Optimal Online Tuning of an Adaptive Controller

by

Jesse Huebsch

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Applied Science

in

Chemical Engineering

Waterloo, Ontario, Canada, 2004

I herby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

A novel adaptive controller, suitable for linear and non-linear systems was developed. The controller is a discrete algorithm suitable for computer implementation and is based on gradient descent adaptation rules. Traditional recursive least squares based algorithms suffer from performance deterioration due to the continuous reduction of a covariance matrix used for adaptation. When this covariance matrix becomes too small, recursive least squares algorithms respond slow to changes in model parameters. Gradient descent adaptation was used to avoid the performance deterioration with time associated with regression based adaptation such as Recursive Least Squares methods. Stability was proven with Lyapunov stability theory, using an error filter designed to fulfill stability requirements. Similarities between the proposed controller with PI control have been found.

A framework for on-line tuning was developed using the concept of estimation tracks. Estimation tracks allow the estimation gains to be selected from a finite set of possible values, while meeting Lyapunov stability requirements. The trade-off between sufficient excitation for learning and controller performance, typical for dual adaptive control techniques, are met by properly tuning the adaptation and filter gains to drive the rate of adaptation in response to a fixed excitation signal. Two methods for selecting the estimation track were developed. The first method uses simulations to predict the value of the bicriteria cost function that is a combination of prediction and feedback errors, to generate a performance score for each estimation track. The second method uses a linear matrix inequality formulation to find an upper bound on feedback error within the range

of uncertainty of the plant parameters and acceptable reference signals. The linear matrix inequality approach was derived from a robust control approach.

Numerical simulations were performed to systematically evaluate the performance and computational burden of configuration parameters, such as the number of estimation tracks used for tuning. Comparisons were performed for both tuning methods with an arbitrarily tuned adaptive controller, with arbitrarily selected tuning parameters as well as a common adaptive control algorithm.

Acknowledgments

I would like to thank my supervisor, Dr. Hector Budman, for his time, advice and support.

I would also like to thank my fiancée, Alyssa, for her love and patience.

Table of Contents

List of Figures

List of Tables

1 Introduction

Computer systems supporting advanced control algorithms have advanced to the point where high performance controllers beyond the standard PID do not have an onerous computational burden. In general, these high performance control systems rely on a mathematical model for controller design. A common first mathematical approximation is a model with linear dynamics, which can be represented as follows:

$$y_{k+1} = \sum_{i=0}^{n} a_i y_{k-i} + \sum_{j=0}^{m} b_j u_{k-j} \tag{1.1}$$

For conventional, non-adaptive control, the controller is constructed such that the closed loop combination of the controller and the plant model results in some desired closed loop transfer function. For many processes this calculation is not performed explicitly, and instead an empirical tuning method is used, such as the Ziegler and Nichols tuning technique (Seborg et. al., 1989). In the conventional linear control approach, although the plant may change over time, the controller parameters are kept constant. However, the changes in the model parameters if severe enough, can cause the closed-loop system to perform poorly or even become unstable, even if the original open-loop plant is stable. The conventional approach to avoiding this problem is to design the controller with very conservative parameter values and manually re-tune the controller periodically.

Adaptive and robust control methods were developed to counteract the problems with conventional controllers described above. In a robust control approach it is assumed that even if the model parameters are not known, the uncertainty of the values is known or

bounded. The controller is designed explicitly to take these uncertainties into account, and guarantee stability and performance through the range of uncertainty. In an adaptive control approach, parameter estimation methods are used to refine or learn a model of the plant over time. Design relations similar to the ones used for conventional controllers are used to calculate the feedback law based on the learned parameters over time. Some adaptive controllers, referred to as *cautious* controllers, use the uncertainty of the model parameters in the feedback law design. Adaptive controllers require an excitation signal to the process for parameter estimation to occur. One important class of controllers, referred to as dual adaptive controllers in the literature, use a combination of cautious control in the face of model uncertainty together with an optimal excitation signal to optimize the performance of the controller. Thus, the key idea behind the dual control concept is to achieve a trade-off between sufficient excitation for model learning and cautiousness or robustness of the controller in the presence of model uncertainty.

In the literature, most adaptive controllers have tuneable parameters that greatly affect their performance, but the methodology for selecting them has been restricted to rules based on a priori knowledge of the true system or ad hoc selection based on extensive trial and error simulations. Neither method is a satisfying solution for a general case.

In chapter two, background material for the current study consisting of brief reviews of concepts regarding discrete parameter estimation, feedback control laws, dual adaptive control, Lyapunov stability theory, and linear matrix inequalities are presented. Most of

the adaptive control methods found in the literature have tuneable parameters, but no systematic methods for selecting these parameters are given.

Chapter three introduces the adaptive control method proposed in this work. Stability is proven using Lyapunov stability theory. A method for maintaining stability in the event of a division by zero situation is also presented. Parallels with PI control are discussed.

A novel concept of estimation tracks is introduced in chapter four. Each estimator requires a constant value for the estimation gains for stability. Estimation tracks are used to provide the basis for on-line tuning by resetting all estimates and filtered errors to the values corresponding to the current best track. This idea consists of conducting parallel closed loop simulations with different set of tuning parameters. Each set is associated to a specific track. The dual adaptive nature of the proposed methods is discussed. Two new tuning methods, that have not been previously reported in the literature but are based on concepts reported in different contexts, are proposed in this work. The first method uses the bicriteria cost function, a combination of prediction and feedback errors, to predict which track will have the best performance. The second method draws from robust control ideas and uses linear matrix inequalities to find an upper bound on the feedback error for each track to find the track with the lowest error in the presence of model uncertainty. Simple examples are given to demonstrate each method in operation.

In chapter five detailed results are presented. First the effects of configuration parameters, such as the number of estimation tracks, time horizon for calculation and

magnitude of the model uncertainty are examined. Next the performance and computation time of the two proposed tuning methods are compared with an arbitrarily tuned system. This last simulation is explicitly conducted to illustrate the effect of a non-optimally tuned system. Finally a comparison with a standard adaptive control method is presented.

Finally, conclusions and recommendations for future work are presented in chapter six.

2 Literature Review

High performance control systems rely on a mathematical model for controller design. This model can be obtained mechanistically or empirically. A mechanistic model uses the chemical, mechanical, electrical or biological properties of the system to derive some equation describing the dynamic response of the system to inputs and disturbances. An empirical model uses the observed response of the system to inputs and disturbances to provide a prediction of the system behaviour after the model parameters are properly adjusted.

For conventional, non-adaptive control, the model of the system is required to design the controller. Once it is designed, it is kept fixed until manual retuning is conducted. One of the major difficulties with these conventional control design methods is their sensitivity to model errors or mismatch. Model mismatch tends to arise from errors in the initial modelling and from changes in the true system over time (e.g. change in feedstock, change in operating conditions, heat-exchanger fouling, etc.). In the presence of model mismatch an unstable closed-loop system or low control performance can result. Methods commonly used to compensate for these problems are robust and adaptive control. This thesis will focus on the topic of adaptive control.

An adaptive control system can be considered to be one where the controller parameters are changed based on observed input-output behaviour of the system. (Astrom and Wittenmark, 1989, chapter 1) Thus, adaptive controllers can clearly be used to deal with model errors or model changes with time.

The variables in a time varying system can be divided into two classes: the dynamic states of the system and model parameters. For example, in a system of the form:

$$y_{k+1} = \sum_{i=0}^{n} a_{i,k} y_{k-i} + \sum_{j=0}^{m} b_{j,k} u_{k-j} \tag{2.1}$$

y is the state, u is the control input and a and b are the model parameters. Dynamic states usually can be measured directly, or inferred with an observer, and are time-varying. Model parameters determine the response of the states to the control inputs. Generally, it is assumed that the model parameters vary slowly with time compared to the states, and are not deterministically affected by the control input. These two types of variables give rise to two elements in adaptive control algorithms: estimation and feedback.

The inner (or fast acting) loop is the feedback/feedforward control law. This loop resembles traditional control algorithms, and generates a control signal based on an error between the observed output and a reference signal. Common forms of control algorithms used are one-step-ahead, pole placement, minimum variance, PID, or minimum control effort techniques.

The outer (or slow acting) loop is used for system identification. This loop is the one that provides the adaptation with time. The model estimates are assumed to change slowly compared to the feedback loop, allowing the model estimate to be updated at slower rates than the feedback loop control action calculations.

If the adaptive element determines the parameters for a model, such as given in equation 2.1, the algorithm is referred to as indirect or explicit adaptive control (Filatov and Unbehauen, 2000). In this type of adaptive controller the estimation problem is done separately from the design problem and the estimation algorithm adapts the parameters of the model. Common types of estimation algorithm are least squares and gradient decent estimators. Based on this adapting model the control law is derived using some design rules or algorithm. Indirect adaptation gives some advantages for analysing and selecting alternatives. Having an explicit estimate model allows for process simulations and easy calculation of the prediction error.

If, instead of above, the feedback law parameters are adapted with time as input-output data becomes available, then the control is referred to as direct or implicit adaptive control. (Ex. Filatov et. al., 1997) In this method the controller's parameters are updated by the process estimator. The error between the observed closed loop system and a reference model is used to drive the adaptation.

Unlike other model estimation methods, the estimation problem in adaptive control is done in closed loop. The selection of the feedback law needs to take into account any requirements for sufficient excitation to the system. Implicit in all forms of adaptive control is a trade-off between instantaneous or short-term tracking performance and the accuracy of the estimated plant model. Long-term tracking performance depends on the plant model. In effect a trade-off is sought between short-term performance for long-

term performance. For a special kind of adaptive control, referred to as dual adaptive control, this trade-off is made explicit as explained in a later section in this chapter.

2.1 Identification of Plant Models

As stated above, one of the key components of adaptive control is system identification. For a linear system, the model used for adaptation is generally given in the Discrete Auto-Regressive Moving Average (DARMA) form:

$$y_{k+1} = \sum_{i=0}^{n} a_i y_{k-i} + \sum_{j=0}^{m} b_j u_{k-j} \tag{2.2}$$

The parameter vector to be considered is:

$$\boldsymbol{\theta} = \left[a_1,...,a_n,b_1,...,b_n\right] \tag{2.3}$$

The vector of parameter estimates at the k^{th} time step is:

$$\hat{\boldsymbol{\theta}}_k = \left[\hat{a}_{1,k},...,\hat{a}_{n,k},\hat{b}_{1,k},...,\hat{b}_{n,k}\right] \tag{2.4}$$

This vector can be expressed in the form of deviation variables:

$$\tilde{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_k - \boldsymbol{\theta} = \left[\tilde{a}_{1,k},...,\tilde{a}_{n,k},\tilde{b}_{1,k},...,\tilde{b}_{n,k}\right] \tag{2.5}$$

Convergence of the estimation requires that the norm of vector $\tilde{\theta}_k$ goes to 0 as k goes to infinity, i.e.

$$\lim_{k\to\infty}\left\|\tilde{\boldsymbol{\theta}}_k\right\| = 0 \tag{2.6}$$

A regression vector is defined as a function composed of past input-output data up to the order of the process model as follows:

$$\mathbf{X} = \left[y_k, y_{k-1} \ldots y_{k-n}, u_k, u_{k-1}, \ldots, u_{k-m} \right] \tag{2.7}$$

This work will focus on recursive algorithms, which have a basic form of the parameter update equation as follows: (Goodwin and Sin, 1984)

$$\hat{\boldsymbol{\theta}}_{k+1} = f\left(\hat{\boldsymbol{\theta}}_k, \mathbf{X}_k, k\right) \tag{2.8}$$

A widely used special case of this form is the following linear recursive representation:

$$\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k-1} + \mathbf{M}_{k-1} \mathbf{X}_{k-1} e_k \tag{2.9}$$

Where, the future value of the estimate vector is an algebraic function of the current value of the estimate vector, the current and past input-output data, and the time step. Recursive estimation is needed for adaptive control schemes since the new parameter value is required to calculate the new control action, and the computation time available for this operation relatively short. A non-recursive method which uses all past data will have a computation time that is a monotonically increasing function of k, and will eventually require more time to complete the calculations than the step interval available for this calculation.

2.1.1 Projection Algorithm

The projection algorithm is one of the most basic adaptation schemes reported in the literature. It is based on the following parameter update equation:

$$\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k-1} + \frac{\mathbf{X}_{k-1}}{\mathbf{X}_{k-1}{}^T \mathbf{X}_{k-1}} \left[y_k - \mathbf{X}_{k-1}{}^T \hat{\boldsymbol{\theta}}_{k-1} \right] \tag{2.10}$$

With $\hat{\boldsymbol{\theta}}_0$ known.

This equation is based on the following recursive form:

$$\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k-1} + \mathbf{M}_{k-1} \mathbf{X}_{k-1} e_k \tag{2.11}$$

Where,

$$\mathbf{M}_{k-1} = \frac{\mathbf{X}_{k-1}}{\mathbf{X}_{k-1}{}^T \mathbf{X}_{k-1}}, \; e_k = \left[y_k - \mathbf{X}_{k-1}{}^T \hat{\boldsymbol{\theta}}_{k-1} \right] \tag{2.12}$$

The error e_k is the error in prediction of the current observed value, based on the last estimated parameter set. The basic form of the projection algorithm is prone to division by zero. To avoid this singularity it is modified as follows:

$$\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k-1} + \frac{a \cdot \mathbf{X}_{k-1}}{c + \mathbf{X}_{k-1}{}^T \mathbf{X}_{k-1}} \left[y_k - \mathbf{X}_{k-1}{}^T \hat{\boldsymbol{\theta}}_{k-1} \right] \tag{2.13}$$

With $\hat{\boldsymbol{\theta}}_0$ known and $c > 0; \; 0 < a < 2$

This algorithm is known as the normalized least-mean-squares (NLMS). In Goodwin (Goodwin and Sin, 1984), proof is provided that $\left\| \hat{\boldsymbol{\theta}}_{k-1} \right\|$ is non-increasing, and that the parameter set is only guaranteed to converge if the vector $\mathbf{X}_{k-1}$ is orthogonal to $\mathbf{X}_k$. These results provide the motivation for the orthogonalized projection algorithm described in the next section.

2.1.2 Orthogonalized Projection Algorithm (Recursive Least Squares)

$$\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k-1} + \frac{\mathbf{P}_{k-2}\mathbf{X}_{k-1}}{c + \mathbf{X}_{k-1}{}^T\mathbf{P}_{k-2}\mathbf{X}_{k-1}}\left[y_k - \mathbf{X}_{k-1}{}^T\hat{\boldsymbol{\theta}}_{k-1}\right] \tag{2.14}$$

$$\mathbf{P}_{k-1} = \mathbf{P}_{k-2} - \frac{\mathbf{P}_{k-2}\mathbf{X}_{k-1}{}^T\mathbf{X}_{k-1}\mathbf{P}_{k-2}}{c + \mathbf{X}_{k-1}{}^T\mathbf{P}_{k-2}\mathbf{X}_{k-1}} \tag{2.15}$$

With $\hat{\boldsymbol{\theta}}_0$ known and $c > 0;\ 0 < a < 2$. The initial covariance matrix, $\mathbf{P}_{-1}$ is positive definite.

If one set c = 1, then the resulting algorithm is referred to as the recursive least-squares algorithm (RLS). The least-squares method has some key advantages. It may converge faster than the projection algorithm, provided that a good initial guess for the covariance matrix, P is available. It is also less sensitive to noise. The projection algorithm is easier to calculate for systems with a large number of states (Astrom and Wittenmark, 1989). The disadvantages of RLS are that its performance depends on the initial value of $\mathbf{P}_{-1}$. Also, $\|\mathbf{P}_k\|$ tends to 0 as the algorithm converges. Therefore, the basic form is not suitable for time varying systems since no further adaptation occurs after the system converges to an initial set of parameters. There are several variations on the RLS algorithm that can be implemented to deal with time varying systems. Examples are given in the following two sections. However, these methods require a fair amount of trial and error or ad-hoc tuning.

2.1.3 Exponential Data Weighting (Forgetting Factor)

This is a variant of the recursive least-squares algorithm, where the newest data is assumed to be more important than the old data.

$$\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k-1} + \frac{\mathbf{P}_{k-2}\mathbf{X}_{k-1}}{\alpha_{k-1} + \mathbf{X}_{k-1}{}^T\mathbf{P}_{k-2}\mathbf{X}_{k-1}}\left[y_k - \mathbf{X}_{k-1}{}^T\hat{\boldsymbol{\theta}}_{k-1}\right] \tag{2.16}$$

$$\mathbf{P}_{k-1} = \frac{1}{\alpha_{k-1}}\left[\mathbf{P}_{k-2} + \frac{\mathbf{P}_{k-2}\mathbf{X}_{k-1}{}^T\mathbf{X}_{k-1}\mathbf{P}_{k-2}}{\alpha_{k-1} + \mathbf{X}_{k-1}{}^T\mathbf{P}_{k-2}\mathbf{X}_{k-1}}\right] \tag{2.17}$$

With $\hat{\boldsymbol{\theta}}_0$ and $P_{-1} > 0$ known.

The parameter α_k is the forgetting factor and is generally selected ad-hoc. The excitation of the system is particularly important with this method.

2.1.4 Covariance Resetting

The standard recursive least-squares method is used, in combination with frequent resetting of the covariance matrix; $\mathbf{P}_k$, becomes $\mathbf{K}_k$*I. When the covariance matrix is reset at every time interval, this method becomes equivalent to the projection algorithm. In both of these variants, the performance depends on the selection of α or the resetting time interval. Of course, in the case of time varying parameters, it is not clear when to reset the covariance since the times at which the changes occur are unknown a priori.

2.2 Feedback Laws for Adaptive Control

As stated above, in explicit adaptive control methods the model estimator is designed separately from the control law. Although the estimation and control are chosen separately, some forms of estimation and control may compliment each other better than others. Traditional forms of feedback use the *certainty equivalence* (Bar-Shalom and Tse, 1974) principle, where the plant model estimate parameters are assumed to be the true values of the real system for the purpose of designing the feedback mechanism.

Goodwin (Goodwin and Sin, 1984) identifies several common forms of feedback laws in his text and these are further discussed in the following subsections.

2.2.1 One-Step-Ahead Control

The error criteria to be minimised is the output error at the next step. This method has the advantage of using the model parameters directly in the control law, so no estimation algorithm per se needs to be considered.

The cost function to be minimised is:

$$J_{k+1} = \left\{ \frac{1}{2} \left[y_{k+1} - ysp_{k+1} \right]^2 \right\} \tag{2.18}$$

With a standard DARMA model:

$$y_{k+1} = \sum_{i=1}^{n} a_i y_{k-i} + \sum_{j=1}^{m} b_j u_{k-j} \tag{2.19}$$

The control law is given as follows:

$$u_k = \left(ysp_{k+1} - \sum_{i=1}^{n} a_i y_{k-i} - \sum_{j=2}^{m} b_j u_{k-j} \right) \cdot b_1^{-1} \tag{2.20}$$

If the true model parameters are unavailable, the estimated model parameters are used as follows:

$$u_k = \left(ysp_{k+1} - \sum_{i=1}^{n} \hat{a}_i y_{k-i} - \sum_{j=2}^{m} \hat{b}_j u_{k-j} \right) \cdot \hat{b}_1^{-1} \tag{2.21}$$

The closed loop equation when (2.19) and (2.21) are combined is:

$$y_{k+1} = b_1 \cdot \hat{b}_1^{-1} ysp_{k+1} + \sum_{i=1}^{n} \left(a_i - \left(b_1 \cdot \hat{b}_1^{-1} \right) \hat{a}_i \right) y_{k-i} + \sum_{j=2}^{m} \left(b_j - \left(b_1 \cdot \hat{b}_1^{-1} \right) \hat{b}_j \right) u_{k-j} \tag{2.22}$$

If the estimated values are accurate, $\hat{b}_j = b_j, \hat{a}_i = a_i$ then the closed loop system follows:

$$y_{k+1} = ysp_{k+1} \tag{2.23}$$

One disadvantage of this method is that it may give excessive control actions if there is a step change in the set-point. To solve this problem with the one-step-ahead control a weight is added to the control input in the cost function to be minimised as follows:

$$J_{k+1} = \left\{ \frac{1}{2} \left[y_{k+1} - ysp_{k+1} \right]^2 + \frac{\lambda}{2} u_k \right\} \tag{2.24}$$

Where λ is a tuning parameter that provides the input weighting.

An alternative method to deal with excessive control inputs is to filter the reference signal with a desired tracking model as shown in Figure 2.1.

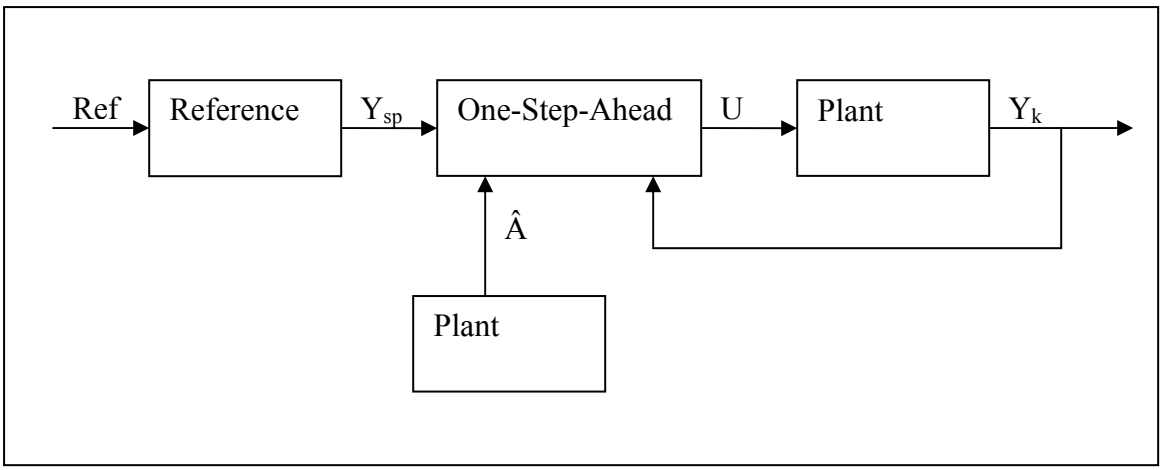

Figure 2.1: Schematic of a one-step-ahead controller with a reference model

Figure 2.1 represents the basic idea for model reference control, where the desired closed-loop performance is given in the form of a desired transfer function or if pole placement control is desired, by a set of desired poles.

2.2.2 Model Reference Control

For model reference control the reference signal is simultaneously fed to the controller with a transfer function C(z), and to a reference model with transfer function G(z) (Goodwin and Sin, 1984). The reference model is a known transfer function that is stable, and has a delay at least as long as that of the system to be controlled. The desired controller will give feedback such that the closed loop system of C(z) and the plant P(z) gives an output that is identical to the output of G(z); Y^*, when both are driven to the reference signal, R(z). The preceding system is illustrated in Figure 2.2, as follows:

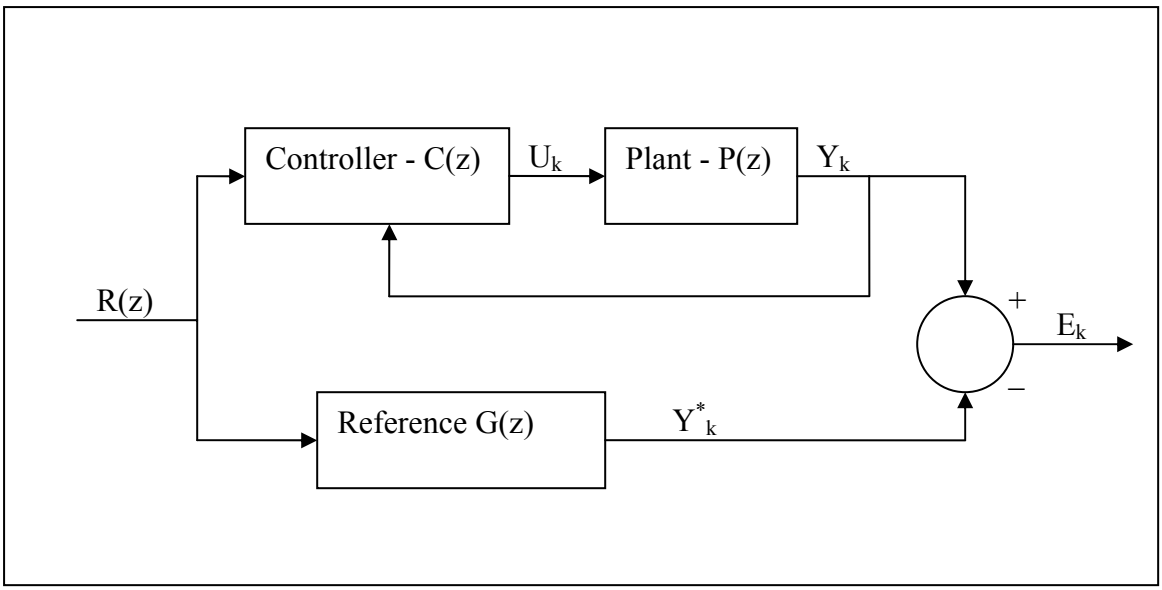

Figure 2.2: Schematic of a model reference controller

In an adaptive control context, the tracking error and the input-output data are used in an adaptive mechanism to update the controller C(z). Usually this is referred to as a direct adaptive controller.

2.2.3 Cautious Control

(Astrom and Wittenmark, 1989)

If the controller is selected as a one-step-ahead controller, the control action calculation is the following control law, from equation (2.20):

$$u_k = \frac{ysp_{k+1} - [0, u_{k-2}, ..., u_{k-n}, -y_{k-1}, ..., -y_{k-n}]\boldsymbol{\theta}_{k+1}}{b_{1,k+1}} \tag{2.25}$$

The certainty equivalence controller is equal to the one-step-ahead controller but it uses the parameter estimates instead of the true values as follows:

$$u_k = \frac{ysp_{k+1} - [0, u_{k-2}, ..., u_{k-n}, -y_{k-1}, ..., -y_{k-n}]\hat{\boldsymbol{\theta}}_{k+1}}{\hat{b}_{1,k+1}} \tag{2.26}$$

On the other hand, for cautious control the uncertainty in the estimates is considered. For example, with a recursive least-squares algorithm, the uncertainty is accounted through the covariance matrix, P_k, as follows:

$$u_k = \frac{\hat{b}_{1,k+1} \cdot ysp_{k+1} - [0, u_{k-2}, ..., u_{k-n}, -y_{k-1}, ..., -y_{k-n}]\left[\hat{b}_{1,k+1} \cdot \hat{\boldsymbol{\theta}}_{k+1} + \mathbf{P}_k \cdot [1,0,...,0]\right]}{\hat{b}^2_{1,k+1} + p_{b1,k+1}} \tag{2.27}$$

where $p_{b1,k+1}$ is the variance of the parameter $\hat{b}_{1,k+1}$.

When the covariance $\mathbf{P}_k = 0$ then the cautious controller given in equation (2.27) is equivalent to the certainty equivalence controller given in equation (2.26). The situation where $\mathbf{P}_k = 0$ indicates that the parameter estimation has converged, and that the adapted model is the best estimate of the true plant.

For example, for a pure integrating controller the certainty equivalence controller is:

$$u_k = -\frac{1}{\hat{b}_{k+1}} y_k \tag{2.28}$$

The corresponding cautious controller is:

$$u_k = -\frac{\hat{b}_{k+1}}{\hat{b}^2_{k+1} + p_{b,k+1}} y_k \tag{2.29}$$

The net effect of accounting for uncertainty with P is a reduction of the controller gain, making the performance less aggressive while the uncertainty is large. The downside to this approach is that the input gain decreases as the uncertainty increases. This can become a problem when the new input lacks sufficient excitation to better the estimation and consequently, to reduce the uncertainty. This is called the *turn-off phenomenon* (Astrom and Wittenmark, 1989). The dual adaptive control methodology presented in the sequel, avoids this problem by ensuring sufficient excitation for adaptation.

2.2.4 Dual Control

Implicit in all forms of adaptive control is a trade-off between instantaneous or short-term tracking performance and the accurate long-term estimation of the plant model. Long-term tracking performance depends on the plant model, so in effect there is a trade-off between short-term control performance and long-term performance. For dual adaptive control this trade-off is made explicit. Unfortunately, the optimal dual problem is only numerically tractable for very simple examples. Also, analytical solutions are only available for very simple systems. Thus, suboptimal approximations of optimal dual control are used.

Sternby (1976) gives an example of a system where an analytical solution to the optimal dual control problem can be found. The system used is a Markov chain with a finite set of states and no system dynamics.

Filatov et. al. (1997) introduce the bicriterial approach to a pole placement dual control. With the bicriterial approach there are two cost functions to be minimised,

$$J_k^c = E\left\{B^2\left[y_n(k+1) - y(k+1)\right]^2 \mid \Im_k\right\}, \qquad (2.30)$$
$$J_k^a = E\left\{\left[\bar{y}(k+1) - \hat{\mathbf{p}}^T(k)\mathbf{m}(k)\right]^2 \mid \Im_k\right\}$$

y_n represents the nominal or desired trajectory, $\bar{y}(k+1)$ represents the output with no disturbance, $\hat{\mathbf{p}}^T(k)$ is the vector of estimated parameters and $\mathbf{m}(k)$ is the input-output data vector. These two terms represent the expectations of the feedback error and the prediction error respectively.

Filatov and Unbehauen (1998) extend dual adaptive control to a continuous system. A dither signal is used for excitation, with an amplitude dependant on the uncertainty of the estimate.

Dumont (Dumont and Astrom, 1987, Allison et. al., 1995) reported the implementation of a suboptimal dual controller on a wood chip refiner. The primary purpose of the controller is to detect and counteract a process gain sign reversal. Heuristic elements are added to the control algorithm to allow for quick response in the event of a gain reversal.

Veres and Xia (1998) examine the worst case transient performance for adaptive control systems. They use the context of an airplane that is damaged, causing a sudden change in the true plant. In this case the eventual convergence of the adaptive control is insufficient to guarantee overall stability of the algorithm; thus, the states of the system must be kept stable while adaptation occurs.

Filatov et. al. (1996) use a direct dual adaptive control on a laboratory scale mechanical example. They use the bicriterial method for designing a feedback law. The setup demonstrates performance improvements with the dual control over a normal adaptive controller for an unstable system. No tuning guidelines are given for this algorithm.

Sanner and Slotine (1992) introduce a continuous time framework for adaptive control of nonlinear systems. A Gaussian network on a fixed grid is used to approximate the nonlinear system. The network gains are directly updated by the adaptation mechanism. For this method to work it is necessary for the system to satisfy assumptions about relative smoothness and bandwidth limitations. A dead-zone around the set-point and sliding control at the edge of the modeled region are used to ensure stability using Lyapunov stability criteria. This algorithm, that serves as a basis for the techniques used in the current study, includes several tuning parameters that are selected ad-hoc or by trial and error. This is also one of the key disadvantages of dual adaptive algorithms.

