2001

Simulation study and instability of adaptive control

Zhongshan Wu
Louisiana State University and Agricultural and Mechanical College, zwu@ece.lsu.edu

SIMULATION STUDY AND INSTABILITY OF ADAPTIVE CONTROL

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

by
Zhongshan Wu
B.S.E.E., Northeastern University, China, 1996
December 2001

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Guoxiang Gu, for his knowledge and support in helping me throughout my research. I appreciate his enlightening guidance and advice in helping me complete this study. Especially his serious attitude on research and his pursuit for the perfect work will help me in a long run.

I also sincerely thank my committee member, Dr. Kemin Zhou and Dr. Peter Wolenski, for their patience and kind support for completing this thesis. Their lectures help me fulfill this research. Dr. Kemin Zhou's humor shows me how to have fun from my research.

This research is supported by Air Force Office for Scientific Research.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF FIGURES v

ABSTRACT vii

CHAPTER 1 INTRODUCTION 1
1.1 Review of Literature 1
1.2 S-function Simulation 3
1.3 Scope of Work 3

CHAPTER 2 SELF-TUNING REGULATORS 4
2.1 Introduction 4
2.1.1 Adaptive Control 4
2.1.2 Self-tuning Regulators (STR) 5
2.2 Estimation Algorithms 6
2.2.1 Process Model 6
2.2.2 Least-squares Estimation Algorithm 7
2.2.3 Projection Algorithm 9
2.3 Control Algorithm 10
2.3.1 A Linear Controller of General Structure 10
2.3.2 Model Following 11
2.3.3 Compatibility Condition 13

CHAPTER 3 SIMULATION OF ADAPTIVE CONTROL SYSTEMS 16
3.1 Introduction to S-function 16
3.1.1 What Is an S-function 16
3.1.2 When to Use an S-function 17
3.1.3 How S-functions Work 17
3.1.4 A Simple Example of S-function 22
3.2 Simulation of RLS Estimator 26
3.2.1 Plant Model and Estimation Algorithm 26
3.2.2 Simulation Experiments 27
3.3 Simulation of MDPP Controller 32
3.3.1 Simulation Steps 32
3.3.2 Solving the Diophantine Equation with Euclid's Algorithm 33
3.3.3 Two Simulation Examples 36

CHAPTER 4 ROBUSTNESS ANALYSIS AND SIMULATION 41
4.1 Frequency Analysis of the Convergent Adaptive Controller 41
4.1.1 Stability Margin 42
4.1.2 Adaptive Controller Under Unmodeled Dynamics 46

