapter 7

Conclusions

> *The problem of stability is vastly more difficult when artificial neural networks are used either for identification or control and the system is nonlinear. Unlike linear systems, simple algebraic conditions are not available for assuring the stability of the overall system. ... All these indicate that our knowledge of the stability of dynamical adaptive systems using artificial neural networks is quite rudimentary at the present time and that considerable work remains to be done. ... It is precisely in problems where the system has to adapt to large uncertainty that controllers based on neural networks will be needed in practical applications. For such problems, new concepts and methods based on stability theory will have to be explored.*
>
> Adaptive Control Using Neural Networks [61], K.S. Narendra

The combination of reinforcement learning and recurrent neural networks provides a powerful architecture for automatic control. On the other hand, both the nonlinearity and the adaptation make the stability analysis of such control systems challenging. Almost twenty years after [61] was published, the quote from Narendra remains true. In this work, several steps toward a practical, adaptive, neural control system have been reported.

State-of-the-art analysis of the stability properties of neural networks involves the construction and solution of linear matrix inequality problems. These conditions are computationally expensive and generally not exact. One contribution of this work is a reduction in the conservativeness of the existing stability analysis for recurrent neural networks. For time-invariant networks, this reduction comes from the use of better IQCs for the description of the network's nonlinearity. For time-varying networks, this reduction is achieved by a novel formulation of the time-varying network equations. Reducing the conservativeness in the analysis allows a larger class of systems to be proved stable. In addition to the reduction in conservativeness, some reductions in computational complexity were achieved by analysis of the LMI problems. Some of the constraints on the decision variables in these problems were shown to be unnecessary. Finally, it was shown that the augmented Lagrangian, or penalty-barrier, method is much more efficient than standard interior point methods for the LMI problems of interest.

The algorithm presented in Chapter 5 for maintaining the stability of adaptive, recurrent neural networks has its roots in the work of Kretchmar in [49]. One of the main contributions of this work is the development of techniques for reducing the computational cost of the algorithm. Two developments allowed the reduction in computational cost. The bounds on the allowable weight variation were solved for directly, rather than through a bisection approach as in [49]. The cost

of this step was thus reduced to that of solving a single LMI, or possibly BMI, problem. More importantly, a stability bias was introduced in Chapter 5 that biased the weight trajectories of an adapting network away from the boundary of the stable weight set. Use of this bias reduces the number of times the variation bounds must be computed. Additionally, the stability bias improves the behavior of the constrained reinforcement learning algorithm even when only step-wise stability is enforced. A reasonable computational cost for the stability bias was achieved by applying a novel approach to warm-starting the augmented Lagrangian SDP solver.

7.1 Summary

In Chapter 3, a stability analysis was derived for continuous time recurrent neural networks from the theory of integral quadratic constraints. The resulting optimization problems were solved efficiently using a recently developed approach [47] to semidefinite programming that avoids the explicit formulation and solution of Newton equations.

In Chapter 4 the conservativeness of an existing analysis from [79] of recurrent neural networks with time-varying weights was reduced. A new formulation of the time-varying RNN equations was developed which improved the accuracy of the stability analysis and allowed more of the power of certain IQCs to be applied to the problem. Additionally, the problem of finding maximal bounds on the variation of an RNN's weights under which stability can be assured was addressed. In [79] this problem was addressed using an LMI approximation of an underlying BMI problem. In Chapter 4 the BMI problem was solved directly and shown to produce qualitatively better results. The LMI approximations seems to suffer from some deficiency which causes it to set many of the variation bounds to zero. The BMI solutions did not exhibit this pathology.

In Chapter 5 an algorithm for maintaining the stability of adaptive, recurrent neural networks was developed. By restricting the variations in an RNN's weights to satisfy the type of variation bounds computed in Chapter 4, stability of the evolving network can be assured. A basic problem with this approach is that, in general, the learning algorithm in charge of adapting the weights of the network has no knowledge of the stability constraints. This ignorance can lead to situations where the weight trajectory evolves along the border of the set of stable weight matrices. When this occurs the stable learning algorithm must repeatedly compute new sets of constraints on the weight variations. Because this computation is expensive, its occurrence should be minimized. Knowledge about the stability properties of the network can be included in the weight updates through the use of a stability bias. The stability bias is the gradient of an $\mathcal{L}_2$-gain bounding function with respect to the weights of the network. It points to the interior of the set of stable weight matrices, and its magnitude grows proportionally to the closeness to the estimated stability boundary.

A weighting rule for the stability bias was devised that limits the impact of the bias when the weights are far from the boundary of the stable weight set. More importantly, it was shown how the cost of computing the stability bias could be drastically reduced by applying warm-start methods to the necessary semidefinite programs. Experimental analysis revealed that augmented Lagrangian methods, such as [47], for solving SDPs are capable of capitalizing on available warm-start information. If a sequence of slightly perturbed problems must be solved — this is the case for computing the stability bias — initializing the SDP solver using the previous problem solutions reduced the cost of solving the perturbed problem. A perturbation analysis of linear SDPs given in [27] was applied to improve the quality of the warm start information. The improvement, however, comes at the cost of solving a linear program and may not provide an overall benefit in terms of run time.