Fabri and Kadirkamanathan (1998) demonstrate the applicability of explicit dual control to nonlinear systems. They use a fixed grid mesh of radial basis functions to estimate a nonlinear function. The neuron gains are adapted, but not the spacing or variance of the neurons. The uncertainty of the estimate is taken into account in the design of the feedback law; providing this algorithm with dual adaptive control features.

2.3 Analysis Techniques of Control

2.3.1 Linear Matrix Inequalities

A linear matrix inequality formulation can be used to evaluate the performance of a system with uncertain parameters, while taking into account every possible combination of parameters within the range of uncertainty.

A Linear Matrix Inequality (LMI) has the form:

$$\mathbf{A}(x) = \mathbf{A}_0 + x_1\mathbf{A}_1 + \ldots + x_n\mathbf{A}_n < 0 \tag{2.31}$$

Where

$x = [x_1 \ \ldots \ x_n]$ is a vector with unknown values, known as the *optimization variable*.

$\mathbf{A}_0, \ldots, \mathbf{A}_n$ are known symmetric matrices.

And $\mathbf{A}(x)$ is negative definite (i.e. all eigenvalues of $\mathbf{A}(x)$ are negative,

or $\boldsymbol{\eta}^T \mathbf{A}(x)\boldsymbol{\eta} < 0$ for all nonzero $\boldsymbol{\eta} \in \Re^n$

The LMI's (2.31) can be rearranged to represent $\mathbf{A}(x) > 0$ as $-\mathbf{A}(x) < 0$ and

$\mathbf{A}(x) < \mathbf{B}(x)$ as $\mathbf{A}(x) - \mathbf{B}(x) < 0.$

Note that $\mathbf{A}(y) < 0, \mathbf{A}(z) < 0 \Rightarrow \mathbf{A}\left(\frac{y+z}{2}\right) < 0$, thus, the LMI's (2.31) is a convex constraint on x. The key properties of the LMI formulation are that its solution set is a convex subset of $\Re^n$ and if there is a solution to (2.31) finding the solution is a convex

optimisation problem. The important thing to note about a convex optimisation is that even though (2.31) has no analytic solution; a numerical solver can be guaranteed to converge to a solution, provided a solution exists.

The formulation to be solved that is relevant to this work is the generalized eigenvalue problem (GEVP). The GEVP is to find the minimum value of the maximum generalized eigenvalue of a pair of matrices that are affine functions of the LMI optimization variable. The GEVP is formulated as follows:

$$\text{Minimize } \lambda \text{ subject to: } \lambda\mathbf{B}(x) - \mathbf{A}(x) > 0, \mathbf{B}(x) > 0, \mathbf{C}(X) > 0 \tag{2.32}$$

Where **A**, **B**, **C** are symmetric matrices that are affine functions of the optimisation variable x. The result of this computation is λ_{max}, the largest eigenvalue of the GEVP, after the minimization of (2.32).

In the case that **B**(x) is positive semi-definite, and not positive definite, the LMI solvers available in Matlab may not be able to calculate a feasible solution. This is important for problems where **B**(x) has the structure

$$\mathbf{B}(x) = \begin{bmatrix} \mathbf{B}_1(x) & 0 \\ 0 & 0 \end{bmatrix}, \mathbf{B}_1(x) > 0 \tag{2.33}$$

By replacing the constraints of (2.32),

$$\lambda\mathbf{B}(x) - \mathbf{A}(x) > 0, \mathbf{B}(x) > 0, \mathbf{C}(x) > 0 \tag{2.34}$$

with the following constraints:

$$\mathbf{A}(x) < \begin{bmatrix} \mathbf{Y} & 0 \\ 0 & 0 \end{bmatrix}, \mathbf{Y} < \lambda \mathbf{B}_1(x), \mathbf{B}_1(x) > 0 \qquad (2.35)$$

where $\mathbf{Y}^T = \mathbf{Y}$

the resulting problem is equivalent to the original GEVP problem as given in equation (2.32), and can be solved with the GEVP Matlab function.

Kothare et. al. (1994, 1996) use LMI's to design a robust model predictive controller (MPC). They use this LMI formulation to design a control law that results in the least bad case design. This provides robust performance for the nonlinear process represented by a robust model composed of a nominal model supplemented by a model error representation.

Ozkan et. al. (2000) subdivide the control structure of a nonlinear system into pieces sufficiently small to be represented by piecewise linear models. This representation proves to be particularly suited to deal with problems with saturation, relays and dead zones.

2.3.2 Linear Representations of Non-Linear Systems

A linear system with structured uncertainties in the parameters can be represented using the form of equation (2.31) as follows:

$$x_{k+1} = \mathbf{A}(\boldsymbol{\delta}_k)x_k + \mathbf{B}(\boldsymbol{\delta}_k)u_k \tag{2.36}$$
$$x_{k+1} = \left[\mathbf{A}_0 + \delta_{1,k}\mathbf{A}_1 + \ldots + \delta_{n,k}\mathbf{A}_n\right]x_k + \left[\mathbf{B}_0 + \delta_{1,k}\mathbf{B}_1 + \ldots + \delta_{n,k}\mathbf{B}_n\right]u_k$$

where,

$$\mathbf{A}(\boldsymbol{\delta}_k) < 0, \mathbf{B}(\boldsymbol{\delta}_k) < 0$$

and

$$\boldsymbol{\delta}_k = \{\delta_1, \cdots, \delta_n\} \in \Re^n$$

If the values of $\delta_1, \cdots, \delta_n$ are time varying and bounded equation (2.36) describes a time varying linear system as follows:

$$x_{k+1} = \mathbf{A}_k x_k + \mathbf{B}_k u_k \tag{2.37}$$

The bounds on $\delta_1, \cdots, \delta_n$ and the structure of (2.36) imply that $\mathbf{A}_k$ and $\mathbf{B}_k$ are bounded by a polytope of matrices. This polytope is structured as follows:

$$\left[\mathbf{A}_k, \mathbf{B}_k\right] = \sum_{i=1}^{L} \alpha_{i,k}\left[\mathbf{A'}_i, \mathbf{B'}_i\right] \tag{2.38}$$
$$\alpha_{i,k} \geq 0, \quad \sum_{i=1}^{L} \alpha_{i,k} = 1$$

The state matrices used in (2.36) can be represented as a linear combination of a set of i invariant sub-matrices (Kothare et. al., 1994).

Results from Liu (1968) allow us to approximate a non-linear system as a time varying linear system. Provided that the Jacobian below is contained in a polytope of matrices, where the Jacobian is evaluated using the extreme values of the range of the uncertainty in the parameters of the nonlinear system, i.e.:

$$
\begin{aligned}
x_{k+1} &= f(x_k, u_k, k) \\
y_k &= g(x_k, k) \\
\left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial u}, \frac{\partial g}{\partial x}\right] &= \sum_{i=1}^{L} \alpha_{i,k} \left[\mathbf{A'}_i, \mathbf{B'}_i, \mathbf{C'}_i\right]
\end{aligned}
\tag{2.39}
$$

The matrices $[\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i]$ give us the 'vertices' of the parameter space where the true system can exist. The location of the vertices depends on the uncertainty involved in estimating the true parameters of the system.

The polytope resulting from the Jacobian of the nonlinear system is in a form that can be used in the LMI formulation (2.31).

An example of the Jacobian calculation follows for a simple system:

$$
\begin{aligned}
&x_{k+1} = x_k^{\,2} + bu_k \\
&0 \le x \le x_{\max}
\end{aligned}
\tag{2.40}
$$

The Jacobian (2.40) is:

$$
\begin{aligned}
\left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial u}\right] &= \left[\frac{\partial x_k^{\,2}}{\partial x_k}, \frac{\partial b u_k}{\partial u_k}\right] \\
&= [2x_k, b]
\end{aligned}
\tag{2.41}
$$

Using the known bounds on the system, the polytope that bounds (2.40) is:

$$
\begin{aligned}
&x_{k+1} = \begin{bmatrix} \mathbf{A'}_k & \mathbf{B'}_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \\
&\textit{where} \\
&\begin{bmatrix} \mathbf{A'}_k & \mathbf{B'}_k \end{bmatrix} = \alpha_{1,k} \begin{bmatrix} 0 & b \end{bmatrix} + \alpha_{2,k} \begin{bmatrix} 2x_{\max} & b \end{bmatrix}
\end{aligned}
\tag{2.42}
$$

2.3.3 Lyapunov Stability

Lyapunov methods are used to mathematically prove the stability of a system. The two Lyapunov methods are linearization of the system or indirect method, and the Lyapunov direct method.

The linearization method is used to find the stability in an infinitesimal neighbourhood of an equilibrium point. The theorem relates the local stability of the linearized system with the local stability of the non-linear stability, as follows (Slotine and Li, 1991):

- If the linearized system is strictly stable (i.e. $\|\lambda_{\max}\| < 1$), then the equilibrium point of the original non-linear system is asymptotically stable.
- If the linearized system is unstable (i.e. $\|\lambda_{\max}\| > 1$), then the equilibrium point of the original non-linear system is unstable.
- If the linearized system is marginally stable (i.e. $\|\lambda_{\max}\| = 1$), then no conclusion regarding the stability of the equilibrium point of the original non-linear system can be reached.

The disadvantage to the linearization method is that there is no easy determination of how big the stable neighbourhood around the equilibrium point is. This disadvantage, along with the lack of a guarantee of the results for the actual nonlinear system motivates the Lyapunov direct method.

The Lyapunov direct method uses the concept of physical energy, where stability is assured if the energy constantly decreases. The basis of the direct method is to construct a function that gives a scalar value with properties similar to energy, referred to as the *Lyapunov function*, (i.e. there is a unique input that will return a value of zero, and all other inputs will return values larger than zero) that monotonically decreases to zero. Finding an energy function where this condition is not true does not prove instability; i.e. stability may be possible to prove for a different function. Therefore, the disadvantage of this method is that there is no systematic way of selecting the least conservative Lyapunov function. Slotine (Slotine and Li, 1991) has given some techniques that are useful for searching for an appropriate Lyapunov function for linear systems. Given a linear time-invariant system of the form $\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k)$, a *quadratic Lyapunov function* has the form, as follows:

$$V(k) = \mathbf{x}(k)^T \mathbf{P}\mathbf{x}(k) \tag{2.43}$$

Were $V(k)$ is the Lyapunov energy and **P** is a symmetric positive-definite matrix. The Lyapunov direct method for global stability has the requirements (Slotine and Li, 1991):

*Assume that there exists a scalar function V of the state **x**, with continuous first-order derivatives such that*

- $V(k) = \mathbf{x}(k)^T \mathbf{P}\mathbf{x}(k)$ *is positive-definite*
- $V(k+1) - V(k) < 0$
- $V(k) \to \infty$ *as* $\|\mathbf{x}(k)\| \to \infty$

Then the equilibrium point at the origin is globally asymptotically stable.

In this work the Lyapunov direct method is applied to the stability of the adaptive mechanism proposed in chapter 3.

Stability proofs using the quadratic Lyapunov function (2.43) can be referred to as *quadratic Lyapunov stability* proofs.

2.3.4 Quadratic Lyapunov Performance

To be able to evaluate the performance of a controller a measurement method is needed. For control systems, an energy based L_2-norm is usually used. The L_2-norm is defined as:

$$\|\mathbf{e}\|_{L_2} = \int_{-\infty}^{\infty} \mathbf{e}(t)^T \mathbf{e}(t) dt \tag{2.44}$$

The subscript can be omitted for simplicity and the norm is written as $\|\mathbf{e}\|$.

The following system is used for the analysis:

$$\begin{bmatrix} \boldsymbol{\eta}_{k+1} \\ e_k \end{bmatrix} = \begin{bmatrix} \mathbf{A}(\boldsymbol{\delta}_k) & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{\eta}_k \\ \nu_k \end{bmatrix} \tag{2.45}$$

$\boldsymbol{\eta}_0$ is known

where $\boldsymbol{\delta}_k = \{\delta_1, \cdots, \delta_n\} \in \Re^n$ is a vector of uncertain but bounded time-varying real parameters.

ν_K may represent disturbances to the system or alternatively set point changes.

For the following analysis the following assumptions are used:

- $\delta_{i,k}$ is bounded by known values $\underline{\delta_i}$ and $\overline{\delta_i}$, thus $\delta_{i,k} \in \left[\underline{\delta_i}, \overline{\delta_i}\right]$

- The state matrix, $\mathbf{A}(\boldsymbol{\delta}_k)$ is affinely dependant on the parameters as shown in (2.36)

The assumptions restrict the parameter vector $\boldsymbol{\delta}_k$ to existing in a space that forms a hyper-cube with 2^n vertices, which will be referred to as the *parameter space*, as follows:

$$\mathbf{W} = \left\{ (w_1, \cdots, w_n) : w_i \in \left\{ \underline{\delta_i}, \overline{\delta_i} \right\} \right\} \tag{2.46}$$

Quadratic Lyapunov performance is defined as the following (Gao and Budman, 2003):

The system (2.45) *has a zero initial state, satisfies quadratic Lyapunov stability and*

$$\|\mathbf{e}\|_{L_2} < \gamma \|\mathbf{v}\|_{l_2} \tag{2.47}$$

for all L_2-bounded inputs v if there exists $\mathbf{P} > 0, \mathbf{P} = \mathbf{P}^T$ *and a positive definite quadratic Lyapunov function* $V(t) = \boldsymbol{\eta}(t)^T \mathbf{P} \boldsymbol{\eta}(t)$, $V(t) > 0$, *such that*

$$V(t+1) - V(t) + e^T(t)e(t) - \gamma^2 v^T(t)v(t) < 0 \tag{2.48}$$

for all admissible uncertainties $\boldsymbol{\delta}_t$ *and for zero initial conditions* $\boldsymbol{\eta}_0$.

Inequality (2.48) is true if and only if

$$\begin{bmatrix} \mathbf{A}(\boldsymbol{\delta}_i)^T \mathbf{P} \mathbf{A}(\boldsymbol{\delta}_i) - \mathbf{P} & \mathbf{A}(\boldsymbol{\delta}_i)^T \mathbf{P} \mathbf{B} & \mathbf{C}^T \\ \mathbf{B}^T \mathbf{P} \mathbf{A}(\boldsymbol{\delta}_i) & \mathbf{B}^T \mathbf{P} \mathbf{B} - \gamma^2 \mathbf{I} & \mathbf{D}^T \\ \mathbf{C} & \mathbf{D} & -\mathbf{I} \end{bmatrix} < 0 \tag{2.49}$$

holds for all admissible trajectories and initial values of the uncertain parameter vector $\boldsymbol{\delta}_t$. Evaluating (2.49) is not tractable in general because it imposes an infinite number of constraints on **P**. Under the affine parameter dependence assumptions, as shown in equation (2.36) Gao and Budman (2003) have proposed a theorem that shows that (2.49) holds if and only if **P** satisfies the following system of LMI's.

Consider the time-varying system (2.45) *where* $\boldsymbol{\delta}_t, \mathbf{A}(\boldsymbol{\delta}_t)$ *and* $\mathbf{W}$ *are defined as in section 2.3.3. A sufficient condition for quadratic Lyapunov stability of this system is the existence of* $\mathbf{P} > 0, \mathbf{P} = \mathbf{P}^T$ *such that*

$$\begin{bmatrix} \mathbf{A}(w_i)^T \mathbf{P}\mathbf{A}(w_i) - \mathbf{P} & \mathbf{A}(w_i)^T \mathbf{P}\mathbf{B} & \mathbf{C}^T \\ \mathbf{B}^T \mathbf{P}\mathbf{A}(w_i) & \mathbf{B}^T \mathbf{P}\mathbf{B} - \gamma^2 \mathbf{I} & \mathbf{D}^T \\ \mathbf{C} & \mathbf{D} & -\mathbf{I} \end{bmatrix} < 0, \textit{ for all } w \in \mathbf{W} \tag{2.50}$$

A proof is given in (Gao and Budman, 2003).

Inequality (2.50) can be solved as a general eigenvalues problem (GEVP) as in (2.32), to minimise the performance index γ. This minimisation gives the worst expected effect of the disturbance $\|\mathbf{v}\|$ on the error $\|\mathbf{e}\|$ for all the models included in the family of models defined by equation (2.45).

3 Controller and Estimator Design

In this section a novel discrete adaptive controller is developed, based on the continuous time adaptive controller version proposed by Sanner (Sanner and Slotine, 1992). This discrete adaptive controller has not been reported in the literature. This adaptive controller is based on a gradient descent based parameter estimation algorithm combined with a cautious one-step-ahead feedback control law. The one-step-ahead controller is used for simplicity of the mathematical development, but the stability and tuning results are valid for other forms of feedback laws, such as pole-placement algorithms.

3.1 Definitions

Given a DARMA (Discrete Autoregressive Moving Average) model of a system that is n^{th} order with respect to the state and m^{th} order with respect to the input:

$$y_{k+1} = \sum_{i=0}^{n-1} a_i y_{k-i} + \sum_{j=0}^{m-1} b_j u_{k-j} \tag{3.1}$$

The vectors of the parameters a_i and b_j are defined as follows:

$$\mathbf{A} = \begin{bmatrix} a_0 & \cdots & a_{n-1} \end{bmatrix}^T \tag{3.2}$$

$$\mathbf{B} = \begin{bmatrix} b_0 & \cdots & b_{m-1} \end{bmatrix}^T \tag{3.3}$$

The parameter estimate vectors are defined as follows:

$$\hat{\mathbf{A}}_k = \begin{bmatrix} \hat{a}_{0,k} & \cdots & \hat{a}_{n-1,k} \end{bmatrix}^T \tag{3.4}$$

$$\hat{\mathbf{B}}_k = \begin{bmatrix} \hat{b}_{0,k} & \cdots & \hat{b}_{m-1,k} \end{bmatrix}^T \tag{3.5}$$

Let the values of past input and output data be given by the following vectors:

$$\mathbf{Y}_k = \begin{bmatrix} y_k & \cdots & y_{k-n+1} \end{bmatrix}^T \tag{3.6}$$

$$\mathbf{U}_k = \begin{bmatrix} u_k & \cdots & u_{k-m+1} \end{bmatrix}^T \tag{3.7}$$

Then, using equations (3.2), (3.3), (3.6) and (3.7), the DARMA model given by equation (3.1), can be reformulated in terms of the input and output vectors as follows:

$$y_{k+1} = \mathbf{A}^T \mathbf{Y}_k + \mathbf{B}^T \mathbf{U}_k \tag{3.8}$$

Also for the purpose of designing an implementable controller, let

$$\hat{\mathbf{B}}_k^{old} = \begin{bmatrix} \hat{b}_{1,k} & \cdots & \hat{b}_{m-1,k} \end{bmatrix}^T \tag{3.9}$$

and,

$$\mathbf{U}_k^{old} = \begin{bmatrix} u_{k-1} & \cdots & u_{k-m+1} \end{bmatrix}^T \tag{3.10}$$

A filtered feedback error, to be justified in sections 3.2 and 3.3, is given as follows:

$$s_k = \frac{-\hat{\mathbf{A}}_k{}^T \mathbf{Y}_{k-1} - \hat{\mathbf{B}}_k{}^T \mathbf{U}_{k-1} + 2y_k + (1-K_D)s_{k-1} - \hat{\mathbf{A}}_{k-1}{}^T \mathbf{Y}_{k-1} - \hat{\mathbf{B}}_{k-1}{}^T \mathbf{U}_{k-1}}{1+K_D} \tag{3.11}$$

For simplicity, an adaptive algorithm based on a one-step-ahead controller (equation 2.18) will be used. A term proportional to the filtered error, s_k, is added, multiplied by a tuning parameter K_D, as follows:

$$u_k = \left(y_{sp} - \hat{\mathbf{A}}_k{}^T \mathbf{Y}_k - \hat{\mathbf{B}}_k^{old\,T} \mathbf{U}_k^{old} + (1-K_D)s_k \right) \cdot \hat{b}_{0,k}{}^{-1} \tag{3.12}$$

Where $K_D > 0$

The errors in the estimated parameters are defined in the form of deviation variables as follows:

$$\hat{\mathbf{A}}_k = \widetilde{\mathbf{A}}_k + \mathbf{A} \tag{3.13}$$

$$\hat{\mathbf{B}}_k = \widetilde{\mathbf{B}}_k + \mathbf{B} \tag{3.14}$$

The gradient descent method is used to formulate the parameter update equations where the error used for updating is the sum of the current and past value of the filtered errors, $(s_k + s_{k-1})$:

$$\hat{\mathbf{A}}_k = \hat{\mathbf{A}}_{k-1} + \mathbf{K}_\mathbf{A}\mathbf{Y}_{k-1}(s_k + s_{k-1}) \tag{3.15}$$

$$\hat{\mathbf{B}}_k = \hat{\mathbf{B}}_{k-1} + \mathbf{K}_\mathbf{B}\mathbf{U}_{k-1}(s_k + s_{k-1}) \tag{3.16}$$

$\mathbf{K}_\mathbf{A}, \mathbf{K}_\mathbf{B}$ are matrices of adaptation gains, with diagonalstructure as given below:

$$\mathbf{K}_\mathbf{A} = \begin{bmatrix} K_{A,0} & & 0 \\ & \ddots & \\ 0 & & K_{A,n-1} \end{bmatrix} \tag{3.17}$$

$\mathbf{K}_\mathbf{A}$ is invertible and $\mathbf{K}_{\mathbf{A},i,i} > 0, i \in \{0,1,...,n-1\}$

and,

$$\mathbf{K}_\mathbf{B} = \begin{bmatrix} K_{B,0} & & 0 \\ & \ddots & \\ 0 & & K_{B,m-1} \end{bmatrix} \tag{3.18}$$

$\mathbf{K}_\mathbf{B}$ is invertible and $\mathbf{K}_{\mathbf{B},j,j} > 0, j \in \{0,1,...,m-1\}$

3.2 Proof of Controller Stability

To ensure that the controller defined in the previous section is stable and that the parameters converge to their true values when the model structure is correct, a Lyapunov stability proof is presented. It should be recalled that Lyapunov methods require a positive definite 'energy' function. This implies that the energy is zero at the origin and greater than zero at any other state. If the energy can be shown to be non-increasing with time, then the estimate is stable, and if excitation conditions occur such as this energy function never converges to a non-zero condition, the estimates will converge to their true values. The error filter, s_k, has to be designed to fulfill this Lyapunov criterion.

The Lyapunov function used is a quadratic function made of the combination of the squares of the estimation errors and the filtered error as follows:

$$V_k = \widetilde{\mathbf{A}}_k{}^T \mathbf{K}_{\mathbf{A}}{}^{-1} \widetilde{\mathbf{A}}_k + \widetilde{\mathbf{B}}_k{}^T \mathbf{K}_B{}^{-1} \widetilde{\mathbf{B}}_k + s_k^2 \tag{3.19}$$

Examining the structure of equation (3.19), it can be seen that every term is composed of a tuning factor multiplied by the square of the error of each one of the parameter estimates, or by the square of s_k. Clearly each term is positive definite with respect to an origin that corresponds to zero filtered error and convergence of the parameter estimates to their true values.

Substituting equation (3.12) into the system equation, (3.8) results in the following:

$$y_{k+1} = \mathbf{A}^T \mathbf{Y}_k + \mathbf{B}_k^{old\,T} \mathbf{U}_k^{old} + b_0 \cdot \hat{b}_{0,k}{}^{-1} \left(y_{sp} - \hat{\mathbf{A}}_k{}^T \mathbf{Y}_k - \hat{\mathbf{B}}_k^{old\,T} \mathbf{U}_k^{old} + (1 - K_D) s_k \right) \tag{3.20}$$

When the Lyapunov energy converges to zero, $\hat{\mathbf{A}}_k = \mathbf{A}, \hat{\mathbf{B}}_k = \mathbf{B}$ and $s_k = 0$. When these values are used in equation (3.20), the equation reduces to the following:

$$y_{k+1} = \mathbf{A}^T \mathbf{Y}_k - \hat{\mathbf{A}}_k{}^T \mathbf{Y}_k + \mathbf{B}_k^{old\,T} \mathbf{U}_k^{old} - \hat{\mathbf{B}}_k^{old\,T} \mathbf{U}_k^{old} + b_0 \cdot \hat{b}_{0,k}{}^{-1} \left(y_{sp} + (1 - K_D) s_k \right)$$

$$y_{k+1} = y_{sp} \tag{3.21}$$

Thus, proof of the convergence of the estimated values, to the true values, ensures that the closed loop system tracks the set-point. It should be noted that any controller that is stable when designed with the true system parameters (i.e. pole-placement or model reference controllers) will ensure closed loop stability.

Substituting the Lyapunov convergence conditions, and $s_k = s_{k-1} = 0$ into equation (3.11) results in the following:

$$y_k = 0.5 \left(\hat{\mathbf{A}}_k + \hat{\mathbf{A}}_{k-1} \right)^T \mathbf{Y}_{k-1} + 0.5 \left(\hat{\mathbf{B}}_k + \hat{\mathbf{B}}_{k-1} \right)^T \mathbf{U}_{k-1} \tag{3.22}$$

The RHS of equation (3.22) is the average prediction for y using the parameter estimates at the current time step, and the last time step. When the estimates have converged, $\hat{\mathbf{A}}_k = \hat{\mathbf{A}}_{k-1}$ and $\hat{\mathbf{B}}_k = \hat{\mathbf{B}}_{k-1}$, which results in the following equation:

$$y_k = \hat{\mathbf{A}}_k{}^T \mathbf{Y}_{k-1} + \hat{\mathbf{B}}_k{}^T \mathbf{U}_{k-1} \tag{3.23}$$

Thus, when the estimators converge, the prediction error for y is zero when there is no measurement noise.

After substituting equations (3.13) and (3.14) into equation (3.19):

$$V_k = \left(\hat{\mathbf{A}}_k - \mathbf{A}\right)^T \mathbf{K_A}^{-1}\left(\hat{\mathbf{A}}_k - \underline{\mathbf{A}}\right) + \left(\hat{\mathbf{B}}_\mathbf{k} - \mathbf{B}\right)^T \mathbf{K_B}^{-1}\left(\hat{\mathbf{B}}_\mathbf{k} - \mathbf{B}\right) + s_k^2 \tag{3.24}$$

And expanding terms:

$$\begin{aligned} V_k = {} & \hat{\mathbf{A}}_k \mathbf{K_A}^{-1^T} \hat{\mathbf{A}}_k + \mathbf{A}^T \mathbf{K_A}^{-1} \mathbf{A} - \hat{\mathbf{A}}_k{}^T \mathbf{K_A}^{-1} \mathbf{A} - \mathbf{A}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k \\ & + \hat{\mathbf{B}}_\mathbf{k}{}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_\mathbf{k} + \mathbf{B}^T \mathbf{K_B}^{-1} \mathbf{B} - \hat{\mathbf{B}}_\mathbf{k}{}^T \mathbf{K_B}^{-1} \mathbf{B} - \mathbf{B}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_\mathbf{k} + s_k^2 \end{aligned} \tag{3.25}$$

K_A and K_B are symmetric, thus:

$$\hat{\mathbf{A}}_k{}^T \mathbf{K_A}^{-1} \mathbf{A} = \mathbf{A}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k \tag{3.26}$$

$$\hat{\mathbf{B}}_\mathbf{k}{}^T \mathbf{K_B}^{-1} \mathbf{B} = \mathbf{B}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_\mathbf{k} \tag{3.27}$$

Substituting equations (3.26) and (3.27) into (3.25) gives:

$$\begin{aligned} V_k = {} & \hat{\mathbf{A}}_k{}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k + \mathbf{A}^T \mathbf{K_A}^{-1} \mathbf{A} - 2\hat{\mathbf{A}}_k{}^T \mathbf{K}_A{}^{-1} \mathbf{A} \\ & + \hat{\mathbf{B}}_\mathbf{k}{}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_\mathbf{k} + \mathbf{B}^T \mathbf{K_B}^{-1} \mathbf{B} - 2\hat{\mathbf{B}}_\mathbf{k}{}^T \mathbf{K_B}^{-1} \mathbf{B} + s_k^2 \end{aligned} \tag{3.28}$$

For Lyapunov stability the 'energy' is required to be a non-increasing function as given below:

$$V_{k+1} - V_k \leq 0 \tag{3.29}$$

To satisfy this criteria, equation (3.28) is formulated at interval k+1 and from the resulting equation, equation (3.28), is subtracted, resulting in the following expression:

$$\begin{aligned} V_{k+1} - V_k = {} & \hat{\mathbf{A}}_{k+1}{}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_{k+1} - \hat{\mathbf{A}}_k{}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k + \mathbf{A}^T \mathbf{K_A}^{-1} \mathbf{A} - \mathbf{A}^T \mathbf{K_A}^{-1} \mathbf{A} \\ & - 2\hat{\mathbf{A}}_{k+1}{}^T \mathbf{K_A}^{-1} \mathbf{A} + 2\hat{\mathbf{A}}_k{}^T \mathbf{K_A}^{-1} \mathbf{A} + \hat{\mathbf{B}}_{\mathbf{k}+1}{}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k}+1} - \hat{\mathbf{B}}_\mathbf{k}{}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_\mathbf{k} \\ & + \mathbf{B}^T \mathbf{K_B}^{-1} \mathbf{B} - \mathbf{B}^T \mathbf{K_B}^{-1} \mathbf{B} - 2\hat{\mathbf{B}}_{\mathbf{k}+1}{}^T \mathbf{K_B}^{-1} \mathbf{B} + 2\hat{\mathbf{B}}_\mathbf{k}{}^T \mathbf{K_B}^{-1} \mathbf{B} + s_{k+1}^2 - s_k^2 \end{aligned} \tag{3.30}$$

After collecting like terms:

$$V_{k+1} - V_k = \hat{\mathbf{A}}_{k+1}{}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_{k+1} - \hat{\mathbf{A}}_k{}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k - 2\left(\hat{\mathbf{A}}_{k+1} - \hat{\mathbf{A}}_k\right)^T \mathbf{K_A}^{-1} \mathbf{A} + \hat{\mathbf{B}}_{\mathbf{k+1}}{}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k+1}} - \hat{\mathbf{B}}_{\mathbf{k}}{}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k}} - 2\left(\hat{\mathbf{B}}_{\mathbf{k+1}} - \hat{\mathbf{B}}_{\mathbf{k}}\right)^T \mathbf{K_B}^{-1} \mathbf{B} + s_{k+1}^2 - s_k^2 \quad (3.31)$$

Completing the square as follows:

$$\hat{\mathbf{A}}_{k+1}{}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_{k+1} - \hat{\mathbf{A}}_k{}^T \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k = \left(\hat{\mathbf{A}}_{k+1} - \hat{\mathbf{A}}_k\right)^T \left(\mathbf{K_A}^{-1} \hat{\mathbf{A}}_{k+1} + \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k\right) \quad (3.32)$$

$$\hat{\mathbf{B}}_{\mathbf{k+1}}{}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k+1}} - \hat{\mathbf{B}}_{\mathbf{k}}{}^T \mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k}} = \left(\hat{\mathbf{B}}_{\mathbf{k+1}} - \hat{\mathbf{B}}_{\mathbf{k}}\right)^T \left(\mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k+1}} + \mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k}}\right) \quad (3.33)$$

Equation (3.32) and (3.33) are substituted into (3.31), and then rearranged as follows:

$$V_{k+1} - V_k = \left(\hat{\mathbf{A}}_{k+1} - \hat{\mathbf{A}}_k\right)^T \left(\mathbf{K_A}^{-1} \hat{\mathbf{A}}_{k+1} + \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k - 2\mathbf{K_A}^{-1} \mathbf{A}\right) + \left(\hat{\mathbf{B}}_{\mathbf{k+1}} - \hat{\mathbf{B}}_{\mathbf{k}}\right)^T \left(\mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k+1}} + \mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k}} - 2\mathbf{K_B}^{-1} \mathbf{B}\right) + \left(s_{k+1} + s_k\right)\left(s_{k+1} - s_k\right) \quad (3.34)$$

After calculating equations (3.15) and (3.16) at interval k+1 and substituting the result into equation (3.34), the following expression results:

$$V_{k+1} - V_k = \left(s_{k+1} + s_k\right)^T \mathbf{Y}_k{}^T \mathbf{K_A}^T \left(\mathbf{K_A}^{-1} \hat{\mathbf{A}}_{k+1} + \mathbf{K_A}^{-1} \hat{\mathbf{A}}_k - 2\mathbf{K_A}^{-1} \mathbf{A}\right) + \left(s_{k+1} + s_k\right)^T \mathbf{U}_k{}^T \mathbf{K_B}^T \left(\mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k+1}} + \mathbf{K_B}^{-1} \hat{\mathbf{B}}_{\mathbf{k}} - 2\mathbf{K_B}^{-1} \mathbf{B}\right) + \left(s_{k+1} + s_k\right)\left(s_{k+1} - s_k\right) \quad (3.35)$$

Finally, after rearranging equation (3.35):

$$V_{k+1} - V_k = \left(s_{k+1} + s_k\right)\left[\mathbf{Y}_k{}^T \left(\hat{\mathbf{A}}_{k+1} + \hat{\mathbf{A}}_k - 2\mathbf{A}\right) + \mathbf{U}_k{}^T \left(\hat{\mathbf{B}}_{\mathbf{k+1}} + \hat{\mathbf{B}}_{\mathbf{k}} - 2\mathbf{B}\right) + \left(s_{k+1} - s_k\right)\right] \quad (3.36)$$

The filtered error is defined by the implicit equation given below:

$$\left[\mathbf{Y}_k{}^T \left(\hat{\mathbf{A}}_{k+1} + \hat{\mathbf{A}}_k - 2\mathbf{A}\right) + \mathbf{U}_k{}^T \left(\hat{\mathbf{B}}_{\mathbf{k+1}} + \hat{\mathbf{B}}_{\mathbf{k}} - 2\mathbf{B}\right) + \left(s_{k+1} - s_k\right)\right] = -K_D\left(s_{k+1} + s_k\right) \quad (3.37)$$

Substitute equation (3.37) into (3.36) as given below:

$$V_{k+1} - V_k = -K_D\left(s_{k+1} + s_k\right)^2 \tag{3.38}$$

Equation (3.38) satisfies the requirement that the Lyapunov function is decreasing with time for any positive value of K_D.