4.1.3 Classic Feedback Controller 50
4.2 Noise Contamination 52

REFERENCES 56

APPENDIX SIMULATION PROGRAMS 57

VITA 67

LIST OF FIGURES

Figure 1: Block diagram of an adaptive system 4

Figure 2: Block Diagram of a Self-tuning Regulator 5

Figure 3: A General Linear Controller with Two Degrees of Freedom 10

Figure 4: A S-function Block, Its Dialog Box and the Source M-file 18

Figure 5: How Simulink Performs Simulation 21

Figure 6: 2 Equivalent Simulink Models 22

Figure 7: State-space Model and its equivalent S-function model 23

Figure 8: Estimation Block Diagram 27

Figure 9: S-function Dialog Box of Example 1 29

Figure 10: Simulink Block Diagram of Example 1 29

Figure 11: Simulation Results of Example 30

Figure 12: Simulation Block Diagram of A Second Order System 37

Figure 13: Output of A 2-ord System 37

Figure 14: Output of A 3-ord System 39

Figure 15: Simulation Block Diagram of A Third Order System 40

Figure 16: Step Response of Adaptive Control System 43

Figure 17: Adaptive Control System 44

Figure 18: Adaptive Controller 45

Figure 19: Bode Plot of the Adaptive Control System 45

Figure 20: Adaptive Controller Under Unmodeled Dynamics 47

Figure 21: System Output 47

Figure 22: Frequency Response of Unmodeled Dynamics..... 48

Figure 23: System Output..... 48

Figure 24: Frequency Response of Unmodeled Dynamics..... 49

Figure 25: System Output..... 49

Figure 26: Frequency Response of Unmodeled Dynamics..... 50

Figure 27: Feedback Controller Under Unmodeled Dynamics..... 50

Figure 28: System Output..... 51

Figure 29: System Output..... 51

Figure 30: Type 1 Noise Signal Contamination..... 52

Figure 31: Estimation Under Noise Signal (1)..... 53

Figure 32: Type 2 Noise Signal Contamination..... 54

Figure 33: Estimation Under Noise Signal (2)..... 55

ABSTRACT

The Minimum-degree Pole Placement algorithm for Self-tuning Regulator (STR) design and the Recursive Least-square (RLS) method and the projection algorithm for plant estimation are studied first in this thesis. Simulation studies for the estimator and controller algorithms are mainly undertaken after describing how to use MATLAB S-function in detail. Not only do S-function simulation experiments illustrate how and how well the MDPP and RLS algorithms work, but also show how to write and debug MATLAB codes for S-function programs. The robustness of the adaptive control system is intensively discussed subsequently. By using an estimator resistant to the noise contamination, the adaptive control system can not be destablized by the introduced noise at the input of the plant or the estimator. However, the adaptive control system lacks stability robustness in presence of the unmodeled dynamics that have a magnitude response like an impulse with peak value at the crossover frequency of the system. Simulation results also show that a classic feedback controller has a better performance, compared with the adaptive controller.

CHAPTER 1 INTRODUCTION

1.1 Review of Literature

In common sense, "to adapt" means to change a behavior to conform to new circumstances. Intuitively, an adaptive controller is thus a controller that can modify its behavior in response to the changing dynamics of the process and the character of the disturbances. Since ordinary feedback also attempts to reduce the effects of disturbances and plant uncertainty, the question of the difference between feedback control and adaptive control immediately arised. At an early symposium in 1961 a long discussion ended with the following suggestion: "An adaptive system is any phyisical system that has been designed with an adaptive viewpoint". There is a consensus that a constant-gain feedback system is not an adaptive system.

In the early 1950s, there was extensive research on adaptive control in connection with the design of autopilots for high-performance aircrafts. Such aircraft operated over a wide range of velocity and altitude. It was found that the ordinary constant-gain, linear feedback control could work well in one operating point but not over the whole flight regime. A more sophiscated controller that could work well over a wide range of operating conditions was therefore needed. After much research effort it was found that gain sheduling was a feasible technique for flight control system. The interests in adaptive control diminished partly because the adaptive control problem was too hard to handle using the techniques that were available at the time.

In the 1960s, there was a major development in control theory that contributed to the development of adaptive control. State space and stability theory were introduced. There were also important results in stochastic control theory. Dynamics programming, introduced by Bellman [12], increased the understanding of adaptive processes. Fundamental contribution were also made by Tsypkin [13], who showed that many schemes for learning and adaptive control could be described in a common framework. There were also major developments in system identification. A renaissance of adaptive control ocurred in the 1970s [1], when different estimation schemes were combined with various design methods. Many applications were reported, but theoretical results were very limited.

In the late 1970s and early 1980s, proofs for stability of adaptive systems appeared [3], [4], [6], though under very restricted assumptions. The efforts to merge ideas of robust control and system identification are of particular relevance. Research of the necessity of those assumptions sparked new and interesting research into the robustness of adaptive control, as well as into controllers that are globally stabilizing.

In the late 1980s and 1990s, research [5], [7], [8] gave new insights into the robustness of adaptive controllers. Investigation of nonliearn systems also led to significantly increased understanding of adaptive control.

The theory of robust adaptive control has been well established and understood in 1990s [9], [10], [11]. Essentially, the design of a robust adaptive controller involves appropriate modifications of the conventional adaptive laws. Various modification approaches have been proposed for both the direct model

reference adaptive schemes in [6]-[9] and the indirect schemes in [10]. These include normalization with parameter projection, σ-modification plus normalization, ε_1-modification and the use of deadzone. However, there has not appeared a robust adaptive control algorithm capable of tolerating unmodeled dynamics of reasonable "size".

1.2 S-function Simulation

We use a specifically structured function—S-function of MATLAB in our simulation experiments. S-function can simulate the dynamics of a system, but it is realtively difficult to be used it correctly. Simulation experiments further our understanding of the Self-tuning Regulator and check the stability margin of the adaptive control systems. Detailed usage of S-function simulation is described in Chapter 3.

1.3 Scope of Work

This thesis includes the following parts

1) Introduce the RLS algorithm and projection algorithm, and present the MDPP algorithm.
2) Describe in detail how to use the S-function by examples. S-function program for estimation and controller design prsented in 1) are developed.
3) Analyze the stability of adaptive control system, and give the comparison between adaptive controller and ordinary feedback controller.

