A MATLAB-BASED TOOLBOX FOR THE SIMULATION AND DESIGN OF L1 ADAPTIVE CONTROLLERS

BY

ALAN BELGRAVE GOSTIN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Advisers:

Professor Naira Hovakimyan
Associate Professor Daniel M. Liberzon

ABSTRACT

This thesis introduces the $\mathcal{L}_1$ Adaptive Control Toolbox, a set of tools implemented in Matlab that aid in the design process of an $\mathcal{L}_1$ adaptive controller and enable the user to construct simulations of the closed-loop system to verify its performance. Following a brief review of the existing theory on $\mathcal{L}_1$ adaptive controllers, the interface of the toolbox is presented, including a description of the functions accessible to the user. Two novel algorithms for determining the required sampling period of a piecewise constant adaptive law are presented and their implementation in the toolbox is discussed. The detailed description of the structure of the toolbox is provided as well as a discussion of the implementation of the creation of simulations. Finally, the graphical user interface is presented and described in detail, including the graphical design tools provided for the development of the filter $C(s)$. The thesis closes with suggestions for further improvement of the toolbox.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Often in control systems, the control designer is unable to completely characterize the system and is forced to design a controller that can deal with the uncertainties that arise from the incomplete characterization. From this fundamental problem, the idea of adaptive controllers arose. The underlying concept behind adaptive control is simple: during operation, monitor the behavior of the system and generate estimates of the system's uncertainties that can be used to create the control input fed back into the system. Many of the classical adaptive controllers based on this concept are presented in [1] and [2] and provide guaranteed performance bounds on the system's output. Ideally, an adaptive controller would correctly respond to all the changes in the system's initial conditions, reference inputs, and uncertainties by quickly identifying a set of control parameters that would provide a satisfactory system response. However, to be able to quickly respond to these changes requires a fast estimation scheme with high adaptation rates. These high adaptation rates, in turn, can create high frequencies in the control signals and increased sensitivity to time delays. Therefore, a common concern with adaptive controllers is their ability to guarantee robustness in the presence of fast adaptation. Several papers, including those by Ioannou and Kokotović [3–5], Peterson and Narendra [6], Kresselmeier and Narendra [7], and Narendra and Annaswamy [8], investigated the robustness of adaptive controllers and proposed modifications to the adaptive laws to prevent instability. However, these modifications were unable to provide an analytical quantification of the relationship between the rate of adaptation, the transient response, and the robustness margins. Therefore, it became clear that a new architecture for adaptive controllers needed to be created that would allow for guaranteed robustness in the presence of fast adaptation and provide a means of quantifying the trade-off between the two.

The $\mathcal{L}_1$ adaptive controller was first proposed by Cao and Hovakimyan in [9] and describes such an architecture that decouples adaptation from the robustness of the system and also provides performance bounds for both the input and the output of the plant. The key underlying concept behind $\mathcal{L}_1$ adaptive controllers is that the controller should only attempt to control the plant within the bandwidth of the control channel. By doing so, the system can achieve fast adaptation, and therefore good performance, without allowing high frequencies to enter the control signals, thus maintaining the system's robustness. The theory of $\mathcal{L}_1$ adaptive controllers has since been extended for use with systems with time-varying uncertainties in [10], for systems where only output feedback is available in [11], and most recently, multiple input, multiple output systems with unmatched nonlinearities in [12] by Xargay, Hovakimyan, and Cao. Additionally, a modification of the standard $\mathcal{L}_1$ adaptive controller which uses a piecewise constant adaptive law was first proposed in [13]. $\mathcal{L}_1$ adaptive controllers have found numerous applications in flight control such as the NASA AirSTAR flight test vehicle [14] and Boeing's X-48B blended wing aircraft [15], among others [16] – [17].

However, as the number of applications of $\mathcal{L}_1$ adaptive controllers has grown, it has become increasingly clear that a set of tools to aid in the design and development of these controllers is necessary. This paper presents the $\mathcal{L}_1$ Adaptive Control Toolbox, a new set of tools implemented in Matlab that:

- Aid in the design of $\mathcal{L}_1$ adaptive controllers by enabling the user to quickly tune the controller to achieve the desired performance, thereby reducing the development time of a new controller.
- Enable users to easily construct and configure simulations of $\mathcal{L}_1$ adaptive controllers.
- Dynamically check the assumptions and requirements from the theory, thereby ensuring that the user's final design is valid for the given plant.

Chapter 2 will present a brief review of the existing theory for state feedback $\mathcal{L}_1$ adaptive controllers, and discuss the user interface for specifying and simulating state feedback controllers. The individual functions accessible to the user and their uses are presented. The chapter concludes with the algorithm used for designing the sampling period in the case of

the piecewise constant adaptive law.

Chapter 3 covers the toolbox's treatment of output feedback $\mathcal{L}_1$ adaptive controllers. A brief review of the existing theory for output feedback controllers is presented, followed by an in-depth discussion of the user interface for specifying and simulating output feedback controllers, including the individual functions accessible to the user and their various uses. Finally, the algorithm used for designing the sampling period in the case of non-strictly positive real models is presented.

Chapter 4 discusses the implementation of the $\mathcal{L}_1$ Adaptive Control Toolbox in detail. First, the `L1Controller` class, which contains all of the simulation tools and capabilities, is presented. The internal structure of the data stored in the class is explained first, followed by a detailed step-by-step description of how simulations are constructed and run. In the second section, the `L1gui` class, which contains the graphical user interface (GUI) and all of the dynamic interactions with the user, is presented. First the underlying structure of the class is discussed including how it handles data as it is entered by the user and the interactions between the `L1gui` class and the `L1Controller` class. Then the graphical interface between the user and the `L1gui` class is described in detail, including how the interface dynamically reconfigures itself as the user specifies the system to be simulated. The section concludes by discussing the tools provided for designing the low pass filter in the control law, $C(s)$.

Chapter 5 presents a summary of the major features discussed in this thesis and possible future improvements to the $\mathcal{L}_1$ Adaptive Control Toolbox.

Finally, note that any text that is presented in fixed-width typewriter font such as `L1gui` represents actual Matlab code or variables from the toolbox and is displayed differently to emphasize the difference between theoretical values and the toolbox implementation.

CHAPTER 2

STATE FEEDBACK

2.1 Mathematical Preliminaries

The general form of the class of systems that can be stabilized by a state feedback $\mathcal{L}_1$ adaptive controller is the following:

$$\begin{aligned} \dot{x}(t) &= A_m x(t) + B_m\big(\omega u(t) + f_1(t, x(t), z(t))\big) + B_{um} f_2(t, x(t), z(t))\,, \quad x(0) = x_0\,, \\ \dot{x}_z(t) &= g(t, x(t), x_z(t))\,, \quad z(t) = g_0(t, x_z(t))\,, \quad x_z(0) = x_{z0}\,, \\ y(t) &= Cx(t)\,, \end{aligned} \tag{2.1.1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, which can be measured, $u(t) \in \mathbb{R}^m$ is the control signal, with $m \leq n$, $y(t) \in \mathbb{R}^m$ is the output of the system, and $z(t)$ and $x_z(t)$ are the output and state vector of any unmodeled dynamics which are internal to the system and cannot be measured. In addition, $A_m \in \mathbb{R}^{n\times n}$ is a known Hurwitz matrix that describes the desired dynamics of the closed loop system, $B_m \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{m\times n}$ are known matrices such that (A_m, B_m) is controllable and (A_m, C) is observable, $B_{um} \in \mathbb{R}^{n\times(n-m)}$ is a known matrix such that $[B_m,\ B_{um}]$ is nonsingular and $B_m^\top B_{um} = 0$, $\omega \in \mathbb{R}^{m\times m}$ is an unknown matrix representing the uncertainty in the gain of the system, and $f_1(\cdot)$, $f_2(\cdot)$, $g_0(\cdot)$, and $g(\cdot)$ are unknown nonlinear functions representing the uncertainty in the plant dynamics.

The basic outline of the $\mathcal{L}_1$ adaptive control architecture for state feedback controllers is to obtain estimates of the plant's uncertainties by using a fast estimation scheme, and then combine these estimates and the reference signal to create the input to a low-pass filter, which then outputs the control signal for the plant. While this architecture is similar to model reference adaptive control (MRAC) architectures, the inclusion of the low-pass filter before

the control signal improves upon MRAC by decoupling adaptation and robustness. This filter ensures that the control signal only tries to cancel the uncertainties within the bandwidth of the control channel, and prevents the high frequencies that result from fast adaptation from entering the plant. Therefore, the trade-off between performance and robustness can be managed by tuning this filter.

Before the more general controller for (2.1.1) is presented, several special cases which allow simpler versions of the $\mathcal{L}_1$ adaptive controller to be used will be presented.

2.1.1 SISO Plant with Linear Matched Uncertainties and Known Input Gain

In the simplest case, the plant is single-input, single-output (SISO) and the only uncertainties present in the plant are in the function $f_1(\cdot)$, which is also known to be linear in x. Therefore, $f_1(\cdot)$ can be written as $\theta^\top(t)x(t) + \sigma(t)$. The equations of the plant, (2.1.1), then simplify to become

$$\begin{aligned} \dot{x}(t) &= A_m x(t) + b\big(u(t) + \theta^\top(t)x(t) + \sigma(t)\big)\,, \quad x(0) = x_0\,, \\ y(t) &= c^\top x(t)\,. \end{aligned} \tag{2.1.2}$$

We assume here that (A_m, b) is controllable, and $\forall t \geq 0,\ \theta(t) \in \Theta$, where Θ is a known compact subset of $\mathbb{R}^n$, and $|\sigma(t)| \leq \Delta$, where Δ is a known (conservative) bound on the $\mathcal{L}_\infty$-norm of σ. In addition, we assume that θ and σ are continuously differentiable and their derivatives are uniformly bounded.

$$\left\|\dot{\theta}(t)\right\|_2 \leq d_\theta < \infty\,, \qquad |\dot{\sigma}(t)| \leq d_\sigma < \infty$$

These two bounds should be known, but may be arbitrarily large. The rest of the $\mathcal{L}_1$ adaptive control architecture is introduced below.

State Predictor:

$$\dot{\hat{x}}(t) = A_m \hat{x}(t) + b\big(u(t) + \hat{\theta}^\top(t)x(t) + \hat{\sigma}(t)\big)\,, \quad \hat{x}(0) = x_0\,, \tag{2.1.3}$$

Adaptive Laws:

$$\begin{aligned}\dot{\hat{\theta}}(t) &= \Gamma \mathrm{Proj}(\hat{\theta}(t), -(\tilde{x}^\top(t)Pb)x(t))\,,\\ \dot{\hat{\sigma}}(t) &= \Gamma \mathrm{Proj}(\hat{\sigma}(t), -\tilde{x}^\top(t)Pb)\,,\end{aligned} \tag{2.1.4}$$

where Proj is the projection operator defined in [18], Γ is the adaptive gain, $\tilde{x}(t) = \hat{x}(t) - x(t)$, and P is the solution to the Lyapnuov equation $A_m^\top P + PA_m = -Q$ for some positive definite Q.

Control Law:

$$u(s) = C(s)\big(k_g r(s) - \hat{\eta}_1(s)\big)\,, \tag{2.1.5}$$

where $C(s)$ is a low pass filter with $C(0) = 1$, $k_g = -1/(c^\top A_m b)$ is a gain designed to ensure the closed loop system has DC gain 1, $u(s)$ and $r(s)$ are the Laplace transforms of $u(t)$ and $r(t)$, respectively, and $\hat{\eta}_1(s)$ is the Laplace transform of $\hat{\eta}_1(t) = \hat{\theta}^\top(t)x(t) + \hat{\sigma}(t)$.

Due to the presence of the low pass filter, the objective of this controller is to have the output y track the output of an *ideal* (non-adaptive) version of the adaptive control system which only assumes cancellation of the uncertainties within the bandwidth of the control channel. In this sense, this ideal reference model, at low frequencies, has the desired dynamics, chosen via the A_m matrix, without uncertainties, while at high frequencies, the uncertainties are still present and largely unaltered. It is important to note, however, that since the original plant is strictly proper (since there is no D matrix in (2.1.2)), at high frequencies, the effects of the uncertainties are attenuated by the low-pass filter nature of the original plant. The reference system that the closed-loop system specified by (2.1.2)–(2.1.5) tracks is presented below:

Reference System:

$$\begin{aligned}x_{ref}(s) &= H(s)\big(u_{ref}(s) + \eta_{1,ref}(s)\big) + x_{in}(s)\,,\\ u_{ref}(s) &= C(s)\big(k_g r(s) - \eta_{1,ref}(s)\big)\,,\\ y_{ref}(s) &= c^\top x_{ref}(s)\,,\end{aligned} \tag{2.1.6}$$

where $H(s) = \big(s\mathbb{I} - A_m\big)^{-1}b$, $\eta_{1,ref}(s)$ is the Laplace transform of $\eta_{1,ref}(t) = \theta^\top(t)x_{ref}(t) +$

$\sigma(t)$, and $x_{in}(s) = \big(s\mathbb{I} - A_m\big)^{-1} x_0$. From (2.1.6), it is straightforward to show that

$$x_{ref}(s) = H(s)C(s)k_g r(s) + H(s)\big(1 - C(s)\big)\eta_{1,ref}(s) + x_{in}(s).$$

The primary difference between the reference system and the original closed loop system specified by (2.1.2)–(2.1.5) is that the reference system assumes that all the uncertainties are known. Therefore, this reference system represents the best that any controller, either adaptive or non-adaptive, can hope to do within the bandwidth of the control channel.

Lemma 2.1.1 *The reference system specified in Equation* (2.1.6) *is bounded-input bounded-state (BIBS) stable if:*

$$\|G(s)\|_{\mathcal{L}_1} L \leq 1, \tag{2.1.7}$$

where $G(s) = H(s)\big(1 - C(s)\big)$, *and* $L = \max_{\theta \in \Theta} \|\theta\|_1$.

Proof The proof is presented in detail in [10], and is omitted here. □

Note that $\|G(s)\|_{\mathcal{L}_1}$ can be reduced simply by increasing the bandwidth of $C(s)$. Therefore, (2.1.7) essentially places a lower bound on the bandwidth of $C(s)$. This means that the control channel must be able to cancel out enough of the uncertainties within the bandwidth of the plant in order to ensure stability.

Theorem 2.1.1 *The transient performance of the closed-loop system specified by* (2.1.2)–(2.1.5), *subject to the constraint* (2.1.7), *tracks the reference system* (2.1.6) *in both transient and steady-state with the following error bounds:*

$$\begin{aligned} \|x - x_{ref}\|_{\mathcal{L}_\infty} &\leq \frac{\gamma_1}{\sqrt{\Gamma}}, \\ \|u - u_{ref}\|_{\mathcal{L}_\infty} &\leq \frac{\gamma_2}{\sqrt{\Gamma}}, \end{aligned} \tag{2.1.8}$$

where

$$\gamma_1 = \frac{\|C(s)\|_{\mathcal{L}_1}}{1 - \|G(s)\|_{\mathcal{L}_1} L} \sqrt{\frac{\theta_m}{\lambda_{\min}(P)}}\,,$$
$$\gamma_2 = \|C(s)\|_{\mathcal{L}_1} L\gamma_1 + \left\| C(s) \frac{1}{c_0^\top H(s)} c_0^\top \right\|_{\mathcal{L}_1} \sqrt{\frac{\theta_m}{\lambda_{\min}(P)}}\,,$$
$$\theta_m = \max_{\theta \in \Theta} \sum_{i=1}^{n} 4\theta_i^2 + 4\Delta^2 + 4\frac{\lambda_{\max}(P)}{\lambda_{\min}(Q)} \left(\max_{\theta \in \Theta} \|\theta\|_2 \, d_\theta + d_\sigma \Delta \right)\,,$$

and $c_0 \in \mathbb{R}^n$ is a vector chosen such that $c_0^\top H(s)$ is minimum phase and has relative degree one.

Proof The proof is presented in detail in [10], and is omitted here. □

The important thing to note here is that the performance bounds can be reduced simply by increasing the adaptive gain, Γ, while stability of the closed-loop adaptive system is ensured by the constraint in (2.1.7). Therefore, $C(s)$ can be chosen to ensure that the $\mathcal{L}_1$-norm condition is satisfied and thus stability is achieved, and then Γ can be chosen to achieve the desired performance bounds.

2.1.2 SISO Plant with Unknown High Frequency Input Gain

Relaxing the requirement in the previous section that the input gain be known yields a system that at first glance seems very similar to (2.1.2),

$$\begin{aligned} \dot{x}(t) &= A_m x(t) + b\big(\omega u(t) + \theta^\top(t) x(t) + \sigma(t)\big)\,, \quad x(0) = x_0\,, \\ y(t) &= c^\top x(t)\,, \end{aligned} \tag{2.1.9}$$

where the only difference is that now there is an unknown gain $\omega \in [\omega_l,\ \omega_h] \subset \mathbb{R},\ \omega_l,\ \omega_r > 0$. We again assume that (A_m, b) is controllable, $\theta(t) \in \Theta$, $|\sigma(t)| \leq \Delta$, $\forall t \geq 0$, and that θ and σ are continuously differentiable with uniformly bounded derivatives. The inclusion of ω requires that we make the following changes to the $\mathcal{L}_1$ adaptive controller:

State Predictor:

$$\dot{\hat{x}}(t) = A_m\hat{x}(t) + b\big(\hat{\omega}(t)u(t) + \hat{\theta}^\top(t)x(t) + \hat{\sigma}(t)\big)\,, \quad \hat{x}(0) = x_0\,, \tag{2.1.10}$$

Adaptive Laws:

$$\begin{aligned}
\dot{\hat{\theta}}(t) &= \Gamma\text{Proj}(\hat{\theta}(t), -(\tilde{x}^\top(t)Pb)x(t)) \\
\dot{\hat{\sigma}}(t) &= \Gamma\text{Proj}(\hat{\sigma}(t), -\tilde{x}^\top(t)Pb) \\
\dot{\hat{\omega}}(t) &= \Gamma\text{Proj}(\hat{\omega}(t), -(\tilde{x}^\top(t)Pb)u(t))
\end{aligned} \tag{2.1.11}$$

Control Law:

$$u(s) = KD(s)\hat{\eta}(s)\,, \tag{2.1.12}$$

where $\hat{\eta}(s)$ is the Laplace transform of $\hat{\eta}(t) = k_g r(t) - \hat{\theta}^\top(t)x(t) - \hat{\sigma}(t) - \hat{\omega}(t)u(t)$, $K \in \mathbb{R}$, and $D(s)$ is a strictly proper SISO filter. K and $D(s)$ must be chosen so that

$$C(s) = \frac{\omega KD(s)}{1 + \omega KD(s)} \tag{2.1.13}$$

is a strictly proper and stable transfer function, with $C(0) = 1$, for all $\omega \in [\omega_l,\ \omega_h]$. Since the plant is SISO, then so are $C(s)$ and $D(s)$. Therefore, $D(s)$ can be rewritten as $\frac{D_n(s)}{D_d(s)}$, and $C(s) = \frac{\omega KD_n(s)}{D_d(s)+\omega KD_n(s)}$. Since $C(0) = 1$, then we must have $D_d(0) = 0$. Therefore, in the SISO case, $D(s)$ must contain a pure integrator.