3.3 Implementation of the adaptive estimation method

In this section an implementable and causal version of the control law given in equation (3.12) will be presented. Equation (3.12) is not directly implementable because it depends on values available at time k. The implementable form of u_k should be a function of values observable at time k-1.

Given equation (3.37) and expanding terms, the following expression is obtained:

$$\begin{aligned}&\mathbf{Y}_k{}^T\hat{\mathbf{A}}_{k+1} + \mathbf{Y}_k{}^T\hat{\mathbf{A}}_k - 2\mathbf{Y}_k{}^T\mathbf{A} + \mathbf{U}_k{}^T\hat{\mathbf{B}}_{\mathbf{k}+1} + \mathbf{U}_k{}^T\hat{\mathbf{B}}_{\mathbf{k}} \\ &- 2\mathbf{U}_k{}^T\mathbf{B} + \left(K_D + 1\right)s_{k+1} + \left(K_D - 1\right)s_k = 0\end{aligned} \tag{3.39}$$

After substituting equation (3.8) into equation (3.39),

$$\begin{aligned}&\mathbf{Y}_k{}^T\hat{\mathbf{A}}_{k+1} + \mathbf{Y}_k{}^T\hat{\mathbf{A}}_k - 2y_{k+1} + 2\mathbf{U}_k{}^T\mathbf{B} + \mathbf{U}_k{}^T\hat{\mathbf{B}}_{\mathbf{k}+1} \\ &+ \mathbf{U}_k{}^T\hat{\mathbf{B}}_{\mathbf{k}} - 2\mathbf{U}_k{}^T\mathbf{B} + \left(K_D + 1\right)s_{k+1} + \left(K_D - 1\right)s_k = 0\end{aligned} \tag{3.40}$$

After collecting terms in equation (3.40):

$$\mathbf{Y}_k{}^T\hat{\mathbf{A}}_{k+1} + \mathbf{Y}_k{}^T\hat{\mathbf{A}}_k - 2y_{k+1} + \mathbf{U}_k{}^T\hat{\mathbf{B}}_{\mathbf{k}+1} + \mathbf{U}_k{}^T\hat{\mathbf{B}}_{\mathbf{k}} + \left(K_D - 1\right)s_k = -\left(K_D + 1\right)s_{k+1} \tag{3.41}$$

Or explicitly, from equation (3.41) in terms of time k:

$$s_k = \hat{\mathbf{A}}_k{}^T \left[\frac{-\mathbf{Y}_{k-1}}{1+K_D}\right] + \underline{\hat{B}_k}{}^T \left[\frac{-\mathbf{U}_{k-1}}{1+K_D}\right] + \left[\frac{2y_k + (1-K_D)s_{k-1} - \hat{\mathbf{A}}_{k-1}{}^T \mathbf{Y}_{k-1} - \hat{\mathbf{B}}_{\mathbf{k-1}}{}^T \mathbf{U}_{k-1}}{1+K_D}\right] \tag{3.42}$$

From equation (3.42), the following terms are defined:

$$\mathbf{S}_{A,k} = \left[\frac{-\mathbf{Y}_{\mathbf{k-1}}}{1+K_D}\right] \tag{3.43}$$

$$\mathbf{S}_{B,k} = \left[\frac{-\mathbf{U}_{k-1}}{1+K_D}\right] \tag{3.44}$$

$$S_{f,k} = \left[\frac{2y_k + (1-K_D)s_{k-1} - \hat{\mathbf{A}}_{k-1}{}^T \mathbf{Y}_{k-1} - \hat{\mathbf{B}}_{k-1}{}^T \mathbf{U}_{k-1}}{1+K_D}\right] \tag{3.45}$$

And from equation (3.15) and (3.16) define:

$$\hat{\mathbf{A}}_k = s_k \left[\mathbf{K}_{\mathbf{A}} \cdot \mathbf{Y}_{k-1}\right] + \left[\hat{A}_{k-1} + \mathbf{K}_{\mathbf{A}} \cdot \mathbf{Y}_{k-1} s_{k-1}\right] \tag{3.46}$$

$$\mathbf{A}_{S,k} = \left[\mathbf{K}_{\mathbf{A}} \cdot \mathbf{Y}_{k-1}\right] \tag{3.47}$$

$$\mathbf{A}_{f,k} = \left[\hat{\mathbf{A}}_{k-1} + \mathbf{K}_{\mathbf{A}} \cdot \mathbf{Y}_{k-1} s_{k-1}\right] \tag{3.48}$$

$$\hat{\mathbf{B}}_k = s_k \left[\mathbf{K}_{\mathbf{B}} \cdot \mathbf{U}_{k-1}\right] + \left[\hat{\mathbf{B}}_{k-1} + \mathbf{K}_{\mathbf{B}} \cdot \mathbf{U}_{k-1} s_{k-1}\right] \tag{3.49}$$

$$\mathbf{B}_{S,k} = \left[\mathbf{K}_{\mathbf{B}} \cdot \mathbf{U}_{k-1}\right] \tag{3.50}$$

$$\mathbf{B}_{f,k} = \left[\hat{\mathbf{B}}_{k-1} + \mathbf{K}_{\mathbf{B}} \cdot \mathbf{U}_{k-1} s_{k-1}\right] \tag{3.51}$$

Substitutions of equations (3.49) and (3.46) into equation (3.42), and use of definitions (3.43), (3.44), (3.45), (3.47), (3.48), (3.50) and (3.51), result in the following expression:

$$s_k = \left(s_k \mathbf{A}_{S,k} + \mathbf{A}_{f,k}\right)^T \mathbf{S}_{A,k} + \left(s_k \mathbf{B}_{S,k} + \mathbf{B}_{f,k}\right)^T \mathbf{S}_{B,k} + S_{f,k} \tag{3.52}$$

Or, rearranging equation (3.52):

$$s_k \left[1 - \mathbf{A}_{S,k}{}^T \mathbf{S}_{A,k} - \mathbf{B}_{S,k}{}^T \mathbf{S}_{B,k}\right] = \mathbf{A}_{f,k}{}^T \mathbf{S}_{A,k} + \mathbf{B}_{f,k}{}^T \mathbf{S}_{B,k} + S_{f,k} \tag{3.53}$$

Finally, solving for s_k:

$$s_k = \frac{\mathbf{A}_{f,k}{}^T \mathbf{S}_{A,k} + \mathbf{B}_{f,k}{}^T \mathbf{S}_{B,k} + S_{f,k}}{1 - \mathbf{A}_{S,k}{}^T \mathbf{S}_{A,k} - \mathbf{B}_{S,k}{}^T \mathbf{S}_{B,k}} \tag{3.54}$$

In the last equation, all the expressions in the right hand side are given by definitions (3.43) - (3.51). All of these definitions are causal, i.e. they are functions of values obtained at time k-1, and can be measured or calculated, and therefore can be implemented on line. Using expressions (3.54), (3.49) and (3.46) the control action can be calculated online using equation (3.12).

3.4 Avoidance of Division by Zero

In equation (3.12) there is a division by $\hat{b}_{0,k}$. This is an estimated parameter value, and the Lyapunov stability criteria only guarantees that the estimate will converge for $t \to \infty$. However, during transients, this parameter estimate may reach a zero value. When the parameter $\hat{b}_{0,k}$ reaches a value of zero, numerical problems in calculating the next control input will arise due to a division by zero in equation (3.12). To avoid the numerical

problems if a division by zero is calculated all estimation values should be reset to the last time step. For the algorithm considered in this work the values that would be reset are $\hat{\mathbf{A}}_k = \hat{\mathbf{A}}_{k-1}$, $\hat{\mathbf{B}}_k = \hat{\mathbf{B}}_{k-1}$, $u_k = u_{k-1}$ and $s_k = s_{k-1}$.

The question for stability considerations is whether the situation that leads to a division by zero error will occur for two or more time intervals, that is:

$$\hat{b}_{k,0} = s_k \left[K_{B,0} u_{k-1} \right] + \left[\hat{b}_{k-1,0} + K_{B,0} u_{k-1} s_{k-1} \right] = 0 \tag{3.55}$$

If $\hat{\mathbf{A}}_{k-1}, \hat{\mathbf{B}}_{k-1}$ and s_{k-1} and therefore u_{k-1} are held constant then the only variable in equation (3.55) that can change from one interval to the next is s_k.

According to equation (3.54):

$$s_k = \alpha y_k + \beta \tag{3.56}$$

Where α and β are functions of the variables $\hat{\mathbf{A}}_{k-1}, \hat{\mathbf{B}}_{k-1}, s_{k-1}$ and y_k is the observed output at time k.

Then, it is obvious that for $\hat{b}_{k,0}$ to remain zero, it is necessary y_k or s_k remain constant from interval to interval. However it is clear that in a dynamic system, y_k will change with time. Even in the case that the system is at steady state, y_k will always be corrupted by random measurement error and therefore will change. Thus, equation (3.55) will not occur for an infinite number of consecutive time steps. Consequently, Lyapunov function

will eventually continue to decrease towards the origin corresponding to convergence of the parameters to their actual values and convergence of the feedback error to zero.

3.5 Parallels with PI Control

In this section, the similarities between the controller algorithm proposed in this study and conventional PI controllers will be explained.

A conventional discrete PI controller has been given in the literature (e.g. Seborg, Edgar and Mellichamp,1989), as follows:

$$U(z) = K_C\left[1 + \frac{\Delta t}{\tau_I}\left(\frac{1}{1-z^{-1}}\right)\right]E(z) \tag{3.57}$$

The control law consists of a proportional term, $K_C E(z)$ and an integral term, $\frac{\Delta t}{\tau_I}\left(\frac{1}{1-z^{-1}}\right)E(z)$. The similarities between the controller presented in this section and a PI controller may be established by inspection of equation (3.12) as follows:

$$u_k = \left(y_{sp} - \hat{\mathbf{A}}_k{}^T\mathbf{Y}_k - \hat{\mathbf{B}}_k^{old\,T}\mathbf{U}_k^{old} + (1-K_D)s_k\right)\cdot \hat{b}_{0,k}{}^{-1}$$

The term $(1-K_D)s_k \cdot \hat{b}_{0,k}{}^{-1}$ is equivalent to a nonlinear proportional action because $\hat{b}_{0,k}$ is a function of s_k. The term $y_{sp} - \hat{\mathbf{A}}_k{}^T\mathbf{Y}_k - \hat{\mathbf{B}}_k^{old\,T}\mathbf{U}_k^{old}$ represents the prediction error, which fulfills the function of an integrator as shown in the sequel. The estimates update equation, (3.15):

$$\hat{\mathbf{A}}_k = \hat{\mathbf{A}}_{k-1} + \mathbf{K}_A \mathbf{Y}_{k-1}\left(s_k + s_{k-1}\right)$$

This equation can be represented as a z-transform as follows:

$$\hat{\mathbf{A}}(z) = \frac{1}{\left(1 - z^{-1}\right)} \cdot Z\left[\mathbf{K}_\mathbf{A} f\left(\mathbf{Y}_{k-1}, s_k, s_{k-1}\right)\right] \tag{3.58}$$

Thus, $\hat{\mathbf{A}}$ is the integration of a non-linear function of $\mathbf{Y}_{k-1}, s_k$ and s_{k-1}.

4 Theory and Methods

One of the significant features of dual control is the combination of cautious control with some form of probing. As discussed in section 2.2.3, cautious control reduces the aggression of the feedback law to take into account the uncertainty and the noise in the system, and the probing capability refers to the addition of an excitation signal to the control action to ensure the convergence of the parameter estimates. In most dual control studies the control action, which includes the excitation signal for probing, is designed where the adaptation gains are selected ad-hoc, or rules of thumb are used to achieve optimal control performance. For this work, the amount of excitation or probing will be considered to be fixed, and it is contained in the reference signal. On the other hand, the adaptation gains K_A and K_B are used to adjust the rate of adaptation. Selecting a high value for the adaptation gains will tend to speed adaptation, while causing the estimates to be more sensitive to measured noise. The filtered error gain, K_D, determines the system response to s_k. The trade-off between cautiousness and probing in this work will be achieved by adjusting the adaptation and filter gains in an attempt to balance between adaptation speed and fast oscillations that may be caused by aggressive adaptation in the presence of noise and model uncertainty.

4.1 Tuning using tracks

4.1.1 Rational

The stability proofs for the discrete adaptive controller developed in section 3.2, as well as other adaptive control methods in the literature, require that the tuning parameters be constant with respect to time. Thus, on-line tuning of the parameters is, in principle, not allowed. However, multiple simultaneous estimations of the plant model parameters can be calculated with the same input-output data but different tuning parameters. In this chapter a method for switching parameters for the purpose of tuning the controller is developed. Each of these simultaneous calculations will be referred to as an *estimation track*. At any given time interval, one of the estimation tracks is selected, where the criteria for selection is as discussed in section 3.2. It will be shown in the next section, that stability and convergence properties can be still maintained using this method of switching between estimation tracks. Then, the values of the plant model parameter estimates, and the filtered error, s_k in the selected estimation track can be used to calculate the control action. For conventional adaptive controllers reported in the literature, the probing element of the controller is introduced in the control action. Thus, tuning is difficult in these circumstances because alternative tuning parameters, and hence control actions cannot be simultaneously calculated and independently implemented.

4.1.2 Stability of Estimation Tracks

In this section the stability of the proposed tuning method is explained. For each estimation track, the estimate is guaranteed to be stable and converge only if the tuning constants, K_A, K_B and K_D, are constant with respect to time. When a new track is deemed to be the best according to an optimisation criteria, the parameter estimates $\underline{\hat{B}}_k$ and $\underline{\hat{A}}_k$ and the filtered error s_k are reset to the values corresponding to the new track. As a result of this, at each time step this switch can cause a local increase in Lyapunov energy. As an example refer to the jump in Lyapunov energy between letters 'A' and 'B' in Figure 4.1. In this figure the curves labelled '1' through '5' refer to the energy of the tracks corresponding to the parameter sets $\left[K_{A,1}, K_{B,1}, K_{D,1}\right]$ through $\left[K_{A,5}, K_{B,5}, K_{D,5}\right]$ respectively. Despite the possibility of temporary jumps in Lyapunov energy all the tracks eventually converge, as shown in Figure 4.1. Thus overall stability, i.e. decrease of the Lyapunov function is obtained although temporary increases in this function may occur.

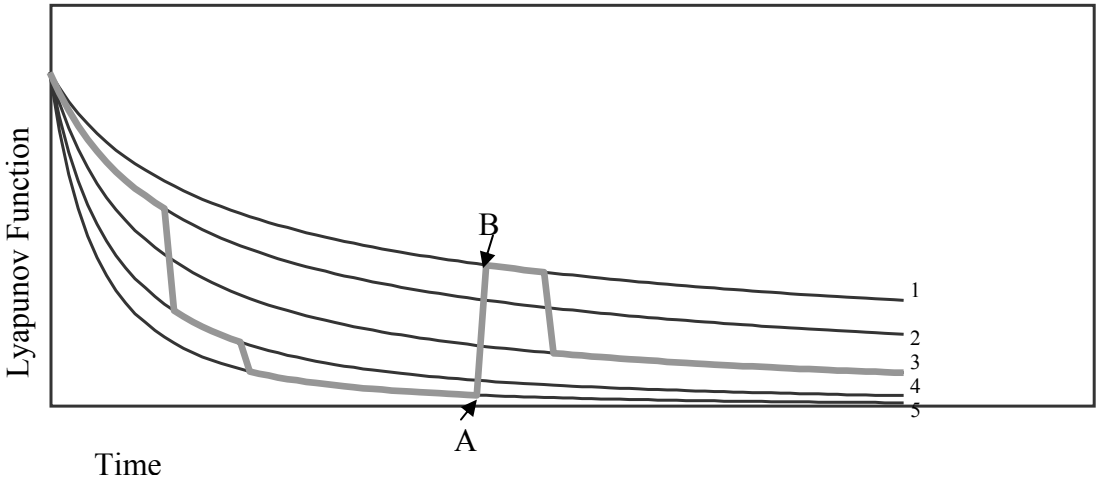

Figure 4.1: Lyapunov energy of each estimation track with the current value superimposed

Also, in section 3.4 a method for avoiding numerical problems caused by division by zero was given where all estimation values are frozen until a non-zero estimate of $\hat{b}_{0,k}$ can be found. When multiple tracks are used as explained above, if at least one track has an estimate for $\hat{b}_{0,k}$ that is non-zero, then the tracks with a zero estimate can be temporarily excluded for consideration for use in designing the control law, at that specific interval. Only if all tracks have an estimate of zero for $\hat{b}_{0,k}$, then all estimates must be frozen. It should be remembered that the condition for a zero estimate of $\hat{b}_{0,k}$ to be calculated for two or more intervals is that y_k or equivalently s_k remain constant from interval to interval. However as explained in section 3.4, this will never happen because either the system will be transient, i.e. y_k will change, or y_k will be corrupted by measurement noise. For this analysis a first order system has been considered, but for a higher order system, there are more parameters in the condition equation which must match between tracks, making the problem of having all estimates for $\hat{b}_{0,k}$ remaining at zero even more remote.

4.1.3 Set selection

Each track has a combination of values for $K_{A,}$ K_B and K_D. There are n tuneable values in $K_{A,}$ m tuneable values in K_B and K_D is scalar, for a total of n+ m + 1 tuneable parameters. The parameter sets to be considered are selected to cover the range of likely

'good' tuning parameters. An example of computer code for the set selection is given below:

```
for a = 0 : (tracks_to_check - 1)
 for b = 0 : (tracks_to_check - 1)
  for d = 0 : (tracks_to_check - 1)
   set_number = set_number + 1;
   set(:,set_number) = [(a*4 + 1) / tracks_to_check);
    … (b*4 + 1) / tracks_to_check);
    … (d*4 + 1) / tracks_to_check)];
  end
 end
end
```

(4.1)

This code selects parameters on a grid, evenly spaced between values of 0 and 4, excluding these values. To select the range of parameters, some a priori knowledge of the approximate values of the model parameters is required, e.g. an approximate value for the time constant for the system. The amount of computational resources available will influence how good this a priori knowledge needs to be.

4.2 Track selection methods

Ideally, the track with the lowest Lyapunov energy should be selected, but to calculate the Lyapunov function the true values of the plant model parameters have to be available. Since the parameters are not known, the Lyapunov energy cannot be calculated online. In the following sub-sections two methods are discussed to select the proper track. The Bicriteria method attempts to identify the track with the lowest expectation for prediction and feedback errors. The second method is based on an LMI formulation that finds the worst case scenario in the range of expected model error or uncertainty for each track. The track with the lowest 'worst-case' score is used.

4.2.1 Bicriteria Method

The bicriteria error function used by Filatov (Filatov et. al., 1997), given before as equation 2.30, is repeated below, in terms of the variables used in the method described in this work:

$$J_k^c = E\left\{\left[y_{sp,k+1} - y_{k+1}\right]^2 \mid \Im_k\right\}, \qquad (4.2)$$

$$J_k^a = E\left\{\left[y_{k+1} - \underline{\hat{A}_k}^T \underline{Y_k} - \underline{\hat{B}_k}^T \underline{U_k}\right]^2 \mid \Im_k\right\}$$

These two terms represent the expectations of the feedback or short-term tracking error and the prediction or long term error respectively. Minimising the error is good for short term purposes; however, if the error is small, learning of the model will be slower and therefore the prediction error will be larger which is detrimental in the long run. The key advantage of using the bicriteria cost functions is that they explicitly describe the trade-off between short and long-term performance that is required in the design of any adaptive controller. An adaptive controller that produces a sequence of control actions that produce the minimum possible value for the combined bicriteria cost functions is referred to as an *optimal controller* in the literature. It was mentioned in section 2.2.4 that an analytic solution of the optimisation of (4.2) is not possible for most systems. Thus, most adaptive control methods focus on suboptimal control; where the control sequence used gives good performance, but not necessarily the best possible performance. For the estimation track tuning method proposed in this work, a finite set of valid tuning parameter combinations is considered, and hence a finite set of valid control actions at any given time interval can be calculated. The set of all valid control

actions does not include all possible control actions. Thus this estimation track tuning method is expected to be suboptimal.

4.2.1.1 Method

The bicriteria cost functions are an expectation of the value of the feedback and prediction errors at the next time step. For this work this expected value is calculated by simulation. The initial value of the plant parameters is taken on a grid to cover the range of uncertainty in the estimates, and weighted by the likelihood that the estimate is the correct one.

The filtered error, s_k can be used to measure the uncertainty in the estimated values of the parameters. The 'energy' or root-mean-square of recent values of s_k will be used as a basis for the calculation of the confidence interval of the parameter estimates. In this work a periodic excitation signal of square wave type has been used. Most adaptation occurs when the excitation signal has large jumps, i.e. at the end of the period, so the time to consider for calculations will be one period for the excitation signal, as follows:

$$\Delta_k = \left[\left(\sum_{i=k-p}^{k} {s_i}^2 \right) (p+1)^{-1} \right]^{0.5} \quad (4.3)$$

Where p is the period of the excitation signal used.

The simulation is initialized with an estimate of the plant parameters. The initial values are chosen on a grid with the current estimate values at the centre, and the corners are

chosen to be, e.g. for a first order system, all the combinations of

$\hat{a}_k \pm M_\Delta \Delta_k$, $\hat{b}_k \pm M_\Delta \Delta_k$. A section of code used to select the points on the grid follows:

```
for a = 1:(points*2+1)
 for b = 1:(points*2+1)
  sim_set(1,a_num) = DeltaK(k,i)*(2*((a-1)/(points*2))-1);
  sim_set(2,a_num) = DeltaK(k,i)*(2*((b-1)/(points*2))-1);
  a_num = a_num+1;
 end
end
```

(4.4)

A simulation is performed for each point in the simulation set, and a predicted bicriteria score is calculated. To calculate the overall score for each estimation track, the weighted sum of all the scores of the simulation runs (i.e. a total of (number of samples for each parameter)^2 for a first order system) is used. Each score is weighted by the likelihood that the corresponding simulation used the true parameters. For this calculation Δ_k is assumed to be proportional to the standard deviation. The range of parameters used in the calculation is equal to $M_\Delta \Delta_k$. Where M_Δ is the number of standard deviations to consider and is referred to as the *uncertainty bounds*. Thus, the overall score is calculated by multiplying the vector of simulation scores that use the parameter estimates in equation (4.4) by the vector of weights given in equation (4.6), as follows for the j^{th} estimation track:

$$score(j) = sim\_score(sim\_set(j))^T * weight \quad (4.5)$$

The values for $\underline{\hat{A}}_k, \underline{\hat{B}}_k$ and s_k are resetted to the values corresponding to the estimation track with the lowest score. A section of code follows:

```
for a = 0:(points*2)
 for b = 0:(points*2)
  it = it + 1;
  weight(it) =
   … normpdf(2*(a/(points*2) - 0.5)*mult,0,1)*
```

(4.6)

```
   … normpdf(2*(b/(points*2) - 0.5)*mult,0,1);
 end
end
```

4.2.1.2 Results (simple example to illustrate method)

A simple example in this section is used to illustrate the bicriteria track selection method. The system used is as follows:

$$y_{k+1} = 1.1y_k + 0.5u_k + N(0,0.005) \quad (4.7)$$

This represents a first order and open loop unstable system, with Gaussian noise centred at zero and a standard deviation of 0.005 added. At time interval 50, b is increased from 0.5 to 0.6 to test the response of the adaptation method to a time varying parameter step change.

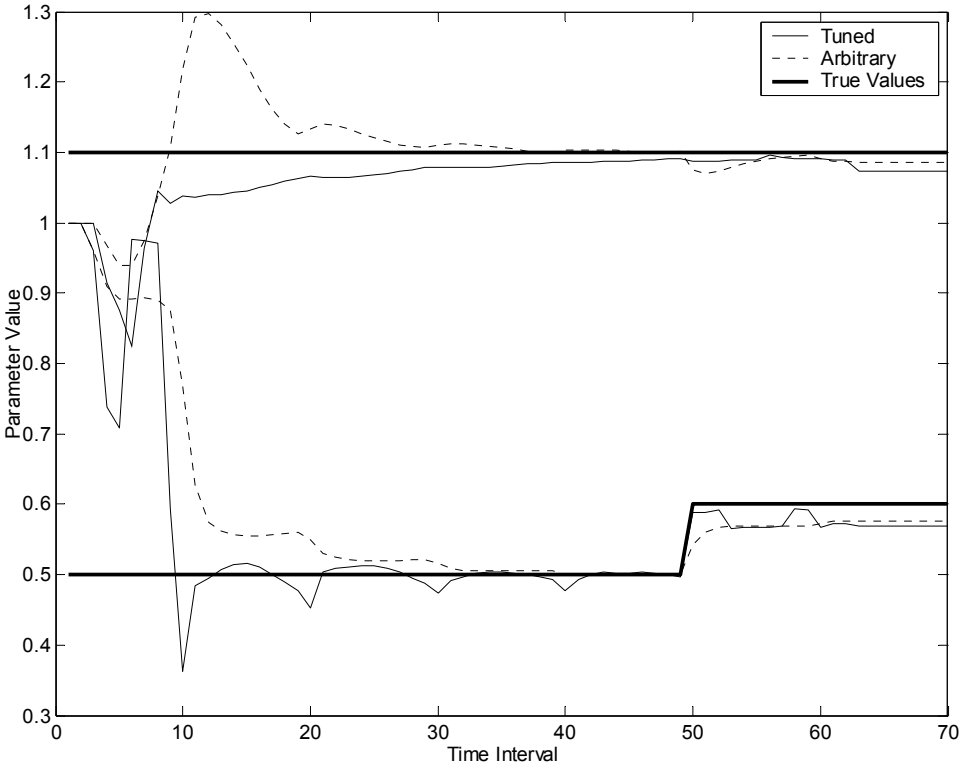

Figure 4.2: Parameter estimates for the default and tuned adaptive controller

In Figure 4.2 above the parameter estimates are compared for the controller tuned with the bicriteria method and a controller with arbitrarily selected parameters K_a and K_b equal to 1. The combined bicriteria error score is 0.254 for the system tuned with the bicriteria method and 0.290 for the default system. The computation time is 10.9 seconds for the bicriteria method and .03 seconds for the default system. Thus an improvement in score is achieved but at the cost of significant increase in computation time. In Figure 4.3, the track selection is indicated. The system tuned with the bicriteria method tends to select an aggressive value for K_b, and a more cautious value for K_a. In this example the initial error in the estimate of b is much larger than for a, so this tuning selection result makes

sense. The oscillations for the estimates of b in the tuned system in Figure 4.2 are a consequence of the use of an aggressive adaptation gain.