The proposed stable learning algorithm and stability bias were applied, in Chapter 6, to the

problem of robust, adaptive, reinforcement learning control of an uncertain, nonlinear, multiple spring-mass-damper system. The stable learning algorithm was extended to ensure stability of the closed loop between the RNN controller and nonlinear plant, even in the presence of uncertainty. Unfortunately, conservatism in the stability analysis makes application of the algorithm computationally expensive. A modified algorithm was proposed that ensures a weaker *step-wise* stability of the learning system. The weakening of the stability constraint allowed the computational complexity of the algorithm to be reduced. Even under the weaker stability conditions the algorithm was shown to prevent instability during the adaptation of the RNN controller. Use of the stability bias further improved the behavior of the algorithm and allowed the actor-critic system to improve the performance of the controller in the presence of stability constraints.

7.2 Future Work

The algorithm presented in Chapters 5 and 6 is a very general purpose tool for stable, adaptive, neural control in the presence of uncertainty in the plant behavior. If an uncertainty model can be constructed for a given plant in terms of IQCs the algorithm can be applied to ensure stability of the closed loop control system. The adaptation of the RNN controller can be performed by any algorithm since the stable learning algorithm is a general purpose filter for weight updates. The combination of the stable learning algorithm with reinforcement learning is particularly interesting, however, as it provides an approach to solving difficult control problems with guarantees of stability. The reinforcement learning algorithm applied in Chapter 6 was rather basic and made inefficient use of the available data. The resulting control performance, was therefore not extremely good. The stable learning algorithm can be applied as-is to more advance reinforcement learning designs such as dual heuristic programming approaches [78]. The improved learning efficiency of these algorithms combined with the explicit stability guarantees of the stable learning algorithm could make a powerful general purpose tool for adaptive control.

Two major barriers to the practical application of the proposed algorithm point the way toward future research. The first barrier is the computation expense of the algorithm. Even with the improvements made by application of warm-start methods, it is unlikely that the solution of the stability LMI problem could be performed in real-time. One possibility, however, might be to adapt the work in [40] that describes an analog neural circuit for real-time semidefinite programming. Other possibilities include heuristic approaches that limit the computation of the stability bias to certain time samples or based on the time constraints of the system. Even if such advances were made, application to very large recurrent neural networks will probably remain elusive. The high cost of the using the KYP lemma in systems with many states requires more analysis in terms of the specific structure of the LMI related to an RNN's stability. As in [89] it might be possible to take advantage of some particular feature of the LMIs to decrease the size of the resulting problems.

The second barrier to practical application of the algorithm is the remaining conservativeness in the IQC analysis of the recurrent neural network. As mentioned in Chapter 6, the analysis ignores the boundedness of the nonlinearity. Following up on the recent work in [35] could prove extremely fruitful. The analysis in [35] explicitly account for boundedness in the modeling of nonlinear operators by covering the nonlinear system with a set of switched, saturated, linear systems for which a tractable analysis can be performed. Because the boundedness of $\phi(\cdot)$ is an essential feature and differentiates the dynamics of the RNN from a linear network with similar weights accounting for it in the analysis could provide a drastic reduction in conservatism. Additional conservatism enters the analysis throughout the consideration of the time varying weights. An alternative approach that limits the rate at which the controller switches between different sets of

weights might allow the strong dynamic stability constraint to be recovered in what is currently the algorithm that ensures step-wise stability of the system. Some relevant work can be found in [62] where a similar idea is applied to linear, adaptive systems. Also, the vast literature on switching systems, for example [77], might also provide insight into this problem.

REFERENCES

[1] B.D.O. Anderson, M. Mansour, and F.J. Kraus. A new test for strict positive realness. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 42(4):226–229, 1995.

[2] C.W. Anderson, P.M. Young, M. Buehner, J.N. Knight, K.A. Bush, and D.C. Hittle. Robust reinforcement learning control using integral quadratic constraints for recurrent neural networks. *IEEE Transactions on Neural Networks*, 18(4):993–1002, 2006.

[3] A.F. Atiya and A.G. Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709, 2000.

[4] G. Balas, R. Chiang, A. Packard, and M.G. Safonov. *Robust Control Toolbox 3, User's Guide*, 2008.

[5] N.E. Barabanov and D.V. Prokhorov. Stability analysis of discrete-time recurrent neural networks. *IEEE Transactions on Neural Networks*, 13(2):292–303, 2002.

[6] R. D. Beer. Parameter space structure of continuous-time recurrent neural networks. *Neural Computation*, 18(12):3009–3051, 2006.