We may now define the reference system, which represents the ideal version of the controller, or in other words, a version where all the uncertainties are known.

Reference System:

$$\begin{aligned}
\dot{x}_{ref}(t) &= A_m x_{ref}(t) + b\left(\omega u_{ref}(t) + \theta^\top(t)x_{ref}(t) + \sigma(t)\right) \\
y_{ref}(t) &= c^\top x_{ref}(t) \\
u_{ref}(s) &= \frac{C(s)}{\omega}\left(k_g r(s) - \eta_{ref}(s)\right)\,,
\end{aligned} \tag{2.1.14}$$

where $\eta_{ref}(s)$ is the Laplace transform of $\eta_{ref}(t) = \theta^\top(t)x_{ref}(t) + \sigma(t)$. The following lemma

and theorem were first presented and proved in [19] and are presented here without proof.

Lemma 2.1.2 *The reference system specified in* (2.1.14) *is BIBS stable if* $D(s)$ *and* K *are chosen to satisfy*

$$\|G(s)\|_{\mathcal{L}_1} L \leq 1, \tag{2.1.15}$$

where $G(s) = (s\mathbb{I}_n - A_m)^{-1} b\big(1 - C(s)\big)$, *and* $L = \max_{\theta \in \Theta} \|\theta\|_1$.

Theorem 2.1.2 *The transient performance of the closed-loop system specified by* (2.1.9)–(2.1.12), *subject to the constraint* (2.1.15), *tracks the reference system* (2.1.14) *in both transient and steady-state with the following error bounds:*

$$\begin{aligned} \|x - x_{ref}\|_{\mathcal{L}_\infty} &\leq \frac{\gamma_1}{\sqrt{\Gamma}}, \\ \|u - u_{ref}\|_{\mathcal{L}_\infty} &\leq \frac{\gamma_2}{\sqrt{\Gamma}}, \end{aligned} \tag{2.1.16}$$

where

$$\begin{aligned} \gamma_1 &= \frac{\|C(s)\|_{\mathcal{L}_1}}{1 - \|G(s)\|_{\mathcal{L}_1} L} \sqrt{\frac{\theta_m}{\lambda_{\min}(P)}}, \\ \gamma_2 &= \left\| \frac{C(s)}{\omega} \right\|_{\mathcal{L}_1} L\gamma_1 + \left\| \frac{C(s)}{\omega} \frac{1}{c_0^\top H(s)} c_0^\top \right\|_{\mathcal{L}_1} \sqrt{\frac{\theta_m}{\lambda_{\min}(P)}}, \\ \theta_m &= \max_{\theta \in \Theta} \sum_{i=1}^{n} 4\theta_i^2 + 4\Delta^2 + 4(\omega_h - \omega_l)^2 + 4\frac{\lambda_{\max}(P)}{\lambda_{\min}(Q)} \left(\max_{\theta \in \Theta} \|\theta\|_2 \, d_\theta + d_\sigma \Delta \right), \end{aligned}$$

and $c_0 \in \mathbb{R}^n$ *is a vector chosen such that* $c_0^\top H(s)$ *is minimum phase and has relative degree one.*

2.1.3 MIMO Plant with Nonlinear Unmatched Uncertainties

This section treats the $\mathcal{L}_1$ adaptive controller for the general system expressed in Equation (2.1.1). The material presented here is presented in [20]. To simplify notation, we define $X = [x^\top,\ z^\top]^\top$ and use this to redefine $f_i(t, X) = f_i(t, x, z)$, $i = 1, 2$. Then, we place the following assumptions on the system.

Assumption 2.1.1 *There exist $B_i > 0$ such that $\|f_i(t,0)\|_\infty \leq B_i$ holds for all $t \geq 0$, for $i = 1, 2$.*

Assumption 2.1.2 *For arbitrary $\delta > 0$, there exist positive K_1, K_2, $d_{ft1}(\delta)$, and $d_{ft2}(\delta)$ such that for all $\|X\|_\infty < \delta$, the partial derivatives of $f_i(t, X)$ are piecewise constant and bounded uniformly in t:*

$$\left\| \frac{\partial f_i(t,X)}{\partial X} \right\|_\infty \leq K_i \,, \quad \left\| \frac{\partial f_i(t,X)}{\partial t} \right\|_\infty \leq d_{fti}(\delta) \,, \quad i = 1, 2.$$

Assumption 2.1.3 *The matrix ω is assumed to be nonsingular, strictly row diagonally dominant, and to reside within a known compact convex set $\Omega \subset \mathbb{R}^{m \times m}$. It is also assumed that $sgn(\omega_{ii})$ is known, $i = 1, 2, \ldots, m$.*

Assumption 2.1.4 *The transfer function $H_m(s) = C(s\mathbb{I} - A_m)^{-1}B_m$ is assumed to have all of its zeros in the open left half-plane.*

Assumption 2.1.5 *The x_z dynamics represented by the functions g and g_0 in Equation (2.1.1) are assumed to be bounded-input bounded-output (BIBO) stable with respect to both their initial condition x_{z0} and their input x. More specifically, there exist L_z, $B_z > 0$ such that for all $t \geq 0$*

$$\|z_t\|_{\mathcal{L}_\infty} \leq L_z \|x_t\|_{\mathcal{L}_\infty} + B_z \,.$$

We use the simplified notation $\|f_t\|_{\mathcal{L}_\infty}$ with the subscript t to represent the truncated norm $\left\|f_{[t_0,\, t]}(\tau)\right\|_{\mathcal{L}_\infty}$ where

$$f_{[t_0,\, t]}(\tau) = \begin{cases} 0, & \tau < t_0 \\ f(\tau), & t_0 \leq \tau \leq t \\ 0, & \tau > t \end{cases} \,.$$

Note that in [20], Assumption 2.1.2 allows the variables K_i to be based on δ. However, for the purposes of the $\mathcal{L}_1$ Adaptive Control Toolbox, we require that a single Lipschitz constant is known for all X within the region the system will operate in. In addition, we

must define a Lipschitz constant that combines the effects of the nonlinearities and the unmodeled dynamics. Therefore, for every $\nu > 0$, let

$$L_{i\nu} = \frac{M}{\nu} K_i \,, \tag{2.1.17}$$

where $M = \max\{\nu + \bar{\gamma}_x, L_z(\nu + \bar{\gamma}_x) + B_z\}$, and $\bar{\gamma}_x$ is an arbitrary positive constant representing the desired bound on the error $\|x - x_{ref}\|_{\mathcal{L}_\infty}$.

As in the previous sections, the goal is to have the output y track the output of a desired transfer function $C(s\mathbb{I}_n - A_m)^{-1}B_m k_g$ to a bounded reference input r. Note that while k_g could be any transfer function, it will be assumed here that $k_g = -(CA_m^{-1}B_m)^{-1}$ so that the DC gain of the desired transfer function is $\mathbb{I}_m$.

As in Section 2.1.2, rather than define $C(s)$, we must define $K \in \mathbb{R}^{m\times m}$ and $D(s)$, a strictly proper transfer matrix with m inputs and m outputs, such that

$$C(s) = \omega K D(s)(\mathbb{I}_m + \omega K D(s))^{-1} \tag{2.1.18}$$

is a strictly proper and stable transfer function with $C(0) = \mathbb{I}_m$, for all $\omega \in \Omega$. We must also define the following:

$$\begin{aligned}
H_{xm}(s) &= (s\mathbb{I}_n - A_m)^{-1}B_m \\
H_{xum}(s) &= (s\mathbb{I}_n - A_m)^{-1}B_{um} \\
H_m(s) &= C(s\mathbb{I}_n - A_m)^{-1}B_m \\
H_{um}(s) &= C(s\mathbb{I}_n - A_m)^{-1}B_{um} \\
G_m(s) &= H_{xm}(s)(\mathbb{I}_m - C(s)) \\
G_{um}(s) &= (\mathbb{I}_n - H_{xm}(s)C(s)H_m^{-1}(s)C)H_{xum}(s)\,.
\end{aligned}$$

In addition, the choices of K and $D(s)$ must ensure that $C(s)H_m^{-1}(s)$ is a stable proper transfer matrix and that there exists $\rho_r > 0$ such that

$$\begin{aligned}\|G_m(s)\|_{\mathcal{L}_1}\left(L_{1\rho_r}\rho_r + B_1\right) + \|G_{um}(s)\|_{\mathcal{L}_1}\left(L_{2\rho_r}\rho_r + B_2\right) + \\ \|H_{xm}(s)C(s)k_g\|_{\mathcal{L}_1}\|r\|_{\mathcal{L}_\infty} + \rho_{ic} < \rho_r\,,\end{aligned} \tag{2.1.19}$$

where $\rho_{ic} = \|s(s\mathbb{I}_n - A_m)^{-1}\|_{\mathcal{L}_1}\rho_0$ and ρ_0 is a known bound on the initial conditions, $\|x_0\|_\infty \leq \rho_0$. Also, let

$$\rho_x = \rho_r + \bar{\gamma}_x \tag{2.1.20}$$

and let

$$\gamma_x = \frac{\|H_{xm}(s)C(s)H_m^{-1}(s)C\|_{\mathcal{L}_1}}{1 - \|G_m(s)\|_{\mathcal{L}_1}L_{1\rho_r} - \|G_{um}(s)\|_{\mathcal{L}_1}L_{2\rho_r}}\bar{\gamma}_0 + \epsilon \tag{2.1.21}$$

where $\bar{\gamma}_0$ and ϵ are arbitrary positive constants such that $\gamma_x \leq \bar{\gamma}_x$. Finally, let

$$\rho_u = \rho_{u_r} + \gamma_u \tag{2.1.22}$$

where

$$\begin{aligned}\rho_{u_r} = \left\|\omega^{-1}C(s)\right\|_{\mathcal{L}_1}\left(L_{1\rho_r}\rho_r + B_1\right) + \left\|\omega^{-1}C(s)H_m^{-1}(s)H_{um}(s)\right\|_{\mathcal{L}_1}\left(L_{2\rho_r}\rho_r + B_2\right) \\ + \left\|\omega^{-1}C(s)k_g\right\|_{\mathcal{L}_1}\|r\|_{\mathcal{L}_\infty}\,,\end{aligned} \tag{2.1.23}$$

and

$$\gamma_u = \left(\left\|\omega^{-1}C(s)\right\|_{\mathcal{L}_1}L_{1\rho_r} + \left\|\omega^{-1}C(s)H_m^{-1}(s)H_{um}(s)\right\|_{\mathcal{L}_1}L_{2\rho_r}\right)\gamma_x + \left\|\omega^{-1}C(s)H_m^{-1}(s)C\right\|_{\mathcal{L}_1}\bar{\gamma}_0\,. \tag{2.1.24}$$

An issue with nonlinear uncertainties is that it is unclear from Equation (2.1.1) what exactly should be estimated in the closed-loop adaptive system. However, the following lemma, first presented in [21], allows the uncertainties to be rewritten in a more useful form.

Lemma 2.1.3 *For the system in Equation* (2.1.1)*, if*

$$\|x_\tau\|_{\mathcal{L}_\infty} \leq \rho_x\,, \quad \|u_\tau\|_{\mathcal{L}_\infty} \leq \rho_u\,,$$

then, for all $t \in [0,\ \tau]$*, there exist differentiable* $\theta_1(t) \in \mathbb{R}^m$*,* $\sigma_1(t) \in \mathbb{R}^m$*,* $\theta_2(t) \in \mathbb{R}^{n-m}$*, and* $\sigma_2(t) \in \mathbb{R}^{n-m}$ *such that*

$$\begin{aligned} \|\theta_i(t)\|_\infty < L_{i\rho_x}\,, \quad \|\sigma_i(t)\|_\infty < L_{i\rho_x} B_z + B_i + \epsilon_i\,, \\ f_i(t, x(t), z(t)) = \theta_i(t)\,\|x_t\|_{\mathcal{L}_\infty} + \sigma_i(t)\,, \end{aligned} \tag{2.1.25}$$

where $\epsilon_i > 0$ *is an arbitrarily small constant and the derivatives of* θ_i *and* σ_i *are bounded.*

Using this lemma, we may introduce the rest of the closed-loop adaptive system.

State Predictor:

$$\begin{aligned} \dot{\hat{x}}(t) =& A_m\hat{x}(t) + B_m\big(\hat{\omega}(t)u(t) + \hat{\theta}_1(t)\,\|x_t\|_{\mathcal{L}_\infty} + \hat{\sigma}_1(t)\big) \\ &+ B_{um}\left(\hat{\theta}_2(t)\,\|x_t\|_{\mathcal{L}_\infty} + \hat{\sigma}_2(t)\right)\,, \quad \hat{x}(0) = x_0\,, \end{aligned} \tag{2.1.26}$$

where $\hat{\omega}(t) \in \mathbb{R}^{m\times m}$, $\hat{\theta}_1(t) \in \mathbb{R}^m$, $\hat{\sigma}_1(t) \in \mathbb{R}^m$, $\hat{\theta}_2(t) \in \mathbb{R}^{n-m}$, and $\hat{\sigma}_2(t) \in \mathbb{R}^{n-m}$ are the estimates of the plant's uncertainties.

Adaptive Laws:

$$\begin{aligned} \dot{\hat{\theta}}_1(t) &= \Gamma\mathrm{Proj}(\hat{\theta}_1(t), -(\tilde{x}^\top(t)PB_m)^\top\,\|x_t\|_{\mathcal{L}_\infty}) \\ \dot{\hat{\sigma}}_1(t) &= \Gamma\mathrm{Proj}(\hat{\sigma}_1(t), -(\tilde{x}^\top(t)PB_m)^\top) \\ \dot{\hat{\theta}}_2(t) &= \Gamma\mathrm{Proj}(\hat{\theta}_2(t), -(\tilde{x}^\top(t)PB_{um})^\top\,\|x_t\|_{\mathcal{L}_\infty}) \\ \dot{\hat{\sigma}}_2(t) &= \Gamma\mathrm{Proj}(\hat{\sigma}_2(t), -(\tilde{x}^\top(t)PB_{um})^\top) \\ \dot{\hat{\omega}}(t) &= \Gamma\mathrm{Proj}(\hat{\omega}(t), -(\tilde{x}^\top(t)PB_m)^\top u^\top(t)) \end{aligned} \tag{2.1.27}$$

where P is the solution to the Lyapunov equation $A_m^\top P + PA_m = -Q$ for some $Q = Q^\top > 0$, Γ is the adaptive gain, and the projection bounds are $\left\|\hat{\theta}_i(t)\right\|_\infty \leq L_{i\rho_x}$, $\|\hat{\sigma}_i(t)\|_\infty \leq L_{i\rho_x}B_z + B_i + \epsilon_i$, and $\hat{\omega}(t) \in \Omega$.

Control Law:

$$u(s) = KD(s)\hat{\eta}(s)\,, \tag{2.1.28}$$

where $\hat{\eta}(s)$ is the Laplace transform of $\hat{\eta}(t) = k_g r(t) - \hat{\eta}_1(t) - \hat{\eta}_{2m}(t) - \hat{\omega}(t)u(t)$, $\hat{\eta}_{2m}(s) = H_m^{-1}(s)H_{um}(s)\hat{\eta}_2(s)$, and $\hat{\eta}_i(t) = \hat{\theta}_i(t)\,\|x_t\|_{\mathcal{L}_\infty} + \hat{\sigma}_i(t)$.

As in the previous sections, the reference system, which represents the best that the closed-loop adaptive system can do, is found by assuming that all uncertainties are known. Thus, we get the following:

Reference System:

$$\begin{aligned}
\dot{x}_{ref}(t) =& A_m x_{ref}(t) + B_m\big(\omega u_{ref}(t) + f_1(t, x_{ref}(t), z_{ref}(t))\big) \\
&+ B_{um} f_2(t, x_{ref}(t), z_{ref}(t))\,, \quad x_{ref}(0) = x_0 \\
\dot{x}_{z,ref}(t) =& g(t, x_{ref}(t), x_{z,ref}(t))\,, \quad z_{ref}(t) = g_0(t, x_{z,ref}(t))\,, \quad x_{z,ref}(0) = x_{z0} \\
y_{ref}(t) =& C x_{ref}(t)\,, \\
u_{ref}(s) =& \omega^{-1} C(s)\left(k_g r(s) - \eta_{1ref}(s) - H_m^{-1}(s)H_{um}(s)\eta_{2ref}(s)\right)\,,
\end{aligned} \tag{2.1.29}$$

where $\eta_{1ref}(t) = f_1(t, x_{ref}(t), z_{ref}(t))$ and $\eta_{2ref}(t) = f_2(t, x_{ref}(t), z_{ref}(t))$.

The following lemma and theorem were first presented and proved in [20], and are presented without proof.