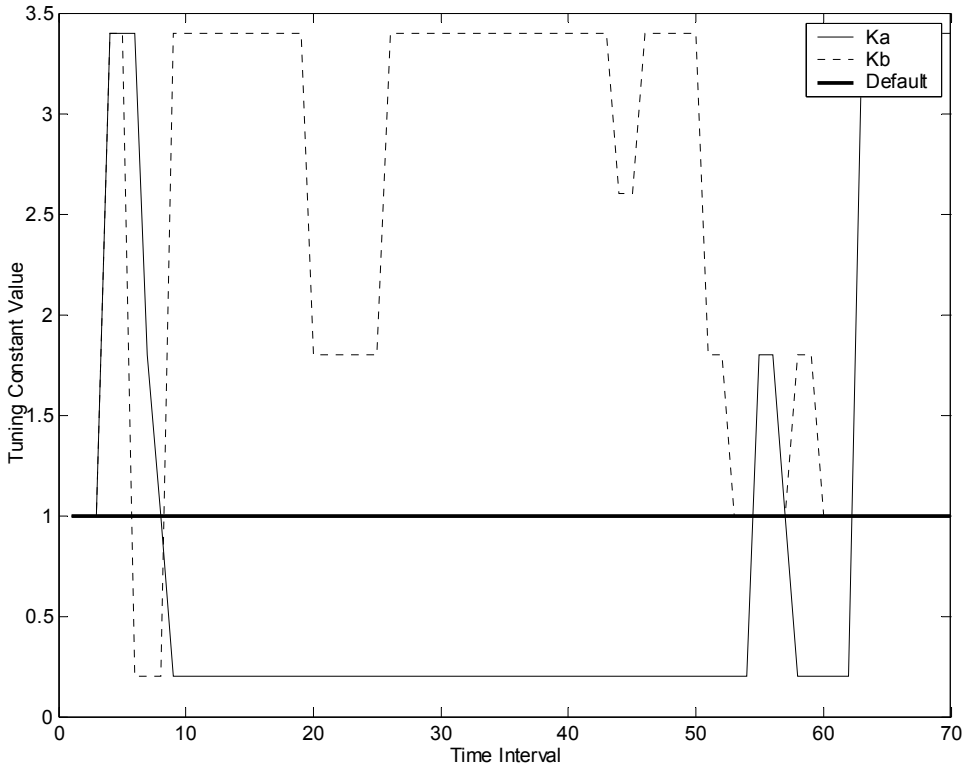

Figure 4.3: Estimation track used at each time interval, given by tuning constants

4.2.2 Linear Matrix Inequalities

Using the bicriteria cost function, an optimisation cost is constructed to identify the estimation track which is expected to offer the best performance at that time interval. The downside of that method is that when the parameter estimates are not close to their true values, or the disturbances are significant, the predictions will be inaccurate and that selection criteria can result in selecting an estimation track with poor performance. For

the LMI method the goal is to test the performance for all possible parameter combinations in the range of parameter uncertainty considered for calculation. This is in contrast with the bicriteria method, where only a finite number of combinations of parameters are considered. It should be noted that the performance score for the LMI method is not weighted by how likely that combination of parameters to occur is, whereas for the bicriteria method the expected performance for the current parameter estimates is weighted the highest.

4.2.2.1 Method

The LMI (Linear Matrix Inequality) formulation used in this method is derived from a test previously used for robust control design, where the norm of the error in the output can be bounded by the damping ratio, γ, times the norm of the input vector $\mathbf{v}$ which in this work corresponds to set-point changes to the process, as follows:

$$\|\mathbf{e}\|_2 < \gamma \|\mathbf{v}\|_2 \tag{4.8}$$

The LMI based test, to be used in this method, requires a system in the form of a linear nominal model with a model uncertainty description given as follows:

$$\boldsymbol{\eta}_{k+1} = \left[\mathbf{A}_0 + \mathbf{A}_1\delta_1 + \cdots + \mathbf{A}_n\delta_n\right]\boldsymbol{\eta}_k + \mathbf{B}\nu_k \tag{4.9}$$

where,
$\{\delta_1 \cdots \delta_n\}$ are the set of uncertainties to consider. In this work $\{\delta_1 \cdots \delta_n\}$ represent the elements of $\{\widetilde{\mathbf{A}}_k, \widetilde{\mathbf{B}}_k, \Delta_D\}$, Δ_D is the amplitude of d_k, a disturbance

Equation (4.9) can be represented as a time varying linear system, as follows:

$$\boldsymbol{\eta}_{k+1} = \mathbf{A}_k \boldsymbol{\eta}_k + \mathbf{B}\nu_k, \tag{4.10}$$

where

$$\mathbf{A}_k = \mathbf{A}_0 + \mathbf{A}_1\delta_1 + \cdots + \mathbf{A}_n\delta_n$$

The adaptive controller used in this work is non-linear based on the parameter estimation algorithm (equations (4.18) and (4.19)), where states are multiplied together, and the control action algorithm (equation (4.16)), where there is a division by $\hat{b}_k$. Also, the closed loop model is nonlinear with respect to the disturbance. As shown in the literature (Liu, 1968), a nonlinear system can be bounded by a set of time-varying linear systems. Thus, Liu showed that if the family of linear time invariant systems satisfy certain stability and performance tests, the original nonlinear system is also guaranteed to satisfy these properties. The general nonlinear system representing the model with the adaptive control algorithm is given as follows:

$$\begin{aligned}
y_{i,k+1} &= e_i\left(y_k, d_{i,k}, \widetilde{\mathbf{A}}_{i,k}, \hat{\mathbf{A}}_k, \widetilde{\mathbf{B}}_{i,k}, \hat{\mathbf{B}}_k, s_k, k\right) \\
s_{i,k+1} &= f_i\left(y_k, y_{sp,k}, d_{i,k}, \widetilde{\mathbf{A}}_{i,k}, \hat{\mathbf{A}}_k, \widetilde{\mathbf{B}}_{i,k}, \hat{\mathbf{B}}_k, s_k, \mathbf{K}_A, \mathbf{K}_B, K_D, k\right) \\
\hat{\mathbf{A}}_{i,k+1} &= g_i\left(y_k, y_{sp,k}, d_{i,k}, \widetilde{\mathbf{A}}_{i,k}, \hat{\mathbf{A}}_k, \widetilde{\mathbf{B}}_{i,k}, \hat{\mathbf{B}}_k, s_k, \mathbf{K}_A, \mathbf{K}_B, K_D, k\right) \\
\hat{\mathbf{B}}_{i,k+1} &= h_i\left(y_k, y_{sp,k}, d_{i,k}, \widetilde{\mathbf{A}}_{i,k}, \hat{\mathbf{A}}_k, \widetilde{\mathbf{B}}_{i,k}, \hat{\mathbf{B}}_k, s_k, \mathbf{K}_A, \mathbf{K}_B, K_D, k\right) \\
\widetilde{\mathbf{A}}_k &\in \left[-\delta\mathrm{a}_\mathrm{k}, \delta\mathrm{a}_\mathrm{k}\right],\ \widetilde{\mathbf{B}}_k \in \left[-\delta\mathrm{b}_\mathrm{k}, \delta\mathrm{b}_\mathrm{k}\right],\ d_k \in \left[-\Delta_\mathrm{D}, \Delta_\mathrm{D}\right]
\end{aligned} \tag{4.11}$$

In this work

$$\nu = y_{ref,k} \tag{4.12}$$

The actual values of $\widetilde{\mathbf{A}}_k$, $\widetilde{\mathbf{B}}_k$ and d_k are not known, but upper and lower bounds are known. As an example, the specific equations used to find f_i and g_i for a first order system are shown, as follows:

$$y_{k+1} = a \cdot y_k + b \cdot u_k + d_k \tag{4.13}$$

The plant model including the disturbance is given in equation (4.13). The parameters a and b are unknown, but the uncertainty in the current estimate for those values can be estimated. Thus equation (4.13) can be represented as follows:

$$y_{k+1} = (\hat{a}_k - \tilde{a}_k) y_k + (\hat{b}_k - \tilde{b}_k) u_k + d_k \tag{4.14}$$

The reference model, with time constant τ_d, used to calculate the set-point, $y_{sp,k+1}$, from the reference signal, $y_{ref,k}$, is as follows:

$$y_{sp,k+1} = \tau_d^{-1} \cdot y_{sp,k} + (1 - \tau_d^{-1}) y_{ref,k} \tag{4.15}$$

The input signal uses the current observed state, the current set-point, the current filtered error, s_k, and the current values of the parameter estimates $\hat{a}_k$ and $\hat{b}_k$ as follows:

$$u_k = (-\hat{a}_k y_k + y_{sp,k} + (1 - K_D) s_k) \hat{b}_k^{-1} \tag{4.16}$$

The next value of the filtered error is calculated by the following set of equations:

$$A_f = \hat{a}_k + K_A y_k s_k$$
$$A_S = K_A y_k$$
$$B_f = \hat{b}_k + K_B u_k s_k$$
$$B_S = K_B u_k$$
$$S_f = \frac{2y_{k+1} - \hat{a}_k y_k - \hat{b}_k u_k + (1 - K_D) s_k}{1 + K_D}$$
$$S_A = \frac{-y_k}{1 + K_D}$$
$$S_B = \frac{-u_k}{1 + K_D}$$

Which all combine to form:

$$s_{k+1} = \frac{S_f + S_A A_f + S_B B_f}{1 - S_A A_S - S_B B_S} \tag{4.17}$$

The update equations for $\hat{a}_{k+1}$ and $\hat{b}_{k+1}$ are:

$$\hat{a}_{k+1} = A_f + A_S \cdot s_{k+1} \tag{4.18}$$

$$\hat{b}_{k+1} = B_f + B_S \cdot s_{k+1} \tag{4.19}$$

Equations (4.13)-(4.19) are combined using the Matlab symbolic manipulation package and put in terms of observable and measurable variables, such as $y_k, s_k, \hat{a}_k, \hat{b}_k, y_{sp,k}$ together with the uncertain but bounded variables $d_k, \tilde{a}_k$ and $\tilde{b}_k$ to give the following equations for e_i, f_i, g_i and h_i :

$$y_{i,k+1} = \mathrm{a}y_k + \left(\mathrm{y_{sp,k}} - \hat{\mathrm{a}}_\mathrm{k}\,\mathrm{y_k} + (1 - K_D)s_{i,k}\right)b \big/ \hat{b}_{i,k} + \mathrm{d_k} \tag{4.20}$$

$$s_{i,k+1} = \frac{\left[2ay_k + 2\Theta\frac{b}{\hat{b}_{i,k}} + 2d_k - \mathrm{y_{sp,k}} - y_k\left(\hat{\mathrm{a}}_\mathrm{i,k} + K_A y_k s_{i,k}\right) - \frac{\Theta\left(\hat{b}_{i,k} + \frac{K_B \Theta s_{i,k}}{\hat{b}_{i,k}}\right)}{\hat{b}_{i,k}} \right]}{\left[1 + K_D + \mathrm{y_k}^2 K_A + \frac{K_B \Theta^2}{\hat{b}_{i,k}^{\,2}} \right]} \tag{4.21}$$

$$\hat{a}_{i,k+1} = \hat{a}_{i,k} + K_A y_k \left(\frac{\left[2ay_k + 2\Theta \frac{b}{\hat{b}_{i,k}} + 2d_k - y_{sp,k} - y_k \left(\hat{a}_{i,k} + K_A y_k s_{i,k} \right) - \frac{\Theta \left(\hat{b}_{i,k} + \frac{K_B \Theta s_{i,k}}{\hat{b}_{i,k}} \right)}{\hat{b}_{i,k}} \right]}{\left[1 + K_D + y_k^{\,2} K_A + \frac{K_B \Theta^2}{\hat{b}_{i,k}^{\,2}} \right]} + s_{i,k} \right) \tag{4.22}$$

$$\hat{b}_{i,k+1} = \hat{b}_{i,k} + K_B \Theta \left(\frac{\left[2ay_k + 2\Theta \frac{b}{\hat{b}_{i,k}} + 2d_k - y_{sp,k} - y_k \left(\hat{a}_{i,k} + K_A y_k s_{i,k} \right) - \frac{\Theta \left(\hat{b}_{i,k} + \frac{K_B \Theta s_{i,k}}{\hat{b}_{i,k}} \right)}{\hat{b}_{i,k}} \right]}{\left[1 + K_D + y_k^{\,2} K_A + \frac{K_B \Theta^2}{\hat{b}_{i,k}^{\,2}} \right]} + \frac{s_{i,k}}{\hat{b}_{i,k}} \right) \tag{4.23}$$

With

$$\Theta := y_{sp,k} - \hat{a}_{i,k} y_k + (1 - K_D) s_{i,k} \tag{4.24}$$

The states that are required for the LMI formulation are: $y_k, s_k, \hat{a}_k, \hat{b}_k, y_{sp,k}$ which can be represented altogether as

$$\boldsymbol{\eta}_k = \left\lfloor y_k, \quad s_k, \quad \hat{a}_k, \quad \hat{b}_k, \quad y_{sp,k} \right\rfloor \tag{4.25}$$

As shown by Kothare et. al. (1994) the nonlinear system defined by equation (4.11) and expressions for e_i, f_i, g_i and h_i as given above can be bounded by taking the Jacobian of the system with every possible combination of upper and lower bound of the uncertain variables, d_k, $\widetilde{\mathbf{A}}_k$ and $\widetilde{\mathbf{B}}_k$. For $\widetilde{\mathbf{A}}_k$ and $\widetilde{\mathbf{B}}_k$ the magnitude of the uncertainty is Δ_k, that is

calculated with equation (4.3) whereas the magnitude of the uncertainty for $dist_k$ is Δ_d, which is based on a priori knowledge about the disturbance's magnitude. The family of models defined by equation (4.10) can be viewed as a hyper-volume with vertices defined by the combinations of the extreme values of the uncertainties Δ_k and Δ_d. As an example, for a first order system, Δ_d and Δ_k are substituted for $\widetilde{a}_k$, $\widetilde{b}_k$ and d_k in equations (4.13) through (4.19) to calculate the i^{th} vertex, as follows:

$$\begin{array}{cccc} i & \underline{\widetilde{a}_k} & \underline{\widetilde{b}_k} & \underline{dist_k} \\ 1 & -\Delta_k & -\Delta_k & -\Delta_d \\ 2 & -\Delta_k & -\Delta_k & +\Delta_d \\ 3 & -\Delta_k & +\Delta_k & -\Delta_d \\ 4 & -\Delta_k & +\Delta_k & +\Delta_d \\ 5 & +\Delta_k & -\Delta_k & \_\Delta_d \\ 6 & +\Delta_k & -\Delta_k & +\Delta_d \\ 7 & +\Delta_k & +\Delta_k & \_\Delta_d \\ 8 & +\Delta_k & +\Delta_k & +\Delta_d \end{array} \tag{4.26}$$

The Jacobian of the function f_i in equation (4.11) is taken with respect to the vector of states, $\boldsymbol{\eta}_k$ and the input, ν_k, for each combination of uncertainties shown in (4.26), as follows:

$$\left[\frac{\partial \boldsymbol{\eta}_{k+1}(\delta_i)}{\partial \boldsymbol{\eta}_k}\right] = \left[\mathbf{A}_{k,i}(\delta_i)\right] \tag{4.27}$$

$$[\mathbf{A}_k] = \sum_{i=1}^{L} \alpha_{i,k}[\mathbf{A}_i(\delta_i)],$$

$$\sum_{i=1}^{L} \alpha_{i,k} = 1,\ \alpha > 0$$

Each set of matrices $\left[\mathbf{A}_{k,i}(\delta_i)\right]$ is the i[th] vertex of a polytope of matrices, where the nonlinear system at time k, can be represented as a linear combination of the vertices of

the polytope of matrices, with the weights $\alpha_{i,k}$ .The overall closed loop system equations are then given as follows:

$$\begin{bmatrix} \boldsymbol{\eta}_{k+1} \\ e_k \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{i,k}(\boldsymbol{\delta}_i) & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{\eta}_k \\ \nu_k \end{bmatrix} \tag{4.28}$$

$\boldsymbol{\eta}_0$ is known

$$\boldsymbol{\eta}_k = \begin{bmatrix} y_k & s_k & \hat{a}_k & \hat{b}_k & y_{sp,k} \end{bmatrix}$$

$$\nu_k = \begin{bmatrix} y_{ref,k} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1-\tau_d^{-1} \end{bmatrix}^T$$

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

$$\mathbf{D} = \begin{bmatrix} 0 \end{bmatrix}$$

The values of $\alpha_{i,k}$ in equation (4.27) are related to the true values of $\widetilde{a}_k$, $\widetilde{b}_k$ and d_k by a simple linear transformation. The linear combination of the polytope resulting from equation (4.27) is equivalent to the form required for a LMI in equation (4.10). As stated above, if the set of linear time varying systems satisfy certain robustness properties, the original non-linear system will also satisfy the same robustness properties. The LMI formulation used to calculate a score for each estimation track is as follows:

$$\gamma_{optimal} = \underset{\{track\}}{Min}\left[\underset{\Delta}{Min}(\gamma)\right] \tag{4.29}$$

s.t.

$$\begin{bmatrix} \mathbf{A}(\boldsymbol{\delta}_i)^T \mathbf{P}\mathbf{A}(\boldsymbol{\delta}_i) - \mathbf{P} & \mathbf{A}(\boldsymbol{\delta}_i)^T \mathbf{P}\mathbf{B} & \mathbf{C}^T \\ \mathbf{B}^T \mathbf{P}\mathbf{A}(\boldsymbol{\delta}_i) & \mathbf{B}^T \mathbf{P}\mathbf{B} - \gamma^2 \mathbf{I} & \mathbf{D}^T \\ \mathbf{C} & \mathbf{D} & -\mathbf{I} \end{bmatrix} < 0$$

where,

track is defined by the set $\{\mathbf{K}_A, \mathbf{K}_B, K_D\}_{track}$

and,

$$\gamma > \frac{\|e\|_2}{\|v\|_2},\ e = f\left(v, \hat{a}_k, \hat{b}_k, y_k, u_{k-1}, \{\mathbf{K}_A, \mathbf{K}_B, K_D\}_{track}\right)$$

Because each element in the linear matrix inequality is either linear or quadratic with respect to the parameters δ's , the space considered for the optimization in equation (4.28) is convex with respect to these uncertainty parameters. Hence, it is sufficient to satisfy the test in equation (4.28) only at the vertices of the polytope of matrices.
The value of γ calculated for each estimation track is the minimum damping ratio which satisfies the general eigenvalue problem (GEVP) over the entire range of parameter uncertainties and disturbances considered. This γ is the worst case performance over the range of models considered for each track. The estimation track selected is the one which has the lowest value of γ over all the tracks considered.

A first order system has one equation for the output, x_k, one for the filtered error, s_k, one for each of the parameters estimated, $\hat{a}_k$ and $\hat{b}_k$, and one for the reference model, $y_{sp,k}$, for a total of five dynamic states in the system. In a general high order system there will be 2n + 2m + p states involved, where n is the order of the system with respect to the past outputs, m is the order with respect to the inputs and p is the order of the reference model. Each estimated parameter in the vectors $\underline{\hat{A}}_k$ and $\underline{\hat{B}}_k$ has an error associated with it, along with the disturbance, d_k, which adds one dimension to the LMI formulation. There are a total of n + m + 1 variables with uncertainty, which require a total of $2^{(n+m+1)}$ vertices to represent the non-linear system of equation (4.11) by a set of linear models as required for the LMI formulation.

The Jacobian at each vertex is calculated using the equation (4.27), as follows:

$$\mathbf{A}_{k,i}(\delta_i)=\begin{bmatrix} \partial x_{k+1}/\partial x_k & \partial x_{k+1}/\partial \hat{a}_k & \partial x_{k+1}/\partial \hat{b}_k & \partial x_{k+1}/\partial s_k & \partial x_{k+1}/\partial ref_k \\ \partial \hat{a}_{k+1}/\partial x_k & \partial \hat{a}_{k+1}/\partial \hat{a}_k & \partial \hat{a}_{k+1}/\partial \hat{b}_k & \partial \hat{a}_{k+1}/\partial s_k & \partial \hat{a}_{k+1}/\partial ref_k \\ \partial \hat{b}_{k+1}/\partial x_k & \partial \hat{b}_{k+1}/\partial \hat{a}_k & \partial \hat{b}_{k+1}/\partial \hat{b}_k & \partial \hat{b}_{k+1}/\partial s_k & \partial \hat{b}_{k+1}/\partial ref_k \\ \partial s_{k+1}/\partial x_k & \partial s_{k+1}/\partial \hat{a}_k & \partial s_{k+1}/\partial \hat{b}_k & \partial s_{k+1}/\partial s_k & \partial s_{k+1}/\partial ref_k \\ \partial ref_{k+1}/\partial x_k & \partial ref_{k+1}/\partial \hat{a}_k & \partial ref_{k+1}/\partial \hat{b}_k & \partial ref_{k+1}/\partial s_k & \partial ref_{k+1}/\partial ref_k \end{bmatrix} \quad (4.30)$$

This results in a set of equations for $\left[\mathbf{A}_{k,i}(\delta_i)\right]$, in terms of variables that can be observed or calculated. These equations are implemented in a Matlab function that takes the parameter estimates, past input and output data, s_k, the reference model, and the uncertainties for one vertex as inputs, and returns the matrices $\left[\mathbf{A}_{k,i}(\delta_i)\right]$ corresponding to the vertex considered. An example of one vertex is as follows:

$$\left[\mathbf{A}_{k,i}(\delta_i)\right]=vertex\left(\hat{a}_k,-\Delta_k,\hat{b}_k,-\Delta_k,+\Delta_d,y_{sp},y_k,y_{k-1},u_{k-1},s_k,s_{k-1}\right) \quad (4.31)$$

Each vertex contributes to the general eigenvalue problem, an inequality of the following form:

$$\begin{bmatrix} \mathbf{A}(\boldsymbol{\delta}_i)^T\mathbf{P}\mathbf{A}(\boldsymbol{\delta}_i)-\mathbf{P} & \mathbf{A}(\boldsymbol{\delta}_i)^T\mathbf{P}\mathbf{B} & \mathbf{C}^T \\ \mathbf{B}^T\mathbf{P}\mathbf{A}(\boldsymbol{\delta}_i) & \mathbf{B}^T\mathbf{P}\mathbf{B}-\gamma^2\mathbf{I} & \mathbf{D}^T \\ \mathbf{C} & \mathbf{D} & -\mathbf{I} \end{bmatrix}<0 \quad (4.32)$$

With the vertices $\left[\mathbf{A}_{k,i}(\delta_i)\right]$ calculated with (4.31). The resulting general eigenvalue problem in equation (4.29) is evaluated using the Matlab function '`gevp`', which returns the best value of γ. The value of γ returned for each track indicates the expected worst-case possible performance within the range of parameter uncertainty used for the calculation. The track with the lowest value of γ is expected to have the best robust

performance out of all the tracks considered, and thus the variables $\hat{a}_k, \hat{b}_k$ and s_k are reset to the values corresponding to this track.

With the LMI method, an individual formulation may have a result that is returned as 'infeasible', which implies that no upper bound can be found for γ with that particular estimation track, for that time interval. Estimation tracks for which this result is obtained are excluded from consideration when they have an 'infeasible' score. If all results are 'infeasible' then a track with associated default tuning parameters a priori selected should be used. This tends to happen for the first few time intervals until sufficient data is available to get a reasonable estimate for the parameter estimates with corresponding small uncertainty values.

4.2.2.2 Results

The LMI method has the potential to identify performance problems along the entire parameter space, whereas the bicriteria method studies the performance for a small number of points in the parameter space. Therefore it is expected that for non-linear systems, if the model includes a large number of parameters, there can be large performance variations within the parameter space considered that will not be accounted for by the bicriteria method. This situation will be clearly illustrated in chapter 5. In the remainder of this chapter, a simple example that demonstrates the method is given. The example system used for the LMI method is the same as the one used to demonstrate the bicriteria method, as given in equation (4.7) with default tuning parameters set to $K_A = 4$, $K_B = 0.25$ and $K_D = 1$.

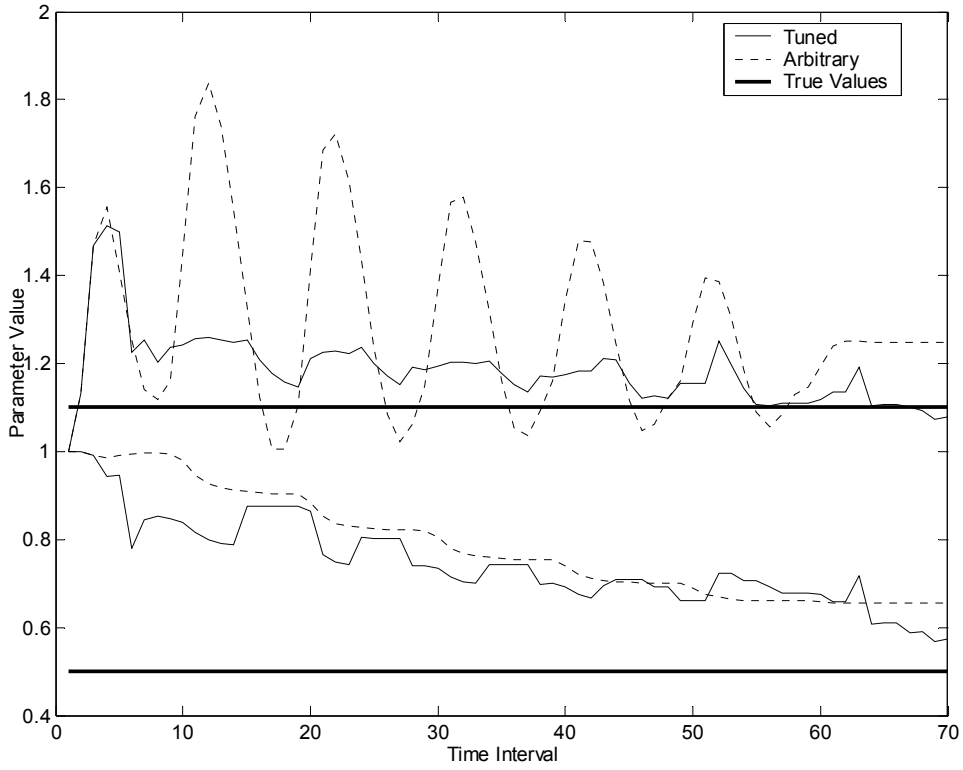

Figure 4.4: Parameter estimates for the default and tuned adaptive controller

In Figure 4.4 the parameter estimates are compared for the controller tuned with the bicriteria method and a controller with arbitrarily selected parameters K_a and K_b equal to 1. The actual value for γ is 0.0231 for the system tuned with the LMI method and 0.0385 for the system with arbitrarily selected parameters. The computation time is 795 seconds for the LMI method and 0.016 seconds for the default system. Thus, although the LMI method leads to improvement in control performance, it requires a large amount of computation time. Hence, it would only be considered for cases where there is a clear performance advantage over other tuning techniques, such as the bicriteria method. The LMI method is design to select the most robust estimation track. It is possible to find sets

of tuning parameters that perform better than the LMI selected tracks for the specific system and noise set used, but there is no way to determine what these estimation tracks are a priori.

In Figure 4.5 the parameters selected at each interval corresponding to different tracks are indicated. The LMI method, when used for a simple linear system, tends to switch the estimation track more often than when using the bicriteria method.

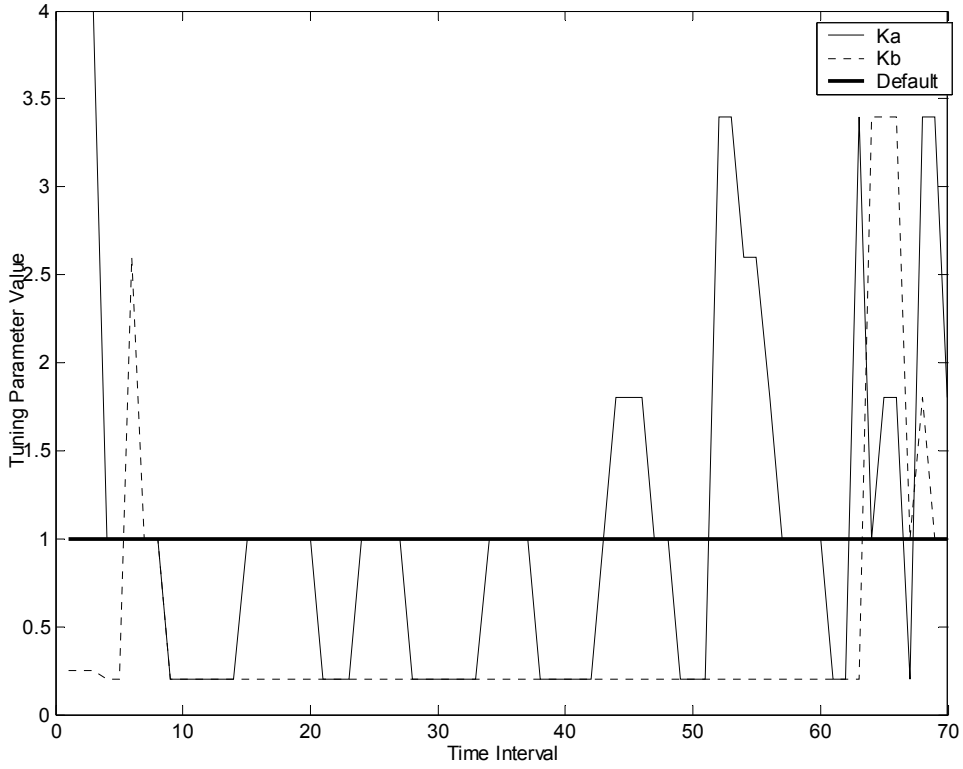

Figure 4.5: track used at each time interval, given by tuning constants

With the LMI selection method there is a significant amount of overlap in the parameter space considered for each estimation track, as illustrated in Figure 4.6. For example, with

a first order linear system the performance score changes slowly with changes in the estimated parameters. Hence, the worst case performance calculated (i.e. the score calculated for γ) shows minor variations between the estimation tracks, and small changes in the noise and estimations can lead to frequent changing of the track. Also, the LMI method tends to select a more conservative set of tuning constants. This can lead to slower adaptation, but there is less of a risk of inducing a large error through high frequency oscillations due to overly aggressive tuning.

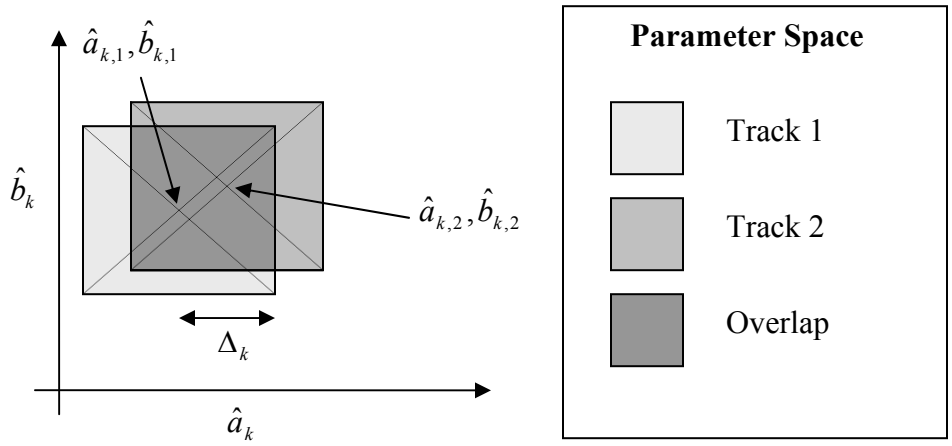

Figure 4.6: Overlap in parameter space for two tracks in the LMI method

5 Results

In this section results obtained for systems of first and higher order than one, the effect of tuning based on the performance tests presented in Chapter 4, and the behaviour of the adaptive controller in the presence of deterministic disturbances will be presented. First the BC and LMI methods will be examined in detail. The two tuning methods will be compared with a system tuned with arbitrarily selected parameters for the following cases: a first order system with white noise disturbances, a first order system with a deterministic disturbance, and for a higher order system. The BC and LMI tuning methods will also be compared with a traditional adaptive control method, the Recursive Least Squares (RLS) algorithm. The following first order system will be used for the comparisons:

$$
\begin{aligned}
&x_{k+1} = a_k x_k + b_k u_k \\
&y_k = x_k + \eta_k \\
&\textit{where} \\
&k \in \{1,\cdots,200\} \\
&a_k = \begin{cases} 1.05 & k < 100 \\ 0.95 & 100 \le k \le 200 \end{cases} \\
&b_k = \begin{cases} 0.5 & k < 150 \\ 0.6 & 150 \le k \le 200 \end{cases} \\
&\eta_k = (1-\beta)d_k + \beta\eta_{k-1},\ \beta = 0.75, d \in N(0,0.005)
\end{aligned}
\qquad (5.1)
$$

It is assumed that during operation, this process undergoes a step change in each of the model parameters, *a* and *b*. The system given by equation (4.2) is open-loop unstable for the first 100 time intervals, and open-loop stable for the last 100 time intervals. The measurement noise is low-pass filtered, as noise near or above the Nyquist frequency of a discrete type system cannot be dampened effectively due to aliasing.