CHAPTER 2 SELF-TUNING REGULATORS

2.1 Introduction

2.1.1 Adaptive Control

In common sense, "to adapt" means to change a behavior or characteristic to conform to a new and unknown circumstances. In the sense of control theory and engineering, an adaptive controller is an "intelligent" controller that can modify its behavior in response to the variations in the dynamics of the process and the character of the disturbances. As defined in [1], an adaptive controller is a controller with adjustable parameters and a mechanism for adjusting the parameters. Simply speaking, an adaptive control system consists of two closed loops. One loop is a normal feedback control with the plant and the controller, and the other loop is the parameter adjustment loop, which are shown in the following diagram.

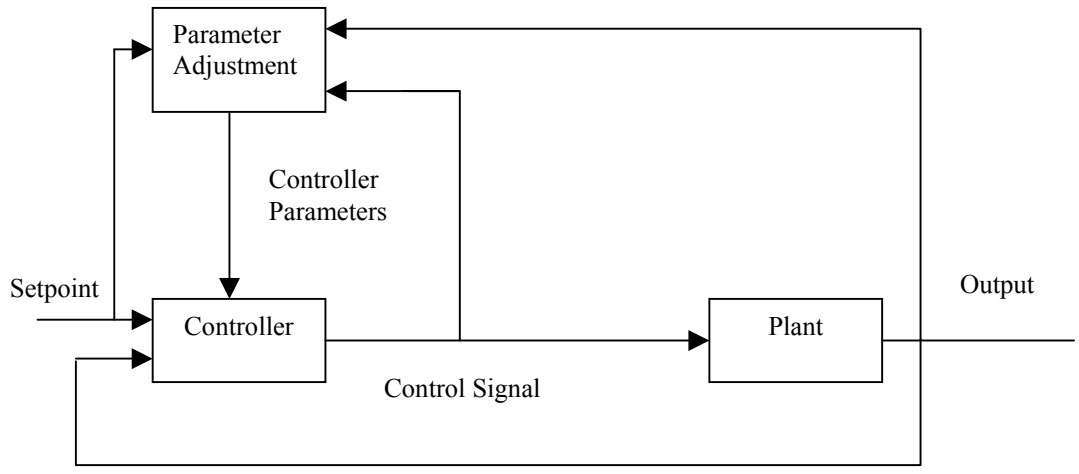


Figure 1: Block diagram of an adaptive system

2.1.2 Self-tuning Regulators (STR)

Usually there are four types of adaptive control schemes: self-tuning regulators, model-reference adaptive control, gain scheduling and dual control. This section focuses on self-tuning regulators. The block diagram of a self-tuning regulator is shown in Fig. 2 as follows.