[7] D.P. Bertsekas, V.S. Borkar, and A. Nedic. Improved temporal difference methods with linear function approximation. In J. Si, A.G. Barto, W.B. Powell, and D. Wunsch, editors, *Handbook of Learning and Approximate Dynamic Programming*. Wiley-IEEE Press, 2004.

[8] R. Bhatia. *Positive Definite Matrices*. Princeton University Press, 2007.

[9] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994.

[10] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[11] R.W. Brockett and J.L. Willems. Frequency domain stability criteria: Part i and ii. *IEEE Transactions on Automatic Control*, 10:255–261 and 401–413, 1965.

[12] M. Buenher and P. Young. A tighter bound on the echo state property. *IEEE Transactions on Neural Networks*, 17(3):820–824, 2006.

[13] K. Bush. *An Echo State Model of Non-Markovian Reinforcement Learning*. PhD thesis, Colorado State University, 2008.

[14] J. Cao and J. Wang. Global asymptotic stability of a general class of recurrent neural networks with time-varying delays. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 50(1):34–44, 2003.

[15] R.S. Chandra and R. D'Andrea. A scaled small gain theorem with applications to spatially interconnected systems. *IEEE Transactions on Automatic Control*, 51(3):465–469, March 2006.

[16] T. Chu, C. Zhang, and Z. Zhang. Necessary and sufficient condition for the absolute stability of normal neural networks. *Neural Networks*, 16:1223–1227, 2003.

[17] Y.-C. Chu. *Control of Systems with Repeated Scalar Nonlinearities*. PhD thesis, University of Cambridge, 1998.

[18] Y.-C. Chu. Bounds on the incremental gain for discrete time recurrent neural networks. *IEEE Transactions on Neural Networks*, 13(5):1087–1098, 2002.

[19] F.J. D'Amato, M.A. Rotea, A.V. Megretski, and U.T. Jönsson. New results for analysis of systems with repeated nonlinearities. *Automatica*, 37:739–747, 2001.

[20] C.A. Desoer and M. Vidyasagar. *Feedback Systems: Input-Output Properties*. Academic Press, 1975.

[21] K. Doya. Bifurcations in the learning of recurrent neural networks. In *Proceedings of 1992 IEEE International Symposium on Circuits and Systems*, pages 2777–2780, 1992.

[22] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, January 2000.

[23] G.E. Dullerud and F. Paganini. *A Course in Robust Control Theory*. Springer, 2000.

[24] J.L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[25] B. Fares, D. Noll, and P. Apkarian. Robust control via sequential semidefinite programming. *SIAM J. Control Optim.*, 40(6):1791–1820, 2001.

[26] R. W. Freund, F. Jarre, and C. H. Vogelbusch. Nonlinear semidefinite programming: sensitivity, convergence, and an application in passive reduced-order modeling. *Math. Program.*, 109(2):581–611, 2007.

[27] R.W. Freund and F. Jarre. A sensitivity analysis and a convergence result for a sequential semidefinite programming method. Technical Report Numerical Analysis Manuscript 03-4-08, Bell Laboratories, 2003.

[28] R.W. Freund and F. Jarre. A sensitivity result for semidefinite programs. *Operations Research Letters*, 32:126–132, 2004.

[29] V. Fromion. Lipschitz continuous neural networks on $\mathcal{L}_p$. *Proceedings of the 39th IEEE Conference on Decision and Control*, 4:3528–3533, 2000.

[30] V. Fromion and M.G. Safonov. Popov-zames-falb multipliers and continuity of the input/output map. In *Proceedings of the IFAC Symposium on Nonlinear Control Systems (NOLCOS 2004)*, Stutgart, Germany, 2004.

[31] M. Fu, S. Dasgupta, and Y.C. Soh. Integral quadratic constraint approach vs. multiplier approach. *Automatica*, 41:281–287, 2005.

[32] K. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Netorks*, 6:801–806, 1993.

[33] L. El Ghaoui and S. Niculescu, editors. *Advances in Linear Matrix Inequality Methods in Control*. Advances in Design and Control. SIAM, 2000.

[34] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge, 1985.

[35] T. Hu, B. Huang, and Z. Lin. Absolute stability with a generalized sector condition. *IEEE Transactions on Automatic Control*, 49(4):535–548, April 2004.

[36] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical report, German National Research Institute for Computer Science, 2001.

[37] H. Jaeger, M. Lukośevićius, D. Popovici, and U. Seiwert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20:335–352, 2008.

[38] H. Jaeger, W. Maass, and J. Principe. Special issue on echo state networks and liquid state machines: Editorial. *Nerual Networks*, 20(3):287–289, 2007.

[39] F. Jarre. On an approximation of the hessian of the lagrangian. Technical report, Universität Düsseldorf, 2003.