5.1 Bicriteria Tuning (BC) Method

For the BC tuning method the number of estimation tracks, the number of simulation tracks used for each estimation track, the simulation time horizon, and the ranges of uncertainty in the model parameters to be considered in the simulations have to be all set a priori. These factors, especially the first three, have major impact on the computation time required. The effect of all these factors has been investigated and the results are summarized in the following subsections. When each one of these factors is individually investigated, the other factors are set at the default values as follows: 28 estimation tracks, 49 simulation tracks, a simulation time horizon of 2, and an uncertainty bounds multiple, M_Δ, of 2. The uncertainty bounds multiple is a scalar that defines the amount of model parameter uncertainty for each parameter. For example, for a model parameter *a,* the range of uncertainty in this parameter is defined as follows

$$\left[\hat{a}_k - M_\Delta \Delta_k, \hat{a}_k + M_\Delta \Delta_k\right]$$

5.1.1 Estimation Tracks

In Figure 5.1, the effect of the number of estimation tracks used is examined. It should be recalled that the estimation tracks are distributed according to the code given in equation (4.1), which is a multidimensional numerical grid, with one dimension per tuning parameter considered, and a total of $(\text{k\_points})^{n+m+1}+1$ estimation tracks used, where n is the order of the input, m is the order of the output, one adaptation gain per

parameter to be estimated and the filter gain K_D, for a total of n+m+1 tuning parameters. The tuning parameters are selected to be evenly spaced in a fixed range $(0,4)$ thus, as the number of tracks is increased the spacing between the tuning parameter values associated with the estimation tracks becomes smaller.

It should be remembered that the excitation signal is given by the time-varying set-point; in this work a periodic signal with a period of 20 intervals is used. The diamonds in Figure 5.1 indicate the error in the system when the simulation score, as represented by the value of the BC cost function in equation 2.30, is used at each time step and the X's indicate the error in the system when the simulation score results are averaged over one excitation period. Both ways of taking into account the scores, i.e. by considering the instantaneous score or the average score, have similar results for up to 28 estimation tracks. Beyond this number of estimation tracks the differences between tuning parameter sets associated to the different tracks are becoming small, and differences in scores corresponding to the different tracks are very sensitive to measurement noise. Between step changes in the square wave excitation signal, i.e. during periods when the excitation signal is constant, the simulation scores are very strongly affected by measurement noise because there are no significant input changes to drive the adaptation process. Thus, the best track, i.e. the track with the lowest BC score, will change frequently, even if the true best performing track does not. The true best performing track is referred to the one that would be selected if one had perfect knowledge of the model parameters and disturbances and consequently will result in the lowest BC score. By averaging the simulation scores over one excitation period, the random effects of measurement noise can be reduced, at the cost of a slower response to a change in model

parameters. The computation time is proportional to the number of tracks used, which results in a rapid increase in computation time beyond a small number of estimation tracks.

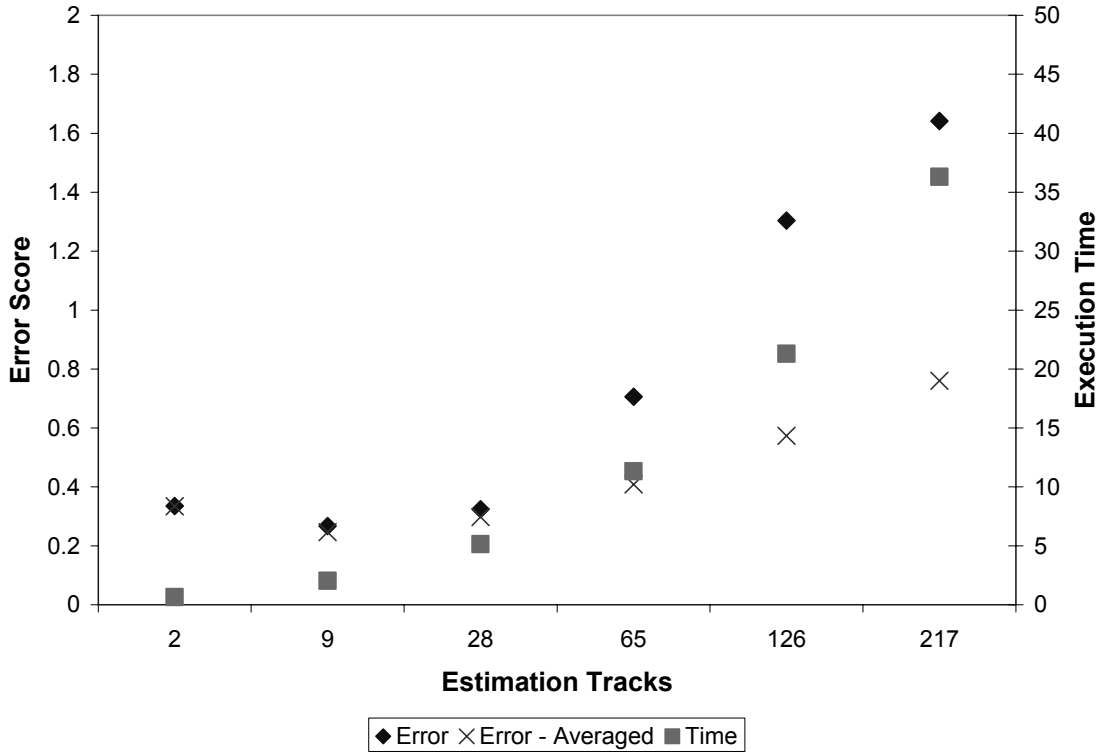

Figure 5.1: Effects of the number of estimation tracks on BC performance

5.1.2 Simulation Tracks

In Figure 5.2, the effects of varying the number of simulation tracks used to predict the bicriteria error for each of the estimation tracks are examined. It should be remembered that each track is associated to a specific set of tuning parameters. At any given interval of time, certain values of model parameter estimates are computed with some corresponding uncertainty in these parameters. Then the different simulations for that track used a specific set of tuning parameters but are conducted for different

combinations of model parameter values based on the calculated uncertainty in these model parameters. Thus, the simulations use parameters in the range $[\hat{a}_k - M_\Delta \Delta_k, \hat{a}_k + M_\Delta \Delta_k]$ and $\lfloor \hat{b}_k - M_\Delta \Delta_k, \hat{b}_k + M_\Delta \Delta_k \rfloor$ where M_Δ is constant.

The score corresponding to '1' simulation track is equivalent to the certainty equivalence (CE) principle found in the literature, where the current plant parameter estimates are assumed to be the correct ones.

Thus increasing the number of simulation tracks increase the density of the parameter combinations in this range, but the overall range is constant. Almost all the benefit of simulating different possible combinations of parameter uncertainty is realised by a single layer of parameter values (i.e. 9 in Figure 5.2) combinations around the CE design trajectory. Additional combinations do not improve the controller performance much in terms of the error as shown in Figure 5.2. The computation time required is proportional to the number of simulation trajectories used.

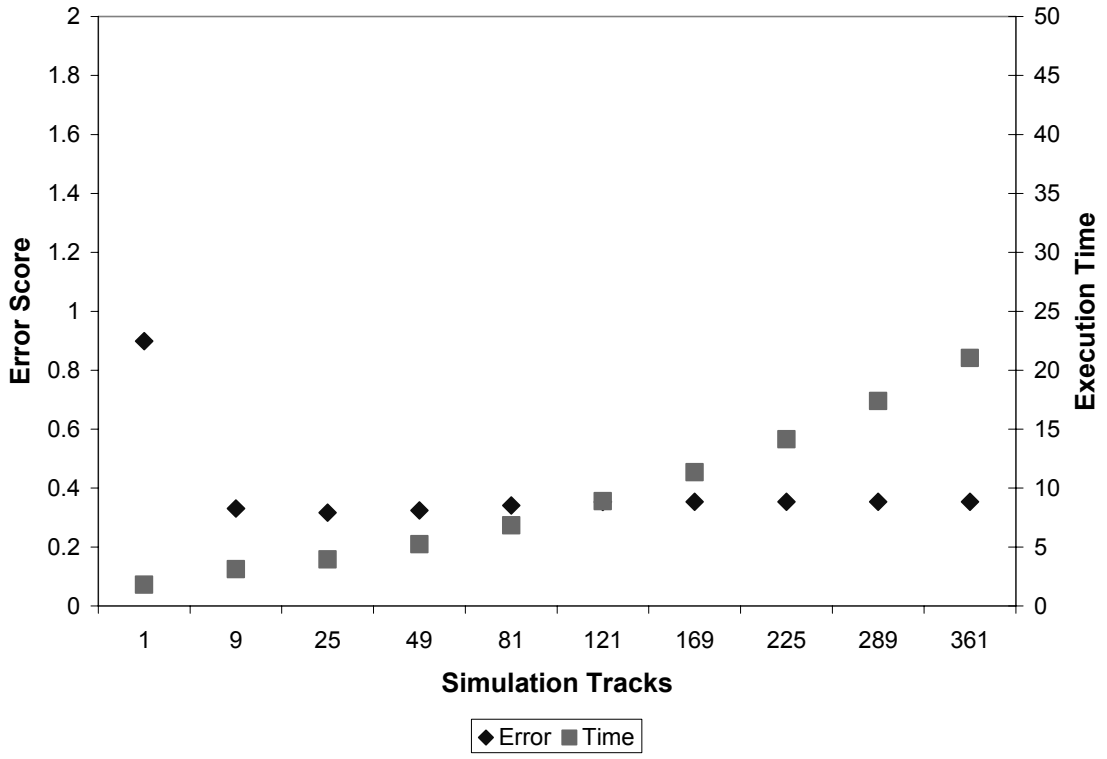

Figure 5.2: Effects of the number of simulation tracks on BC performance

5.1.3 Simulation Horizon

In Figure 5.3, the effect of varying the number of time steps used for the calculation of the bicriteria error for each one of the estimation tracks is examined. A simulation horizon of '1' uses only current time step values, and is equivalent to basing the selection on the prediction error only, and not the feedback error. The reason for this is that the feedback error is equal for all tracks at the current time step, so the only difference in the calculation for the different tracks is in the prediction error, thus feedback error does not contribute to the differences in the BC scores for track selection. Beyond 3 time steps the uncertainty compounds to the point that the estimation track selection is not reliable, thus performance is inconsistent for a simulation horizon of more than 3 time steps.

The computation time changes corresponding to an increase in the number of estimation tracks, the number of simulations tracks per estimation track, or the simulation horizon compound with each other in an approximately multiplicative fashion (i.e. if each one of the 3 factors is doubled, the overall computation time required increases approximately by 8 times).

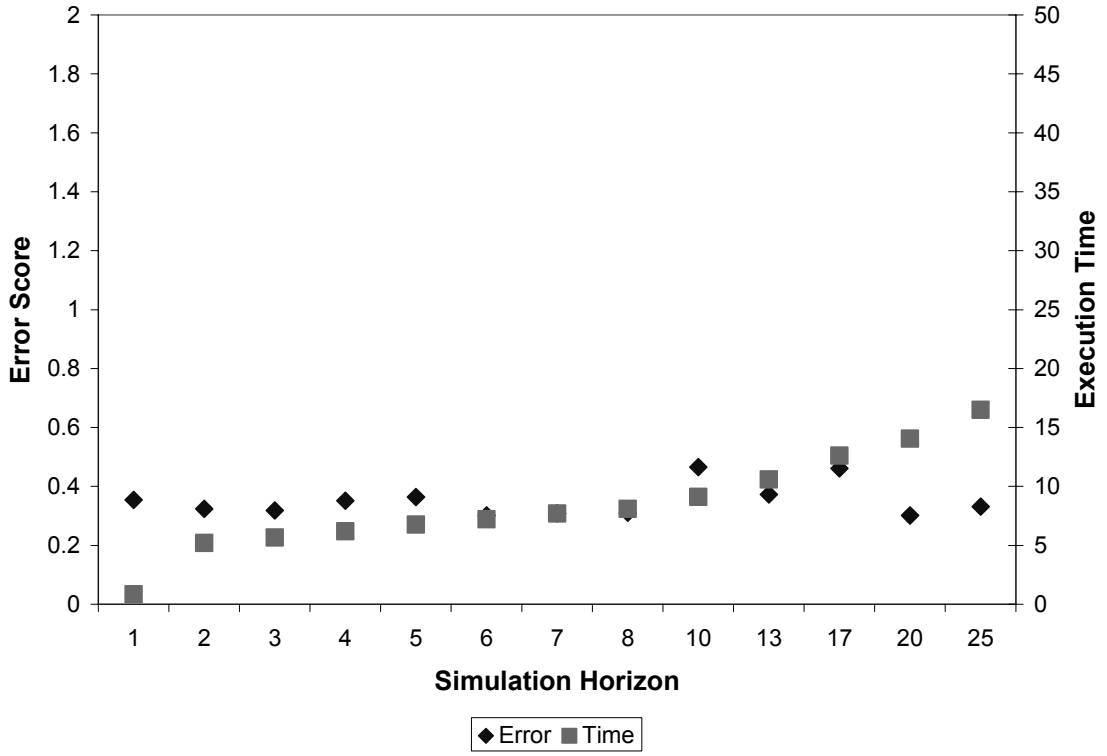

Figure 5.3: Effects of the simulation horizon on BC performance

5.1.4 Effect of the uncertainty bounds (M_Δ) on the simulations carried out around each track

In Figure 5.4, the effects the uncertainty bounds, M_Δ, are examined. The entry '0' indicates that the only simulation track uses the CE assumptions, as per section 5.1.2.

The entry '1' uses all possible combinations of $\{\hat{a}_k + \frac{1}{3}\Delta_k, \hat{a}_k, \hat{a}_k - \frac{1}{3}\Delta_k\}$ and $\{\hat{b}_k + \frac{1}{3}\Delta_k, \hat{b}_k, \hat{b}_k - \frac{1}{3}\Delta_k\}$ as the initial values of the plant models for the simulation. For the entry '2', all possible combinations of $\{\hat{a}_k + \frac{2}{3}\Delta_k, \hat{a}_k + \frac{1}{3}, \Delta_k, \hat{a}_k, \hat{a}_k - \frac{1}{3}\Delta_k, \hat{a}_k - \frac{2}{3}\Delta_k\}$ and $\{\hat{b}_k + \frac{2}{3}\Delta_k, \hat{b}_k + \frac{1}{3}\Delta_k, \hat{b}_k, \hat{b}_k - \frac{1}{3}\Delta_k, \hat{b}_k - \frac{2}{3}\Delta_k\}$ are used, and for the entry 'n', $\{\hat{a}_k + \frac{n}{3}\Delta_k, \hat{a}_k, \hat{a}_k - \frac{n}{3}\Delta_k\}$ and $\{\hat{b}_k + \frac{n}{3}\Delta_k, \hat{b}_k, \hat{b}_k - \frac{n}{3}\Delta_k\}$ are used. This method gives an indication of what the effect is of adding each additional set of simulation trajectories. The plot indicates that the first layer of simulation tracks has a large effect on the system performance. Additional layers do not have a significant impact on performance, but they do add significant computation time.

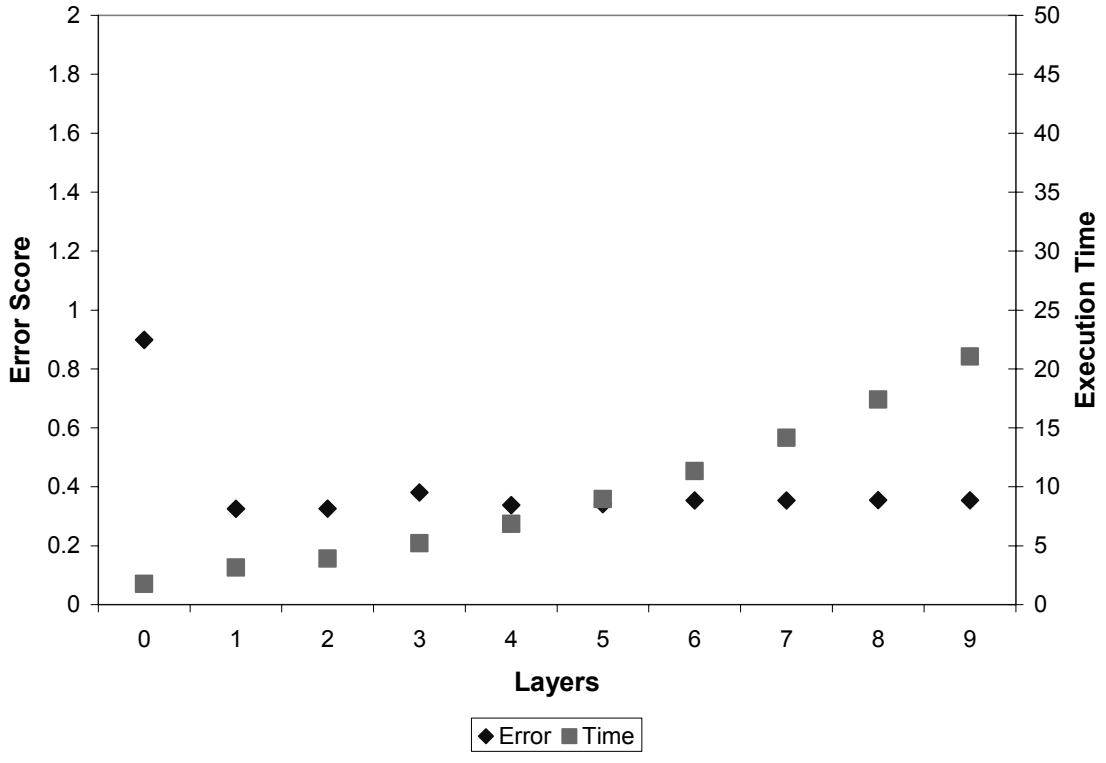

Figure 5.4: Effects of adding simulation layers on BC performance

5.1.5 Conclusions for BC Method

With the BC method, some design factors need to be selected a priori. The number of estimation tracks, the number of simulation tracks, and the simulation horizon are all constrained by the computational resources. The tests above indicate that these three factors are important up to certain small values, but beyond these values performance improvements are relatively small. Moreover, in the case of the number of estimation tracks, having too many tracks has been shown to be not useful because of sensitivity to noise. In summary, the BC method can result in significant benefits with a reasonable computational burden, in the range of 0.025 seconds of computation time per time step on a 2 GHz PC (see Table 5.1).

5.2 LMI Configuration

For the LMI tuning method, the number of estimation tracks and the uncertainty bounds, M_{Δ}, have to be selected a priori. It should be recalled that the uncertainty bounds are used when calculating the vertices of the polytope of the LMI system. The effect of these two parameters on the controller performance has been investigated and the results are presented in the following subsections. The system (4.2) is used, with default values of 28 estimation tracks, and a multiplier of 2. When one particular parameter is investigated the other one is kept at its default value. The system under study is the one used for examining the configuration of the BC method in section 5.1, above.

5.2.1 Estimation Tracks

In Figure 5.5 below, the effect of the number of estimation tracks is examined. It was found that the computation time required is proportional to the number of estimation tracks, and is significantly larger than for the BC method, at approximately 90 seconds per estimation track for a 200 time interval simulation (Table 5.1). Similarly to the BC method, having a large number of estimation tracks can be counterproductive, as shown in Figure 5.2 due to sensitivity to noise. Also, it was found that as the number of estimation tracks is increased, the possibility of obtaining infeasible results will be gradually reduced.

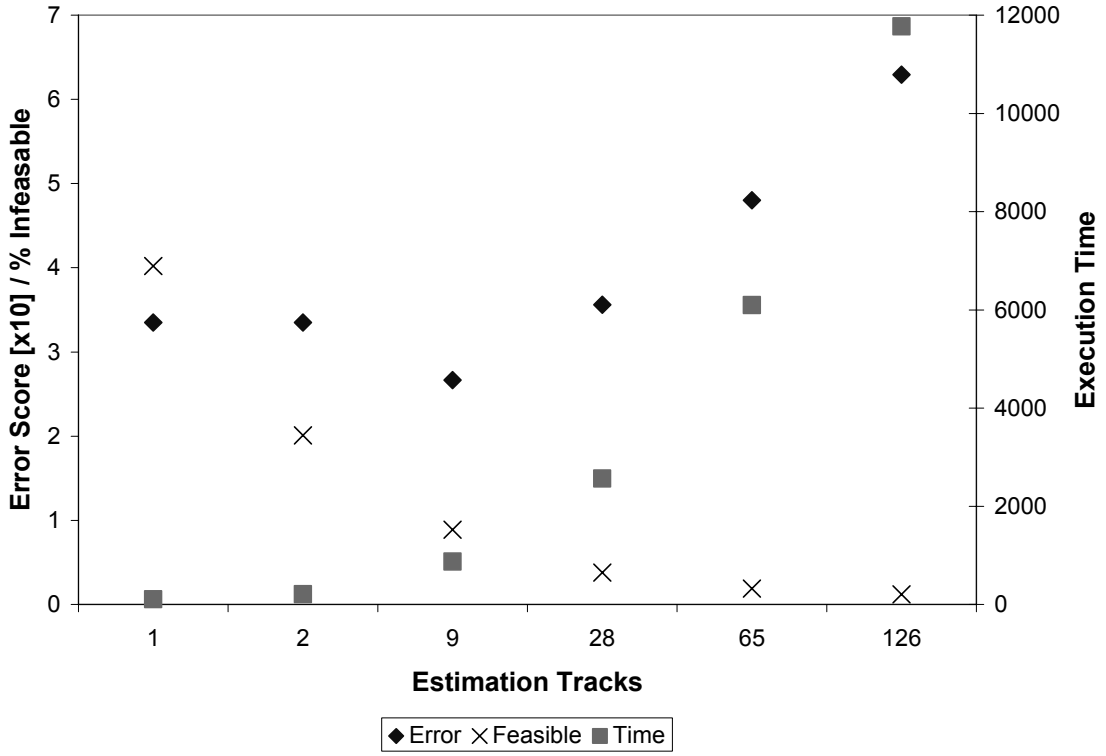

Figure 5.5: Effects of number of estimation tracks on LMI performance

5.2.2 Uncertainty bounds

In Figure 5.6 , the effects of the multiplier M_Δ, used in the calculation of the uncertainty bounds as given by equation (4.3), are studied. The performance effect is small until a threshold is reached, where the number of 'infeasible' results increases rapidly. If an estimation track has an 'infeasible' result, it cannot be evaluated and thus selected at that time step, even if it would be the true best performing track. The true best performing track is referred to the one that result in the smallest value of gamma if all model parameters and disturbances are perfectly known. There is only a small effect on the computation time. Unlike the BC method, the MatLab LMI solver is iterative, thus the same number of LMI solutions can take differing amounts of computation time.

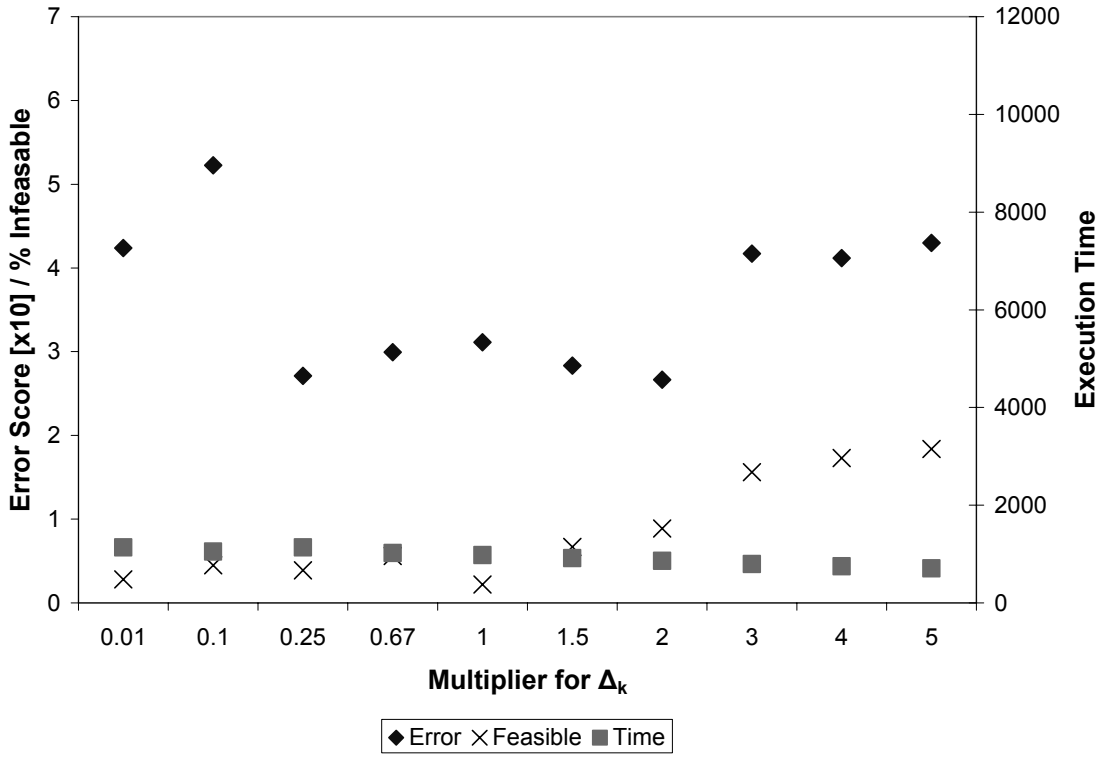

Figure 5.6: Effects of the multiplier for Δ_k on LMI performance

5.2.3 Conclusions for LMI Method

For the LMI method the design parameters that need to be selected a priori are the number of estimation tracks and the uncertainty multiplier $M_{\Delta},$. For the first order linear system under investigation it was found that the multiplier that determines the uncertainty bounds only has a significant detrimental effect beyond a value that is large enough to cause an increase in 'infeasible' optimization results. Similar to the BC method, an intermediate range of estimation tracks is ideal. For a first order system with a grid distribution as used in (4.2) 9 to 28 estimation tracks resulted in the best performance. For the best range of estimation tracks each step requires approximately 12 seconds per time interval (Table 5.1). For higher order systems the number of parameters to be estimated increases along with the adaptation gains to be tuned. The dimension of the matrix used in the LMI increases with the states of the closed loop system and the number of estimation tracks increase exponentially with the number of adaptation gains. Thus the LMI evaluation time is longer and more evaluations are required, resulting in a very large increase in computation time. In an extreme case, the LMI track selection can be performed once per cycle of the excitation signal, which is the external reference signal with a period of 20 time intervals in this work.

5.3 Tuning Method Comparison

In this section the performance and computation time of the LMI and BC tuning methods are compared. Several simulations conducted with arbitrarily chosen tuning parameter

are also included to demonstrate the importance of systematic selection of the adaptation gains. Without sufficient a priori knowledge the only way to evaluate the adaptation gains online is through tuning. The systems used for comparison are; i- first order with Gaussian measurement noise, ii- second order with Gaussian measurement noise, and iii- first order with an unmeasured deterministic disturbance. The same random seed for Gaussian noise is used for each simulation.

5.3.1 First Order

The system given in (4.2) is used, along with the default configurations given in sections 5.1 and 5.2 for the BC and LMI tuning methods respectively. Two simulations with arbitrarily selected parameters are included in the comparison, one that performs well, and one with poor performance. The control performance measured by either the bicriteria error score or by the true value for γ_{sp} as defined by the inequality $\|y_{sp} - y\| < \gamma_{sp}\|y_{sp}\|$ used by the LMI method and the execution time for each method are summarized in Table 5.1. For the LMI method, 99.89% of the calculations returned a 'feasible' result, or equivalently only 6 'infeasible' results were evident.

It should be emphasized that for the simulations based on arbitrarily selected tuning parameters, the performance can be better or worse by chance, as compared to the tuning parameters systematically selected by the BC or LMI methods as shown in Table 5.1. The BC tuning method requires, as expected, significantly more computation time than the simulations using arbitrary tuning parameters. The LMI method provides the best

performance but at the cost of an extremely high computation time. The plant simulation for all four trials are identical, thus the increase in computation time is almost entirely the result of the track selection method.

Method	Error Score	γ_{sp}	Execution Time (s)
Arbitrary tuning 1	0.335134	0.008489	0.016
Arbitrary tuning 2	0.885266	0.018235	0.015
BC Tuning	0.325586	0.008744	7.08
LMI Tuning	0.277526	0.007469	3449

Table 5.1: Test results for a linear first order system

The overall system response is plotted in Figure 5.7 along with a detailed section of this later figure shown in Figure 5.8. The detailed section shows the response during a step change in the parameter 'b'.