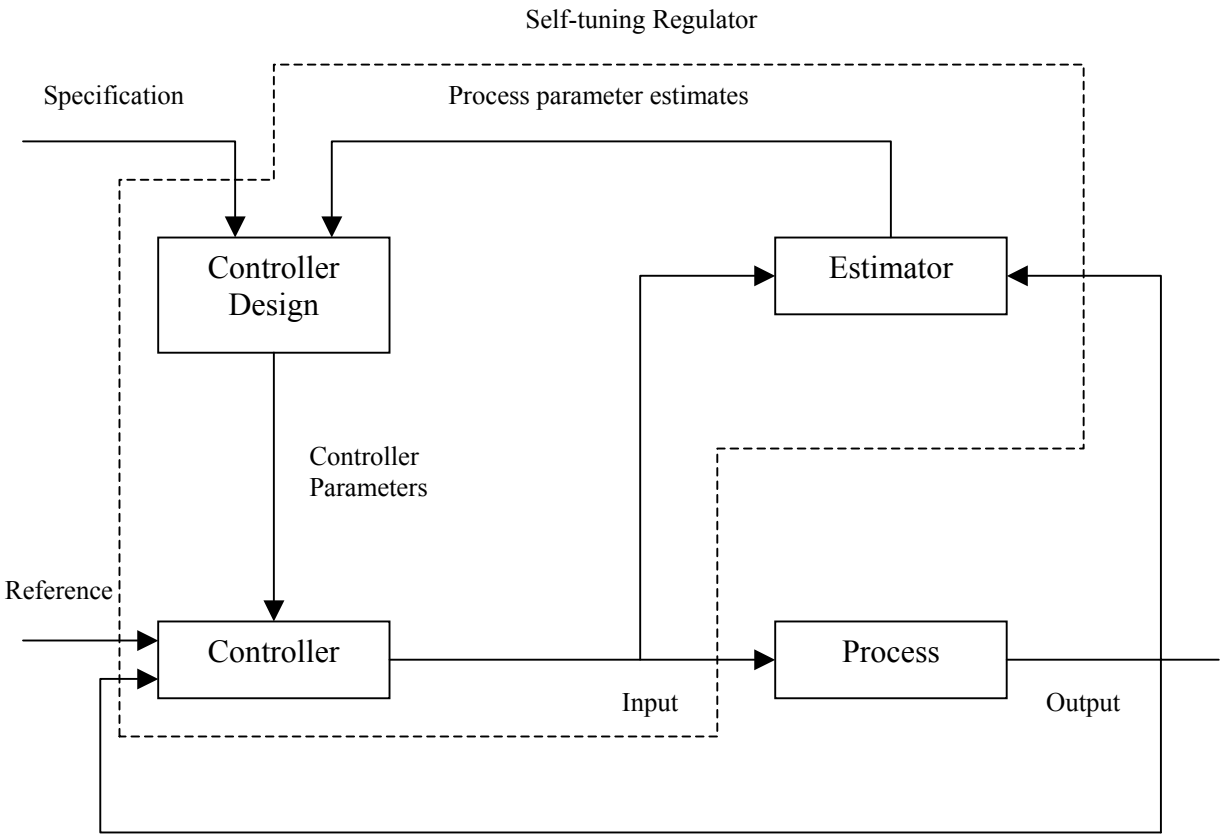


Figure 2: Block Diagram of a Self-tuning Regulator

The block labeled "Estimator" represents an on-line estimation of the process parameters using least-squares or projection algorithms. The block labled "Controller Design" represents an on-line solution to a design problem for a system with known parameters or with estimated parameters. The block labled "Controller" is to

calculate the control action with the controller parameters computed by its proceeding block. The system can be viewed as an automation of processing modeling/estimation and design, in which the process model and the control design are updated at each sampling interval. Sometimes the STR algorithm can be simplified by reparametizing and directly estimating the controller parameters, not the processs parameters alone. It is flexible in that the STR scheme can be implemented by different choices of the underlying design and estimation methods. It is subject to the performance requirement and the practical conditions.

2.2 Estimation Algorithms

It is important to estimate the process parameters on-line in adaptive control. For an adaptive control system, the adaptive mechanism is based on identifying the system first. A self-tuning regulator in Fig. 2 explicitly includes a recursive parameter estimator. Simply speaking, the process parameters estimation is a part of system identification. In a broader sense, system identification is selection of model structure, experiment design, parameter estimation, and validation.

2.2.1 Process Model

It is assumed that the process is described by the single-input, single output (SISO) system

$$A(z^{-1})y(t) = B(z^{-1})(u(t-d_0) + v(t-d_0)) \tag{2.1}$$

where

$$A(z^{-1}) = 1 + a_1 z^{-1} + \ldots + a_n z^{-n}$$

$$B(z^{-1}) = b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}$$

with $m = n - d_0$. In (2.1) y is the output, u is the input of the system, and v is a disturbance. The disturbance can enter the system in many ways. Here it has been assumed that v enters at the process input.

2.2.2 Least-squares Estimation Algorithm

The least-square method is commonly used in system identification. Its principle is that the unknown parameters of a mathematical model should be chosen by minimizing the sum of the square of the difference between the actually observed and the analyically predicted output values with possible weighting that measure the degree of precision. The least-squares criterion is quadratic, so an analytic solution to the least-squares problem exists as long as the measured variable is linear in the unknown parameters. The derivation of the analytic solution is omitted here, since it can be found in books.