[40] D. Jiang and J. Wang. A recurrent neural network for real-time semidefinite programming. *IEEE Transactions on Neural Networks*, 10(1):81–93, 1999.

[41] E. John and E.A. Yildrim. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension (in print). *Computational Optimization and Applications*, 2007.

[42] U. Jönsson. Lecture notes on integral quadratic constraints. Technical report, Department of Mathematics, Royal Institute of Technology, Stockholm Sweden, 2000.

[43] U. Jönsson, C.-Y. Kao, A. Megretski, and A. Rantzer. A guide to IQCβ: A matlab toolbox for robust stability analysis and performance analysis. Technical report, 2004.

[44] H. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[45] J.-J. Kim and T. Singh. Desensitized control of vibratory systems with friction: Linear programming approach. *Optimal Control Applications and Methods*, 25:165–180, 2004.

[46] M. Kocvara and M. Stingl. *High Performance Algorithms and Software for Nonlinear Optimization*, chapter PENNON- A Generalized Augmented Lagrangian Method for Semidefinite Programming, pages 297–315. Kluwer Academic Publishers, 2003.

[47] M. Kocvara and M. Stingl. On the solution of large-scale sdp problems by the barrier method using iterative solvers. *Mathematical Programming*, 109:413–444, 2007.

[48] V.R. Konda and J.N. Tsitsiklis. Actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003.

[49] R.M. Kretchmar. *A Synthesis of Reinforcement Learning and Robust Control Theory*. PhD thesis, Computer Science Department, Colorado State University, 2000.

[50] R.M. Kretchmar, P.M. Young, C.W. Anderson, D.C. Hittle, M.L. Anderson, C.C. Delnero, and J. Tu. Robust reinforcement learning control with static and dynamic stability. *International Journal of Robust and Nonlinear Control*, 11:1469–1500, 2001.

[51] K. Krishnan. *Linear Programming Approaches to Semidefinite Programming Problems*. PhD thesis, Rensselaer Polytechnic Institute, 2002.

[52] V.V. Kulkarni and M.G. Safonov. All multipliers for repeated nonlinearities. *IEEE Transactions on Automatic Control*, 47(7):1209–1212, 2002.

[53] V.V. Kulkarni and M.G. Safonov. Incremental positivity nonpreservation by stability multipliers. *IEEE Transactions on Automatic Control*, 41(1):173–177, 2002.

[54] M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, December 2003.

[55] G.G. Lendaris and J.C. Neidhoefer. Guidance in the use of adaptive critics for control. In J. Si, A.G. Barto, W.B. Powell, and D. Wunsch, editors, *Handbook of Learning and Approximate Dynamic Programming*, pages 97–124. Wiley-IEEE Press, 2004.

[56] A.I. Lur'e and V.N. Postnikov. On the theory of stability of control systems. *Applied Mathematics and Mechanics*, 8(3), 1944. In Russian.

[57] U. Mackenroth. *Robust Control Systems: Theory and Case Studies*. Springer-Verlag, 2004.

[58] C. Makkar, W.E. Dixon, W.G. Sawyer, and G. Hu. A new continuously differentiable friction model for control systems design. *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 600–605, 24-28 July 2005.

[59] A. Megretski and A. Rantzer. System analysis via integral quadratic constraints. *IEEE Transactions on Automatic Control*, 42(6):819–830, 1997.

[60] R. Munos. Error bounds for approximate policy iteration. In T. Fawcett and N. Mishra, editors, *Proceedings of the Twentieth International Conference on Machine Learning*, pages 560–567, Washington, D.C., USA, 2003. AAAI Press.

[61] K.S. Narendra. Adaptive control using neural networks. In W.T. Miller, R.S. Sutton, and P.J. Werbos, editors, *Neural Networks for Control*, pages 125–142. MIT Press, 1990.

[62] A. Ng and H.J. Kim. Stable adaptive control with online learning. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 977–984. MIT Press, 2005.

[63] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3):161–178, 2000.

[64] A. Packard. Gain scheduling via linear fractional transformations. *Systems and Control Letters*, 22(2):79–92, 1994.

[65] P.A. Parrilo. Outer approximation algorithms for kyp-based lmis. In *Proceedings of the American Control Conference*, volume 4, pages 3025–3028, 2001.

[66] B.A. Pealmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

[67] T.J. Perkins and A.G. Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3:803–832, 2002.

[68] T.J. Perkins and D. Precup. A convergent form of approximate policy iteration. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

[69] L. Perko. *Differential Equations and Dynamical Systems*. Springer, 2001.

[70] V.M. Popov. Absolute stability of nonlinear systems of automatic control. *Automation and Remote Control*, 21:961–979, 1961.

[71] D. Precup, R.S. Sutton, and S. Dasgupta. Off-policy temporal-difference learning with function approximation. In C.E. Brodley and A.P. Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 417–424, Williams College, Williamstown, MA, USA, 2001. Morgan Kaufmann.