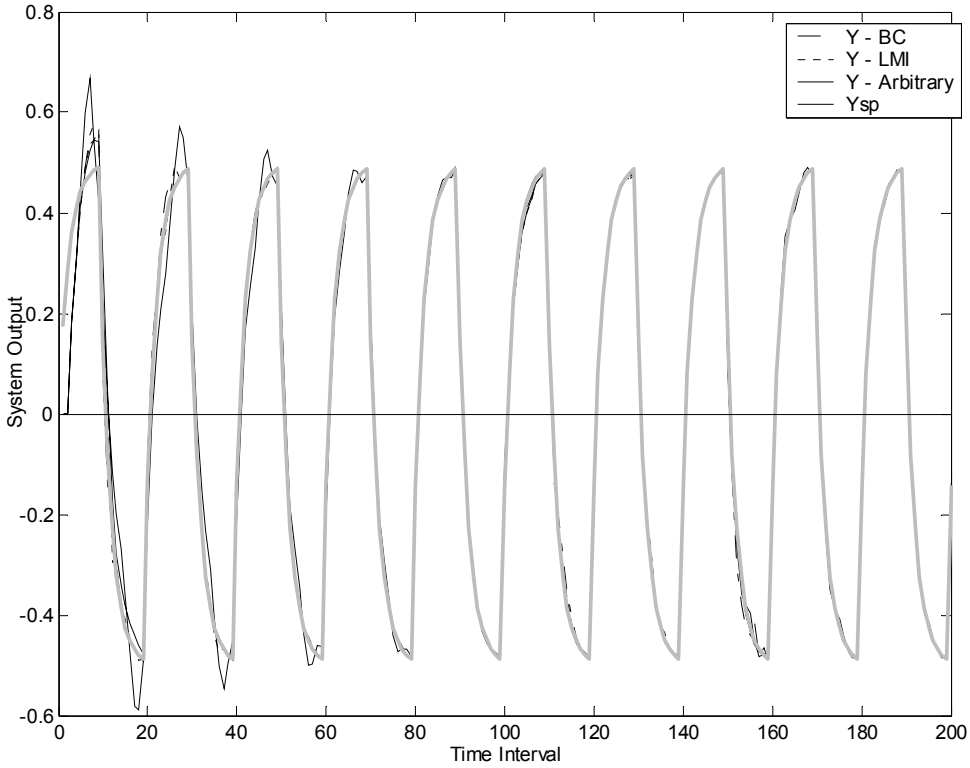

Figure 5.7: First order system response

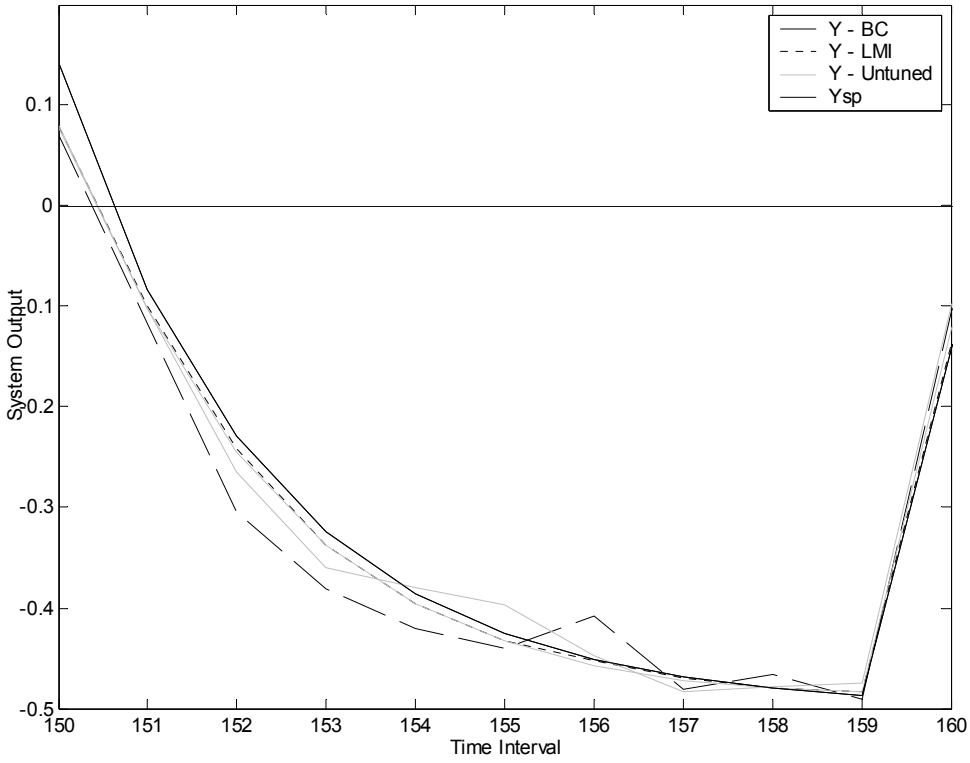

Figure 5.8: First order system response (Detail)

The track selection based on the BC and LMI methodologies is plotted in Figure 5.9. The LMI based track selection method is more conservative than the BC selection as shown by the tendency to select tracks with low adaptation gains. For this particular system, this is an advantage, but for a system with continuously varying parameters, the more aggressive track switching provided by the BC could be advantageous.

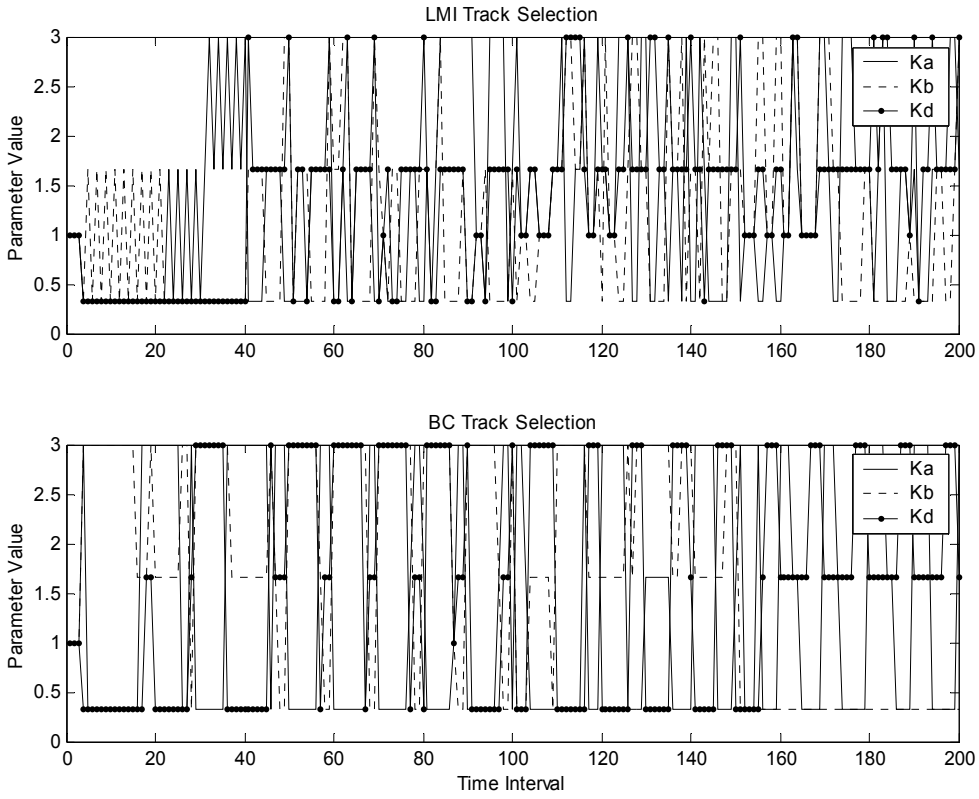

Figure 5.9: First order system track selection

The response in the estimated values for the parameter 'a' and 'b' are plotted in Figure 5.10 and Figure 5.11. The estimation track selection changes correspond to jumps in the value of the estimate. The estimates values calculated by using the controllers based on the BC and LMI methods reach their true values quickly after each step change, but they are slightly oscillatory around the true values. One of the simulations conducted with arbitrary tuning parameters shows very large oscillations, leading to corresponding poor performance.

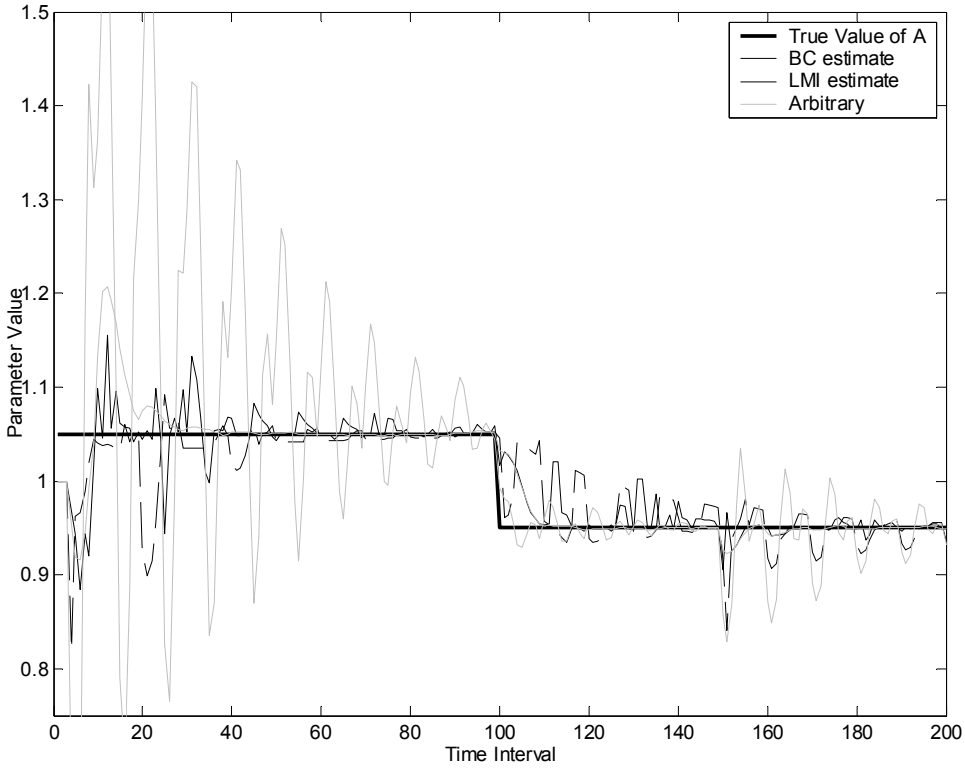

Figure 5.10: First order system estimate for the parameter 'a'

In Figure 5.11 oscillation patterns are visible in the estimated values of the parameter 'b' for the arbitrarily tuned system and the RLS algorithm. Most of the parameter adaptation occurs during the swings of the excitation in the reference signal, suggesting that track selection could be performed once per excitation cycle to save computation time with minimal impacts on performance.

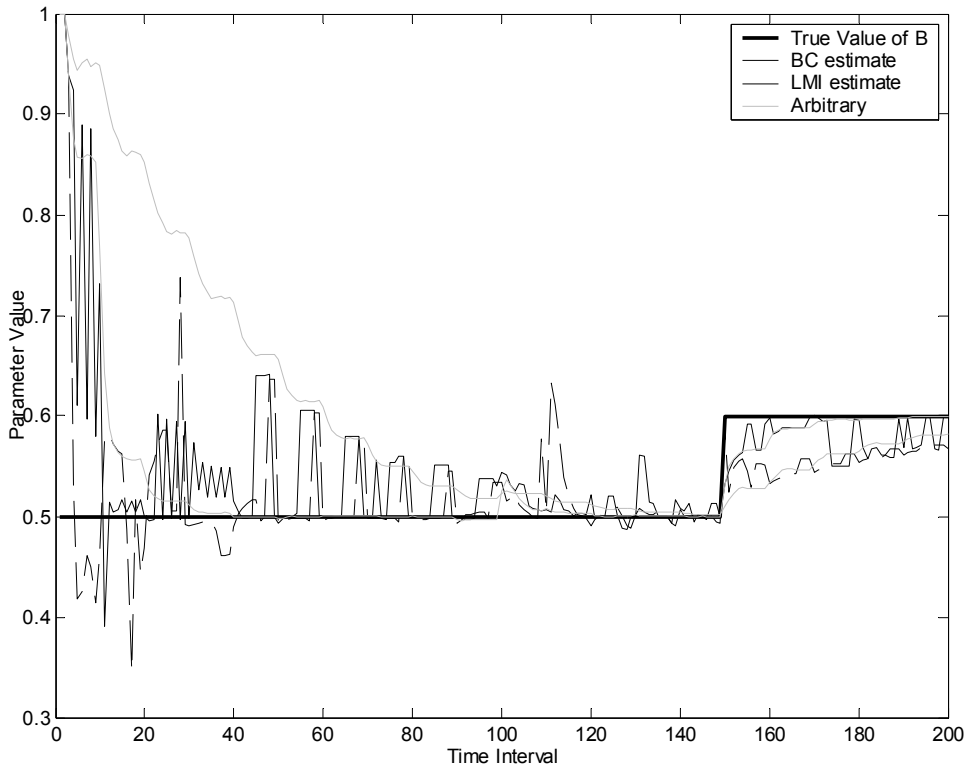

Figure 5.11: First order system estimate for the parameter 'b'

For a first order linear system both tuning methods discussed in this work provide significant performance improvements over an arbitrarily tuned controller. However, the extremely large computation time requirements for the LMI based tuning method system make it a poor choice for linear systems with white noise disturbances where it offers small potential benefits over other tuning methods. The fact that adaptation mostly occurs during drastic changes in the excitation signal seems to indicate that changes in the track are necessary only once per excitation step rather than once per time interval. This may result in significant savings in computation time.

5.3.2 Higher Order Systems

For the third comparison, a system that is second order in the state is used, with the following form:

$x_{k+1} = a_{0,k}x_k + a_{1,k}x_{k-1} + b_k u_k$ $y_k = x_k + \eta_k$ *where* $k \in \{1,\cdots,500\}$ $a_{0,k} = 0.9$ $a_{0,k} = -0.1$ $b_k = 0.5$ $\eta_k = (1-\beta)d_k + \beta\eta_{k-1},\ \beta = 0.75, d \in N(0,0.005)$	(5.2)

A total of 82 estimation tracks are used, along with 126 simulation trajectories and a simulation horizon of 2 for the BC method. The uncertainty bounds use a multiplier, M_Δ, of 2. The LMI method is evaluated over a number of time steps that are multiple of 10, as suggested in section 5.3.1, to reduce the computation time expected due to the additional number of parameters in the second order system.

Test results are summarised in Table 5.2. The BC method provides a significant performance improvement over both the un-tuned, and the LMI tuning method. This test demonstrates the conservative nature of the LMI tuning method.

Method	Error Score	γ_{sp}	Execution Time (s)
Arbitrary tuning	1.241621	0.009198	0.079
BC Tuning	0.574759	0.005864	17.16

LMI Tuning	0.905469	0.007517	6499.9

Table 5.2: Test results for a second order system

The estimates for all three model parameters are shown in Figure 5.12. Both the BC and LMI tuning methods provide some advantage in the estimation performance before the 50^{th} time interval. After this point the BC method still provides a significant estimation advantage, but the LMI tuning results in a conservative track selection, and is almost equivalent to the arbitrarily tuned system, as shown in Figure 5.13.

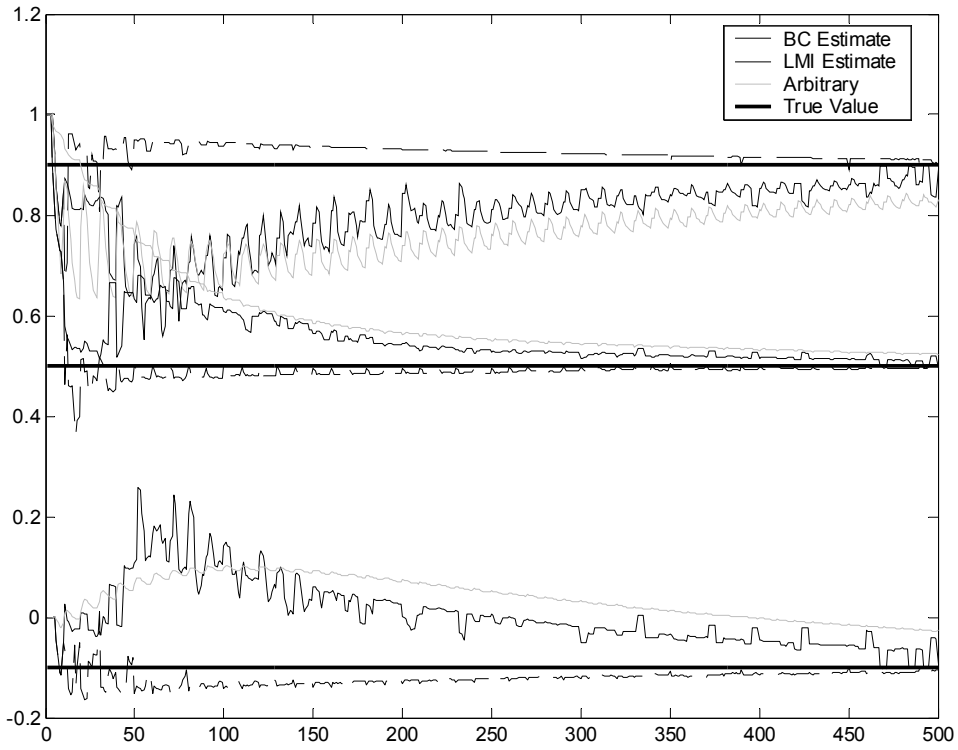

Figure 5.12: Second order system parameter estimates

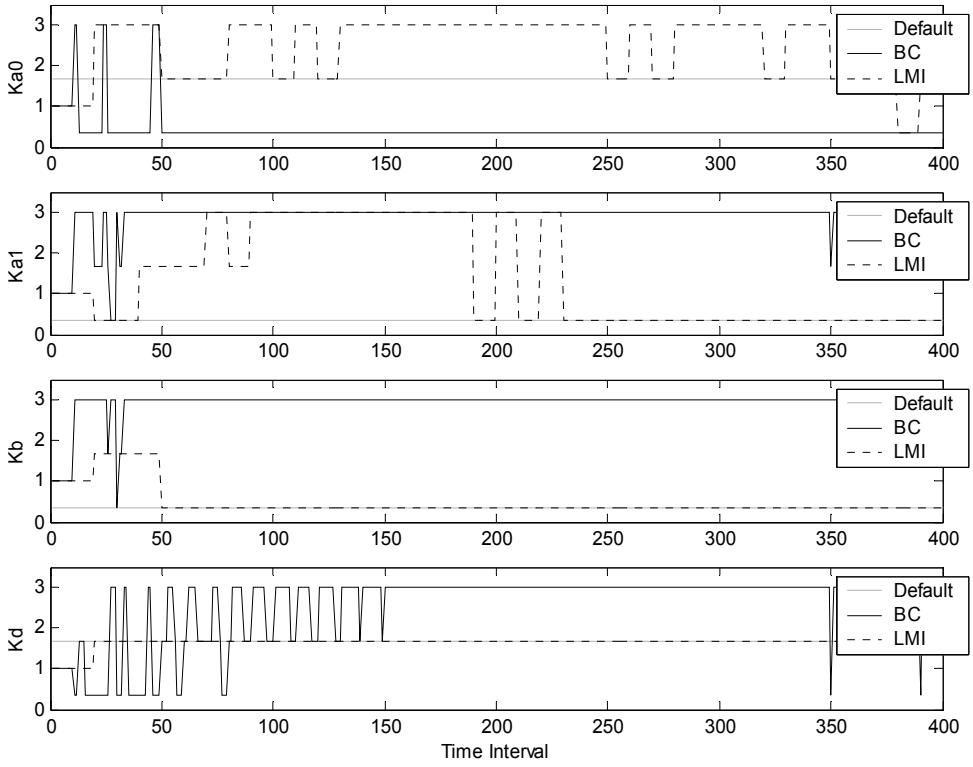

Figure 5.13: Second order system estimation track selection

The advantage of the quick estimation at the beginning of the simulation can be seen in Figure 5.14. The response of the arbitrarily tuned system lags after the set-point while the estimation error is large.

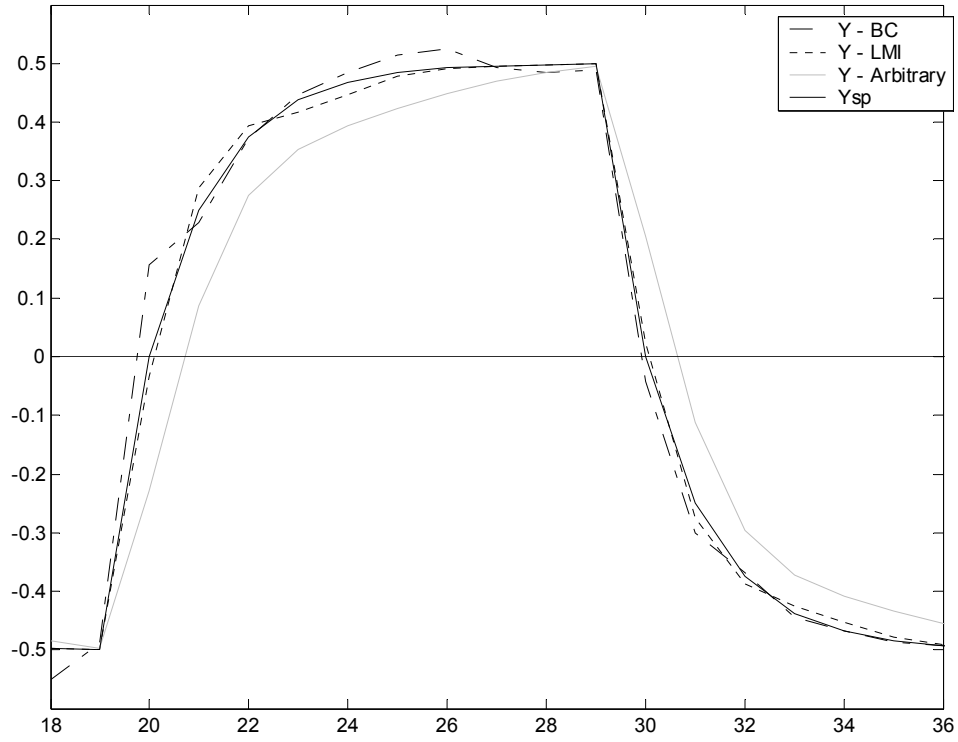

Figure 5.14: Second order system response (Detail)

In summary, the performance potential of a tuned adaptive control system is evident for a system that has higher than first order dynamics.

5.3.3 Deterministic Disturbances

In this section, a situation where the LMI method will offer significant improvement over the BC tuning method, will be illustrated. This situation corresponds to adaptation in the presence of a deterministic square wave disturbance where a bound on its amplitude is assumed to be known a priori but its period is unknown a priori.

The BC selection method is based on prediction of future performance based on the simulated response in the presence of the disturbance. Clearly, a good estimate of the disturbance is needed for the prediction to be accurate. If the assumed disturbance is significantly different from the actual one, the BC method may erroneously select an estimation track that may correspond to a poor performance for the actual disturbance occurring in the system. In the LMI selection method, the disturbance can be represented by an uncertainty parameter, δ_d, where bounds on the amplitude are all that is required to calculate the tuning parameters. Then the uncertainty model for the LMI system with a bounded disturbance can be accordingly represented as follows:

$$\boldsymbol{\eta}_{k+1} = \left[\mathbf{A}_0 + \mathbf{A}_a\delta_a + \mathbf{A}_b\delta_b + \mathbf{A}_d\delta_d\right]\boldsymbol{\eta}_k + \mathbf{B}\nu_k \qquad (5.3)$$

The disturbance is not considered as an input for the LMI calculations, but is instead accounted for as an additional source of uncertainty on the parameters of the state matrix $\mathbf{A}_k$. For example; consider a system with a bounded disturbance

$$x_{k+1} = ay_k + bu_k + d_k \qquad (5.4)$$

Where, d_k is assumed to be a periodic squared-wave disturbance with known amplitude but an unknown period. To accurately model the disturbance by the simulation, used for the BC method; the period, amplitude and phase are all required to be known a priori, whereas for the LMI method only the maximal amplitude is required. The LMI method will find the worst case γ for any disturbance with amplitude less or equal than this maximal amplitude.

In this example the disturbance is assumed, for the calculation of the BC method, to be a square wave of period 2, but the actual used disturbance is a square wave with a period of 8. Tuning is restricted to the parameter K_D for simplicity and clarity, while the other adaptation gains, K_A and K_B, are fixed each at a value of 1.0.

The error score, as given by the observed BC cost function, using different values of K_D for the actual and assumed disturbances is plotted in Figure 5.15. In the parameter space region examined, it was found that the relation between the error score and the value of K_D to obtain the best performance is very different and show opposite trends for disturbances with period of 2 and period 8. The simulations used in the BC tuning method will produce results that are based on the 'Assumed Disturbance' line whereas the actual result obtained is based on the 'true' disturbance line.

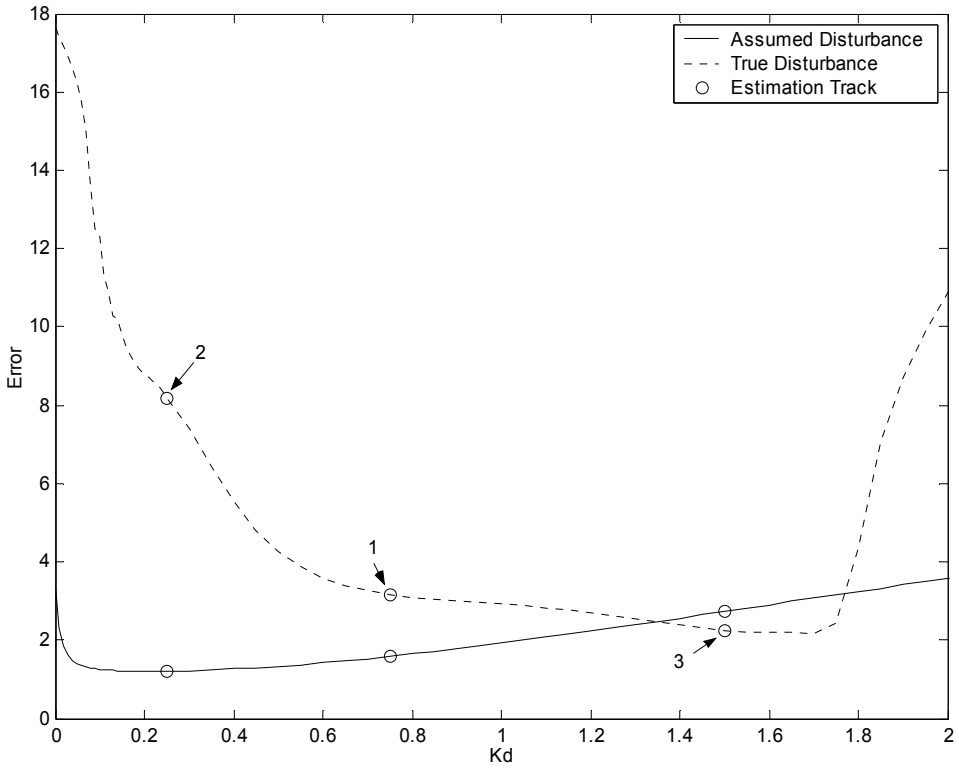

Figure 5.15: System Error vs. K_D value

In principle the best track, i.e. the track that if always selected will result in the smallest error, is labelled 3, with $K_D = 1.5$, but based on the BC simulations the track selected most frequently was the one labelled '2' with $K_D = 0.25$. The LMI method consistently selects track 3, while the BC method leads to frequent switching between tracks and often selects tracks '2' or '1', as shown in Figure 5.16. The results of these simulations are summarised in Table 5.3. The execution time for the LMI method is, as expected, much larger than for the other methods. It should be noted that the LMI method had 'infeasible' solutions for 21% of the calculations, which is a notable increase from the scenario examined in section 5.3. However, despite the longer computation times and

infeasibilities it is clear from the Table that the LMI method results in much better performance than the BC method.

Method	Error Score	Gamma SP	Time (s)
LMI	0.201	0.0111	1203.15
BC	0.427	0.0251	2.34

Table 5.3: Deterministic disturbance system results

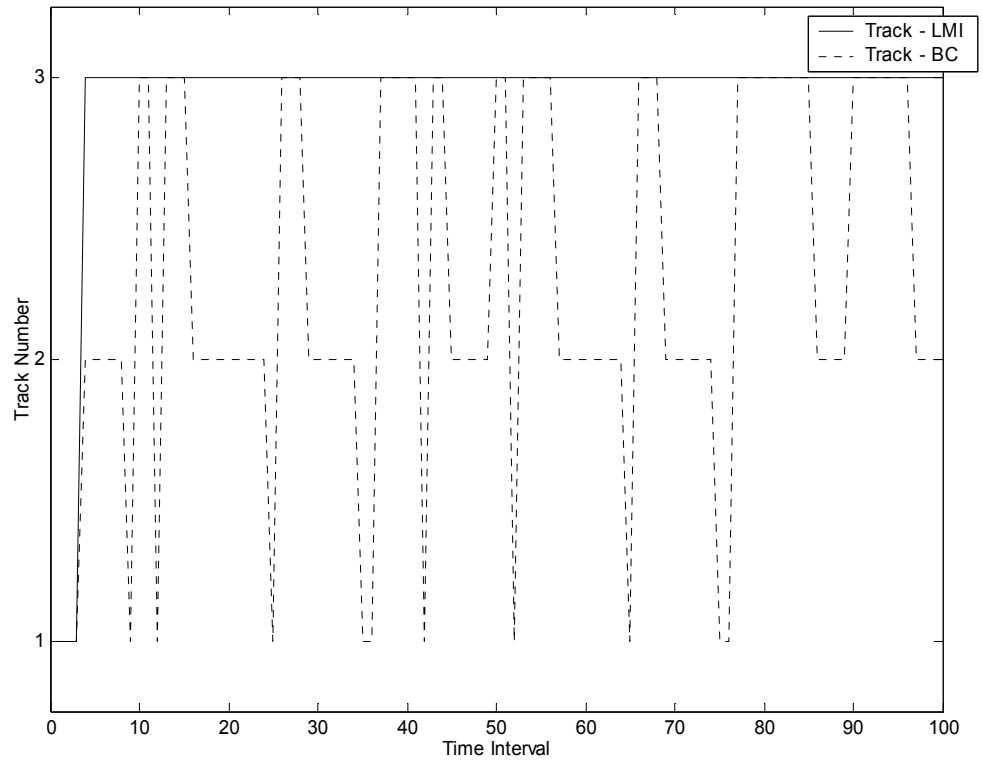

Figure 5.16: Deterministic disturbance system estimation track selection

The oscillatory effect of the frequent track selection changes in the BC method is clear in Figure 5.17.

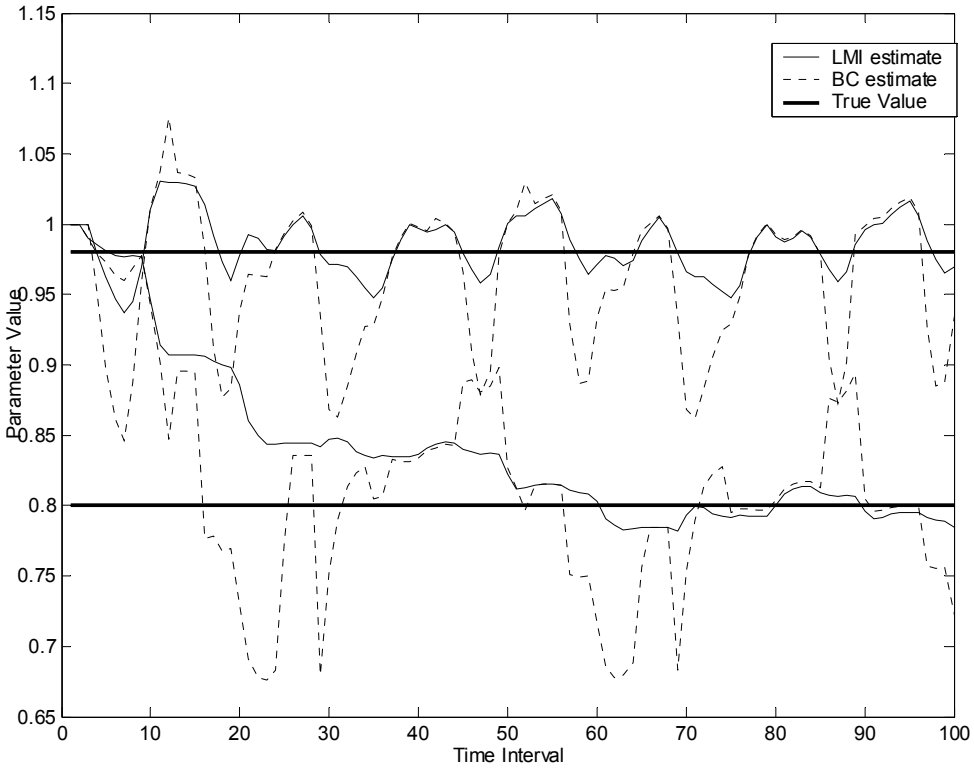

Figure 5.17: Deterministic disturbance system parameter estimates

Figure 5.18 and Figure 5.19, below, show the system output for each method. The BC method clearly has the worse tracking error. The lag in the LMI response is due to the slow adaptation for the parameter 'b'.