Lemma 2.1.4 *For the closed-loop system in Equation* (2.1.29)*, subject to the $\mathcal{L}_1$-norm condition in Equation* (2.1.19)*, if $\|x_0\|_\infty \leq \rho_0$ and $\|z_{ref\ \tau}\|_{\mathcal{L}_\infty} \leq L_z\left(\|x_{ref\ \tau}\|_{\mathcal{L}_\infty} + \gamma_x\right) + B_z$, then $\|x_{ref\ \tau}\|_{\mathcal{L}_\infty} < \rho_r$ and $\|u_{ref\ \tau}\|_{\mathcal{L}_\infty} < \rho_{u_r}$.*

Theorem 2.1.3 *If Γ is sufficiently large and $\|x_0\|_\infty \leq \rho_0$, then the closed-loop system consisting of Equations* (2.1.1) *and* (2.1.26)–(2.1.28)*, subject to the $\mathcal{L}_1$-norm condition in Equation* (2.1.19)*, satisfies the following:*

$$\|x\|_{\mathcal{L}_\infty} \leq \rho_x\,, \quad \|u\|_{\mathcal{L}_\infty} \leq \rho_u\,, \quad \|\tilde{x}\|_{\mathcal{L}_\infty} \leq \bar{\gamma}_0\,,$$

$$\|x - x_{ref}\|_{\mathcal{L}_\infty} \leq \gamma_x\,, \quad \|u - u_{ref}\|_{\mathcal{L}_\infty} \leq \gamma_u\,, \quad \|y - y_{ref}\|_{\mathcal{L}_\infty} \leq \|C\|_\infty\,\gamma_x\,.$$

2.1.4 The Piecewise Constant Adaptive Law

In the previous section, the state predictor was created by using Lemma 2.1.3. However, rather than expressing the unknowns as a function of $\|x_t\|_{\mathcal{L}_\infty}$, it is possible to estimate the aggregate effects of all the uncertainties on the system. This idea was originally created for output feedback systems discussed in Section 3.1.2. Xargay, Hovakimyan, and Cao adapted this for use in state feedback systems in [12] by rewriting the plant in the following way:

$$\begin{aligned} \dot{x}(t) &= A_m x(t) + B_m \omega u(t) + f(t, x(t), z(t)) \,, \quad x(0) = x_0 \,, \\ \dot{x}_z(t) &= g(t, x(t), x_z(t)) \,, \quad z(t) = g_0(t, x_z(t)) \,, \quad x_z(0) = x_{z0} \,, \\ y(t) &= Cx(t) \,, \end{aligned} \tag{2.1.30}$$

where $f(t, x(t), z(t)) = B_m f_1(t, x(t), z(t)) + B_{um} f_2(t, x(t), z(t))$. We may then attempt to estimate the value of f. In an attempt to mimic what a processor would actually do, let us define $T > 0$ as the adaptation sampling period, and assume that the estimates will be constant over each period. This leads to the definition of the rest of the closed-loop adaptive system.

State Predictor:

$$\dot{\hat{x}}(t) = A_m \hat{x}(t) + B_m \omega_0 u(t) + \hat{\sigma}(t) \,, \quad \hat{x}(0) = x_0 \,, \tag{2.1.31}$$

where $\omega_0 \in \mathbb{R}^{m \times m}$ is the best available estimate of ω and $\hat{\sigma}(t) \in \mathbb{R}^n$ is the estimate of the plant's uncertainties.

Adaptive Law:

$$\begin{aligned} \hat{\sigma}(t) &= \hat{\sigma}(iT) \,, \quad t \in [iT, \ (i+1)T) \,, \\ \hat{\sigma}(iT) &= -\Phi^{-1}(T) e^{A_m T} \tilde{x}(iT) \,, \quad i = 0, 1, 2, \dots \,, \end{aligned} \tag{2.1.32}$$

where

$$\Phi(T) = A_m^{-1} \left(e^{A_m T} - \mathbb{I}_n \right) \tag{2.1.33}$$

Control Law:

$$u(s) = KD(s)\hat{\eta}(s)\,, \tag{2.1.34}$$

where $\hat{\eta}(s)$ is the Laplace transform of $\hat{\eta}(t) = k_g r(t) - \hat{\eta}_m(t) - \omega_0 u(t)$, and we define $\hat{\eta}_m(s) = H_m^{-1}(s)H_{\hat{\sigma}}(s)\hat{\sigma}(s)$ and $H_{\hat{\sigma}}(s) = C(s\mathbb{I}_n - A_m)^{-1}$.

It is important to note that changing the adaptive law does not change the reference system. Therefore, the reference system is still Equation (2.1.29) and Lemma 2.1.4 still applies. However, in order to discuss stability of the adaptive closed-loop system, we must define the following:

$$\Delta_1 = \left(\max_{\omega\in\Omega}\left\{\|\omega - \omega_0\|_2\right\}\rho_u + L_{1\rho_x}\rho_x + B_1\right)\sqrt{m}\,, \tag{2.1.35}$$

$$\Delta_2 = \left(L_{2\rho_x}\rho_x + B_2\right)\sqrt{n-m}\,, \tag{2.1.36}$$

$$\kappa_1(T) = \int_0^T \left\|e^{A_m\tau}B_m\right\|_2 d\tau\,, \tag{2.1.37}$$

$$\kappa_2(T) = \int_0^T \left\|e^{A_m\tau}B_{um}\right\|_2 d\tau\,, \tag{2.1.38}$$

$$\varsigma(T) = \kappa_1(T)\Delta_1 + \kappa_2(T)\Delta_2\,, \tag{2.1.39}$$

where ρ_u was defined in Equation (2.1.22), ρ_x was defined in Equation (2.1.20), and $L_{1\rho_x}$ was defined in Equation (2.1.17). Also let

$$\bar{\alpha}_1(T) = \max_{t\in[0,\ T]}\left\{\left\|e^{A_m t}\right\|_2\right\}\,, \tag{2.1.40}$$

$$\bar{\alpha}_2(T) = \int_0^T \left\|e^{A_m\tau}\Phi^{-1}(T)e^{A_m T}\right\|_2 d\tau\,, \tag{2.1.41}$$

$$\gamma_0(T) = \left(\bar{\alpha}_1(T) + \bar{\alpha}_2(T) + 1\right)\varsigma(T)\,. \tag{2.1.42}$$

The following lemma and theorem were originally presented and proven in [12] and are presented here without proof.

Lemma 2.1.5

$$\lim_{T\to 0}\gamma_0(T) = 0$$

Theorem 2.1.4 *If* $\|x_0\|_\infty \leq \rho_0$ *and if* T *is chosen so that*

$$\gamma_0(T) < \bar{\gamma}_0 \,, \tag{2.1.43}$$

where $\bar{\gamma}_0$ *was defined in Equation* (2.1.21)*, then the closed-loop system defined by Equations* (2.1.30)–(2.1.32) *and* (2.1.34)*, subject to the* $\mathcal{L}_1$*-norm condition in* (2.1.19)*, satisfies the following:*

$$\|x\|_{\mathcal{L}_\infty} \leq \rho_x \,, \quad \|u\|_{\mathcal{L}_\infty} \leq \rho_u \,, \quad \|\tilde{x}\|_{\mathcal{L}_\infty} < \bar{\gamma}_0 \,,$$

$$\|x - x_{ref}\|_{\mathcal{L}_\infty} \leq \gamma_x \,, \quad \|u - u_{ref}\|_{\mathcal{L}_\infty} \leq \gamma_u \,, \quad \|y - y_{ref}\|_{\mathcal{L}_\infty} \leq \|C\|_\infty \gamma_x \,.$$

2.2 Toolbox Overview

2.2.1 User Interface

The process of specifying the closed-loop $\mathcal{L}_1$ adaptive control system to be simulated can be expressed in five steps:

1. Specify the matrices A_m, B_m, C, and optionally, Q and the initial condition x_0.
2. Decide if the adaptive law will be the piecewise constant law or the gradient descent law.
3. Specify the plant's uncertainties, and provide any known quantities such as Lipschitz constants, projection bounds, the adaptive gain Γ, or initial estimates. Note that based on the type of adaptive law chosen, all of these values may not be necessary.
4. Specify $C(s)$, or if ω is present, specify $D(s)$ and K.
5. Specify the sampling period T, if necessary.

With these five steps, the closed loop system may be completely specified as described in any of the subsections in Section 2.1. The $\mathcal{L}_1$ Adaptive Control Toolbox uses this process

to build the simulation of the closed-loop system and, in the `L1Controller` class, provides a separate set of functions for each of the above steps.

The function `setPlantModel(obj, Am, Bm, C, Q, IC, ICp)` comprises the first step and establishes the basic plant in Equation (2.1.1) without any of the uncertainties. The inputs `IC` and `ICp` represent the initial conditions of the plant and the state predictor, respectively. Note that the inputs `Q`, `IC`, and `ICp` are optional. If they are not specified, then it is assumed that `Q` $= \mathbb{I}_n$ and `IC` $=$ `ICp` $= 0$. In addition, the functions `setPlantIC(obj, IC)` and `setModelIC(obj, IC)` are provided so that the user may alter the initial conditions of the plant and the state predictor without having to call `setPlantModel` again. Finally, if the plant is nonlinear, then it is recommended, but not required, that the user use the `setICBound(obj, p0)` function to specify the known bound on the initial conditions.

The type of adaptive law can be specified by `usePiecewiseConstantAdaptiveLaw(obj)` or `useGradientDescentAdaptiveLaw(obj)`. They each set a flag internally that modifies the implementation of subsequent functions. This is the primary reason that these two functions must be called at this point, instead of later in the process. Note also that these are the same two functions used for output feedback as well.

A separate function is provided for each of the different types of uncertainties that can be present in a state feedback system. The list of functions is provided below:

- `addUnknownTheta(obj, radius, trueval, gamma, IC)`,
- `addUnknownSigma(obj, maxval, trueval, gamma, IC)`,
- `addUnknownOmega(obj, range, trueval, gamma, IC)`,
- `addMatchedNonlinearity(obj, trueval, K, B, gamma, IC_theta, IC_sigma)`,
- `addUnmatchedNonlinearity(obj, trueval, K, B, gamma, IC_theta, IC_sigma)`,
- `addUnmodeledDynamics(obj, dxzdt, outputFcn, Lz, Bz, ICxz)`,

where in all cases, `trueval` represents the true unknown value of the parameter. This may be supplied as a constant, an anonymous function handle or a string representation of the function. It is required, however, that the arguments of these functions be `t`, `x`, `z`, or `xz`,

representing t, $x(t)$, $z(t)$, and $x_z(t)$, respectively. Any other argument will generate an error. However, if a string is supplied, constants may be defined in the workspace and used inside the function. For example, the function `'k*t'` will execute as `'3*t'` provided that `k` is 3 in the base Matlab workspace at the time the simulation is run. Note that the value does not have to be defined when the function is specified. In the first two functions, `radius` and `maxval` are the known bounds on the 2-norm of the respective parameters, and shall be used as the projection bounds. Note that while the theory specifies projection bounds in terms of the ∞-norm, these functions require the user to transform this into a bound on the 2-norm. For the nonlinear functions, `K` and `B` are the Lipschitz constants specified in Assumptions 2.1.1 and 2.1.2. In `addUnmodeledDynamics`, `dxzdt` represents the equation for $\dot{x}_z(t) = g(t, x(t), x_z(t))$, `outputFcn` represents the equation for $z(t) = g_0(t, x_z(t))$, `Lz` and `Bz` are the Lipschitz constants from Assumption 2.1.5, and `ICxz` is x_{z0}. Note also that in the first five functions, the adaptive gain Γ and the initial estimates, denoted as `IC`, are only required when the system is using gradient descent adaptive laws and are ignored if the system is using piecewise constant laws.

The filter $C(s)$, or the filter $D(s)$ and the gain K if ω is present, can be specified with the function `setCs`. The function maybe called in one of two ways: `setCs(obj, F, K)`, or `setCs(obj, num, den, K)`, where `F` is a transfer function variable from Matlab's Control Systems Toolbox representing either $C(s)$ or $D(s)$, and `num` and `den` are cell matrices where each cell contains a vector of either the numerator's or denominator's coefficients. In other words, the command `tf(num, den)` should create either $C(s)$ or $D(s)$. Additionally, the last input, `K`, may be omitted if it is not necessary. No matter how the function is called, however, a minimal state-space representation of the filter is found and stored internally. At this point, the system is completely specified, with the possible exception of the sampling period, T. Therefore, this function checks the most important requirement in an $\mathcal{L}_1$ adaptive controller: the $\mathcal{L}_1$-norm condition, either from Equation (2.1.7) or (2.1.19), whichever is appropriate for the system specified. If this condition is not satisfied, a warning is presented to the user that the closed-loop adaptive system is not guaranteed to be stable. While theoretically, $C(s)$ could be specified prior to the adaptive law, it is anticipated that most of the tuning the user will perform when creating an $\mathcal{L}_1$ adaptive controller will take place in $C(s)$. Therefore,

Table 2.1: String identifiers for the `sim` function and their meanings

Identifiers	Function Graphed
r	$r(t)$
y	$y(t)$
yhat	$\hat{y}(t)$
ytilde	$\tilde{y}(t) = \hat{y}(t) - y(t)$
yref	$y_{ref}(t)$
eyref	$y(t) - y_{ref}(t)$
u	$u(t)$
uref	$u_{ref}(t)$
euref	$u(t) - u_{ref}(t)$
x	$x(t)$
xhat	$\hat{x}(t)$
xtilde	$\tilde{x}(t) = \hat{x}(t) - x(t)$
xref	$x_{ref}(t)$
exref	$x(t) - x_{ref}(t)$

Identifiers	Function Graphed
xz	$x_z(t)$
z	$z(t)$
theta	$\theta(t)$
thetahat	$\hat{\theta}(t)$
sigma	$\sigma(t)$
sigmahat	$\hat{\sigma}(t)$
omega	$\omega(t)$
omegahat	$\hat{\omega}(t)$
fm	$f_1(t, x(t), z(t))$
fmhat	$\hat{f}_1(t)$
fum	$f_2(t, x(t), z(t))$
fumhat	$\hat{f}_2(t)$
d	$d(t) = f(t, y(t))$

it is assumed that this function will be called last, and thus the verification of the $\mathcal{L}_1$-norm condition is performed here. Again, the only possible exception is that the sampling period, T, will have not been specified yet, but since T does not appear in the $\mathcal{L}_1$-norm condition, it is not beneficial to wait until T is specified to check the condition. Finally, note that `setCs` is the same function used for output feedback as well.

Finally, the function `setSamplingPeriod(obj, Ts)` is used in the case of the piecewise constant adaptive law to specify the sampling period, T. Calling this function when the gradient descent adaptive law is in use produces an error. In addition to storing the sampling period, this function checks the stability condition on T presented in Equation (2.1.43) and provides a warning if it is not satisfied. Note that this is the same function used for output feedback systems as well.

Once the controller has been completely specified by these functions, it may be simulated with the `sim(obj, r, times, varargin)` function, whose inputs are the function $r(t)$, a two element vector containing the start and stop times of the simulation, and a variable number of inputs representing the graphs to generate. Each one of the variable inputs is a string and corresponds to a Matlab figure. This string contains identifiers representing the signals

in the simulation that the user wishes to overlay on the same graph. The list of allowable identifiers and the signal they represent is presented in Table 2.1. As many identifiers as desired may be placed in any one string, and identifiers may be repeated in other strings. In addition, the user may provide as many strings as desired. Note, however, that since each string creates a separate figure, there is a practical limit to the number of strings that should be provided based on the user's computer's capabilities. It should also be noted that the user may specify three outputs from the `sim` function which are the trajectories of every internal state in the entire closed-loop system, the set of times used by the differential equation solver, and the ordering of these internal states.

2.2.2 Sampling Period Calculations

The relationship between the sampling period T and the error bound γ_x has already been established by Theorem 2.1.4. Lemma 2.1.5 guarantees that there exists a T small enough to guarantee any error bound. Given these statements, two obvious questions arise:

1. Given the sampling period of the CPU, is the closed-loop system guaranteed to be stable, and if so, what error bound is guaranteed?

2. Given a desired error bound, how small does the sampling period need to be to guarantee this bound?

The second question may be answered by using the provided error bound to calculate $\bar{\gamma}_0$ and evaluating $\gamma_0(T)$ over a window of values of T and comparing to $\bar{\gamma}_0$. If a suitable value of T is not found, then we may slide the window to search for an appropriate value of T. Interestingly enough, the first question is actually more difficult to answer since it is only possible to determine ρ_r as a function of $\bar{\gamma}_x$, when only T and the controller are specified. However, if a value of $\bar{\gamma}_x$ is supplied as well as T, then the actual achieved error bound, γ_x, may be easily computed. The calculations used are summarized below.