[72] A. Rantzer. On the kalman-yakubovich-popov lemma. *Systems and Control Letters*, 28:7–10, 1996.

[73] A. Ruiz, D.H. Owens, and S. Townley. Existence of limit cycles in recurrent neural networks. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE)*, volume 1, pages 104–108, 1996.

[74] M.G. Safonov, K.C. Goh, and J.H. Ly. Control system synthesis via bilinear matrix inequalities. In *Proceedings of the American Control Conference*, volume 1, pages 45–49, 1994.

[75] U.D. Schiller. *Analysis and Comparison of Algorithms for Training Recurrent Nerual Networks*. PhD thesis, University of Bielefeld, 2003.

[76] R. Schoknecht. Optimality of reinforcement learning algorithms with linear function approximation. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

[77] R. Shorten, F. Wirth, O. Mason, K. Wulff, and C. King. Stability criteria for switched and hybrid systems. *SIAM Review*, 49(4):545–592, 2007.

[78] J. Si, A. Barto, W. Powell, and D. Wunsch, editors. *Handbook of Learning and Approximate Dynamic Programming*. Wiley-Interscience, 2004.

[79] J. Steil. *Input-Output Stability of Recurrent Neural Networks*. PhD thesis, Der Technischen Fakultät der Universität Bielefeld, 1999.

[80] J. Steil. Local stability of recurrent networks with time-varying weights and inputs. *Neurocomputing*, 48(1-4):39–51, 2002.

[81] J. Steil. Online stability of backpropagation-decorrelation recurrent learning. *Neurocomputing*, 69(7–9):642–650, 2006.

[82] J. Steil and H. Ritter. Maximization of stability ranges for recurrent neural networks subject to online adaptation. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 369–374, 1999.

[83] J.F. Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999.

[84] G.-Z. Sun, H.-H. Chen, and Y.C. Lee. Green's function method for fast online learning algorithm of recurrent neural networks. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 333–340. MIT Press, 1991.

[85] R.S. Sutton, A.G. Barto, and R.J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, 1992.

[86] V. Tadic. On the convergence of temporal-difference learning with linear function approximation. *Machine Learning*, 42:241–267, 2001.

[87] O. Toker and H. Ozbay. On the NP-hardness of solving bilinear matrix inequalities and simultaneous stabilization with static output feedback. In *Proceedings of the American Control Conference*, volume 4, pages 2525–2526, 1995.

[88] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690, 1997.

[89] L. Vandenberghe, V.R. Balakrishnan, R. Wallin, A. Hansson, and T. Roh. Interior-point algorithms for semidefinite programming problems derived from the kyp lemma. In D. Henrion and A. Garulli, editors, *Positive Polynomials in Control*, pages 195–238. Springer-Verlag, 2005.

[90] C.H. Vogelbusch. *Numerical Treatment of Nonlinear Semidefinite Programs*. PhD thesis, Heinrich-Heine-Universität Düsseldorf, 2006.

[91] J.C. Willems. *The Analysis of Feedback Systems*. The M.I.T. Press, 1971.

[92] H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of Semidefinite Programming: Theory, Algorithms, and Applications*. Kluwer Academic, 2000.

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

[94] S. Xu, Y. Chu, and J. Lu. New results on global exponential stability of recurrent neural networks with time-varying delays. *Physics Letters A*, 352(4-5):371–379, 2006.

[95] E.A. Yildirim and S.J. Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12(3):782–810, 2002.

[96] H. Yu and D.P. Bersekas. Convergence results for some temporal difference methods based on least squares. Technical Report LIDS Report 2697, MIT, 2006.

[97] G. Zames. On the input-output stability of time-varying nonlinear feedback systems - part i: Conditions derived using concepts of loop gain, conicity and positivity. *IEEE Transactions on Automatic Control*, AC-11:228–238, 1966.

[98] F. Zhang, editor. *The Schur Compliment and Its Applications*. Springer, 2005.

Appendix A

LMIs for Time-Varying RNNs

A recurrent neural network with time-varying weights can be written as the equation

$$\dot{x}(t) = -Cx(t) + \widetilde{W}\Delta(x(t),t)KRx(t) + u(t) \tag{A.1}$$

where

$$\begin{aligned}
\widetilde{W} &= \begin{bmatrix} W & \widehat{W} & -\widehat{W} \end{bmatrix} \in \mathbb{R}^{n\times(n+2n^2)}, \\
\widehat{W} &= \begin{bmatrix} e_1 & \overset{\times n}{\cdots} & e_1 & e_2 & \dots & e_n \end{bmatrix} \in \mathbb{R}^{(n\times n^2)}, \\
K &= \operatorname{diag}\{I, \overline{\Delta}, \underline{\Delta}\}, \\
R &= \begin{bmatrix} I & I & \overset{\times 2n}{\cdots} & I \end{bmatrix}^T \in \mathbb{R}^{(n+2n^2)\times n}, \\
\Delta(x(t),t) &= \operatorname{diag}\{\delta_i(x_i(t)), \overline{\delta}_{ij}(t)\delta_i(x_i(t)), \underline{\delta}_{ij}(t)\delta_i(x_i(t))\} \in \mathbb{R}^{(n+2n^2)\times(n+2n^2)}, \\
& \qquad 0 \le \delta_i(x_i(t)) \le 1,\ 0 \le \overline{\delta}_{ij}(t) \le 1,\ 0 \le \underline{\delta}_{ij}(t) \le 1.
\end{aligned}$$