In adaptive control system the observations are obtained sequentially in real time. Recursive estimation algorithm is desirable. It saves the computation time by using the results obtained at time $t-1$ to get the estimates at time t. Hence, the recursive least-square (RLS) estimation method is used in this section. The process model (2.1) can be rewritten as

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) - \ldots - a_n y(k-n) + b_0 u(k-d_0) + \ldots b_m u(k-d_0-m) \tag{2.2}$$

The model is linear in the parameters and can be written in the vector form as

$$y(k) = \varphi^T(k)\theta \tag{2.3}$$

where

$$\theta = [b_0, b_1, \ldots b_m, a_1, a_2, \ldots, a_n]^T$$

$$\varphi(k) = [u(k-d_0), \ldots u(k-d_0-m), -y(k-1), \ldots -y(k-n)]^T$$

The recursive least-square estimator is given by

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)[y(k) - \varphi^T(k)\hat{\theta}(k-1)]$$

$$K(k) = P(k-1)\varphi(k)(I + \varphi^T(k)P(k-1)\varphi(k))^{-1}$$

$$P(k) = P(k-1) - P(k-1)\varphi(k)[I + \varphi^T(k)P(k-1)\varphi(k)]^{-1}\varphi^T(k)P(k-1)$$

The RLS algorithm above can be interpreted intuitively. The estimate $\hat{\theta}(k)$ is obtained by adding a weighted prediction error term $y(k) - \varphi^T(k)\hat{\theta}(k-1)$ to the previous estimate $\hat{\theta}(k-1)$. The term $\varphi^T(k)\hat{\theta}(k-1)$ can be viewed as the value of y at time k predicted by the model (2.3) with the previous estimates $\hat{\theta}(k-1)$. The elements of the vector $K(k)$ are weighting factors that tell how the correction and the previous estimates should be combined. The symmetric convariance matrix $P(k)$ is defined by $P(k) = \left(\sum_{i=1}^{k} \varphi(i)\varphi^T(i) \right)^{-1}$ with the initial condition $P(0) = P_0$ positive definite. By this definition, it is easy to see that $P(k) = \left(P_0^{-1} + \sum_{i=1}^{k} \varphi(i)\varphi^T(i) \right)^{-1}$. Notice that $P(k)$ can be made arbitrarily close to $\left(\sum_{i=1}^{k} \varphi(i)\varphi^T(i) \right)^{-1}$ by choosing P_0 sufficiently large. Large P_0 implies poor confidence of the initial estimate. It is a trick in estimating the unknown parameters successfully.

RLS estimation algorithm usually has several modified versions suitable for specific applicaitions. For instance, RLS with exponential forgetting algorithm is designed to estimate the system with slowly time-varing parameters, unlike that we assume the parameter vector θ to be constant in model (2.3). In this pragmatic approach, we simply introduce a time-varying weighting of the data. The latest data is weighted by 1, but the data that is n time units old is weighted by λ^n (λ is called the forgetting factor, $0 < \lambda < 1$.). Its basic idea is to assign the time-varing information with different importance. This algorithm is listed as follows.

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)[y(k) - \varphi^T(k)\hat{\theta}(k-1)]$$

$$K(k) = P(k-1)\varphi(k)(\lambda I + \varphi^T(k)P(k-1)\varphi(k))^{-1}$$

$$P(k) = P(k-1)/\lambda - P(k-1)\varphi(k)[I + \varphi^T(k)P(k-1)\varphi(k)]^{-1}\varphi^T(k)P(k-1)/\lambda \tag{2.4}$$

2.2.3 Projection Algorithm

The RLS algorithm given in (2.4) needs to update the parameter vector θ and the convariance matrix P at each step. For a large size vector θ, the updating of θ and P dominates the computing effort. The projection algorithm presented by Kaczmarz can sidepass refreshing the P and θ at the price of slower convergence. It is an engineering trade-off. The simple projection algorithm is briefly introduced in the following. It is also simulated by using S-function in Chapter 3.

$$\hat{\theta}(k) = \hat{\theta}(k-1) + \frac{\gamma\varphi(k)}{\alpha + \varphi^T(k)\varphi(k)}(y(k) - \varphi^T(k)\hat{\theta}(k-1)),\ \alpha \geq 0,\ 0 < \gamma < 2$$

2.3 Control Algorithm

2.3.1 A Linear Controller of General Structure

The process model is described in (2.1) as

$$A(z^{-1})y(t) = B(z^{-1})(u(t-d_0) + v(t-d_0))$$

Assume that the polynomials $A(z^{-1})$ and $B(z^{-1})$ are co-prime, i.e. they do not have any common factors. Furthermore, $A(z^{-1})$ is monic. That is, that the coefficient of the highest power is unity.