The first step in either calculation is to find the value of ρ_r from the provided value of $\bar{\gamma}_x$. From Equation (2.1.17), we see that

$$L_{i\rho_r}\rho_r = M(\rho_r)K_i\,,$$

where

$$M(\rho_r) = \max\{\rho_r + \bar{\gamma}_x, L_z(\rho_r + \bar{\gamma}_x) + B_z\} \tag{2.2.1}$$

is written with ρ_r as an argument explicitly to emphasize their relationship. Using this, we may rewrite the $\mathcal{L}_1$-norm condition from Equation (2.1.19) as

$$\begin{aligned}
&\|G_m(s)\|_{\mathcal{L}_1}(M(\rho_r)K_1 + B_1) + \|G_{um}(s)\|_{\mathcal{L}_1}(M(\rho_r)K_2 + B_2) \\
&\qquad + \|H_{xm}(s)C(s)k_g\|_{\mathcal{L}_1}\|r\|_{\mathcal{L}_\infty} + \rho_{ic} = \\
&\|G_m(s)\|_{\mathcal{L}_1}B_1 + \|G_{um}(s)\|_{\mathcal{L}_1}B_2 + \|H_{xm}(s)C(s)k_g\|_{\mathcal{L}_1}\|r\|_{\mathcal{L}_\infty} + \rho_{ic} \\
&\qquad + \left(\|G_m(s)\|_{\mathcal{L}_1}K_1 + \|G_{um}(s)\|_{\mathcal{L}_1}K_2\right)M(\rho_r) \triangleq \\
&\qquad\qquad c_1 + c_2M(\rho_r) < \rho_r
\end{aligned}$$

Note that the way c_1 and c_2 are defined here ensures that they are not dependent on ρ_r and are therefore known constants once the system and its reference input have been specified. Then the only unknown in this inequality is ρ_r, and we may attempt to solve this equation. Since $\bar{\gamma}_x$ and B_z are positive, then if $L_z \geq 1$, $M = L_z(\rho_r + \bar{\gamma}_x) + B_z$, for any $\rho_r > 0$. However, if $L_z < 1$, then the graph of M in terms of ρ_r is similar to Figure 2.1, which additionally displays examples of the function

$$M = \frac{\rho_r - c_1}{c_2}. \tag{2.2.2}$$

Note that for the $\mathcal{L}_1$-norm condition to hold, we must have $c_2 < 1$, thus the slope of the line in Equation (2.2.2) must be greater than 1, and therefore, greater than the slope of the lines in (2.2.1). Combined with the fact that the y-intercept of Equation (2.2.1) is positive and the y-intercept of Equation (2.2.2) is negative, then there is guaranteed to be an intersection of the two equations for some value of $\rho_r > 0$. To find this intersection point, define ρ_1 and ρ_2 as the values where Equation (2.2.2) intersects with the lines $M = \rho_r + \bar{\gamma}_x$

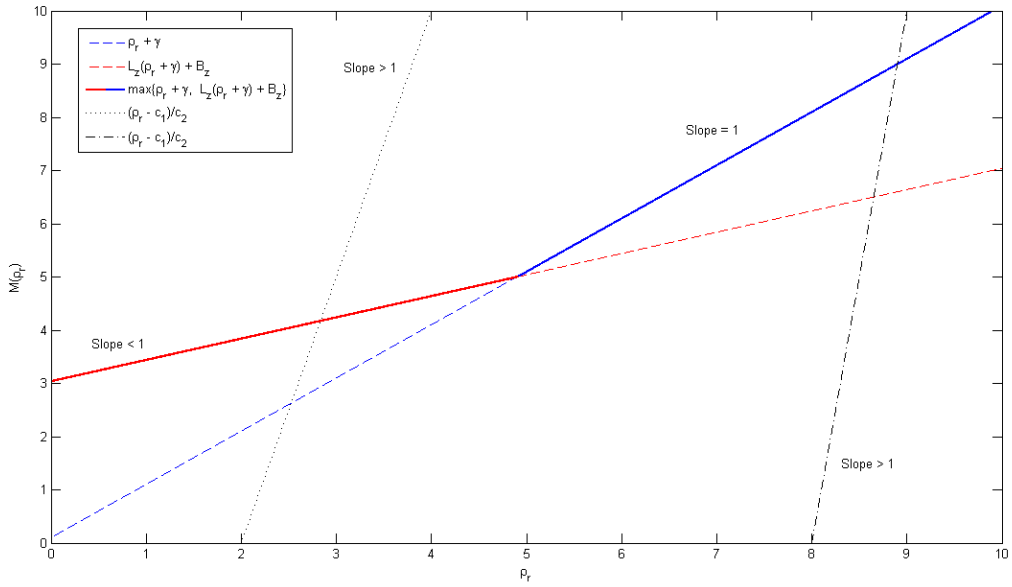

Figure 2.1: Relationships between M and ρ_r. The solid line is the definition of $M(\rho_r)$, while the red and blue dashed lines are the continuations of the line segments of M, and the black lines represent possible graphs of Equation (2.2.2).

and $M = L_z(\rho_r + \bar{\gamma}_x) + B_z$, respectively. Solving for these values yields

$$\begin{aligned} \rho_1 &= \frac{c_1 + c_2\bar{\gamma}_x}{1 - c_2}\,, \\ \rho_2 &= \frac{c_1 + c_2 L_z\bar{\gamma}_x + c_2 B_z}{1 - c_2 L_z}\,. \end{aligned} \tag{2.2.3}$$

If $\rho_1 < \rho_2$, then Equation (2.2.2) intersects $M = L_z(\rho_r + \bar{\gamma}_x) + B_z$ at a higher y value than $M = \rho_r + \bar{\gamma}_x$, and therefore, for this value of ρ_r, $M(\rho_r) = L_z(\rho_r + \bar{\gamma}_x) + B_z$. Otherwise, $M(\rho_r) = \rho_r + \bar{\gamma}_x$. Note that $\rho_1 < \rho_2$ is equivalent to

$$\bar{\gamma}_x < \frac{B_z(1 - c_2)}{1 - L_z} - c_1\,. \tag{2.2.4}$$

Therefore, this leads to the following equation for ρ_r in terms of $\bar{\gamma}_x$:

$$\rho_r = \begin{cases} \frac{c_1 + c_2 L_z\bar{\gamma}_x + c_2 B_z}{1 - c_2 L_z}, & \bar{\gamma}_x < \frac{B_z(1-c_2)}{1-L_z} - c_1 \\ \frac{c_1 + c_2\bar{\gamma}_x}{1 - c_2}, & \text{otherwise} \end{cases} \tag{2.2.5}$$

Now that ρ_r is known, Equations (2.1.20)–(2.1.24) and (2.1.35)–(2.1.36) can be used to calculate Δ_i and all the error bounds.

The L1Controller class provides the function `calcErrorBound(obj, r_bound, gammaxbar)` to determine the actual error bound γ_x. The above calculation is used to determine ρ_r using the provided `gammaxbar` as $\bar{\gamma}_x$ and `r_bound` as the $\mathcal{L}_\infty$-norm bound on the reference input. The sampling period T is specified in the object and Equations (2.1.37)–(2.1.42) can be used to determine $\gamma_0(T)$, which is then compared to the value of $\bar{\gamma}_0$ chosen in Equation (2.1.21). If $\gamma_0(T) < \bar{\gamma}_0$, then the system is stable and the error bound γ_x is returned. Otherwise, the system cannot achieve the error bound requested and the user must provide a larger value for `gammaxbar` or a smaller value for T.

Similarly, the function `calcMaxTs(obj, error_bound, r_bound)` uses the provided error bound as both $\bar{\gamma}_x$ and γ_x and the provided bound on $r(t)$ to calculate Δ_i, $i = 1, 2$. Then the function $\gamma_0(T)$ is calculated for a range of T values and is compared to $\bar{\gamma}_0$. The function searches for the value $T_{\max}$ that makes $\gamma_0(T) < \bar{\gamma}_0 \quad \forall T < T_{\max}$. The search is performed as follows. The algorithm begins by calculating $\gamma_0(T)$ for 1001 values of T, evenly spaced from 0 up to T_{win}, which is initially 1 ms. Then it searches this vector of $\gamma_0(T)$ values for the smallest value T_0 that makes $\gamma_0(T_0) \geq \bar{\gamma}_0$. Then let the estimate of $T_{\max}$ be called $\hat{T}_{\max} = T_0 - (T_{win}/1000)$. If $T_{win}/10 \leq \hat{T}_{\max} < T_{win}$, then $\hat{T}_{\max}$ is accurate to within 1% of the true value, and the program finishes. If the estimate is not in that range, then it updates T_{win} with a new value $T_{win,new}$ according to Equation (2.2.6), shown below, recalculates $\gamma_0(T)$ for 1001 values evenly spaced from 0 to $T_{win,new}$ and repeats the search. In this way the search repeatedly alters the window size, T_{win}, until an appropriate value of $T_{\max}$ can be found.

$$T_{win,new} = \begin{cases} \frac{T_{win}}{1000}\,, & \hat{T}_{\max} = 0 \\ 2\hat{T}_{\max}\,, & 0 < \hat{T}_{\max} < \frac{T_{win}}{10} \\ 100T_{win}\,, & \hat{T}_{\max} = T_{win} \end{cases} . \tag{2.2.6}$$

CHAPTER 3

OUTPUT FEEDBACK

3.1 Mathematical Preliminaries

The general form of the class of systems that can be stabilized by an output feedback $\mathcal{L}_1$ adaptive controller is the following:

$$y(s) = A(s)\left(u(s) + d(s)\right) , \tag{3.1.1}$$

where $y(t) \in \mathbb{R}$, $u(t) \in \mathbb{R}$, $A(s)$ is an unknown SISO strictly proper transfer function, and $d(t) = f(t, y(t))$, where f is an unknown function representing all the (possibly nonlinear) uncertainties in the plant, subject to the following assumptions:

Assumption 3.1.1 *There exist constants $L > 0$ and $L_0 > 0$, such that the following inequalities hold uniformly in t:*

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2| , \qquad |f(t, y)| \leq L|y| + L_0 .$$

Assumption 3.1.2 *There exist constants $L_1 > 0$, $L_2 > 0$, and $L_3 > 0$, such that for all $t \geq 0$*

$$|\dot{d}(t)| \leq L_1|\dot{y}(t)| + L_2|y(t)| + L_3 .$$

The values L, L_0, L_1, L_2, and L_3 here can be arbitrarily large. Just as with state feedback, the basic outline of the $\mathcal{L}_1$ adaptive controller is to first obtain estimates of the uncertainties, generate the input for the plant that would ideally cancel all of the uncertainties, and send

it through a SISO low pass filter $C(s)$ before using it as the input $u(t)$ for the plant. Again, this filter ensures that the control signal only tries to cancel the uncertainties within the bandwidth of the control channel, and prevents any high frequencies that result from the estimation scheme from entering the plant.

The goal of the output feedback $\mathcal{L}_1$ adaptive controller is to have the closed-loop system act like a minimum-phase, strictly proper, linear time-invariant transfer function, $M(s)$. Thus, given a reference input $r(t)$, the goal is to have $y(s) \approx M(s)r(s)$. In light of this, we define

$$\sigma(s) = \frac{(A(s) - M(s))\, u(s) + A(s)d(s)}{M(s)}\,, \tag{3.1.2}$$

which allows us to rewrite (3.1.1) as

$$y(s) = M(s)\big(u(s) + \sigma(s)\big)\,. \tag{3.1.3}$$

From this form, it is clear that if we can obtain accurate estimates of $\sigma(t)$, which will be called $\hat{\sigma}(t)$, then we should be able to approximately achieve our goal using the following control law:

$$u(s) = C(s)\big(r(s) - \hat{\sigma}(s)\big)\,, \tag{3.1.4}$$

where $C(s)$ needs to be a strictly proper SISO filter with $C(0) = 1$ that ensures that

$$H(s) = \frac{A(s)M(s)}{C(s)A(s) + (1 - C(s))M(s)} \tag{3.1.5}$$

is stable and that

$$\|G(s)\|_{\mathcal{L}_1} L < 1\,, \tag{3.1.6}$$

where $G(s) = H(s)\,(1 - C(s))$. In addition, we define the following:

$$H_0(s) = \frac{A(s)}{C(s)A(s) + (1 - C(s))M(s)}\,, \tag{3.1.7}$$

$$H_1(s) = \frac{(A(s) - M(s))C(s)}{C(s)A(s) + (1 - C(s))M(s)}\,, \tag{3.1.8}$$

$$H_2(s) = C(s)H_0(s)\,, \tag{3.1.9}$$

$$H_3(s) = -\frac{M(s)C(s)}{C(s)A(s) + (1 - C(s))M(s)}\,. \tag{3.1.10}$$

Just as with state feedback, we can create the reference system that the closed-loop adaptive system should track merely by assuming that the estimates are exactly correct.

Reference System:

$$\begin{aligned} y_{ref}(s) &= M(s)\big(u_{ref}(s) + \sigma_{ref}(s)\big)\,, \\ u_{ref}(s) &= C(s)\big(r(s) - \sigma_{ref}(s)\big)\,, \\ \sigma_{ref}(s) &= \frac{(A(s) - M(s))\,u_{ref}(s) + A(s)d_{ref}(s)}{M(s)}\,, \end{aligned} \tag{3.1.11}$$

where $d_{ref}(t) = f(t, y_{ref}(t))$. From this, one can derive

$$y_{ref}(s) = H(s)\,(C(s)r(s) + (1 - C(s))d_{ref}(s))\,,$$

which leads to the following lemma, first proved in [11]:

Lemma 3.1.1 *If $C(s)$ and $M(s)$ verify the condition in* (3.1.6)*, the closed-loop reference system in* (3.1.11) *is bounded-input, bounded-output (BIBO) stable.*

We must also define the following:

$$\begin{aligned} \Delta = {} & \|H_1(s)\|_{\mathcal{L}_1} \|r\|_{\mathcal{L}_\infty} + \|H_0(s)\|_{\mathcal{L}_1} (L\rho + L_0) \\ & + \bar{\gamma} \left(\|H_1(s)/M(s)\|_{\mathcal{L}_1} + L \|H_0(s)\|_{\mathcal{L}_1} \frac{\|H_2(s)\|_{\mathcal{L}_1}}{1 - \|G(s)\|_{\mathcal{L}_1} L} \right)\,, \end{aligned} \tag{3.1.12}$$

where $\bar{\gamma} > 0$ is an arbitrary constant and

$$\rho = \frac{\|H(s)C(s)\|_{\mathcal{L}_1} \|r\|_{\mathcal{L}_\infty} + \|G(s)\|_{\mathcal{L}_1} L_0}{1 - \|G(s)\|_{\mathcal{L}_1} L} \,. \tag{3.1.13}$$

The issue that has yet to be addressed, however, is how to obtain the estimate $\hat{\sigma}(t)$. Similar to the state feedback case, there are two different types of adaptive laws available to us: gradient descent and piecewise constant. However, unlike state feedback, there are restrictions on the choices of $M(s)$ that may be used with the gradient descent law. These two laws, and the concerns that arise with each will be covered next.

3.1.1 The Gradient Descent Adaptive Law

The gradient descent adaptive law can only be used when the desired model $M(s)$ is strictly positive real (SPR). For simplicity, we shall assume a first order model with DC gain 1, $M(s) = \frac{m}{s+m}$ where $m > 0$. We may then define the remainder of the $\mathcal{L}_1$ adaptive controller.

State Predictor:

$$\dot{\hat{y}}(t) = -m\hat{y}(t) + m\left(u(t) + \hat{\sigma}(t)\right) \,, \quad \hat{y}(0) = 0 \,, \tag{3.1.14}$$

Adaptive Law:

$$\dot{\hat{\sigma}}(t) = \Gamma \mathrm{Proj}\left(\hat{\sigma}(t), -\tilde{y}(t)\right) \,, \quad \hat{\sigma}(0) = 0 \,, \tag{3.1.15}$$

where $\tilde{y}(t) = \hat{y}(t) - y(t)$, Γ is the adaptive gain, and the projection bound is $|\hat{\sigma}(t)| \leq \Delta$, where Δ was defined in Equation (3.1.12). Then we get the following performance bounds, first presented and proven in [11].

Theorem 3.1.1 *If Γ is sufficiently large, then the closed-loop system specified by Equations* (3.1.1), (3.1.4) *and* (3.1.14)–(3.1.15), *subject to the $\mathcal{L}_1$-norm condition in Equation* (3.1.6), *satisfies the following bounds:*

$$\|\tilde{y}\|_{\mathcal{L}_\infty} < \bar{\gamma} \,,$$
$$\|y - y_{ref}\|_{\mathcal{L}_\infty} \leq \gamma_1 \,, \quad \|u - u_{ref}\|_{\mathcal{L}_\infty} \leq \gamma_2 \,,$$

where $\bar{\gamma}$ was defined in Equation (3.1.12),

$$\gamma_1 = \frac{\|H_2(s)\|_{\mathcal{L}_1}}{1 - \|G(s)\|_{\mathcal{L}_1} L} \bar{\gamma} \,,$$

and

$$\gamma_2 = L \|H_2(s)\|_{\mathcal{L}_1} \gamma_1 + \left\| \frac{H_3(s)}{M(s)} \right\|_{\mathcal{L}_1} \bar{\gamma} \,.$$

3.1.2 The Piecewise Constant Adaptive Law

The piecewise constant adaptive law is necessary when the model $M(s)$ is not SPR, and therefore, the gradient descent adaptive law cannot be used. However, the piecewise constant law is also applicable whenever the gradient descent law is applicable, making the piecewise constant available to a wider class of systems. We assume that $M(s)$ is strictly proper with relative degree d_r. In addition, $A(s)$ has an unknown relative degree n_r, for which only a known lower bound, $n_r \geq d_r$, is available. The same control law, (3.1.4), is still used in this case, but now $C(s)$ must be chosen to have relative degree d_r, in order to ensure that (3.1.7)–(3.1.10) are all proper.

Let (A_m, b_m, c_m) be the minimal state-space realization of $M(s)$. Therefore, (A_m, b_m) is controllable and (A_m, c_m) is observable. Then we may write the state predictor of the $\mathcal{L}_1$ adaptive controller:

State Predictor:

$$\begin{aligned} \dot{\hat{x}}(t) &= A_m \hat{x}(t) + b_m u(t) + \hat{\sigma}(t) \,, \\ \hat{y}(t) &= c_m^\top \hat{x}(t) \,, \end{aligned} \tag{3.1.16}$$

where even though $\sigma(t) \in \mathbb{R}$ is matched, $\hat{\sigma}(t) \in \mathbb{R}^n$ is unmatched.

Since $M(s)$ is stable, then A_m is Hurwitz, and for any positive definite matrix Q, there

exists $P = P^\top > 0$ that solves the Lyapunov equation

$$A_m^\top P + P A_m = -Q\,.$$

Since P is positive definite, there exists non-singular $\sqrt{P}$ such that

$$P = \left(\sqrt{P}\right)^\top \sqrt{P}\,.$$

We may then define $D \in \mathbb{R}^{(n-1)\times n}$ as a matrix containing a basis for the null space of the vector $c_m^\top(\sqrt{P})^{-1}$, meaning that

$$D\left(c_m^\top(\sqrt{P})^{-1}\right)^\top = 0\,.$$

Then define $\Lambda \in \mathbb{R}^{n\times n}$ as

$$\Lambda = \begin{bmatrix} c_m^\top(\sqrt{P})^{-1} \\ D \end{bmatrix}\sqrt{P} = \begin{bmatrix} c_m^\top \\ D\sqrt{P} \end{bmatrix},$$

which is non-singular since it is the product of two non-singular matrices. Hence Λ^{-1} exists.

The idea behind the piecewise constant adaptive law is to sample the error signal $\tilde{y}(t) = \hat{y}(t) - y(t)$ with some constant sampling period, $T > 0$, which can be thought of as the sampling period of the CPU. Then, rather than attempt to estimate $\sigma(t)$, instead calculate the value of $\hat{\sigma}(t)$ that, when applied constantly over the next sampling period, will eliminate the effect of the current error $\tilde{y}(iT)$ on the subsequent sampled error, $\tilde{y}((i+1)T)$. By doing so, we can keep the error signal bounded.