See Chapter 4 for details. The stability of this system is implied by the feasibility of the LMI condition

$$\begin{bmatrix} -CP - PC & P\widetilde{W} + R^T\hat{T} \\ \widetilde{W}^TP + \hat{T}^TR & -(\hat{T} + \hat{T}^T)K^{-1} \end{bmatrix} < 0. \tag{A.2}$$

In this section additional LMI conditions are given which include the Popov IQC and terms for estimating the $\mathcal{L}_2$ gain of the time-varying system. Also, equivalent LMI conditions are given for the time-varying network equations (4.9).

A.1 Adding the Popov IQC

The Popov IQC is a dynamic IQC, and the techniques described in Appendix B are required for constructing the resulting LMIs. The details are left out for conciseness and only the main LMI is given. Define the matrix Q as

$$Q = \operatorname{diag}\{q_1, \dots, q_n, 0, \overset{\times 2n^2}{\cdots}, 0\}$$

where the q_i are positive decision variables. The Popov IQC results in the LMI

$$\begin{bmatrix} RC & -R\widetilde{W} \\ 0 & I \end{bmatrix}^T \begin{bmatrix} 0 & -Q \\ -Q & 0 \end{bmatrix} \begin{bmatrix} RC & -R\widetilde{W} \\ 0 & I \end{bmatrix} \ge 0.$$

The resulting LMI condition is

$$\begin{bmatrix} -CP - PC & P\widetilde{W} - CR^TQ + R^T\hat{T} \\ \widetilde{W}^TP - Q^TRC + \hat{T}^TR & QR\widetilde{W} + \widetilde{W}^TR^TQ - (\hat{T} + \hat{T}^T)K^{-1} \end{bmatrix} < 0.$$

A.2 LMI for Gain Estimation

The gain of a time varying network can be estimating by augmenting the LMI from the previous section. In this case the Popov IQC results in the LMI condition

$$\begin{bmatrix} RC & -R\widetilde{W} & -R \\ 0 & I & 0 \end{bmatrix}^T \begin{bmatrix} 0 & -Q \\ -Q & 0 \end{bmatrix} \begin{bmatrix} RC & -R\widetilde{W} & -R \\ 0 & I & 0 \end{bmatrix} \geq 0.$$

The main LMI condition is given by

$$\begin{bmatrix} -CP - PC + I & P\widetilde{W} - CR^TQ + R^T\hat{T} & P \\ \widetilde{W}^TP - Q^TRC + \hat{T}^TR & QR\widetilde{W} + \widetilde{W}^TR^TQ - (\hat{T} + \hat{T}^T)K^{-1} & QR \\ P & R^TQ & -\gamma I \end{bmatrix} < 0.$$

A.3 LMI Conditions for the Modified Time-Varying RNN Equations

A modified version of (A.1) is given in Equation 4.9. These equations are

$$\begin{aligned} \dot{x}(t) &= -Cx(t) + \widetilde{W} \begin{bmatrix} I & 0 \\ 0 & \widetilde{\Delta}(t) \end{bmatrix} KR\widehat{\Delta}(x(t))x(t) + u(t) \\ \widehat{\Delta}(x(t)) &= \text{diag}\{\delta_i(x_i(t))\} \\ \widetilde{\Delta}(t) &= \text{diag}\{\overline{\delta}(t), \underline{\delta}(t)\} \end{aligned} \tag{A.3}$$

It is easier to write the LMI stability condition derived from this formulation in its unexpanded form. The feedback system is defined by

$$\begin{gathered} A = -C,\ B_1 = I,\ B_2 = \widetilde{W}, \\ C_1 = I_n,\ C_2 = \begin{bmatrix} I_n \\ 0 \end{bmatrix} \in \mathbb{R}^{(n+2n^2)\times n}, \\ D_{11} = 0 \in \mathbb{R}^{n\times(n+2n^2)},\ D_{12} = 0 \in \mathbb{R}^{n\times n}, \\ D_{21} = \begin{bmatrix} 0 & 0 \\ \widetilde{R} & 0 \end{bmatrix} \in \mathbb{R}^{(n+2n^2)\times(n+2n^2)}, \\ D_{22} = 0 \in \mathbb{R}^{(n+2n^2)\times n}, \end{gathered}$$

where I_n is the $n \times n$ identity matrix and

$$\widetilde{R} = \begin{bmatrix} I & \overset{\times 2n}{\cdots} & I \end{bmatrix}^T.$$