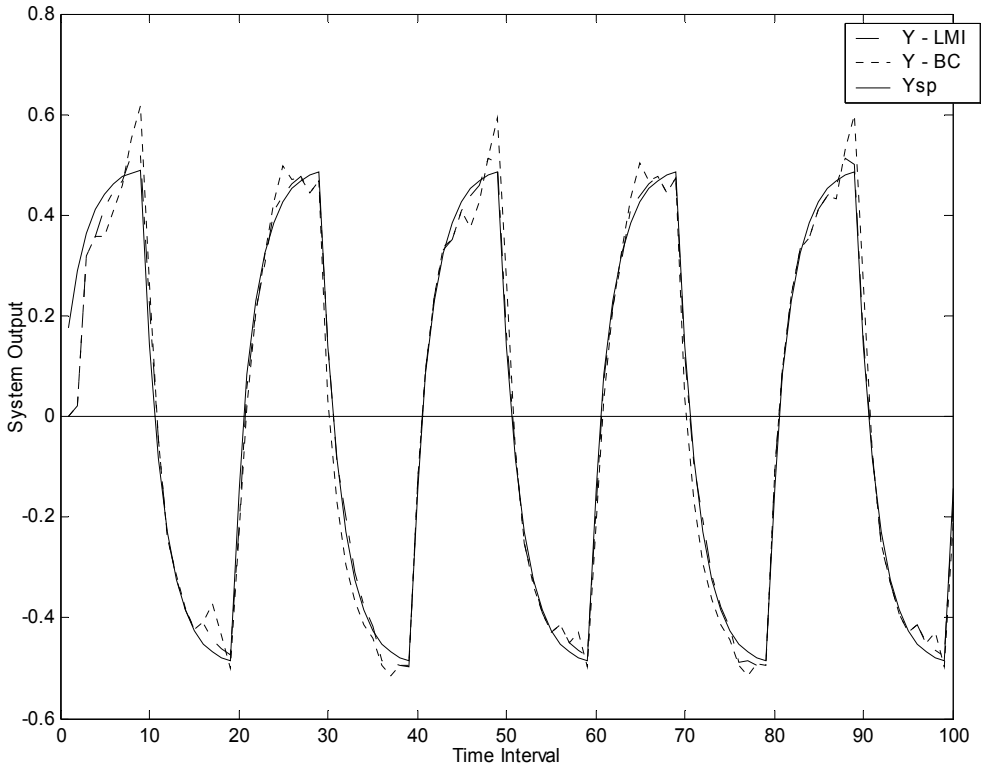

Figure 5.18: Deterministic disturbance system response

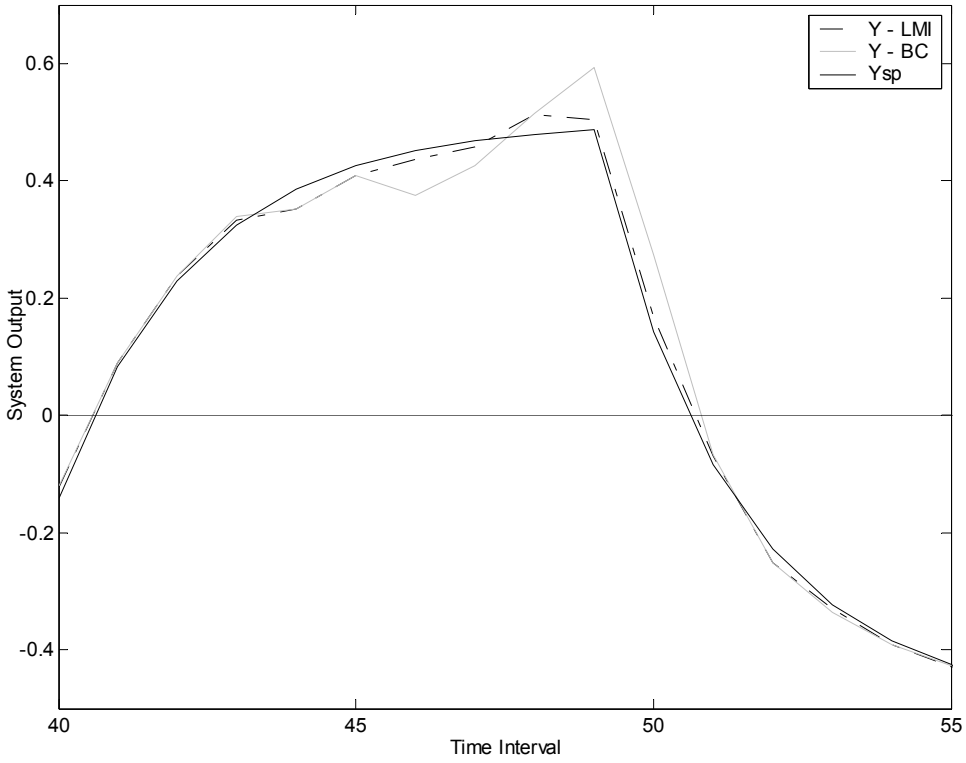

Figure 5.19: Deterministic disturbance system response (detail)

The system used in this section demonstrates how the BC method can erroneously select a track in the presence of an incorrectly modelled disturbance. On the other hand, the LMI tuning method uses a bound on the magnitude of the disturbance to select the correct track. Due to the computation time requirements for the LMI method, actual use of the LMI method would likely be limited to systems of high order for which the improvement in estimation is expected to be significant.

5.4 Comparison with RLS

The BC and LMI tuning methods are compared with the RLS algorithm using the same system (4.2) and setup as used in section 5.3.1. The results are summarised in Table 5.4, together with one of the simulations of the arbitrarily tuned systems. Both the BC and LMI tuning methods provide significant performance improvements over RLS. The arbitrarily tuned controller has the potential to perform better if good parameters are selected by chance. The execution time of the RLS method is similar to the arbitrarily tuned system, and substantially faster than for the BC method.

Method	Error Score	γ_{sp}	Execution Time (s)
Un-tuned 1	0.335134	0.008489	0.016
RLS	0.474401	0.010956	0.047
BC Tuning	0.325586	0.008744	7.08
LMI Tuning	0.277526	0.007469	3449

Table 5.4: Test results for a linear first order system with RLS

The plant model parameter estimates are shown in Figure 5.20 and Figure 5.21. The RLS estimate responds slowly to step changes but it has very little oscillations. The RLS has a slower response as the step changes in the parameters occur later in the simulation since the co-variance matrix $\mathbf{P}_k$ slowly converges to zero as explained in Chapter 2, while the methods used in this work do not have this time dependent deterioration of performance.

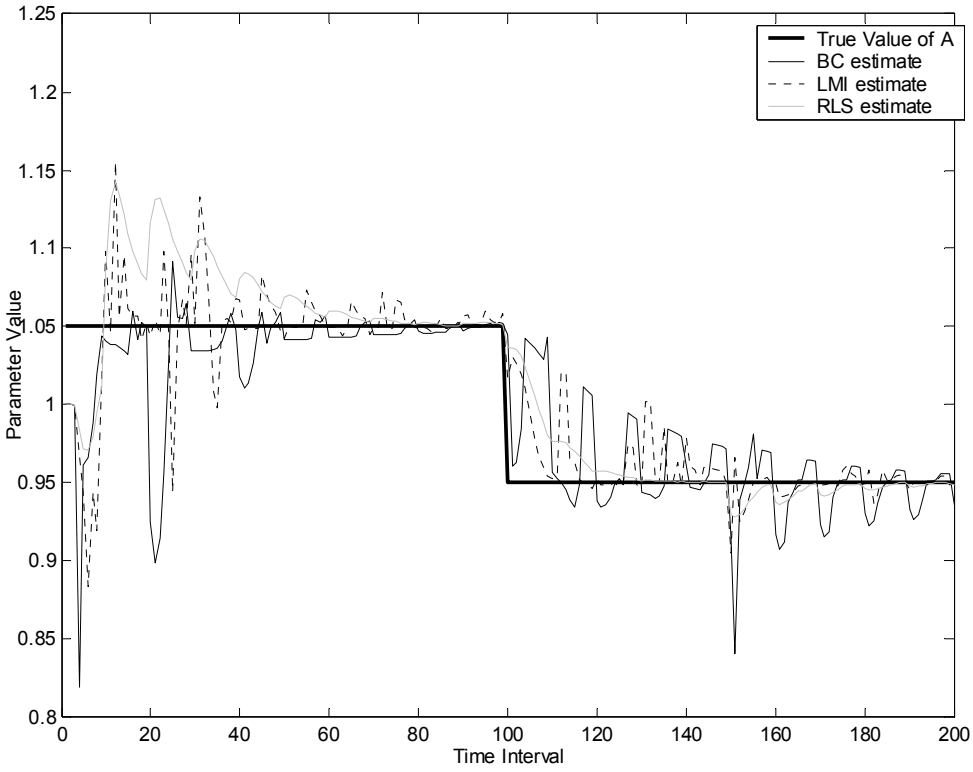

Figure 5.20: RLS comparison estimate for the parameter 'a'

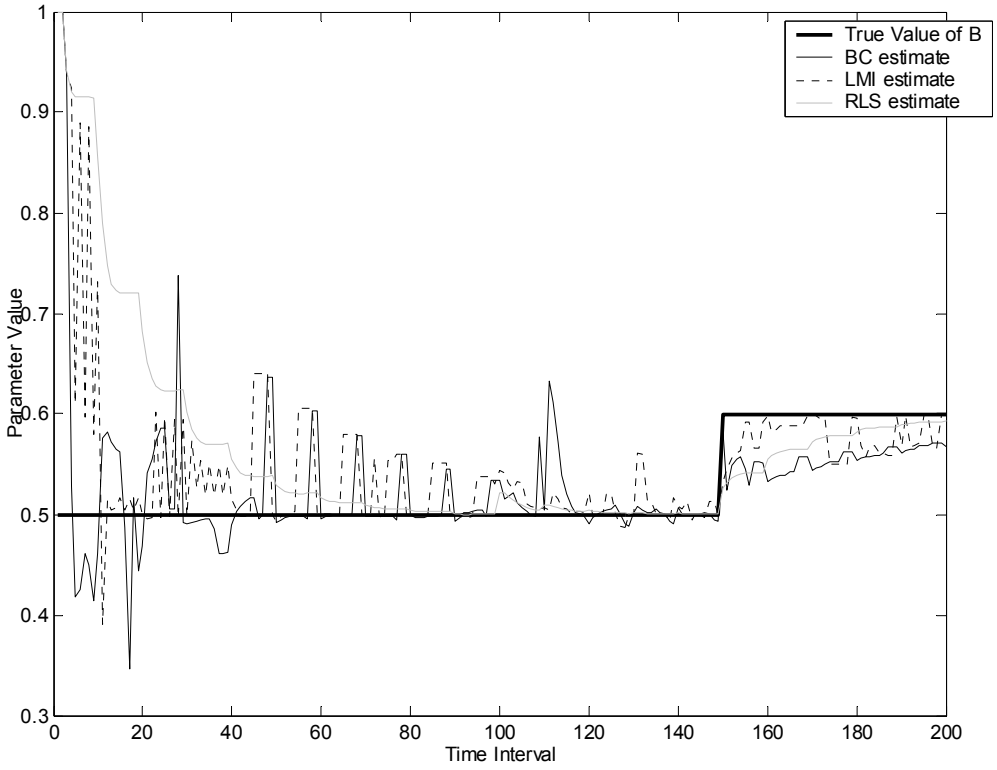

Figure 5.21: RLS comparison estimate for the parameter 'b'

A detail of the system response is shown in Figure 5.22. The overall system response is similar to the ones shown in Figure 5.7, and therefore is not shown again for brevity. The curve corresponding to the RLS method shows a consistent undershoot due to the slow adaptation of the plant model after the model parameter step change.

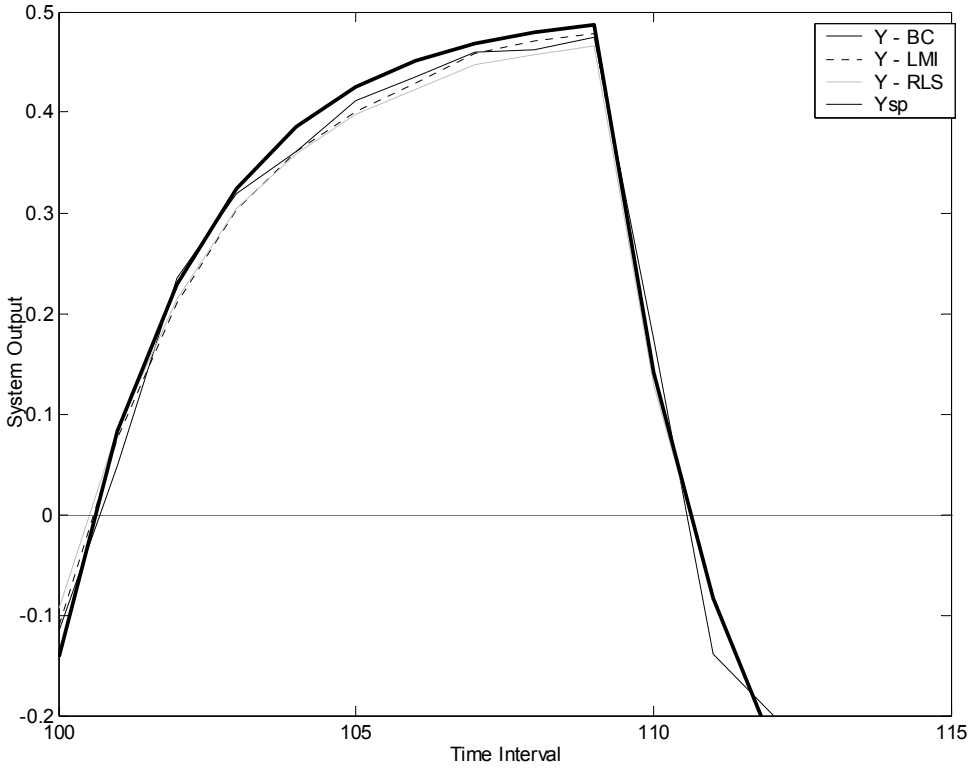

Figure 5.22: RLS comparison system response (detail)

6 Conclusions

A novel adaptive control method was developed. The plant model estimator uses a gradient descent approach with estimator gains suitable for on-line tuning. The error filter used was designed to meet the Lyapunov direct stability theory. A method for avoiding numerical division by zero errors was proposed. Two novel tuning methods were developed to avoid extensive trial and error simulations for proper tuning. Performance tests were performed to compare the novel adaptive control scheme and both tuning methods developed with the well known RLS algorithm.

The stability proofs required that adaptation gains remain constant with respect to time. A tuning framework using *estimation tracks* was proposed to meet this constraint. Dual adaptive control methods in the literature use an excitation signal to adjust the rate of adaptation, but simultaneous evaluation of the effect of different possible excitation signals is not possible, thus online tuning methods are limited in the design approaches that can be used. The tuning framework developed in this work uses multiple simultaneous estimates of the model parameters based on estimators with different estimation gains. The potential performance of each of these estimation tracks is evaluated, and the one with the best performance score is selected, thus adjusting the rate of adaptation, and meeting the dual adaptive control goals.

Two methods for evaluating and selecting the estimation tracks were developed. The first uses a prediction of the bicriteria (BC) cost function for each estimation track as a scoring method. This cost function includes the prediction and feedback errors, and was

developed to explicitly measure the principle design trade-off implicit in dual adaptive control, i.e. robustness in the face of uncertainty versus fast learning of the process. The second method uses the linear matrix inequalities (LMI) to find an upper bound on the feedback error, scored as γ_{sp}, given any admissible reference signal and bounds of the uncertainty in the model parameters.

The measures of system performance used for comparison were the observed value of the bicriteria cost function, the observed value of γ_{sp}, and the computation time used. The bicriteria cost function and γ_{sp} were the optimisation criteria used for the BC and LMI tuning methods respectively, and the computation time is critical for implementation considerations.

In the first set of tests the configuration parameters for the BC tuning method were investigated. There was a 20% performance improvement by using 9 estimation tracks over no-tuning. Performance decreases for more than 28 estimation tracks. The BC method is equivalent to a certainty equivalence controller when only one step of simulation is used to predict the BC error score. There is a 63% performance improvement by using 9 simulations to cover the range of parameter uncertainty instead of using only the current estimated value, but there is not significant change if more than 9 simulation tracks are used. There is a 9% performance improvement by using a 2 or 3 time step simulation horizon. For longer horizons there is not performance benefit. For each of these factors, computation time is approximately proportional to the increase in each one of the tuning parameters, i.e. number of estimation tracks, number of

simulations per track or time horizon for prediction, with a multiplicative effect for changes in two or more of these factors. It was also found that the spacing of the simulation trajectory initial parameter values little impact unless the uncertainty multiplier, M_Δ, is greater than 2.

The number of estimation tracks and the uncertainty bounds, as indicated by M_Δ, were investigated for the LMI tuning approach. There was a 20% performance improvement over the system with arbitrarily chosen tuning parameters when 9 estimation tracks were used. With 28 or more tracks deterioration in performance was observed due to sensitivity to measurement noise. The computation time was proportional to the number of estimation tracks used, and the execution time was approximately 100 times longer for the LMI tuning than for the BC tuning method when the same number of estimation tracks was used for both methods. Values of M_Δ between 0.25 and 2.0 provided a 15% performance improvement over a system with arbitrarily chosen parameters. Values outside this range resulted in decreased performance. The value of M_Δ has a small, but unpredictable effect on computation time.

The BC and LMI tuning methods were compared to each other and to a controller with arbitrarily selected parameters under three scenarios. For the first scenario a first order system with step-changes in each parameter and Gaussian measurement noise were used. The BC and LMI tuning methods were configured to use the best performing combination of number of estimation tracks, number of simulation tracks, simulation horizon and uncertainty bounds identified in the previous testing. The BC method

resulted in a 3% and 63% performance improvement compared to the best and worst arbitrarily tuned systems respectively. It should be remembered that with arbitrarily selected parameters the performance potential is unknown until after the test is over. The simulation methods can address many possible combinations of tuning parameters simultaneously. The LMI tuning method provided an additional 15% performance improvement over the BC method, at the cost of a 490 times increase in the computation time. The BC tuning method requires 217 times the computation time of a system with arbitrarily chosen parameters.

The second scenario examined was a second order linear system. The LMI and BC methods resulted in a 27% and 53% performance improvements over the system with arbitrarily chosen parameters used for comparison. Similarly to the first order test, the LMI method had a computation time that was 380 times longer than the BC method, even when the optimizations were conducted infrequently, i.e. once per period of the external periodic set point signal.

The third scenario studied was designed to illustrate a typical scenario where the LMI tuning method would provide significant performance advantages over the BC method. The system was first order with an unknown deterministic disturbance being introduced to the process. For the LMI approach, only the amplitude of the disturbance is required for track selection. For the BC approach a specific guess of both magnitude and period of the disturbance signal was required. It was shown that if the guess of the period, phase or wave-form is inaccurate, the BC prediction will give inaccurate results. Consequently,

for the case study, a 53% performance gain was achieved for the LMI tuning method over the BC tuning method.

For a final comparison the recursive least squares algorithm was used on the first order system from the first scenario. Performance improvements of 31% and 41% over the RLS algorithm were achieved with the BC and LMI tuning methods, respectively. The RLS algorithm requires some form of resetting or exponential forgetting factor. Otherwise, adaptation will cease as the covariance matrix, $\mathbf{P}_K$, converges to zero. However, there are no systematic ways to reset the covariance in the presence of frequent step changes in the model parameters unless the timing of these changes is a priori known. In this case both the BC criteria and LMI methods proposed in this thesis are offering a clear advantage over the traditional RLS algorithm.

The LMI method required a very large computation effort, and thus it is not recommendable for systems where the BC method provides similar performance. An example of a system where an LMI based method provides a clear advantage is a system with the unknown deterministic type disturbances. In summary both tuning methods offer a systematic approach for selecting tuning parameters, as compared to the trial and error approach often used by practitioners.

7 Recommendations

1- The cost function used to evaluate performance should include manipulated variable changes as a new term. The manipulated variable moves are in many practical situations a key consideration for tuning of the controller. Additionally, penalization of the manipulated variable moves in the cost functions may help to reduce the oscillatory behaviour of the parameter estimates but at the cost of a slower response to changes in the model parameters.

2- Other suitable feedback control laws can be tested in combination with the adaptive estimator such as pole-placement and model reference controllers.

3- To limit the computational burden of the LMI approach, the track selection method can be performed less frequently than the sampling frequency of the system. The timing of the on-line tuning calculations with respect to the excitation signal, and the frequency of the track selection were not explicitly investigated and may be an interesting subject for future investigations.

4- To improve performance after the parameter estimates have converged, a dead-zone approach should be investigated to deal with measurement noise, and to avoid possible parameter drift problems common when the model structure selected for adaptation is not sufficiently accurate to model the actual behaviour of the system. This would prevent the system from using noise to drive the adaptation when there is insufficient excitation.

5- The adaptive control scheme proposed in this study could be extended to multivariable systems.

6- The adaptive controller studied in this work is suitable as well for a special class of non-linear system represented by an artificial neural network (ANN) model with linear gains of the following form:

$$x_k = \sum_{i=1}^{n} a_{i,k} f(x_{k-i}) + \sum_{i=1}^{m} b_{i,k} f(u_{k-i}) \qquad (7.1)$$

where the functions *f*'s are nonlinear basis functions such as Radial Basis Gaussian functions or wavelets. These models have been previously used in the literature for adaptive control of nonlinear systems. Due to the linear dependence of this model with respect to the model parameters all the developments in the current study are applicable to nonlinear models of the form given by equation (4.2).

References

[1] Allison B.J., J.E. Cairniello, P.J.C. Tessier and G.A. Dumont, 1995, Dual Adaptive Control of Chip Refiner Motor Load, *Automatica*, **31**, 1169-1184.

[2] Astrom, K.J., B. Wittenmark, 1989, *Adaptive Control*, Addison-Wesley

[3] Dumont G.A., and K.J. Astrom, 1987, Wood Chip Refiner Control, *American Control Conference.*

[4] Fabri S., K. Kadirkamanathan, 1998, Dual Adaptive Control of Nonlinear Stochastic Systems using Neural Networks, *Automatica*, **34**, 245-253

[5] Filatov N.M. and H. Unbehauen, 1998, Adaptive control for continuous-time systems: A simple example. *UKACC Intl. Conf. on control*, 39-43

[6] Filatov N.M. and H. Unbehauen, 2000, Survey of Adaptive Dual Control Methods, *IEE Proc.-Control Theory Appl*., **147**, 118-128

[7] Filatov N.M., H. Unbehauen and U. Keuchel, 1997, Dual pole placement controller with direct adaptation. *Automatica*, **33**, 113-117.

[8] Filatov N.M., U. Keuchel and H. Unbehauen, 1996, Dual Control for an Unstable Mechanical Plant, Third *IEEE Conference on Control Applications*.

[9] Gao and Budman (2004) In Proceedings of ADCHEM 2003 (International Symposium on Advanced Control of Chemical Processes), Hong Kong, January 2004

[10] Goodwin, G.C., K.S. Sin, 1984, *Adaptive Filtering Prediction and Control*, Prentice-Hall

[11] Kothare M.V., V. Balakrishnan and M. Morari, 1996, Robust Constrained Model Predictive Control using Linear Matrix Inequalities, *Automatica*, **32**, 1361-1379.

[12] Kothare M.V., V. Balakrishnan and M. Morari, 1994, Robust Constrained Model Predictive Control using Linear Matrix Inequalities, *American Control Conference*, **WM2 – 2:50**, 440-444.

[13] Liu, R.W., 1968, Convergent Systems, *IEEE Trans. Aut. Control*, **13(4)**, 384-391

[14] Ozkan L., M.V. Kothare and C. Georgakis, 2000, Model Predictive COntrol of Nonlinear Systems using Piecewise Linear Models, *Computers and Chemical Engineering*, **24**, 793-799

[15] Sanner R.M. and J-J.E. Slotine, 1992, Gaussian Networks for Direct Adaptive Control, *IEEE Trans. On Neural Networks*, **3**, 837-862

[16] Seborg, D.E., T.F. Edgar and D.A. Mellichamp, 1989, *Process Dynamics and Control*, John Wiley & Sons

[17] Slotine, J-J E. and W. Li, 1991, Applied Nonlinear Control, Prentice hall, New Jersey.

[18] Sternby J., 1976, A Simple Dual Control Problem with an Analytical Solution, IEEE Trans. Automatic Control, 840-844

[19] Veres S. M. and H Xia, Dual Predictive Control for Fault Tolerant Control, 1998, *UKACC Intl. Conf. on Control*.

Appendix A: Code for Initialization

This section initializes the variables used in the code in appendices B and C. Parameter sets associated with estimation tracks are picked, as is noise and the reference signal.

```
function main(done)

% done - number of cycles to evaluate
% Ka, Kb, Kd, - Intial/default values of adpatation parameters
% alpha - damping factor for set point (reference model)
% beta - damping factor on noise - for stability - unknown value
% Areal, Breal - True plant parameters - unknown values, a > 1 is
unstable
% Ainit, Binit, yinit, uinit - initial values for the system

close all;
k_points    = 3;        % number of points to sample in each parameter
a_points    = 3;        % number fo points to sample for each estimate
noise       = 0.005;    % size of one standard deviation of gaussian
noise

Kainit      = 1;        % default tuning parameters for untuned systems
Kbinit      = 1;
Kdinit      = 1;
Kainit4     = 4;
Kbinit4     = .25;
Kdinit4     = .5;

alpha       = 0.65;     % time constant for reference model
beta        = 0.75;     % time sonstant for Nyquist limit low pass
filter

Areal       = 1.05;     % true plant
Breal       = 0.5;

Ainit       = 1;        % initialization values
Binit       = 1;
yinit       = 0;
uinit       = 0;

period      = 10;       % values for reference signal
amp         = 0.5;

% parameters for optimizations

horizon     = 2;        % Future sampples for path following technique
start_cycle = 3;        % begin optimization after ___ iterations
std_hor     = 20;       % number of samples to use for std dev. (max)
multiples   = 2;        % Standard deviations for estimate bounds
mult_lmi    = 2;        % same but for LMI
check       = 3;        % Check first to k_space parameters
```

```
% loop control and system order
Vmax        = 100000;   % break if out of control
n           = 1;        % order of system
seed        = 1;        % random seed

% variable initializations
small       = 0;        % time weighted lyapunov
feedback    = 0;        % feedback performace index (Bi-criteria)
pred        = 0;        % prediction performance index (Bi-criteria)
k_weight    = 5;        %start_cycle;
k_weight_l  = 1;        % number of cycles to average predictions over
A           = ones(1,max(done,400))*Areal;
B           = ones(1,max(done,400))*Breal;

% add step changes in parameter values
A(100:400)  = A(100:400) - 0.1;
B(150:400)  = B(150:400) + 0.1;

% more initialization
Aest        = zeros(1,done);
Aest_path   = ones(done,k_points^check+1);
Best        = zeros(1,done);
Best_path   = ones(done,k_points^check+1);
u           = zeros(done,1);
y           = zeros(done,1);
x           = zeros(done,1);
s           = zeros(done,1);
s_path      = zeros(done,k_points^check+1);
V           = zeros(done,1);
Ka          = ones(done,1)*Kainit;
Kb          = ones(done,1)*Kbinit;
Kd          = ones(done,1)*Kdinit;
k_set       = zeros(3,k_points^check+1);
ypred_path  = zeros(done,k_points^check+1);
ypred_pick  = zeros(k_points^check+1);
a_set       = zeros(2,a_points^2);
a_set       = zeros(2,(a_points*2+1)^2);
a_weight    = zeros(1,(a_points*2+1)^2);
ypred_pick  = zeros(done,k_points^check+1);
score       = zeros(k_points^check+1,a_points^2);
Srun        = zeros(k_points^check+1,done);
Save        = zeros(k_points^check+1,done);
Sconf       = zeros(k_points^check+1,done);
time        = (1:(done+1+horizon));
ref         = zeros(1,done+1+horizon);


%use same noise set for each run
randn('state',seed);
rand_set1    = randn(done,1);
% filter the noise (low pass filter, weak noise at nyquist frequencies)
rand_set = filter(1-beta,[1, -beta],rand_set1);

% weighting matrix for a
% used for bi-criteria method
a_it = 0;
if a_points == 0
```

```
    a_weight = 1;
else
    for aa = 0:(a_points*2)
        for ab = 0:(a_points*2)
            a_it = a_it + 1;
            a_weight(a_it) = normpdf(2*(aa/(a_points*2) -
0.5)*multiples,0,1)*normpdf(2*(ab/(a_points*2) - 0.5)*multiples,0,1);
        end
    end
end

% define sweep space used in prediction.
% versions for checking Ka, and Kb, as well as both plus Kd
% note this starts at column 2, the first one is the default values
k_it = 1;
for aa = 0:(k_points-1)
    for ab = 0:(k_points-1)
        for ad = 0:(k_points-1)
            k_it = k_it + 1;
            k_set(:,k_it) = [aa*(4/k_points) + 1/k_points;
ab*(4/k_points) + 1/k_points; ad*(4/k_points) + 1/k_points];
        end
    end
end
% Add default sweep space
k_set(:,1) = [Ka(1); Kb(1); Kd(1)];

for i = time
    if mod(i,period*2) < period
        ref(i) = yinit + amp;
    else
        ref(i) = yinit - amp;
    end
end

% filter by first order reference model, with time constant, alpha
ysp = filter(1-alpha,[1, -alpha],ref);
```