Adaptive Law:

$$\begin{aligned} \hat{\sigma}(t) &= \hat{\sigma}(iT)\,, \quad \forall\ t \in [iT,\ (i+1)T)\,, \\ \hat{\sigma}(iT) &= -\Phi^{-1}(T)e^{\Lambda A_m \Lambda^{-1} T}\mathbf{1}_1\tilde{y}(iT)\,, \quad i = 0, 1, 2, 3, \dots\,, \end{aligned} \tag{3.1.17}$$

where $\tilde{y}(t) = \hat{y}(t) - y(t)$, $\mathbf{1}_1 = [1,\ 0,\ \ldots,\ 0]^\top \in \mathbb{R}^n$, and

$$\Phi(T) = \int_0^T e^{\Lambda A_m \Lambda^{-1}(T-\tau)} \Lambda \, d\tau \,. \tag{3.1.18}$$

It is clear that for very large values of T, the estimates will not update often, thus severely hampering the ability of the control law to regulate the system effectively and potentially allowing the closed-loop system to become unstable. This implies that there is some sort of upper bound on the choice of T that could guarantee closed-loop stability. This notion is formalized below.

Let $\eta_1(t) \in \mathbb{R}$ and $\eta_2(t) \in \mathbb{R}^{n-1}$ be defined as

$$\left[\eta_1(t)\,, \quad \eta_2^\top(t)\right] = \mathbf{1}_1^\top e^{\Lambda A_m \Lambda^{-1} t} \,. \tag{3.1.19}$$

Additionally, let

$$\kappa(T) = \int_0^T |\mathbf{1}_1^\top e^{\Lambda A_m \Lambda^{-1}(T-\tau)} \Lambda b_m| \, d\tau \,, \tag{3.1.20}$$

$$\varsigma(T) = \|\eta_2(T)\|_2 \sqrt{\frac{\alpha}{\lambda_{\max}(P_2)}} + \kappa(T)\Delta \,, \tag{3.1.21}$$

$$\alpha = \lambda_{\max}\left(\Lambda^{-\top} P \Lambda^{-1}\right) \left(\frac{2\Delta \left\| \Lambda^{-\top} P b_m \right\|_2}{\lambda_{\min}\left(\Lambda^{-\top} Q \Lambda^{-1}\right)} \right)^2 \,, \tag{3.1.22}$$

where $P_2 = (DD^\top)^{-1} > 0$. Now let

$$\begin{aligned} \beta_1(T) &= \max_{t \in [0,\ T]} |\eta_1(t)| \,, \quad \beta_2(T) = \max_{t \in [0,\ T]} \|\eta_2(t)\|_2 \,, \\ \beta_3(T) &= \max_{t \in [0,\ T]} \eta_3(t) \,, \quad \beta_4(T) = \max_{t \in [0,\ T]} \eta_4(t) \,, \end{aligned} \tag{3.1.23}$$

where

$$\begin{aligned} \eta_3(t) &= \int_0^t |\mathbf{1}_1^\top e^{\Lambda A_m \Lambda^{-1}(t-\tau)} \Lambda \Phi^{-1}(T) e^{\Lambda A_m \Lambda^{-1} T} \mathbf{1}_1| \, d\tau \,, \\ \eta_4(t) &= \int_0^t |\mathbf{1}_1^\top e^{\Lambda A_m \Lambda^{-1}(t-\tau)} \Lambda b_m| \, d\tau \,. \end{aligned} \tag{3.1.24}$$

Finally, let

$$\gamma_0(T) = \beta_1(T)\varsigma(T) + \beta_2(T)\sqrt{\frac{\alpha}{\lambda_{\max}(P_2)}} + \beta_3(T)\varsigma(T) + \beta_4(T)\Delta\,. \tag{3.1.25}$$

The following lemma and theorem were proven in [13], and are presented here without proof.

Lemma 3.1.2

$$\lim_{T\to 0}\gamma_0(T) = 0$$

Theorem 3.1.2 *Given the system in* (3.1.1)*, and the* $\mathcal{L}_1$ *adaptive controller in* (3.1.4)*,* (3.1.16)*, and* (3.1.17)*, subject to the constraint* (3.1.6)*, if we choose* T *to ensure that*

$$\gamma_0(T) < \bar{\gamma}\,, \tag{3.1.26}$$

where $\bar{\gamma}$ *was defined in* (3.1.12) *then the following are true:*

$$\|\tilde{y}\|_{\mathcal{L}_\infty} < \bar{\gamma}\,,$$
$$\|y - y_{ref}\|_{\mathcal{L}_\infty} < \gamma_1\,, \quad \|u - u_{ref}\|_{\mathcal{L}_\infty} < \gamma_2\,,$$

where

$$\gamma_1 = \frac{\|H_2(s)\|_{\mathcal{L}_1}}{1 - \|G(s)\|_{\mathcal{L}_1} L}\bar{\gamma}\,, \tag{3.1.27}$$

and

$$\gamma_2 = L\,\|H_2(s)\|_{\mathcal{L}_1}\,\gamma_1 + \left\|\frac{H_3(s)}{M(s)}\right\|_{\mathcal{L}_1}\bar{\gamma}\,.$$

Note that Lemma 3.1.2 implies that by picking T small enough, we can make $\gamma_0(T)$ arbitrarily small. Then, by Theorem 3.1.2, we obtain the error bounds for the output y and the input u. Thus, these error bounds can be made arbitrarily small by reducing T.

3.2 Toolbox Overview

3.2.1 User Interface

The process of specifying the closed-loop $\mathcal{L}_1$ adaptive control system to be simulated can be expressed in five steps:

1. Specify the plant $A(s)$ and desired model $M(s)$.
2. Decide if the adaptive law will be the piecewise constant law or the gradient descent law.
3. Specify the disturbance $d(t) = f(t, y(t))$, and provide known bounds such as the Lipschitz constants, and if necessary, the projection bounds and the initial estimate $\hat{\sigma}(0)$.
4. Specify $C(s)$.
5. Specify the sampling period T, if necessary.

With these five steps, the closed loop system is specified as (3.1.1), (3.1.4) and then either (3.1.14) and (3.1.15) or (3.1.16) and (3.1.17), based on which type of adaptive law is chosen. The $\mathcal{L}_1$ Adaptive Control Toolbox uses this process to build the simulation of the closed-loop system and, in the `L1Controller` class, provides a separate function for each of the steps. This section shall cover these functions and how they are used.

The function `setOutputFeedbackPlantModel` comprises the first step and can be called in one of four ways:

1. `setOutputFeedbackPlantModel(obj, A, M)`,
2. `setOutputFeedbackPlantModel(obj, An, Ad, M)`,
3. `setOutputFeedbackPlantModel(obj, A, Mn, Md)`,
4. `setOutputFeedbackPlantModel(obj, An, Ad, Mn, Md)`,

where `obj` is the object of the `L1Controller` class that is being modified, the variables `A` and `M` are transfer functions variables provided by the Matlab Control System Toolbox, and the extra characters `n` and `d` represent that the variables are vectors of real numbers representing the coefficients of the numerator or the denominator, respectively, in order from the highest power of s to the constant term. The function then ensures that the assumptions on $A(s)$ and $M(s)$ specified in Section 3.1 hold, and saves the variables internally.

The type of adaptive law can be specified by `usePiecewiseConstantAdaptiveLaw(obj)` or `useGradientDescentAdaptiveLaw(obj)`. They each set a flag internally that modifies the implementation of subsequent functions. This is the primary reason that these two functions must be called at this point, instead of later in the process. Note also that these are the same two functions used for state feedback as well.

The function `addOutputFeedbackNonlinearity(obj, trueval, L, L0, gamma, bound, IC)` adds the $d(t)$ term into Equation (3.1.1), where the function $f(t, y(t))$ is specified by the input `trueval`. The Lipschitz constants for $f(t, y(t))$ are then specified by `L` and `L0`. Note that while there are three more Lipschitz constants, L_1, L_2, and L_3, these are only necessary for the analysis and need not be specified. The final three inputs are only necessary when the gradient descent adaptive law is used. They specify the value of Γ, the projection bounds, and the initial estimate $\hat{\sigma}(0)$, respectively. This function then uses the provided inputs to create the appropriate adaptive law and stores this law internally.

The filter $C(s)$ can be specified with the function `setCs(obj, num, den)`, where `num` and `den` are vectors of the numerator's and denominator's coefficients, respectively. Similar to `setOutputFeedbackPlantModel`, however, `setCs` can also be called with a transfer function variable in place of the two coefficient vectors. Either way, a minimal state-space representation of $C(s)$ is found and stored internally. At this point, the system is completely specified, with the possible exception of the sampling period, T. Therefore, this function checks the most important requirement in an $\mathcal{L}_1$ adaptive controller: the $\mathcal{L}_1$-norm condition from Equation (3.1.6). If it is not satisfied, a warning is presented to the user that the closed-loop adaptive system is not guaranteed to be stable. While theoretically, $C(s)$ could be specified prior to the adaptive law, it is anticipated that most of the tuning the user will perform when creating an $\mathcal{L}_1$ adaptive controller will take place in $C(s)$. Therefore, it

is assumed that this function will be called last, and thus the verification of the $\mathcal{L}_1$-norm condition is performed here. Again, the only possible exception is that the sampling period, T, will have not been specified yet, but since T does not appear in Equation (3.1.6), it is not beneficial to wait until T is specified to check the condition. Finally, note that `setCs` is the same function used for state feedback as well.

Finally, the function `setSamplingPeriod(obj, Ts)` is used in the case of the piecewise constant adaptive law to specify the sampling period, T. Calling this function when the gradient descent adaptive law is in use produces an error. In addition to storing the sampling period, this function checks the stability condition on T presented in Equation (3.1.26) and provides a warning if it is not satisfied. Note that this is the same function used for state feedback systems as well.

Once the controller has been completely specified by these functions, it may be simulated with the `sim(obj, r, times, varargin)` function, whose inputs are the function $r(t)$, a two element vector containing the start and stop times of the simulation, and a variable number of inputs representing the graphs to generate. Each one of the variable inputs is a string and corresponds to a Matlab figure. This string contains identifiers representing the signals in the simulation that the user wishes to overlay on the same graph. The list of allowable identifiers and the signal they represent is presented in Table 2.1 on page 21. As many identifiers as desired may be placed in any one string, and identifiers may be repeated in other strings. In addition, the user may provide as many strings as desired. Note, however, that since each string creates a separate figure, there is a practical limit to the number of strings that should be provided based on the user's computer's capabilities. It should also be noted that the user may specify three outputs from the `sim` function which are the trajectories of every internal state in the entire closed-loop system, the set of times used by the differential equation solver, and the ordering of these internal states.

3.2.2 Sampling Period Calculations

The relationship between the sampling period T and the error bound γ_1 has already been established by Theorem 3.1.2. Lemma 3.1.2 guarantees that there exists a T small enough

to guarantee any error bound. Given these statements, two obvious questions arise:

1. Given the sampling period of the CPU, is the closed-loop system guaranteed to be stable, and if so, what error bound is guaranteed?

2. Given a desired error bound, how small does the sampling period need to be to guarantee this bound?

The first question is relatively straightforward to answer, though complicated slightly by the inclusion of $\bar{\gamma}$ in (3.1.12), which is used often in the equations leading up to (3.1.25). However, calculating a value for T that answers the second question is considerably more complicated and finding a solution analytically would be difficult. The $\mathcal{L}_1$ Adaptive Control Toolbox answers the first question by providing an algorithm to efficiently calculate the function $\gamma_0(T)$. Then to answer the second question, γ_0 may be evaluated over a narrow window of values of T followed by sliding the window to search for an appropriate value of T. The method of calculating γ_0 more efficiently is presented first, followed by a more detailed explanation of the search for T.

The key to calculating γ_0 more efficiently is to think of it as a function of two variables, $\gamma_0(T, \bar{\gamma})$, and rewriting all of its components in a similar way. In this way, we define

$$c_1 = \|H_1(s)\|_{\mathcal{L}_1} \|r\|_{\mathcal{L}_\infty} + \|H_0(s)\|_{\mathcal{L}_1} (L\rho + L_0), \tag{3.2.1}$$

$$c_2 = \|H_1(s)/M(s)\|_{\mathcal{L}_1} + L \|H_0(s)\|_{\mathcal{L}_1} \frac{\|H_2(s)\|_{\mathcal{L}_1}}{1 - \|G(s)\|_{\mathcal{L}_1} L}, \tag{3.2.2}$$

which allows (3.1.12) to be rewritten as

$$\Delta(\bar{\gamma}) = c_1 + c_2 \bar{\gamma}. \tag{3.2.3}$$

By defining

$$c_3 = \lambda_{\max}\left(\Lambda^{-\top} P \Lambda^{-1}\right) \left(\frac{2 \left\| \Lambda^{-\top} P b_m \right\|_2}{\lambda_{\min}\left(\Lambda^{-\top} Q \Lambda^{-1}\right)} \right)^2,$$

we can rewrite (3.1.22) as

$$\alpha(\bar{\gamma}) = c_3 \left(\Delta(\bar{\gamma})\right)^2. \tag{3.2.4}$$

Similarly,

$$c_4(T) = \frac{\|\eta_2(T)\|_2}{\sqrt{\lambda_{\max}(P_2)}}$$

transforms (3.1.21) into

$$\varsigma(T, \bar{\gamma}) = c_4(T)\sqrt{\alpha(\bar{\gamma})} + \kappa(T)\Delta(\bar{\gamma}) = \left(c_4(T)\sqrt{c_3} + \kappa(T)\right)\Delta(\bar{\gamma})\,, \tag{3.2.5}$$

and

$$c_5 = \frac{1}{\sqrt{\lambda_{\max}(P_2)}}$$

yields an alternate version of (3.1.25):

$$\begin{aligned}\gamma_0(T, \bar{\gamma}) &= \beta_1(T)\varsigma(T) + \beta_2(T)c_5\sqrt{\alpha(\bar{\gamma})} + \beta_3(T)\varsigma(T) + \beta_4(T)\Delta(\bar{\gamma})\,, \\ &= \big((\beta_1(T) + \beta_3(T))(c_4(T)\sqrt{c_3} + \kappa(T)) + \beta_4(T) + \beta_2(T)c_5\sqrt{c_3}\big)\Delta(\bar{\gamma})\,, \\ &\triangleq k(T)\Delta(\bar{\gamma}) = k(T)(c_1 + c_2\bar{\gamma})\,.\end{aligned} \tag{3.2.6}$$

This separation of variables is key to this algorithm as it reduces the computational complexity to merely calculating $k(T)$. From this, the stability requirement from Equation (3.1.26), becomes

$$k(T)(c_1 + c_2\bar{\gamma}) < \bar{\gamma}\,, \tag{3.2.7}$$

or

$$k(T)c_1 < (1 - c_2k(T))\bar{\gamma}\,. \tag{3.2.8}$$

Therefore, we obtain the following corollary to Theorem 3.1.2:

Corollary 3.2.1 *Given the system in* (3.1.1)*, and the* $\mathcal{L}_1$ *adaptive controller in* (3.1.4)*,* (3.1.16)*, and* (3.1.17)*, subject to the constraint* (3.1.6)*, the closed-loop system is BIBO stable if* $c_2k(T) < 1$*.*

Proof Due to the norms inside the integrals, for any finite $T > 0$, then $\beta_1(T)$, $\beta_2(T)$, $\beta_3(T)$, $\beta_4(T)$, and $\kappa(T)$ are all positive and finite. Additionally, since $P_2 > 0$, $\lambda_{\max}(P_2) > 0$, and then $c_4(T)$ and c_5 are both positive and finite. Since Q and Λ are both non-singular,

$\Lambda^{-\top}Q\Lambda^{-1}$ is non-singular, $\lambda_{\min}(\Lambda^{-\top}Q\Lambda^{-1}) \neq 0$, and c_3 is positive and finite. Therefore, $k(T)$ exists and is positive and finite.

From Equations (3.1.7)–(3.1.9), $H_0(s)$ and $H_2(s)$ are stable and proper, and $H_1(s)$ is stable and strictly proper with relative degree d_r. Since $M(s)$ is required to be minimum-phase, stable and strictly proper with relative degree d_r, then $H_1(s)/M(s)$ is stable and proper. This, combined with the requirement in Equation (3.1.6), proves that all the $\mathcal{L}_1$ norms in (3.1.13), (3.2.1), and (3.2.2) exist. By assumption, r is bounded, and therefore, c_1 and c_2 are positive and finite.

Thus, the left-hand side of (3.2.8) is always positive. Then if $c_2k(T) < 1$, $\bar{\gamma}$ may be chosen so that $\bar{\gamma} > \frac{k(T)c_1}{(1-c_2k(T))}$. The derivation of (3.2.8) proves that this choice of $\bar{\gamma}$ will satisfy (3.1.26), and by Theorem 3.1.2, y is bounded. □

The `L1Controller` class provides the function `calcErrorBound(obj, r_bound)` which uses the above corollary and Equation (3.2.8) to calculate γ_1, the bound on $y - y_{ref}$ in Theorem 3.1.2. It first calculates c_1, c_2, and $k(T)$ using the bound on r provided by `r_bound` and the stored value of T previously provided by the user and then checks if $c_2k(T) < 1$. If it is true, then it assigns $\bar{\gamma} = \frac{k(T)c_1}{(1-c_2k(T))}(1+\epsilon)$ for some very small $\epsilon > 0$ and calculates γ_1 according to Equation (3.1.27). If $c_2k(T) \geq 1$, then the function returns $\gamma_1 = \infty$ to represent the possibility of instability.