The main LMI is

$$\begin{bmatrix} -CP - PC & P\widetilde{W} & P \\ \widetilde{W}^TP & 0 & 0 \\ P & 0 & 0 \end{bmatrix} + \Sigma_1 + \Sigma_2 + \Sigma_3 \leq 0$$

where

$$\Sigma_1 = \begin{bmatrix} C_1 & D_{11} & D_{12} \\ 0 & 0 & I \end{bmatrix}^T \begin{bmatrix} I & 0 \\ 0 & -\gamma I \end{bmatrix} \begin{bmatrix} C_1 & D_{11} & D_{12} \\ 0 & 0 & I \end{bmatrix},$$

$$\Sigma_2 = \begin{bmatrix} C_1 & D_{21} & D_{22} \\ 0 & I & 0 \end{bmatrix}^T \begin{bmatrix} 0 & \hat{T} \\ \hat{T}^T & -(\hat{T} + \hat{T}^T)K^{-1} \end{bmatrix} \begin{bmatrix} C_1 & D_{21} & D_{22} \\ 0 & I & 0 \end{bmatrix},$$

$$\Sigma_3 = \begin{bmatrix} RC & -R\widetilde{W} & -R \\ 0 & I & 0 \end{bmatrix}^T \begin{bmatrix} 0 & -Q \\ -Q & 0 \end{bmatrix} \begin{bmatrix} RC & -R\widetilde{W} & -R \\ 0 & I & 0 \end{bmatrix}.$$

Appendix B

LMIs from Dynamic IQCs

In this short appendix, an explanation of the implementation of dynamic IQCs is given. Dynamic IQCs are used in Chapter 6 to describe uncertain parameters and unmodeled linear time invariant dynamics. First some basic details are recalled.

IQCs are generally applied to model the operator Δ in the feedback system given by the equations

$$\begin{aligned} \dot{x} &= Ax + B_1 u + B_2 w \\ y &= C_1 x + D_{11} u + D_{12} w \\ v &= C_2 x + D_{21} u + D_{22} w \\ w &= \Delta(v). \end{aligned} \tag{B.1}$$

The IQC terms enter the main LMI from the KYP theorem through the equation

$$\begin{bmatrix} C_2 & D_{21} & D_{22} \\ 0 & 0 & I \end{bmatrix}^T \Pi(\lambda) \begin{bmatrix} C_2 & D_{21} & D_{22} \\ 0 & 0 & I \end{bmatrix}.$$

The structure of this equation comes from the fact that the inputs and outputs of the IQC are constructed by the equation

$$\begin{bmatrix} C_2 & D_{21} & D_{22} \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} x \\ u \\ w \end{bmatrix}.$$

The first n rows of the matrix multiplier construct the input v and the last n rows construct the output w. For implementing dynamic IQCs it is necessary to augment the inputs and outputs of the IQC and thus change the matrix multiplier in the above equations.

B.1 Unmodeled LTI Dynamics

The implementation of the IQC for unmodeled LTI dynamics is illustrated on the simple system shown in Figure B.1. The plant operator is $G(s) = \frac{s}{s+100}$ and the operator $\Delta(s)$ is an LTI operator with $\|\Delta(s)\| \leq 1$. The system can be represented in the format of the feedback system (B.1) by taking

$$\begin{gathered} A = -100, B_1 = 1, B_2 = -1, C_1 = -100, C_2 = -100, \\ D_{11} = 1, D_{12} = -1, D_{21} = 1, D_{22} = -1. \end{gathered}$$

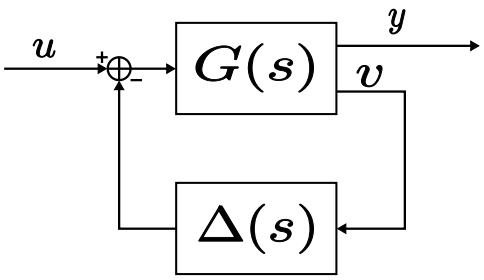

Figure B.1: A feedback system with unmodeled LTI dynamics.

Recall from Chapter 6 that the relation, $w(s) = \Delta(s)v(s)$, where $\Delta(s)$ is an unknown LTI operator satisfying $\|\Delta(s)\| \leq 1$, satisfies the IQC given by

$$\Pi(s) = \begin{bmatrix} x(s) & 0 \\ 0 & -x(s) \end{bmatrix}$$
$$x(s) \geq 0.$$

Here, $x(s)$ is a bounded, measurable function that is represented for computational purposes as

$$x(s) = x_0 + x_1 \frac{1}{s+1}$$

where x_0 and x_1 are real-valued decision variables.