A general linear controller can be described by

$$R(z^{-1})u(t) = T(z^{-1})u_c(t) - S(z^{-1})y(t) \tag{2.5}$$

where $R(z^{-1})$ $S(z^{-1})$, and $T(z^{-1})$ are polynomials in the back shift operator z^{-1}. This controller consists of a feedforward with the transfer operator $\frac{T(z^{-1})}{R(z^{-1})}$ and a feedback with the transfer operator $\frac{S(z^{-1})}{R(z^{-1})}$. It thus has two degrees of freedom. A block diagram of the closed-loop system is illustrated in the following figure 3.

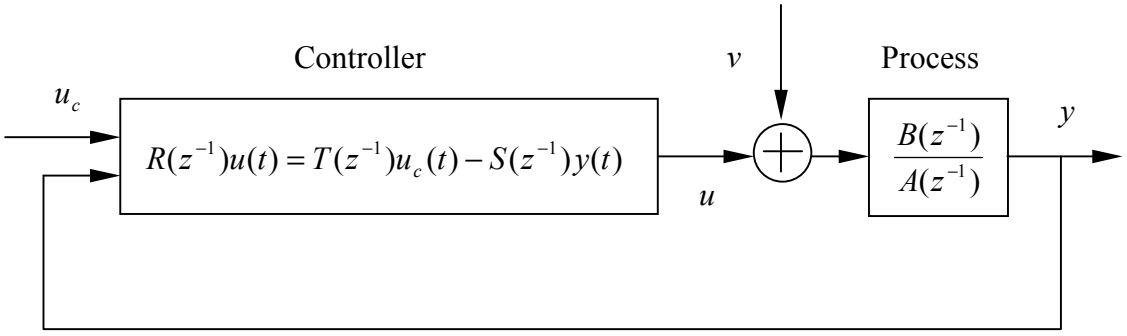


Figure 3: A General Linear Controller with Two Degrees of Freedom

From equations (2.2) and (2.5), we can obtain the following equations for the closed-loop system.

$$y(t) = \frac{B(z)T(z)}{A(z)R(z) + B(z)S(z)} u_c(t) + \frac{B(z)R(z)}{A(z)R(z) + B(z)S(z)} v(t) \tag{2.6}$$

$$u(t) = \frac{A(z)T(z)}{A(z)R(z) + B(z)S(z)} u_c(t) - \frac{B(z)S(z)}{A(z)R(z) + B(z)S(z)} v(t) \tag{2.7}$$

Thus, the closed-loop characteristic polynomial is (for simplicity, the operator z is omitted)

$$A_c = AR + BS \tag{2.8}$$

The key idea of the controller design is to specify the desired closed-loop characteristic polynomial A_c as a design parameter. By solving the Diophantine equation (2.8), the polynomials R and S can be obtained. The closed-loop characteristic polynomial A_c fundamentally determines the property and the performance of the closed system. The Diophantine equation (2.8) always has solutions if the polynomial A and B are co-prime as required. And the solution may be poorly conditioned if the polynomials have factors that are very close. The method to solve the Diophantine equation is presented in the Appendix.

2.3.2 Model Following

The Diophantine equation (2.8) determines only the polynomials R and S. Other conditions must be introduced to calculate the polynomial T in the controller (2.5). To do this, we require that the response from the command signal to the output follow the model

$$A_m y_m(t) = B_m u_c(t) \tag{2.9}$$

From equation (2.6), the following condition

$$\frac{BT}{AR + BS} = \frac{BT}{A_c} = \frac{B_m}{A_m} \tag{2.10}$$

must hold. It then follows from the model-following condition (2.10) that the response of the closed-loop system to command signal is as specified by the model (2.9).

Based on the model-following condition, some constructive conclusions can be deduced. Equation (2.10) implies that there are cancellations of factors of BT and A_c. Factorize the polynomial B as

$$B = B^+ B^- \tag{2.11}$$

where B^+ is a monic polynomial whose zeros are stable and so well damped that they can be canceled by the controller and B^- corresponds to the unstable or poorly damped factors that cannot be canceled. Since B^- remains unchanged, it thus holds that B^- must be a factor of B_m. Therefore

$$B_m = B^- B_m^{'} \tag{2.12}$$

Since B^+ is canceled, it must be a factor of A_c. Furthermore, it follows from equation (2.10) that, A_m is also a factor of A_c. The closed-loop characteristic polynomial A_c thus can be rewritten as

$$A_c = B^+ A_m A_o \tag{2.13}$$

Since A_c and B have the common factor B^+, it follows form equation (2.8) that it must be also a factor of R. Hence

$$R = B^+ R' \tag{2.14}$$

The Diophantine equation (2.8) then can be simplified as

$$AR' + B^- S = A_m A_o = A_c' \tag{2.15}$$

Substituting equation (2.11), (2.12) and (2.13) into equation (2.10), there holds

$$T = B_m' A_o \tag{2.16}$$

2.3.3 Compatibility Condition

To have a control law that is causal in the discrete-time case, we must impose the following conditions upon the polynomials in the control law (2.5).