Similarly, the function `calcMaxTs(obj, error_bound, r_bound)` uses the input `error_bound` as γ_1 and the provided bound on r to calculate $\bar{\gamma}$, c_1, c_2, c_3 and c_5 before calculating the components of $k(T)$ that depend on T. Then, it performs a search for the value $T_{\max}$ that makes $k(T) < \frac{\bar{\gamma}}{c_1+c_2\bar{\gamma}}$, $\quad \forall T < T_{\max}$. The search is performed as follows. The algorithm begins by calculating $k(T)$ for 1001 values of T, evenly spaced from 0 up to T_{win}, which is initially 1 ms. Then it searches this vector of $k(T)$ values for the smallest value T_0 that makes $k(T_0) \geq \frac{\bar{\gamma}}{c_1+c_2\bar{\gamma}}$. Then let the estimate of $T_{\max}$ be called $\hat{T}_{\max} = T_0 - (T_{win}/1000)$. If $T_{win}/10 \leq \hat{T}_{\max} < T_{win}$, then $\hat{T}_{\max}$ is accurate to within 1% of the true value, and the program finishes. If the estimate is not in that range, then it updates T_{win} with a new value $T_{win,new}$ according to Equation (3.2.9), shown below, recalculates $k(T)$ for 1001 values evenly spaced from 0 to $T_{win,new}$ and repeats the search. In this way the search repeatedly alters

the window size, T_{win}, until an appropriate value of $T_{\max}$ can be found.

$$T_{win,new} = \begin{cases} \frac{T_{win}}{1000}\,, & \hat{T}_{\max} = 0 \\ 2\hat{T}_{\max}\,, & 0 < \hat{T}_{\max} < \frac{T_{win}}{10} \\ 100T_{win}\,, & \hat{T}_{\max} = T_{win} \end{cases} . \tag{3.2.9}$$

CHAPTER 4

TOOLBOX IMPLEMENTATION

4.1 L1Controller Implementation

The `L1Controller` class implements state feedback controllers and output feedback controllers with a similar framework internally, despite having separate interfaces for the user. As the user specifies the system, the data entered is stored in a set of pre-defined variables that are applicable to all forms of $\mathcal{L}_1$ adaptive controllers. Most variables, such as A_m, have their own separate variable, but the uncertainties of the plant and all of their parameters are stored internally in a structure called `unknowns`, which contains a unique structure for each uncertainty that has been defined. The fields of this structure are listed in Table 4.1. With all of the variables stored separately and with none of the actual equations of the closed-loop controller created yet, all of the necessary conditions may still be checked, and as discussed in the previous chapters, they are all checked as soon as the user enters the necessary specifications. However, until the `sim` function is called, none of the equations are formally created inside the system. It is the duty of the `sim` function to construct the entire closed-loop system, simulate it, and return the data to the user. Therefore, many of the interesting implementation details are contained within the `sim` function, and its implementation shall be the major focus of this section. However, first, some details of how the variables are internally stored shall be discussed.

Since $\mathcal{L}_1$ adaptive controllers and the systems they control can take on many different forms, the `L1Controller` class needs to keep track of which configuration is being used. All of the configuration information is stored internally as a structure of Boolean variables called `status`. It is here that flags for whether the plant offers state feedback or output feedback and whether the piecewise constant adaptive law or the gradient descent adaptive law are

Table 4.1: The fields of the `unknowns` structure in the `L1Controller` class

Field Name	Description
`Value`	The true value of the uncertainty. Assumed to be unknown, yet must be specified for the system to be simulated. This field is also used for unmodeled dynamics to store the function g_0, defined in (2.1.1).
`AdaptiveLaw`	The adaptive law used to update the estimate of this uncertainty. Not applicable for piecewise constant adaptive laws. This field is also used for unmodeled dynamics to store the function g, defined in (2.1.1).
`AdaptiveGain`	The adaptive gain, Γ. Not applicable for piecewise constant adaptive laws.
`Bound`	The known bound on the estimates used as the projection bound. Not applicable for piecewise constant adaptive laws.
`Size`	The dimensions of the uncertainty. This is included so that matrix estimates, such as $\hat{\omega}(t)$ in a MIMO system, can be reshaped into vectors for the adaptive law, and then reshaped back into matrices for use in the control law and state predictor.
`LipschitzConstant`	For nonlinear uncertainties or unmodeled dynamics, this stores the Lipschitz constants of the uncertainty.

kept, in addition to a flag corresponding to every type of uncertainty that an $\mathcal{L}_1$ adaptive controller can handle.

A large number of the values entered by the user are always constants and may be stored as numbers internally. For example, the desired model is always stored in a state-space representation in which all the matrices are constant. However, for values that may not necessarily be constant, such as the uncertainties, the method of storing these needs to be able to account for constants as well as functions. Therefore, any value entered by the user that could be a function is stored internally in a string representation. In this way, a constant can be stored internally as the string `'2'` and a function could also be stored as `'sin(t)'`. This helps simplify the `sim` function by unifying its interface with the `L1Controller` data and allowing for manipulations on the data to be performed with regular expressions. The only exception to this rule is the gradient descent adaptive laws. These are created as function handles when the uncertainties are specified and stored within the `unknowns` structure. The reason for this is that for each type of uncertainty, the gradient descent adaptive laws are specified *a priori* and can be encapsulated as an anonymous function using only the values

of the plant and the parameters of the uncertainty. Therefore, these laws will not need to be modified in the `sim` function, and thus, they are stored in their final form as anonymous functions.

Additionally, as the user specifies the system, the `L1Controller` object also keeps a running total of all the necessary states of the system. The states that can be included in the variable `statespace` include x, $\hat{x}$, $\hat{y}$ (used only in the case of output feedback with the gradient descent adaptive law), x_z, the internal states of $C(s)$, and the estimates of any uncertainties when using the gradient descent adaptive law. As the object realizes that additional states are required, it appends them on to the end of the list and specifies their initial conditions. Note, however, that while the ordering of the states internally is deterministic, there is no guaranteed ordering. Therefore, the third output of the `sim` function, which lists the ordering of the states, must be taken into account for the first two outputs to be useful.

As stated earlier, the `sim` function is responsible for creating the closed-loop adaptive system in a general form that can be applied to any $\mathcal{L}_1$ adaptive controller. This general form groups the estimates of the uncertainties not by where they appear in the plant, but by where they enter the state predictor. This is a more useful way of grouping the estimates since it allows the state predictor to be more easily constructed, and since, in the control law, estimates that are unmatched in the state predictor are often passed through an additional filter before being passed through $C(s)$, or $D(s)$ if ω is present. The general form combines this additional filter with either $C(s)$ or $D(s)$ to create a new filter which is called $C_{um}(s)$ regardless of whether ω is present, and then performs the addition in the control law after passing the two groups of estimates through their respective filters, rather than before. In this way, certain cases, such as an output feedback system using the piecewise constant adaptive law where a matched uncertainty may be estimated by an unmatched estimate,

can be easily dealt with. The general form is:

$$\dot{x}(t) = A_m x(t) + B_m \left(\omega u(t) + f_1(t, x(t), z(t))\right) + B_{um} f_2(t, x(t), z(t)) \,, \tag{4.1.1}$$

$$\dot{\hat{x}}(t) = A_m \hat{x}(t) + B_m \left(\hat{\omega}(t) u(t) + \hat{f}_1(t)\right) + B_2 \hat{f}_2(t) \,, \tag{4.1.2}$$

$$\dot{x}_{cs}(t) = A_{cs} x_{cs}(t) + B_{cs} \left(k_g r(t) - \hat{f}_1(t) - I_\omega \hat{\omega}(t) u(t)\right) \,, \tag{4.1.3}$$

$$\dot{x}_{csum}(t) = A_{csum} x_{csum}(t) - B_{csum} \hat{f}_2(t) \,, \tag{4.1.4}$$

$$u(t) = C_{cs} x_{cs}(t) + C_{csum} x_{csum}(t) - D_{csum} \hat{f}_2(t) \,, \tag{4.1.5}$$

where the *cs* subscript corresponds to the state space representation of either $C(s)$ or $D(s)$ depending on whether ω is present, the *csum* subscript corresponds to the state space representation of $C_{um}(s)$, and

$$I_\omega = \begin{cases} 1 & \text{if } \omega \text{ is present} \\ 0 & \text{if } \omega \text{ is not present} \end{cases} \,.$$

Note also that the closed-loop system is completed by the adaptive laws for the individual estimates and the definitions of $\hat{f}_1(t)$ and $\hat{f}_2(t)$, which are based on the types of uncertainties present and are not captured in the general form. The process used by the `sim` function to construct and simulate this general form is outlined below:

1. Read in all the necessary variables from the `L1Controller` object and store them locally. This eliminates the need to look up the value of each variable at every time-step during simulation, and thus drastically reduces simulation time.

2. Create an expression for all of the matched estimates, $\hat{f}_1$, and an expression for all the unmatched estimates, $\hat{f}_2$, if necessary. In most cases, this is often accomplished merely by replacing the unknown parameters in the plant with their estimates. However, in the state feedback case when nonlinearities are present, the $\|x_t\|_{\mathcal{L}_\infty}$ in Equation (2.1.25) is created here.

3. If there is an unmatched estimate, calculate $C_{um}(s)$ and convert it to a minimal state space form such as the one in Equations (4.1.4)–(4.1.5).

4. Using regular expressions, replace all of the unknown variables in the plant with their true expressions. This enables us to simulate the plant. Note that the true expressions can contain the names of states, such as x. For the purposes of this step, the output y is treated as an unknown and replaced with `C_local*x` where `C_local` is the local copy of the matrix C. After this step is performed, a copy of the plant uncertainty expressions is stored for use in the reference system later.

5. Using regular expressions, replace the names of any internal state with the reference to the actual states inside the variable `statespace`. Since the goal is to create a single differential equation with a set of states called `states`, we replace any variable that has been registered with the `statespace` structure with the string `states(low:high)` where the range `low:high` is the range of indices for that variable within `statespace`. If the piecewise constant law is being used, we must create a function that updates an internal variable used to store $\hat{\sigma}(t)$ at every time-step, and a function that may be used to read this internal variable. Then the string `sigmahat` is replaced with the name of the reading function just defined.

6. Convert all strings into anonymous function handles to reduce simulation time. The function handles are evaluated in the base workspace, thus allowing variables defined in the base workspace to become incorporated into the function handle.

7. Create anonymous functions for $\dot{x}$, $\dot{\hat{x}}$, and u, according to Equations (4.1.1), (4.1.2), and (4.1.5).

8. Assemble the derivatives of the internal states in order according to `statespace`, and incorporate them all into a single anonymous function. This is performed merely by iterating through `statespace` and appending the derivatives for each state as they are encountered. However, if the piecewise constant law is being used, then the update function is positioned at the top of the vector of derivative functions, so that it gets called first. This ensures that the estimate is correctly updated before the derivatives are calculated. Note that since the update function returns an empty matrix, the inclusion of this function in the derivative vector does not change the size of the

derivative vector.

9. Simulate using the Matlab differential equation solver `ode15s`. If the piecewise constant adaptive law is being used, force the maximum time step used by the solver to be $T/2$, or half of the sampling period. This guarantees that the differential equation solver places at least one time step within every sampling period, a property that would not be guaranteed by choosing T as the maximum time step due to rounding errors. Also, the `ode15s` solver is meant to work well with stiff differential equations, a necessity for simulating adaptive systems with high adaptive gain. In addition, `ode15s` is more accurate than the other built-in Matlab stiff differential equation solvers, though the added accuracy comes at the cost of longer simulation time. However, the increase in time does not significantly hamper the user's ability to use the toolbox, and therefore the more accurate solver was chosen.

10. If a reference signal was requested by the user, construct the reference system and simulate it. This is done by a separate function within the `L1Controller` class called `genRefSystem`. This function starts with the expressions saved at the end of step 4, constructs a new smaller `statespace`, and then builds the reference system shown below in Equation (4.1.6), which is based off of Equations (4.1.1)–(4.1.5).

11. Generate any graphs requested by the user using the state trajectories returned by `ode15s`. Table 2.1 on page 21 lists the acceptable strings the user can provide. In addition, the user may place numbers or ranges of numbers denoted with a colon after these strings to only graph certain indices of the signal with that name. For example, the string `x1:2` will cause this step to generate graphs of x_1 and x_2, where the subscript represents indexing the vector x. If a signal is specified that is not a state listed in `statespace`, the states are then used to calculate the value of the requested signal at every time step chosen by the differential equation solver in step 9, before generating the graph.

$$
\begin{aligned}
\dot{x}_{ref}(t) =& A_m x_{ref}(t) + B_m \left(\omega u_{ref}(t) + f_1(t, x_{ref}(t), z_{ref}(t))\right) \\
&+ B_{um} f_2(t, x_{ref}(t), z_{ref}(t)) \,, \\
\dot{x}_{cs,ref}(t) =& A_{cs} x_{cs,ref}(t) + B_{cs} \left(k_g r(t) - f_1(t, x_{ref}(t), z_{ref}(t)) - I_\omega \omega u_{ref}(t)\right) \,, \\
\dot{x}_{csum,ref}(t) =& A_{csum} x_{csum,ref}(t) - B_{csum} f_2(t, x_{ref}(t), z_{ref}(t)) \,, \\
u_{ref}(t) =& C_{cs} x_{cs,ref}(t) + C_{csum} x_{csum,ref}(t) - D_{csum} f_2(t, x_{ref}(t), z_{ref}(t)) \,.
\end{aligned}
\tag{4.1.6}
$$

4.2 GUI Implementation

While $\mathcal{L}_1$ adaptive controllers can be created and simulated easily using the `L1Controller` class discussed in the previous section, the $\mathcal{L}_1$ Adaptive Control Toolbox also provides a graphical user interface (GUI) for the user to interact with instead of writing code. The purpose of the GUI is to allow the user to design the $\mathcal{L}_1$ adaptive controller interactively while the toolbox automatically performs error checking on the user's inputs and generates relevant anaylsis graphs for the user as the user specifies the system. With these tools, the process of tuning the controller is greatly simplified and the speed at which the user can iterate their controller design is significantly increased. This section will provide an overview of the interface, how the user interacts with it, and the underlying implementation details.

The GUI is implemented in a separate class called `L1gui`, which is responsible for handling all of the interactions with the user and acts essentially as a middleman between the user and an `L1Controller` object, which is contained inside the `L1gui` object. While simple tasks such as basic error checking and generation of some of the analysis plots are handled by the `L1gui` object, more complicated tasks such as simulating the system, calculating error bounds, and checking the more computationally intensive error conditions are all handled by the `L1Controller` object. As the user enters data, the data is stored locally in a structure called `internalVars` inside the `L1gui` object. Then once the user has specified enough of the system, the `L1gui` object automatically configures the `L1Controller` using the functions described in Sections 2.2.1 and 3.2.1. Note that these are the same functions that are accessible to the user. Since many of these functions have multiple required inputs, often

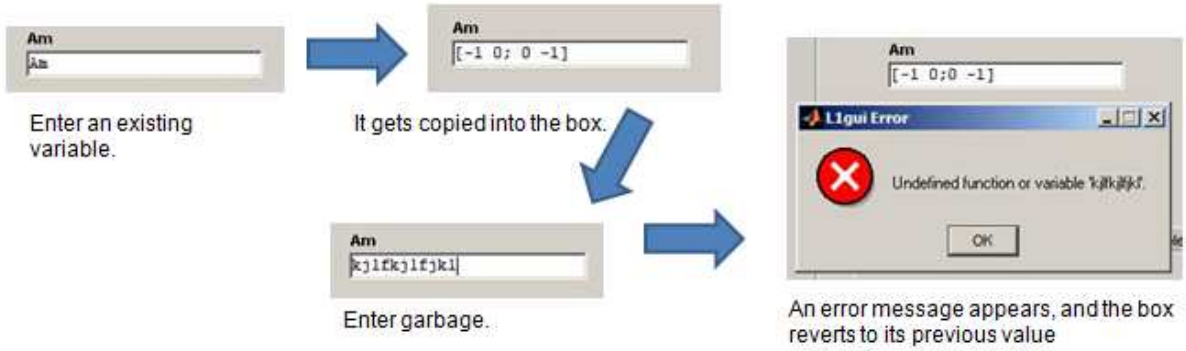

Figure 4.1: Entering data into the GUI with $A = -\mathbb{I}_2$ in the Base Matlab Workspace.

the `L1gui` object will find itself unable to call any of the `L1Controller` functions due to a lack of data and is therefore forced to accumulate the data internally. This is also beneficial as it allows the `L1gui` object to modify the user interface to reflect the options that should be available to the user based on the current knowledge about the system.