Appendix B: Code for LMI Tuning Method

In this section of code the simulation for the plant is performed along with the LMI tuning block. The LMI tuning calls a function 'lmi_test' in Appendix E. After scoring all estimation values are set to the values associated with the current best track.

```
Aest(1)        = Ainit;
Aest_path(1,:) = Aest_path(1,:)*Ainit;
Best(1)        = Binit;
Best_path(1,:) = Best_path(1,:)*Binit;
y(1)           = yinit;
x(1)           = yinit;
u(1)           = uinit;
infeas         = zeros(1,done);
counts         = 0;
tic;

for i = (n+1):done

    % calcualte y, based on old y and inputs
    x(i) = A(:,i)'*x(i-1) + B(:,i)'*u(i-1);
    y(i) = x(i) + noise*rand_set(i);

    % feedback error
    feedback = feedback + (y(i) - ysp(i))^2;

    % predicted value of current output
    ypred(i) = Aest(:,i-1)'*y(i-1) + Best(:,i-1)'*u(i-1);

    % prediction error
    pred = pred + (y(i) - ypred(i))^2;

    % evaluate all k_set updates.
    for k_eval = 1:size(k_set,2)
        % calualte parameters for next error update
        Sa = -y(i-1)/(1 + k_set(3,k_eval));
        Sb = -u(i-1)/(1 + k_set(3,k_eval));
        Sf = (2*y(i) + (1 - k_set(3,k_eval))*s_path(i-1,k_eval) -
Aest_path(i-1,k_eval)'*y(i-1) - Best_path(i-1,k_eval)'*u(i-1))/(1 +
k_set(3,k_eval));
        As = k_set(1,k_eval)*y(i-1);
        Af = Aest_path(i-1,k_eval) + k_set(1,k_eval)*y(i-1)*s_path(i-
1,k_eval);
        Bs = k_set(2,k_eval)*u(i-1);
        Bf = Best_path(i-1,k_eval) + k_set(2,k_eval)*u(i-1)*s_path(i-
1,k_eval);

        % calualte new error metric
        s_path(i,k_eval) = (Af*Sa + Bf*Sb + Sf) / (1 - As*Sa - Bs*Sb);

        % new estimates for A, B
```

```
        Aest_path(i,k_eval) = s_path(i,k_eval)*As + Af;
        Best_path(i,k_eval) = s_path(i,k_eval)*Bs + Bf;
    end

    % uncertainty bounds from fitered error
    for m = 1:size(s_path,2)
        Srun(i,m) = s_path(i,m)^2;
        Sconf(m,i) = sqrt(sum(Srun(max(1,i-std_hor):i,m))/(i-max(1,i-
std_hor)+1));
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%% Begin optimisaton block %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    for k_eval = 1:size(k_set,2)
        % find lmi score
        ypred_path(i,k_eval) = lmi_test(y(i), Sconf(k_eval,i)*mult_lmi,
Aest_path(i,k_eval), Sconf(k_eval,i)*mult_lmi, Best_path(i,k_eval),
k_set(1,k_eval), k_set(2,k_eval), k_set(3,k_eval), s_path(i,k_eval),
ysp(i+1), alpha);
        counts = counts + 1;
        % weight last k_weight predictions averaged together.
        ypred_pick2(k_eval) = mean(ypred_path(max(1,i-
k_weight_l):i,k_eval));
    end

    % count infeasable results
    if ypred_path(i,k_eval) < 0
        infeas(i) = infeas(i) + 1;
        ypred_path(i,k_eval) = Inf;
    end

    % in tie, select current track
    if i > start_cycle
        ypred_pick2(next_k) = ypred_pick2(next_k)*0.999;
    end
    %%%%%%%%%%% End optimisation block %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %  pick next parameters if initialisation time has passed.
    if i > start_cycle
        ypred_pick(i,:) = ypred_path(i,:);
        [score_best, next_k] = min(ypred_pick2(:));
    else
        next_k = 1;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % pick next step parameters
    Ka(i) = k_set(1,next_k);
    Kb(i) = k_set(2,next_k);
    Kd(i) = k_set(3,next_k);

    % new estimates for s, A, B
    s(i)      = s_path(i,next_k);
    Aest(:,i) = Aest_path(i,next_k);
    Best(:,i) = Best_path(i,next_k);
```

```
    % calulate new output - uses the next set point value - we assume
this
    % is known
    u(i) = (ysp(i+1) - Aest(1,i)*y(i)) / Best(1,i);

    % calulate Lyapunov (for simulation purposes only)
    V(i) = (Aest(:,i) - A(:,i))'*Ka(i)*(Aest(:,i) - A(:,i)) +
(Best(:,i) - B(:,i))'*Kb(i)*(Best(:,i) - B(:,i)) + s(i)^2;

    % Print results
    fprintf('I:%i, V:%12.6f, y:%8.3f, s:%8.3f, Aest:%8.3f, Best:%8.3f,
YSP:%8.3f, u%8.3f\n', i, V(i),y(i), s(i), Aest(:,i), Best(:,i), ysp(i),
u(i));

    %break if runaway condition
    if V(i) > Vmax
        break
    end
end
time1 = toc;

time = (1:i)*1;

% calculate error signals
noise_error1 = noise*rand_set(time)'*rand_set(time)*noise;
sp_error1 = ysp(time)*ysp(time)';
out_error1 = (y(time)-ysp(time)')'*(y(time)-ysp(time)');
```

Appendix C: Code for BC Tuning Method

In this section of code the simulation for the plant is performed along with the BC tuning block. The BC tuning calls a function '`single_opt_run2`' in Appendix F. After scoring all estimation values are set to the values associated with the current best track. This code is similar to the code in appendix B, except for the BC evaluation section.

```
% reinitialize all needed variables - not involved with the paths.
small           = 0;        % time weighted lyapunov
feedback        = 0;        % feedback performace index (Bi-criteria)
pred            = 0;        % prediction performance index (Bi-
criteria)

e               = zeros(done,1);
s               = zeros(done,1);
V               = zeros(done,1);
Ka              = ones(done,1)*Kainit;
Kb              = ones(done,1)*Kbinit;
Kd              = ones(done,1)*Kdinit;
next_k3         = ones(done,1);
Aest(1)         = Ainit;
Best(1)         = Binit;
Aest_path(1,:)  = Aest_path(1,:)*Ainit;
Best_path(1,:)  = Best_path(1,:)*Binit;
y(1)            = yinit;
x(1)            = yinit;
u(1)            = uinit;

% time simulation
tic;

%fprintf('\nI:%i, V:%12.6f, y:%8.3f, s:%8.3f, Aest:%8.3f, %8.3f,
Best:%8.3f, %8.3f, YSP:%8.3f, u%8.3f\n', 1, V(1),y(1), s(1), Aest(:,1),
Best(:,1), ysp(1), u(1));

for i = (n+1):done
    fprintf('%i ',i)

    % calcualte y, based on old y and inputs
    x(i) = A(:,i)'*x(i-1) + B(:,i)'*u(i-1);
    y(i) = x(i) + noise*rand_set(i);

    % feedback error
    feedback = feedback + (y(i) - ysp(i))^2;

    % predicted value of current output
    ypred(i) = Aest(:,i-1)'*y(i-1) + Best(:,i-1)'*u(i-1);

    % prediction error
```

```
    pred = pred + (y(i) - ypred(i))^2;

    % evaluate all k_set updates.
    for k_eval = 1:size(k_set,2)
        % calualte parameters for next error update
        Sa = -y(i-1)/(1 + k_set(3,k_eval));
        Sb = -u(i-1)/(1 + k_set(3,k_eval));
        Sf = (2*y(i) + (1 - k_set(3,k_eval))*s_path(i-1,k_eval) -
Aest_path(i-1,k_eval)'*y(i-1) - Best_path(i-1,k_eval)'*u(i-1))/(1 +
k_set(3,k_eval));
        As = k_set(1,k_eval)*y(i-1);
        Af = Aest_path(i-1,k_eval) + k_set(1,k_eval)*y(i-1)*s_path(i-
1,k_eval);
        Bs = k_set(2,k_eval)*u(i-1);
        Bf = Best_path(i-1,k_eval) + k_set(2,k_eval)*u(i-1)*s_path(i-
1,k_eval);

        % calualte new error metric
        s_path(i,k_eval) = (Af*Sa + Bf*Sb + Sf) / (1 - As*Sa - Bs*Sb);

        % new estimates for A, B
        Aest_path(i,k_eval) = s_path(i,k_eval)*As + Af;
        Best_path(i,k_eval) = s_path(i,k_eval)*Bs + Bf;

    end

    % Calculate confidence interval
    for m = 1:size(s_path,2)
        Srun(i,m) = s_path(i,m)^2;
        Sconf(m,i) = sqrt(sum(Srun(max(1,i-std_hor):i,m))/(i-max(1,i-
std_hor)+1));
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%% Begin optimisaton block %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % reset score counter
    score = ones(k_points^check+1,(a_points*2+1)^2);

    for ki = 1:size(k_set,2)
        a_num = 1;

        % calculate the first step of the score - Aest current is not
        % needed for this calculation so the results are the same for
every
        % variation used
        score(ki,:) = score(ki,:)*((y(i) - ysp(i))^2 + (y(i) -
Aest_path(i-1,ki)*y(i-1) + Best_path(i-1,ki)*u(i-1))^2);

        % find combinations of inital values of 'true plant' values to
use
        if horizon > 1
            if a_points == 0
                a_set(1,1) = 0;
                a_set(2,1) = 0;
            else
                for aaa = 1:(a_points*2+1)
```

```
                    for bbb = 1:(a_points*2+1)
                        a_set(1,a_num) = Sconf(ki,i)*(2*((aaa-
1)/(a_points*2))-1)*multiples;
                        a_set(2,a_num) = Sconf(ki,i)*(2*((bbb-
1)/(a_points*2))-1)*multiples;
                        a_num = a_num+1;
                    end
                end
            end

            % calcualte BC cost function prediction
            for ai = 1:size(a_set,1)
                score(ki,ai) = score(ki,ai) + single_opt_run2(horizon,
k_set(:,ki), a_set(1, ai), a_set(2, ai), Aest_path((i-1):i,ki),
Best_path((i-1):i,ki), s_path((i-1):i,ki), y((i-1):i), u(i-1), ysp((i-
1):(i+horizon+1)));
            end

        end

        ypred_pick(i,ki) = score(ki,:)*(a_weight)';
        ypred_pick2(ki) = mean(ypred_pick(max(1,i-k_weight):i,ki));

    end

    if i > start_cycle
        ypred_pick2(next_k3(i)) = ypred_pick2(next_k3(i))*0.999;
    end
    %%%%%%%%%%% End optimisation block %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %  pick next parameters if initialisation time has passed.
    if i > start_cycle
        %ypred_pick(i,:) = ypred_path(i,:);
        [score_best, next_k3(i)] = min(ypred_pick2(:));
    else
        next_k(i) = 1;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % pick next step parameters
    Ka(i) = k_set(1,next_k3(i));
    Kb(i) = k_set(2,next_k3(i));
    Kd(i) = k_set(3,next_k3(i));

    % new estimates for s, A, B
    s(i)      = s_path(i,next_k3(i));
    Aest(:,i) = Aest_path(i,next_k3(i));
    Best(:,i) = Best_path(i,next_k3(i));

    % calulate new output - uses the next set point value - we assume
this
    % is known
    u(i) = (ysp(i+1) - Aest(1,i)*y(i) + (1-Kd(i))*s(i)) / Best(1,i);

    % calulate Lyapunov (for simulation purposes only)
```

```
    V(i) = (Aest(:,i) - A(:,i))'*Ka(i)*(Aest(:,i) - A(:,i)) +
(Best(:,i) - B(:,i))'*Kb(i)*(Best(:,i) - B(:,i)) + s(i)^2;

    %break if runaway condition
    if V(i) > Vmax
        break
    end
end

fprintf('\n');

time3 = toc;

time = (1:i)*1;

% calculate error signals
noise_error3 = noise*rand_set(time)'*rand_set(time)*noise;
sp_error3 = ysp(time)*ysp(time)';
out_error3 = (y(time)-ysp(time)')'*(y(time)-ysp(time)');
```

Appendix D: Code for Display of Results

Creates plots used in chapters 4 and 5. This particular set was used for the linear comparison.

```
figure(5);
subplot(2,1,1)
plot(time, Ka1(time), 'k', time, Kb1(time), 'k:', time, Kd1(time), 'k.-
');
legend('Ka', 'Kb', 'Kd');
title('LMI Track Selection');
ylabel('Parameter Value');
subplot(2,1,2)
plot(time, Ka3(time), 'k', time, Kb3(time), 'k:', time, Kd3(time), 'k.-
');
legend('Ka', 'Kb', 'Kd');
title('BC Track Selection');
ylabel('Parameter Value');
xlabel('Time Interval');

figure(3);
hold on;
plot(time, A(1,time), 'k',time, Aest3(1,time), 'k-.',time,
Aest1(1,time), 'k:',time, Aest0(1,time), 'c', time, Aest4(1,time),
'c');
legend('True Value of A', 'BC estimate', 'LMI estimate', 'Untunded
estimates');
ylabel('Parameter Value');
xlabel('Time Interval');

figure(4);
hold on;
plot(time, B(1,time), 'k',time, Best3(1,time), 'k-.',time,
Best1(1,time), 'k:',time, Best0(1,time), 'c', time, Best4(1,time),
'c');
legend('True Value of B', 'BC estimate', 'LMI estimate', 'Untunded
estimates');
ylabel('Parameter Value');
xlabel('Time Interval');

figure(1);
hold on
plot(time, y3(time), 'k-.',time, y1(time), 'k:',time, y0(time),
'c',time, ysp(time), 'k', time, y4(time), 'c',time, ysp(time),
'k',[1,done],[0,0],'k');
legend('Y - BC','Y - LMI','Y - Untuned','Ysp');
ylabel('System Output');
xlabel('Time Interval');
```

Appendix E: Code for LMI Function Evaluation

This section is called to evaluate the value of γ_{sp} in the LMI tuning method. The C vector used gives an error equal to $y_k - y_{sp}$. A function 'calc_model1' is called to obtain the values for the matrices $\mathbf{A}(\delta_i)$ and $\mathbf{B}$.

```
function tmin = lmi_test(yk, at, ak, bt, bk, ka, kb, kd, sk, rk, alpha)
[Amm, Apm, Amp, App, Bmm, Bpm, Bmp, Bpp] = calc_model1(yk, at, ak, bt,
bk, ka, kb, kd, sk, rk, alpha);

C = [-1,0,0,0,1];
% states are, in order;
% yk, sk, ak, bk, rk

D = [0];

setlmis([]);
r = 1; %inputs
k =1; % outputs
s =max(size(Amm)); % states
% observed = e(k)

P0=lmivar(1,[s 1]);

% positive definite restriction
lmiterm([5 1 1 0],0); %P0 > 0
lmiterm([-5 1 1 P0],1,1);

%vertex 1 of 4 (-delA -delB)
lmiterm([1 1 1 P0],Amm',Amm); % A'PA
lmiterm([1 1 1 P0],-1,1);       %-P
lmiterm([1 1 2 P0],Amm',Bmm); %A'PB
lmiterm([1 1 3 0],C');   % C'
lmiterm([1 2 2 P0],Bmm',Bmm);   %B'PB
lmiterm([1 2 3 0],D');     %D'
lmiterm([1 3 3 0],-eye(k));        %-I

lmiterm([-1 1 1 0],0.000001*eye(s)); % A'PA
lmiterm([-1 2 2 0],eye(r)); % A'PA
lmiterm([-1 3 3 0],0.000001*eye(k)); % A'PA


%vertex 2 of 4 (+delA -delB)
lmiterm([2 1 1 P0],Apm',Apm); % A'PA
lmiterm([2 1 1 P0],-1,1);       %-P
lmiterm([2 1 2 P0],Apm',Bpm); %A'PB
lmiterm([2 1 3 0],C');   % C'
lmiterm([2 2 2 P0],Bpm',Bpm);   %B'PB
lmiterm([2 2 3 0],D');     %D'
lmiterm([2 3 3 0],-eye(k));        %-I
```

```
lmiterm([-2 1 1 0],0.000001*eye(s)); % A'PA
lmiterm([-2 2 2 0],eye(r)); % A'PA
lmiterm([-2 3 3 0],0.000001*eye(k)); % A'PA


%vertex 3 of 4 (-delA +delB)
lmiterm([3 1 1 P0],Amp',Amp); % A'PA
lmiterm([3 1 1 P0],-1,1);       %-P
lmiterm([3 1 2 P0],Amp',Bmp); %A'PB
lmiterm([3 1 3 0],C');   % C'
lmiterm([3 2 2 P0],Bmp',Bmp);   %B'PB
lmiterm([3 2 3 0],D');     %D'
lmiterm([3 3 3 0],-eye(k));        %-I

lmiterm([-3 1 1 0],0.000001*eye(s)); % A'PA
lmiterm([-3 2 2 0],eye(r)); % A'PA
lmiterm([-3 3 3 0],0.000001*eye(k)); % A'PA


%vertex 4 of 4 (+delA +delB)
lmiterm([4 1 1 P0],App',App); % A'PA
lmiterm([4 1 1 P0],-1,1);       %-P
lmiterm([4 1 2 P0],App',Bpp); %A'PB
lmiterm([4 1 3 0],C');   % C'
lmiterm([4 2 2 P0],Bpp',Bpp);   %B'PB
lmiterm([4 2 3 0],D');     %D'
lmiterm([4 3 3 0],-eye(k));        %-I

lmiterm([-4 1 1 0],0.000001*eye(s)); % A'PA
lmiterm([-4 2 2 0],eye(r)); % A'PA
lmiterm([-4 3 3 0],0.000001*eye(k)); % A'PA


lmilio=getlmis;
[tmin,xfeas] = gevp(lmilio,1);

fprintf('tmin: %f, Gamma: %f\n',tmin,sqrt(tmin));
if isempty(tmin)
    tmin = -1;
else
    tmin = sqrt(tmin);
end
```

Appendix F: Code for BC Function Evaluation

This section is called to evaluate the predicted value of the bicriteria error score for the BC tuning method. It is coded as a simulation similar to the main function.

```
function [error] = single_opt_run2(done, k_set, A, B, Ainit, Binit, si,
yi, ui, ysp)
% done is number of steps to execute, 1 will give current conditions
only
% K is thre parameter vector of Ka, Kb, Kd
% A and B are the assumed real values of the parameters
% Ainit and Binit are the inital values of estimates of a and b at i
and
% i-1
% si is 2 vector of s(i-1) and s(i)
% yi is 2 vector of y(i-1) and y(i)
% ui is u(i-1)
% ysp is vector of ysp(i-1:i+done+1)

n = 1;
error = 0;
i = 2;

% Define full sized variables
Aest        = zeros(done+1);
Best        = zeros(done+1);
u           = zeros(done+1);
y           = zeros(done+1);
s           = zeros(done+1);

% set initial values
s(1:2) = si;
y(1:2) = yi;
u(1)   = ui;
Aest(1:2) = Ainit;
Best(1:2) = Binit;
u(2) = (ysp(3) - Aest(2)*y(2)) / Best(2);
feedback = 0;
pred = 0;

for i = (n+2):(done+1)

    % calcualte current y, based on old y and inputs
    y(i) = A*y(i-1) + B*u(i-1);

    % feedback error
    feedback = feedback + (y(i) - ysp(i))^2;

    % prediction error
    pred = pred + (y(i) - Aest(i-1)*y(i-1) + Best(i-1)*u(i-1))^2;

    % calualte parameters for next error update
```

```
    Sa = -y(i-1)/(1 + k_set(3));
    Sb = -u(i-1)/(1 + k_set(3));
    Sf = (2*y(i) + (1 - k_set(3))*s(i-1) - Aest(i-1)'*y(i-1) - Best(i-
1)'*u(i-1))/(1 + k_set(3));
    As = k_set(1)*y(i-1);
    Af = Aest(i-1) + k_set(1)*y(i-1)*s(i-1);
    Bs = k_set(2)*u(i-1);
    Bf = Best(i-1) + k_set(2)*u(i-1)*s(i-1);

    % calualte new error metric
    s(i) = (Af*Sa + Bf*Sb + Sf) / (1 - As*Sa - Bs*Sb);

    % new estimates for A, B
    Aest(i) = s(i)*As + Af;
    Best(i) = s(i)*Bs + Bf;


    % calulate new output - uses the next set point value - we assume
this
    % is known
    u(i) = (ysp(i+1) - Aest(i)*y(i)) / Best(i);

end

error = feedback + pred;
```

Appendix G: Code for LMI Symbolic Jacobian Solver

This section is used to create the function 'calc_model1', which is called from appendix E. Symbolic manipulation is much slower than numerical evaluation of the function, thus this section of code only needs to be executed once.

```
clear all
file = 'calc_model1.m'
f = fopen(file,'wt')
fprintf(f,'function [Amm, Apm, Amp, App, Bmm, Bpm, Bmp, Bpp] =
calc_model1(yk, at, ak, bt, bk, ka, kb, kd, sk, rk, alpha)\n\n');

digits(6);
syms yk xk a at ak b bt bk ka kb kd sk rk ysp alpha dk noise beta;

rk1 = (alpha)*rk + (1-alpha)*ysp;
dk1 = (beta)*dk + (1-beta)*noise;

uk = (-ak*yk + rk + (1-kd)*sk)/bk;

yk1 = a*yk+b*uk;
%yk = xk %+ dk;
ek = yk - rk;

Af = ak + ka*yk*sk;
As = ka*yk;

Bf = bk + kb*uk*sk;
Bs = kb*uk;

Sf = (2*yk1 - ak*yk - bk*uk + (1-kd)*sk)/(1+kd);
Sa = -yk/(1+kd);
Sb = -uk/(1+kd);

sk1 = (Sf + Sa*Af + Sb*Bf)/(1 - Sa*As - Bs*Sb);
ak1 = Af + As*sk1;
bk1 = Bf + Bs*sk1;

A = [yk1,sk1,ak1,bk1, rk1];
y = length(A); % states
fprintf('Vector of [states at k+1]
k\n_____\n');
pretty(A)
Ajs = jacobian(A,[yk,sk,ak,bk, rk]); %dk
Bjs = jacobian(A, ysp);
z = 1; % inputs
Ajs = subs(Ajs,'a','ak-at');
Ajs = subs(Ajs,'b','bk-bt');
%Ajs = subs(Ajs,'xk','yk');

Bjs = subs(Bjs,'a','ak-at');
```

```
Bjs = subs(Bjs,'b','bk-bt');
%Bjs = subs(Bjs,'xk','yk');



fprintf(f,'\n%% -At, -Bt terms\n');

Aj0 = subs(Ajs, 'at', '(-at)');
Aj0 = simplify(subs(Aj0,'bt','(-bt)'));
for m = 1:y
    for n = 1:y
        fprintf(f,'Amm(%s,%s) =
%s;\n',int2str(m),int2str(n),char(Aj0(m,n)));
    end
end
fprintf(f,'\n');
Bj0 = subs(Bjs, 'at', '(-at)');
Bj0 = simplify(subs(Bj0,'bt','(-bt)'));
for m = 1:y
    for n = 1:z
        fprintf(f,'Bmm(%s,%s) =
%s;\n',int2str(m),int2str(n),char(Bj0(m,n)));
    end
end

fprintf(f,'\n%% +At, -Bt terms\n');

Aj0 = subs(Ajs, 'at', '(+at)');
Aj0 = simplify(subs(Aj0,'bt','(-bt)'));
for m = 1:y
    for n = 1:y
        fprintf(f,'Apm(%s,%s) =
%s;\n',int2str(m),int2str(n),char(Aj0(m,n)));
    end
end
fprintf(f,'\n');
Bj0 = subs(Bjs, 'at', '(+at)');
Bj0 = simplify(subs(Bj0,'bt','(-bt)'));
for m = 1:y
    for n = 1:z
        fprintf(f,'Bpm(%s,%s) =
%s;\n',int2str(m),int2str(n),char(Bj0(m,n)));
    end
end

fprintf(f,'\n%% -At, +Bt terms\n');

Aj0 = subs(Ajs, 'at', '(-at)');
Aj0 = simplify(subs(Aj0,'bt','(bt)'));
for m = 1:y
    for n = 1:y
        fprintf(f,'Amp(%s,%s) =
%s;\n',int2str(m),int2str(n),char(Aj0(m,n)));
    end
end
fprintf(f,'\n');
Bj0 = subs(Bjs, 'at', '(-at)');
```

```
Bj0 = simplify(subs(Bj0,'bt','(bt)'));
for m = 1:y
    for n = 1:z
        fprintf(f,'Bmp(%s,%s) =
%s;\n',int2str(m),int2str(n),char(Bj0(m,n)));
    end
end

fprintf(f,'\n%% +At, +Bt terms\n');

Aj0 = subs(Ajs, 'at', 'at');
Aj0 = simplify(subs(Aj0,'bt','bt'));
for m = 1:y
    for n = 1:y
        fprintf(f,'App(%s,%s) =
%s;\n',int2str(m),int2str(n),char(Aj0(m,n)));
    end
end
fprintf(f,'\n');
Bj0 = subs(Bjs, 'at', 'at');
Bj0 = simplify(subs(Bj0,'bt','bt'));
for m = 1:y
    for n = 1:z
        fprintf(f,'Bpp(%s,%s) =
%s;\n',int2str(m),int2str(n),char(Bj0(m,n)));
    end
end

fclose(f)
```

Appendix H: Example of one Element of the Jacobian

In this section one element of one possible combination of uncertainties from the function 'calc_model1' is shown. The full function is not displayed because it is generated from the code in Appendix G, and the full file size is 62 kB of plain text for a first order system with no disturbance, and 387 kB for a second order system.

```
Amm(4,4) = -(kb^2*sk^4*kd^4-2*kb^2*sk^4*kd^3+2*kb^2*sk^4*kd-
2*kb^2*sk^3*rk+2*kb^2*rk^3*sk+kb^2*ak^4*yk^4-
yk^4*ka^2*bk^4+2*bk^2*kb*sk^2-2*bk^4*yk^2*ka+2*kb*rk*yk*bk^2*at-
12*kb^2*ak*yk*rk*sk^2*kd^2+12*kb^2*ak*yk*rk*sk^2*kd+12*kb^2*ak*yk*rk^2*
sk*kd-12*kb^2*ak^2*yk^2*rk*sk*kd-
2*yk^2*ka*bk^2*kb*sk^2*kd+2*yk^2*ka*bk^2*kb*sk*rk-
2*yk^3*ka*bk^2*kb*sk*ak-2*bk^2*kb*ak*yk*sk*kd-2*bk^4*kd*yk^2*ka-
2*bk^2*kb*sk^2*kd^2-bk^4*kd^2+2*bk^2*kb*rk*sk*kd+2*bk^2*kb*sk*rk-
2*bk^2*kb*sk*ak*yk-2*bk^4*kd-
6*kb^2*ak^2*yk^2*sk^2*kd+6*kb^2*ak^2*yk^2*sk*rk+6*kb^2*ak^2*yk^2*rk^2+4
*kb^2*ak^3*yk^3*sk*kd-2*kb^2*ak^3*yk^3*sk-
4*kb^2*ak^3*yk^3*rk+2*yk^2*ka*bk^2*kb*sk^2-6*kb^2*ak*yk*rk^2*sk-
4*kb^2*ak*yk*rk^3+6*kb^2*ak^2*yk^2*sk^2*kd^2-
6*kb^2*sk^3*ak*yk*kd^2+2*kb^2*sk^3*ak*yk+kb^2*rk^4+4*kb^2*ak*yk*sk^3*kd
^3+6*kb^2*sk^3*rk*kd^2+4*bk*bt*kb*ak^2*yk^2+6*kb^2*rk^2*sk^2*kd^2-
6*kb^2*rk^2*sk^2*kd-4*kb^2*rk^3*sk*kd-
4*kb^2*rk*sk^3*kd^3+8*bk*bt*kb*sk*rk-4*bk*bt*kb*sk^2*kd-
4*bk*bt*kb*sk^2*kd^2+4*bk*bt*kd*kb*ak^2*yk^2-8*bk*bt*kd*kb*ak*yk*rk-
8*bk*bt*kb*ak*yk*rk-
8*bk*bt*kb*sk*ak*yk+4*bk*bt*kb*rk^2+4*bk*bt*kb*sk^2-kb^2*sk^4-
2*kb*ak*yk^2*bk^2*at+2*kb*sk*yk*bk^2*at+2*kb*rk*yk*bk^2*at*kd-bk^4-
2*kb*ak*yk^2*bk^2*at*kd-2*kb*sk*kd^2*yk*bk^2*at+6*sk^3*kd*yk*at*kb^2-
6*sk^3*kd^2*yk*at*kb^2+2*sk^3*kd^3*yk*at*kb^2+4*kb*ak^2*yk^4*bk*bt*ka+2
*kb*sk*yk^3*bk^2*at*ka+8*kb*rk*bk*bt*sk*yk^2*ka-
8*kb*rk*bk*bt*sk*kd*yk^2*ka-
2*kb*ak*yk^4*bk^2*at*ka+2*kb*rk*yk^3*bk^2*at*ka+4*kb*bk*bt*sk^2*yk^2*ka
-8*kb*ak*yk^3*bk*bt*rk*ka+4*kb*sk^2*kd^2*bk*bt*yk^2*ka-
8*kb*ak*yk^3*bk*bt*sk*ka+8*kb*ak*yk^3*bk*bt*sk*kd*ka-
8*kb*bk*bt*sk^2*kd*yk^2*ka-
2*kb*sk*kd*yk^3*bk^2*at*ka+4*kb*bk*bt*rk^2*yk^2*ka+2*ak^3*yk^4*at*kb^2-
6*ak^2*yk^3*at*kb^2*sk+6*ak*yk^2*at*kb^2*sk^2-
6*rk*yk^3*at*kb^2*ak^2+6*rk^2*yk^2*at*kb^2*ak+12*rk*yk^2*at*kb^2*sk*ak-
12*rk*yk^2*at*kb^2*ak*sk*kd-2*rk^3*yk*at*kb^2-
6*rk^2*yk*at*kb^2*sk+6*rk^2*yk*at*kb^2*sk*kd-
6*rk*yk*at*kb^2*sk^2+12*rk*yk*at*kb^2*sk^2*kd-
6*rk*yk*at*kb^2*sk^2*kd^2+6*sk*kd*yk^3*at*kb^2*ak^2-
12*sk^2*kd*yk^2*at*kb^2*ak+6*sk^2*kd^2*yk^2*at*kb^2*ak+8*bk*bt*kd^2*kb*
ak*yk*sk+4*bk*bt*kd*kb*rk^2-8*bk*bt*kd^2*kb*rk*sk-
2*sk^3*yk*at*kb^2+4*bk*bt*kd^3*kb*sk^2)/(bk^2+bk^2*kd+yk^2*ka*bk^2+kb*a
k^2*yk^2-2*kb*ak*yk*rk-
2*kb*sk*ak*yk+2*kb*ak*yk*sk*kd+kb*rk^2+2*kb*sk*rk-
2*kb*rk*sk*kd+kb*sk^2-2*kb*sk^2*kd+kb*sk^2*kd^2)^2;
```