The constraint $x(s) \geq 0$ is enforced in the LMI problem by application of the KYP lemma. Rewrite $x(s)$ in the standard operator form, $C_x(sI - A_x)^{-1}B_x + D_x$, by letting $C_x = x_1$, $A_x = -1$, $B_x = 1$, and $D_x = x_0$. The KYP lemma results in the constraint

$$\begin{bmatrix} A_x^T P + P A_x & P B_x - C_x^T \\ B_x^T P - C_x & -(D_x + D_x^T) \end{bmatrix} \leq 0, \quad P = P^T \geq 0,$$

or

$$\begin{bmatrix} -2P & P - x_1 \\ P - x_1 & -2x_0 \end{bmatrix} \leq 0, \quad P \in \mathbb{R}_+$$

in the specific case of interest.

The part of the main LMI contributed by the IQC is constructed in the following way. The dynamic parts of the IQC are extracted leading to the representation

$$\begin{bmatrix} v \\ -w \\ \frac{1}{s+1}v \\ -\frac{1}{s+1}w \end{bmatrix}^T \begin{bmatrix} x_0 & & & \\ & x_0 & & \\ & & x_1 & \\ & & & x_1 \end{bmatrix} \begin{bmatrix} v \\ w \\ v \\ w \end{bmatrix} \geq 0.$$

Multiplying out the left hand side of the condition gives

$$\begin{aligned} v^T x_0 v - w^T x_0 w + \left(\frac{1}{s+1}v\right)^T x_1 v - \left(\frac{1}{s+1}w\right)^T w \\ = v^T \left(x_0 + x_1 \frac{1}{s+1}\right) v - w^T \left(x_0 + x_1 \frac{1}{s+1}\right) w \\ = \begin{bmatrix} v \\ w \end{bmatrix}^T \begin{bmatrix} x(s) & 0 \\ 0 & -x(s) \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}. \end{aligned}$$

To construct the left-hand and right-hand multipliers in the IQC, two states must be added to the system to represent $\frac{1}{s+1}v$ and $\frac{1}{s+1}w$. This means augmenting the main system matrices as

$$A = \begin{bmatrix} -100 & 0 & 0 \\ -100 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, B_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, B_2 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix},$$
$$C_1 = \begin{bmatrix} -100 & 0 & 0 \end{bmatrix}, C_2 = \begin{bmatrix} -100 & 0 & 0 \end{bmatrix}.$$

The augmented system has three states and two signals. The first state of the augmented system is the state of the plant. The second state is $\frac{1}{s+1}v$, and the third state is $\frac{1}{s+1}w$. The left- and right-hand multipliers for the IQC can be constructed by selecting the correct signals and states. The multipliers are

$$L = \begin{bmatrix} -100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}, \text{ and}$$
$$R = \begin{bmatrix} -100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ -100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The LMI for the IQC is constructed by

$$L^T \begin{bmatrix} x_0 & & & \\ & x_0 & & \\ & & x_1 & \\ & & & x_1 \end{bmatrix} R + R^T \begin{bmatrix} x_0 & & & \\ & x_0 & & \\ & & x_1 & \\ & & & x_1 \end{bmatrix} L$$

where the second term exists to ensure symmetry and does not effect the meaning of the constraint. This term must be added to the main KYP constraint and the term for estimating the $\mathcal{L}_2$ gain to construct the complete LMI.

B.2 Uncertain Parameters

The relation $w = \delta v$ with $\delta \in [-1, 1]$ can be described by the IQC

$$\Pi(s) = \begin{bmatrix} x(s) & z(s) \\ z^*(s) & -x(s) \end{bmatrix}$$
$$x(s) \geq 0$$

where $x(s)$ is the same as in the previous section and

$$z(s) = z_1 \frac{1}{s^2 - 1} = -\frac{1}{2} z_1 \left(\left(\frac{1}{s+1} \right)^* + \frac{1}{s+1} \right).$$

The positivity of $x(s)$ can be enforced using the KYP lemma as shown in the previous section.

The IQC has three decision variables and can be written in the alternate form

$$\begin{bmatrix} v \\ w \\ v \\ \frac{1}{s+1}w \\ \frac{1}{s+1}w \\ \frac{1}{s+1}v \end{bmatrix}^T \begin{bmatrix} x_0 & & & & & & \\ & x_0 & & & & & \\ & & x_1 & & & & \\ & & & x_1 & & & \\ & & & & z_1 & & \\ & & & & & z_1 & \end{bmatrix} \begin{bmatrix} v \\ -w \\ \frac{1}{s+1}v \\ -w \\ v \\ -w \end{bmatrix}.$$

Multiplying out the equation and adding to it its transpose gives the desired IQC. The rest of the construction process follows the one used in the previous section for the unmodeled LTI dynamics and is not repeated here.