As mentioned above, the `L1gui` object performs basic error checking on the data as the user provides it. One of the primary error checking features performed by the `L1gui` object is to identify inputs that are unable to be evaluated. This is accomplished by attempting to evaluate the string input in the base workspace immediately after entry. If it successfully evaluates, then the input is accepted and is stored in the `internalVars` structure. However, if the input can not be evaluated, an error is generated and the interface empties that input field and reverts it back to its last known accepted input, if one exists. A beneficial side effect of this policy is that users may define variables in the base Matlab workspace and have them imported into the toolbox merely by entering the variable name into the appropriate field. An example of this is shown in Figure 4.1 where the user first enters `A`, which is defined in the base workspace as `[-1,0;0,-1]`. The input is evaluated and replaced with a string representing the value of `A`. Then a random string, which is unable to be evaluated, is entered, an error is generated, and the text box automatically reverts to the previous value. Note that the `L1gui` object always stores and displays the value of the variables entered, and not the variable names. This is because the variable `A` in the Matlab workspace could be altered or deleted while the GUI is running. Storing only the variable name could allow the altered variable to become incompatible with the rest of the GUI data, possibly causing the

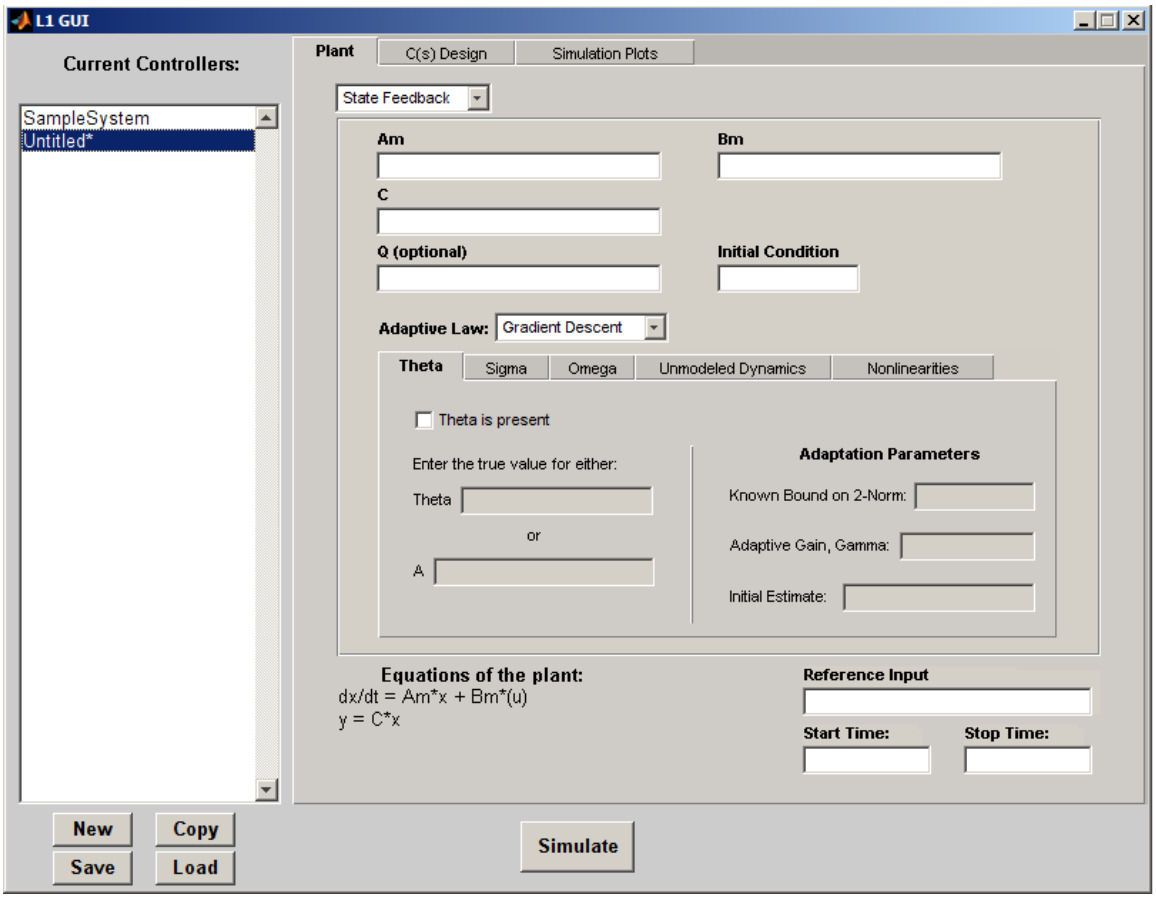

Figure 4.2: A blank plant tab, as it appears upon creation of a new controller.

`L1gui` object to become stuck in an undesirable state. By storing the value at the time the variable was specified, it can be guaranteed that all of the data stored in `internalVars` is valid and does not generate any of the errors the `L1gui` object checks for upon entry.

The GUI, pictured in Figure 4.2, consists of three major components: a list of current controller designs on the left, the main panel for specifying the system which covers the majority of the screen, and a collection of buttons along the bottom representing several common functions. The `L1gui` class allows the user to simultaneously work on multiple different `L1Controller` designs, which are listed on the left of the GUI. The user can switch back and forth from one design to another merely by changing which design is selected in the list. The list of designs is stored internally as an array of `L1Controller` objects and a matching array of `internalVars` structures. Then when the user selects a different design, the index used for these two arrays is changed to point to the newly selected design, and

all of the entry fields are populated with the values stored in that design's `internalVars` structure. In addition, the buttons below the list provide four common actions: creating a new blank design, creating a duplicate of the current design, saving the selected design to disk, and loading an existing design from disk. Designs are stored to disk merely by saving the `internalVars` structure in a `.mat` file. This allows the act of loading from disk to consist of creating a new blank design, importing the `internalVars` structure, constructing the `L1Controller` object, and re-populating the entry fields with the values in `internalVars`. Note also that designs with unsaved changes are displayed with an asterisk at the end of the name, the same paradigm used by the Matlab Editor.

The main panel on the left consists of three tabs called Plant, C(s) Design, and Simulation Plots, the first two of which are responsible for specifying the system while the last is used to configure the plots desired by the user. This tabbed browsing paradigm is useful in this case as it allows a smaller window to be used for the GUI while still providing all of the necessary features. However, while most of the elements in the GUI are built-in Matlab objects such as `uicontrol` or `uipanel`, the tabbed browser is a custom designed object made by arranging `uipanel`s in a tabbed browsing pattern with callbacks on the tabs to control which of the larger panels corresponding to the tabs is visible.

The Plant tab is where the entire system, except for $C(s)$, $D(s)$, or K, is specified. First the user may specify either state feedback or output feedback using the pull-down menu. The use of the pull-down menu instead of another tab represents that state feedback and output feedback are mutually exclusive and forces the user to choose. This choice determines the appearance of the rest of the panel. Note that if state feedback data is entered and then the system is changed to output feedback, the values entered previously for state feedback are not deleted, but are not carried over to specify the output feedback system either. In addition, in the bottom left, the equations of the current system are displayed so that the user may be sure the system has been correctly specified, and in the bottom right, the reference signal r, as well as the start and stop times for the simulation, may be specified.

The state feedback panel, pictured in Figures 4.2 and 4.3, allows the user to specify the base system with the fields in the top half. All of these fields that are not marked as Optional are required, and it is recommended, though not necessary, that the user specify these values

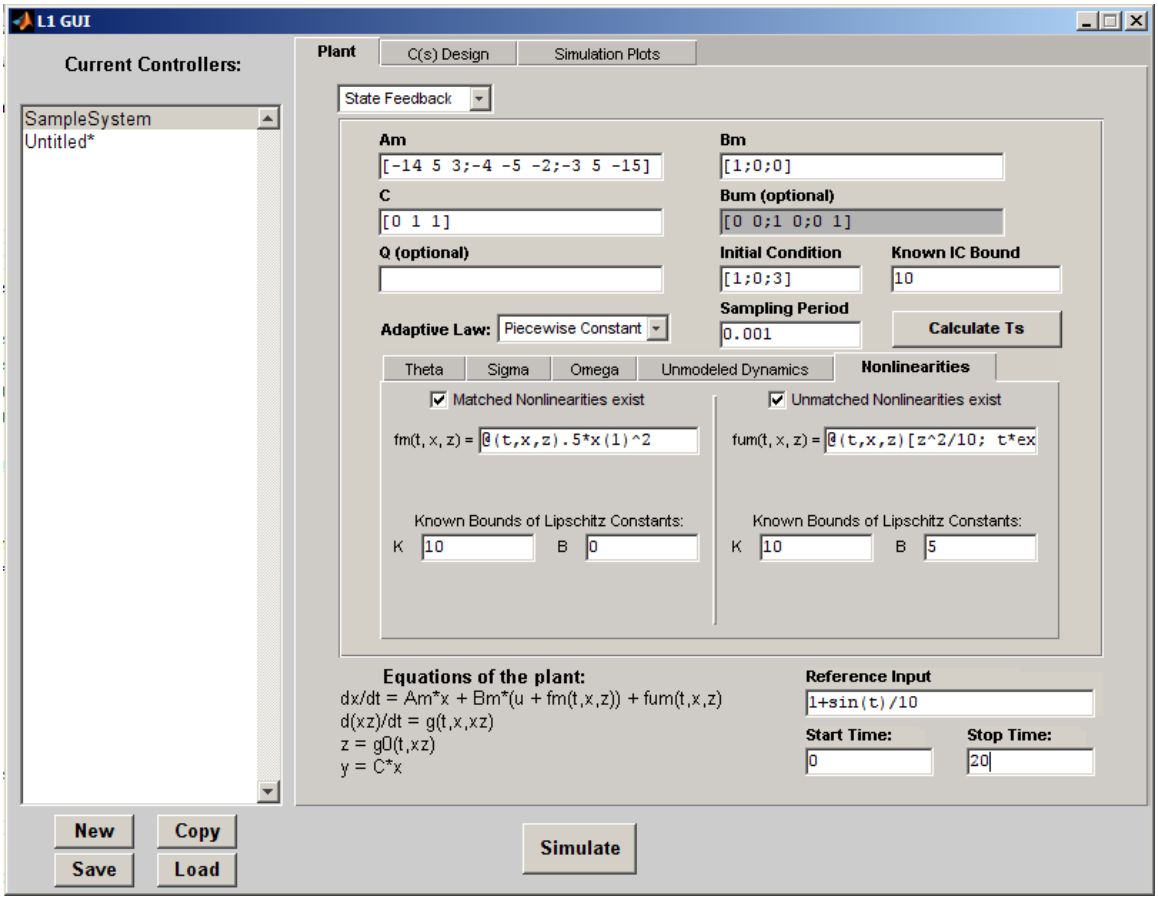

Figure 4.3: A complete plant specification for a SISO plant with unmodeled dynamics and matched and unmatched nonlinearities with a piecewise constant adaptive law. Note that additional fields not present in Figure 4.2 have appeared due to the type of system specified.

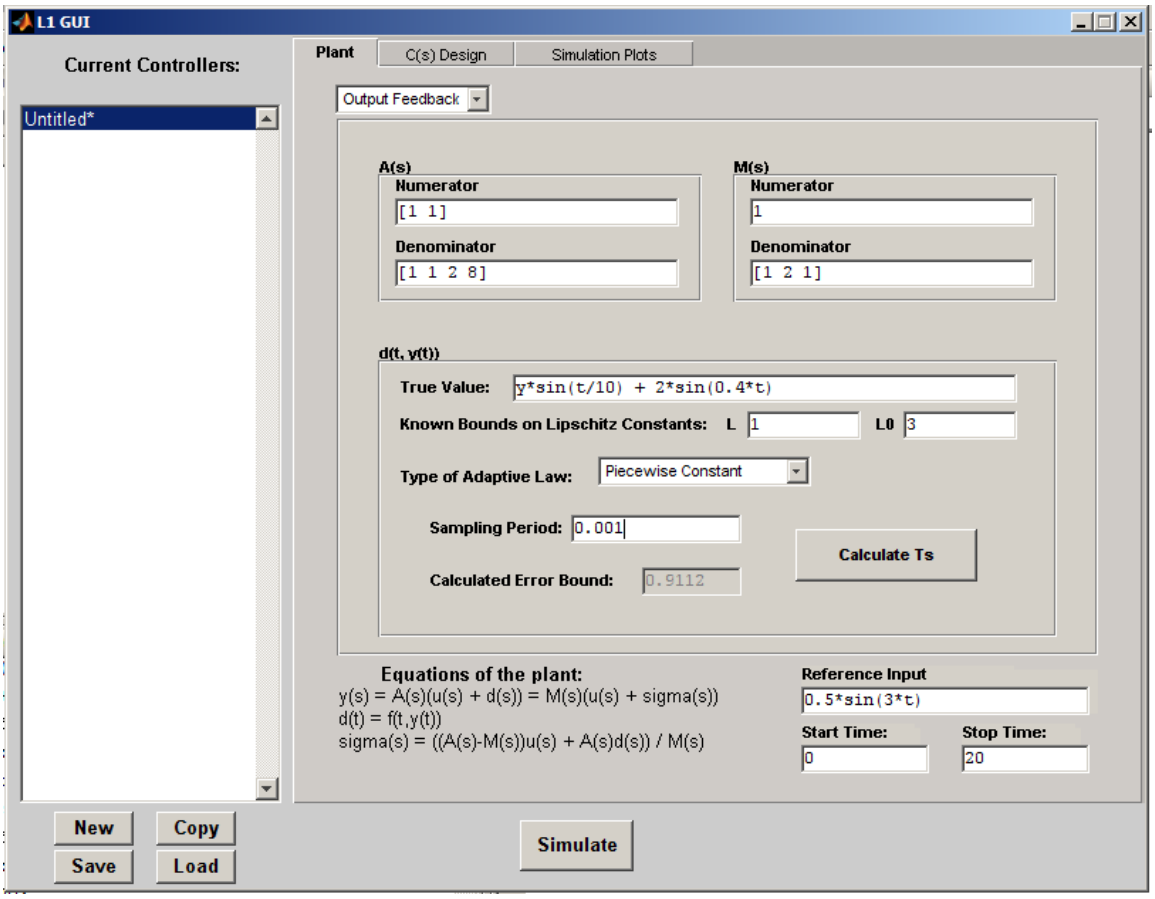

Figure 4.4: An output feedback design with the piecewise constant adaptive law.

first. Then the user may specify either the gradient descent or piecewise adaptive law. The type of adaptive law chosen will reconfigure the tabs below, which allow the user to specify the different kinds of uncertainties that are present in the system. The user may specify as many of these uncertainties as desired with one exception. If nonlinearities are present in the system, the user may not also include θ or σ; these must instead be included as part of the matched nonlinearity. Each tab contains a checkbox to denote if that type of uncertainty is present in the system or not. Checking this box enables the rest of that tab and alters the displayed equations at the bottom. Note that disabling one of these check boxes does not delete any of the data entered in the rest of the tab; it just removes that term from the equation of the plant. Note that the arrangement of the fields in this tab is meant to mirror the order in which the user would specify the system with the `L1Controller` functions.

The output feedback panel, pictured in Figure 4.4, is similarly arranged so as to mirror the

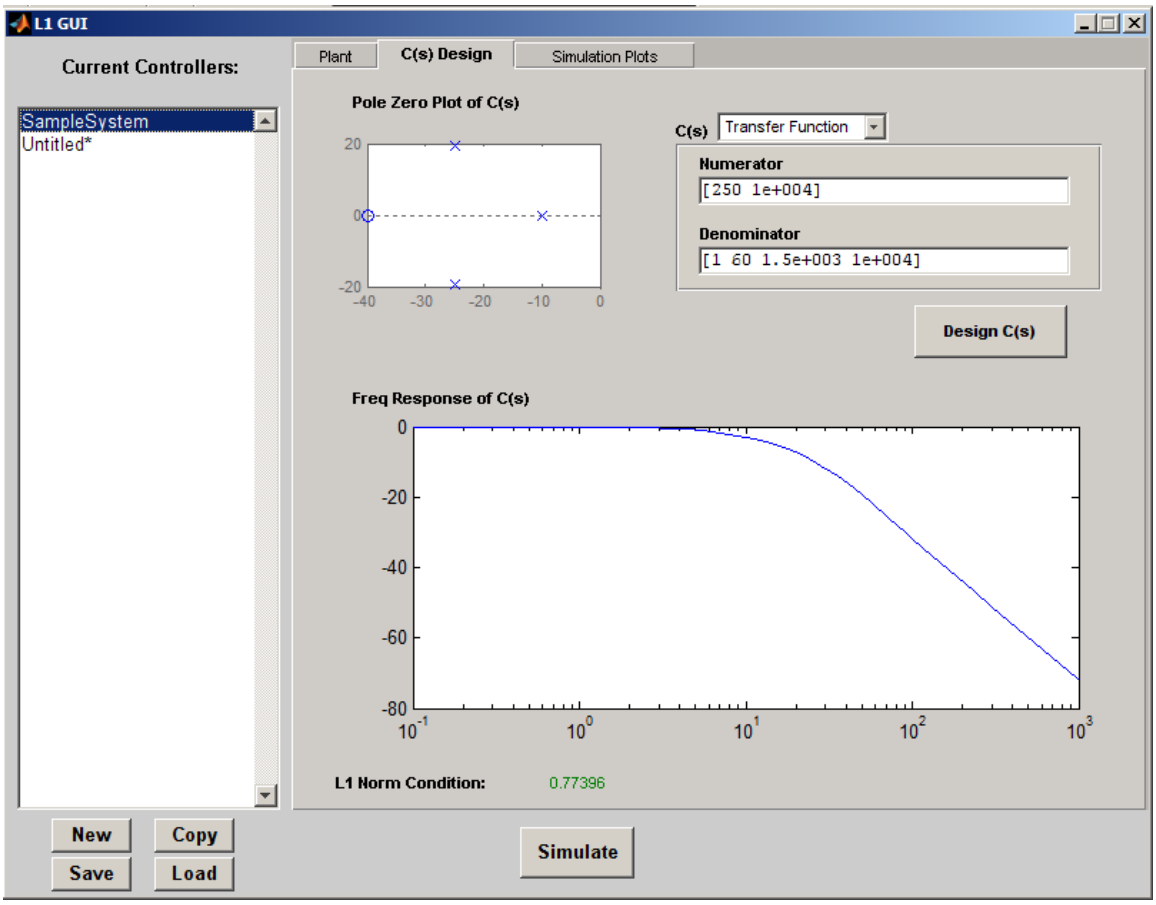

Figure 4.5: The C(s) Design tab, showing a SISO filter design for a system with only matched uncertainties.

order the `L1Controller` functions use. The transfer functions $A(s)$ and $M(s)$ are specified with coefficient vectors, similar to the `setOutputFeedbackPlantModel` function, at the top of the panel. Then below that, the disturbance d may be specified. Since it is assumed that this uncertainty must be present, the true value of the uncertain function $f(t, y(t))$ and the known Lipschitz constants are requested outside of the panel controlled by the adaptive law type selector. The adaptive law type selector reconfigures the fields below it to contain the necessary values. The Calculate Ts button creates a dialog that requests the desired error bound $\bar{\gamma}$, defined in Equation (3.1.12), and then implements the algorithm described in Section 3.2.2.

The C(s) Design tab, pictured in Figure 4.5, is where the filter $C(s)$, or the filter $D(s)$ and the gain K if ω was designated as present in the Plant tab, is specified. These values were

placed in a separate tab since the primary difficulty in designing an $\mathcal{L}_1$ adaptive controller lies in designing the filter $C(s)$. Therefore, this tab provides additional information about the filter such as its Bode plot at the bottom, and its pole-zero diagram in the top left. Note that if $C(s)$ is not SISO, then there will be extra fields for the user to denote which entry should be plotted. In addition, at the bottom of the panel, the $\mathcal{L}_1$-norm condition corresponding to the type of uncertainties chosen in the Plant tab is displayed, allowing the user to determine if the closed-loop system is guaranteed to be stable. This number represents the left-hand side of the condition and is color coded to alert the user: green means the condition is satisfied, yellow means the condition is barely satisfied and that it could be beneficial to alter $C(s)$, and red means the condition is not satisfied. Note that when nonlinear uncertainties are present in a state feedback controller and the $\mathcal{L}_1$-norm condition, Equation (2.1.19), becomes more complicated than the usual equation, such as Equation (2.1.7), then the display changes to show part of the contribution of the matched uncertainty, $\|G_m(s)\|_{\mathcal{L}_1} L_{1\rho_r}$ and part of the contribution of the unmatched uncertainty, $\|G_{um}(s)\|_{\mathcal{L}_1} L_{2\rho_r}$. Note that the sum of these two quantities must be less than 1 in order to guarantee that Equation (2.1.19) is satisfied. Additionally, in this nonlinear case, there is an indicator in the bottom right of the panel showing whether or not the complete condition is satisfied, or in other words, if a ρ_r exists that satisfies the condition.

The Design C(s) button creates a separate window, shown in Figure 4.6, for additional tools used in designing either $C(s)$ or $D(s)$ and K. For the sake of this description, we shall refer to only $C(s)$, though the same description holds for $D(s)$ as well. To keep the interface from becoming too complex and unwieldy, this window requires that $C(s)$ be the product of a SISO transfer function and $\mathbb{I}_m$, thereby providing only one transfer function to design rather than m^2. The list on the left half of the window displays the poles and zeros of $C(s)$. Additional poles and zeros may be added by right-clicking on the list and selecting the type of pole or zero to add from the menu that appears. The panel in the top right of the window displays the current value of the pole or zero and provides a button to delete this pole or zero. Left-clicking on a different pole or zero from the list causes this panel to automatically update to show the data for the newly selected pole or zero. Note that if the selected pole or zero is complex, then it actually represents a complex pair of poles or zeros, though only

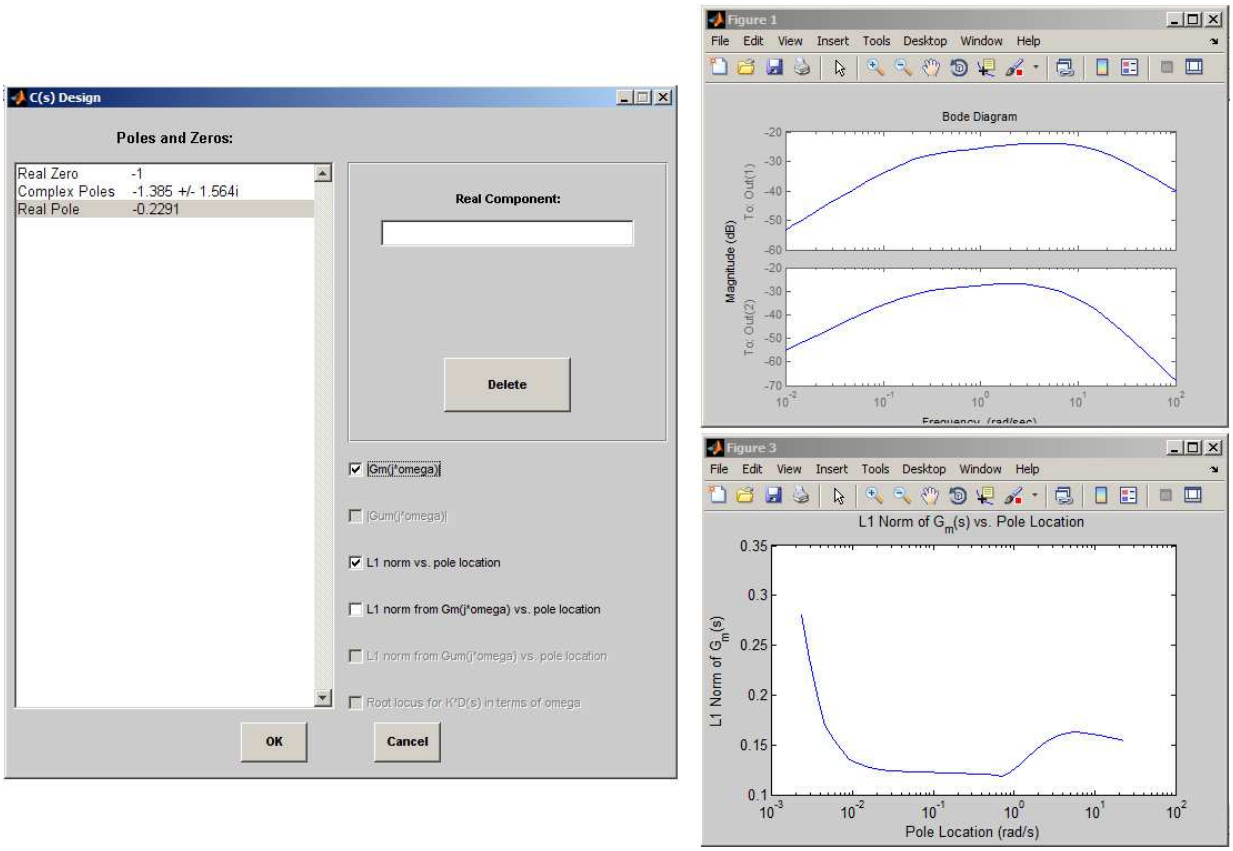

Figure 4.6: The Design C(s) Window, showing a design of $C(s)$ for a system with only matched uncertainties and without ω. Note that the two checked boxes have generated the graphs on the right, and that the unavailable options for this plant have been grayed out.

the positive imaginary value is displayed. Finally, in the bottom right, a list of check boxes corresponding to possible analysis graphs is presented. By checking the box, the graph is generated in a separate figure. The possible graphs can include:

- $|G_m(j\omega)|$, the magnitude of $G_m(j\omega)$ using the current value of the selected pole or zero.
- $|G_{um}(j\omega)|$, the magnitude of $G_{um}(j\omega)$ using the current value of the selected pole or zero.
- The value of the left-hand side of the $\mathcal{L}_1$-norm condition as a function of the location of the selected pole or zero. If a nonlinear state feedback system is specified, then the graph will display $\|G_m(s)\|_{\mathcal{L}_1} L_{1\rho_r} + \|G_{um}(s)\|_{\mathcal{L}_1} L_{2\rho_r}$ as a function of the location of the selected pole or zero.
- The contribution from the matched uncertainties to the $\mathcal{L}_1$-norm, $\|G_m(s)\|_{\mathcal{L}_1} L_{1\rho_r}$, as a function of the location of the selected pole or zero.
- The contribution from the unmatched uncertainties to the $\mathcal{L}_1$-norm, $\|G_{um}(s)\|_{\mathcal{L}_1} L_{2\rho_r}$, as a function of the location of the selected pole or zero.
- The root locus of $C(s)$ using the current values of $D(s)$ and K for all values of $\omega \in \Omega$.

Note that all of these graphs may not be applicable to the plant that is specified in the main GUI. Therefore, only the graphs that are relevant are displayed as options. In addition, for the graphs that are functions of the pole or zero location, when a complex pair is selected, a graph shall be generated as a function of the real component of the pole or zero pair, and a graph shall be generated as a function of the imaginary component of the pole or zero pair. Also, note that the root locus of $C(s)$ is only a root locus in the traditional sense if $m = 1$. If $m > 1$, then a Monte Carlo simulation is run, randomly picking values of $\omega \in \Omega$ and then displaying all the locations where poles of $C(s)$ were found. Whenever a new pole or zero is selected from the list, or when the value of the selected pole or zero changes, all of the figures that have been generated are automatically updated to reflect the change. In this way, the user can change $C(s)$ and quickly see the effect that their changes will have

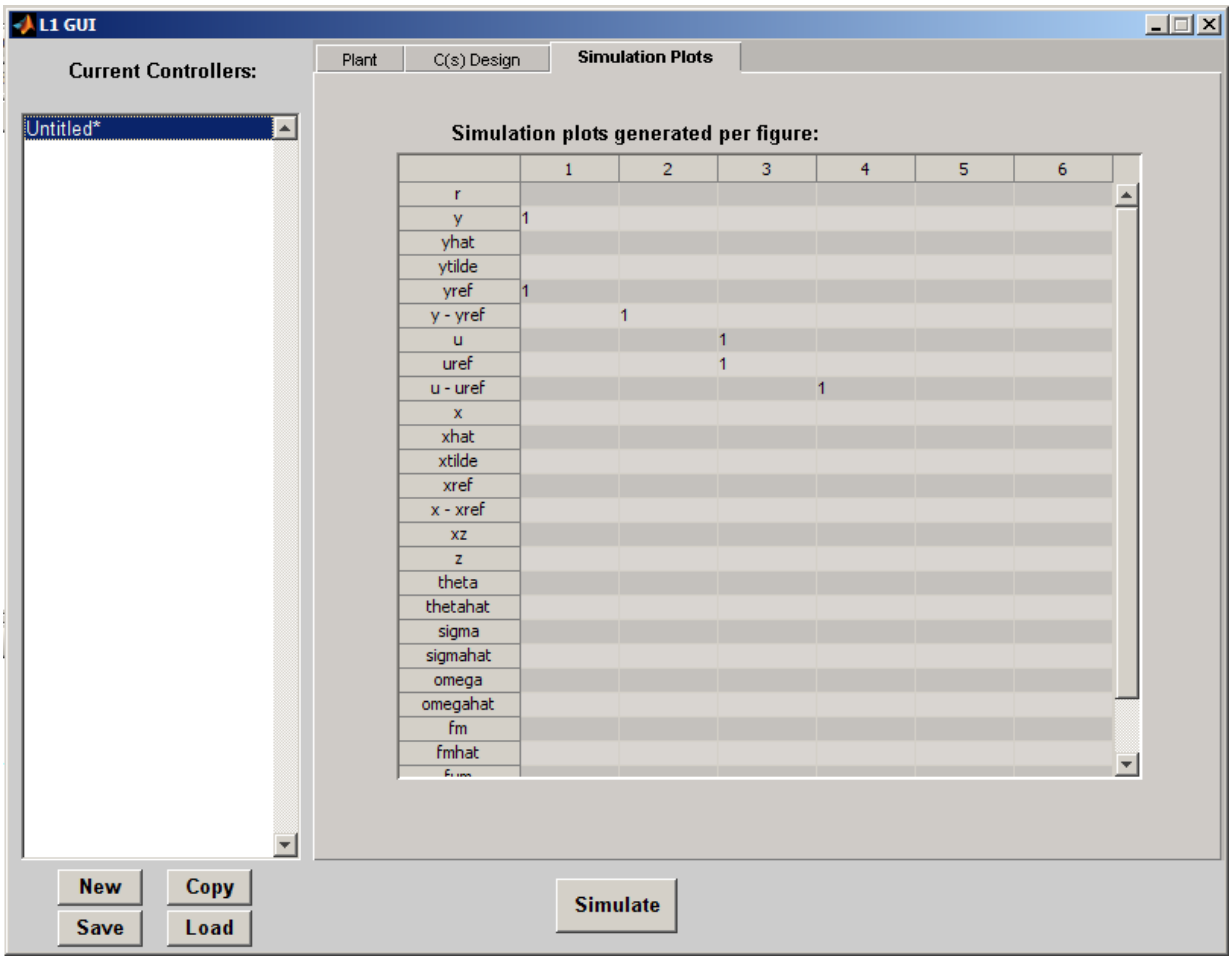

Figure 4.7: The Simulation Plots tab. The particular configuration shown here is to have 4 graphs showing $y(t)$ and $y_{ref}(t)$, $y(t) - y_{ref}(t)$, $u(t)$ and $u_{ref}(t)$, and $u(t) - u_{ref}(t)$.

on the performance of the adaptive system. Once the user decides on a design, the OK button checks to ensure that the design has the correct relative degree and returns the user to the main GUI window with the new value of $C(s)$ automatically entered in. The Cancel button returns the user to the main GUI window and leaves the value of $C(s)$ in the main GUI unchanged. Finally, while this window is open, the user will be unable to go back and modify the main GUI window.

The final tab is the Simulation Plots tab, shown in Figure 4.7, and has the same purpose as the string inputs to the `sim` function described in Section 2.2.1. The list of all the plots that can be generated by the `sim` command is on the left and each column represents a figure. By placing a number or range of numbers in a cell, the Simulate button at the bottom of the

GUI will calculate the signal corresponding to that row, indexed by the numbers or range of numbers provided, and graph it on the figure corresponding to the column. The user may specify as many signals as they wish on any of the figures, and the corresponding graphs will simply be overlayed on that figure.

CHAPTER 5

CONCLUSIONS

As has been shown, the $\mathcal{L}_1$ Adaptive Control Toolbox provides tools that speed up the design process of an $\mathcal{L}_1$ adaptive controller and enable the user to construct simulations of the closed-loop system to verify its performance. The `L1Controller` class has been introduced, and its interface discussed in Chapters 2 and 3. The implementation details were presented in Section 4.1, including the internal structure of the class and a step-by-step description of the `sim` function. The `L1gui` class was presented in Section 4.2 and its interactions with the `L1Controller` class were described as well as the user interface and in particular, the design tools provided for the filter, $C(s)$. In addition, novel algorithms for calculating the necessary sampling period to achieve a given error bound were presented in Sections 2.2.2 and 3.2.2.

Despite the impressive current capabilities of the $\mathcal{L}_1$ Adaptive Control Toolbox, there are future improvements that can be made. These include, but are not limited to, providing algorithms to calculate the time delay margin of the system with the given controller, providing algorithms to find a filter $C(s)$ that is guaranteed to meet a certain specification, such as the algorithm presented in [20] that guarantees a given time-delay margin, providing a method of transforming an `L1Controller` object into a Simulink block diagram, and providing calculations of commonly used performance metrics.

REFERENCES

[1] K. J. Åström and B. Wittenmark, *Adaptive Control*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1994.

[2] M. Krstić, I. Kanellakopoulos, and P. V. Kokotović, *Nonlinear and Adaptive Control Design*. New York, NY: John Wiley & Sons, 1995.

[3] P. A. Ioannou and P. V. Kokotović, "An asymptotic error analysis of identifiers and adaptive observers in the presence of parasitics," *IEEE Transactions on Automatic Control*, vol. 27, no. 4, pp. 921–927, August 1982.

[4] P. A. Ioannou and P. V. Kokotović, *Adaptive Systems with Reduced Models*. Secaunus, NJ: Springer-Verlag New York, Inc., 1983.

[5] P. A. Ioannou and P. V. Kokotović, "Robust redesign of adaptive control," *IEEE Transactions on Automatic Control*, vol. 29, no. 3, pp. 202–211, March 1984.

[6] B. B. Peterson and K. S. Narendra, "Bounded error adaptive control," *IEEE Transactions on Automatic Control*, vol. 27, no. 6, pp. 1161–1168, December 1982.

[7] G. Kresselmeier and K. S. Narendra, "Stable model reference adaptive control in the presence of bounded disturbances," *IEEE Transactions on Automatic Control*, vol. 27, no. 6, pp. 1169–1175, December 1982.

[8] K. S. Narendra and A. M. Annaswamy, "A new adaptive law for robust adaptation without persistent excitation," *IEEE Transactions on Automatic Control*, vol. 32, no. 2, pp. 134–145, February 1987.

[9] C. Cao and N. Hovakimyan, "Design and analysis of a novel $\mathcal{L}_1$ adaptive control architecture with guaranteed transient performance," *IEEE Transactions on Automatic Control*, vol. 53, no. 2, pp. 586–591, March 2008.

[10] C. Cao and N. Hovakimyan, "L1 adaptive controller for systems with unknown time-varying parameters and disturbances in the presence of non-zero trajectory initialization error," *International Journal of Control*, vol. 81, pp. 1147–1161, July 2008.

[11] C. Cao and N. Hovakimyan, "$\mathcal{L}_1$ adaptive output feedback controller for systems of unknown dimension," *IEEE Transactions on Automatic Control*, vol. 53, no. 3, pp. 815–821, April 2008.

[12] E. Xargay, N. Hovakimyan, and C. Cao, "$\mathcal{L}_1$ adaptive controller for multi–input multi–output systems in the presence of nonlinear unmatched uncertainties," in *American Control Conference*, Baltimore, MD, June–July 2010, accepted for publication.

[13] C. Cao and N. Hovakimyan, "$\mathcal{L}_1$ adaptive output-feedback controller for non-strictly-positive-real reference systems: Missile longitudinal autopilot design," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 32, no. 3, pp. 717–726, May-June 2009.

[14] I. M. Gregory, C. Cao, E. Xargay, N. Hovakimyan, and X. Zou, "$\mathcal{L}_1$ adaptive control design for NASA AirSTAR flight test vehicle," in *AIAA Guidance, Navigation and Control Conference*, Chicago, IL, August 2009, AIAA-2009-5738.

[15] T. Leman, E. Xargay, G. Dullerud, and N. Hovakimyan, "$\mathcal{L}_1$ adaptive control augmentation system for the X-48B aircraft," in *AIAA Guidance, Navigation and Control Conference*, Chicago, IL, August 2009, AIAA-2009-5619.

[16] K. Wise, E. Lavretsky, N. Hovakimyan, C. Cao, and J. Wang, "Verifiable adaptive flight control: Ucav and aerial refueling," in *AIAA Guidance, Navigation, and Control Conference*, Honolulu, HI, 2008, AIAA-2008-6658.

[17] E. Kharisov, I. Gregory, C. Cao, and N. Hovakimyan, "L1 adaptive control law for flexible space launch vehicle and proposed plan for flight test validation," in *AIAA Guidance, Navigation and Control Conference*, Honolulu, HI, 2008, AIAA-2008-7128.

[18] J.-B. Pomet and L. Praly, "Adaptive nonlinear regulation: Estimation from the Lyapunov equation," *IEEE Transactions on Automatic Control*, vol. 37, no. 6, pp. 729–740, June 1992.

[19] C. Cao and N. Hovakimyan, "Guaranteed transient performance with $\mathcal{L}_1$ adaptive controller for systems with unknown time-varying parameters: Part I," in *American Control Conference*, New York, NY, July 2007, pp. 3925–3930.

[20] N. Hovakimyan and C. Cao, $\mathcal{L}_1$ *Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation.* Philadelphia, PA: Society for Industrial and Applied Mathematics, to be published in September 2010.

[21] C. Cao and N. Hovakimyan, "$\mathcal{L}_1$ adaptive controller for a class of systems with unknown nonlinearities: Part I," in *American Control Conference*, Seattle, WA, June 2008, pp. 4093–4098.