$$\deg S \le \deg R \tag{2.17}$$

$$\deg T \le \deg R \tag{2.18}$$

In the case of no constraints on the degree of the polynomial, the Diophantine equation (2.8) has many solutions because if R^* and S^* are two specific solutions, then so are

$$R = R^* + MB \tag{2.19}$$

$$S = S^* - MA \tag{2.20}$$

where M is an arbitrary polynomial with any degree. Since there are so many solutions, it is desirable to seek the solution that gives a controller with the lowest degree, i.e. the minimum-degree controller. Given $\deg A > \deg B$, it then follows from equation (2.8) that

$$\deg R = \deg A_c - \deg A \tag{2.21}$$

From equation (2.20), we can always find a solution in which the degree of S is at most $\deg A - 1$. This is defined as the minimum-degree solution to the Diophantine equation (2.8). The condition $\deg S \le \deg R$ thus implies that

$$\deg A_c \ge 2\deg A - 1 \tag{2.22}$$

From equation (2.16), the condition $\deg T \le \deg R$ implies that

$$\deg A_m - \deg B_m \ge \deg A - \deg B = d_0 \tag{2.23}$$

It implies that the time delay of the model must be at least as large as the time delay of the process. It is natural that to get a solution in which the controller has the lowest possible degree. Meanwhile it is reasonable to require that there is no extra delay in the controller. It means that the polynomials R, S and T have the same degrees. Then, we have the following algorithm.

Minimum-degreee Pole Placement (MDPP)

Data: Polynomials A and B.

Specification: Polynomials A_m, B_m and A_o.

Compatibility Conditions:

$\deg A_m = \deg A$

$\deg B_m = \deg B$

$\deg A_o = \deg A - \deg B^+ - 1$

$B_m = B^- B_m^{'}$

Step 1: Decompose B as $B = B^+ B^-$

Step 2: Solve the diophantine equation below to get R' and S with $\deg S < \deg A$.

$$AR' + B^{-}S = A_o A_m$$

Step 3: From $R = B^{+}R'$ and $T = A_0 B_m'$, and compute the control signal from the control law

$$Ru = Tu_c - Sy$$

In this theis, we only consider one special case where no zeros are canceled. Then we have $B^{+} = 1$, $B^{-} = B$, and $B_m = \beta B$,where $\beta = A_m(1)/B(1)$, and $\deg A_o = \deg A - 1$, $T = \beta A_o$. The Diophantine equation in Step 2 becomes

$$AR + BS = A_c = A_o A_m$$

CHAPTER 3 SIMULATION OF ADAPTIVE CONTROL SYSTEMS

In this chapter, we simulate the RLS estimator algorithm and the MDPP controller algorithm by developing the S-function code in MATLAB. S-function is a powerful tool which enables us to add our customrized algorithm block into the Simulink models. We will discuss what the S-function is and how to code with it. We will also give a couple of simulation models for the adaptive control systems by using the S-function later on.

3.1 Introduction to S-function

3.1.1 What Is an S-function

When we create a Simulink model by drawing a block diagram, an S-function is generated with the same name as the model by the Simulink automatically and internally. This S-function is the agent Simulink interacting with for simulation and analysis. Though it is hidden from view, we can call it from the command line like any other MATLAB function. We can sidestep this process by writing an S-function by ourselves. S-functions can be written using MATLAB or C. C language S-functions are compiled as MEX-files (MATLAB Executable files) using the mex utility described in the *Application Program Interface Guide*.

In most basic sense, S-functions are simply MATLAB functions using a special calling syntax that enables us to interact with Simulink's equation solvers. This interaction is very similar to the interaction that takes place between the solvers and built-in Simulink blocks. The form of an S-function is very general and can

accommodate continuous, discrete, and hybrid systems. As a result, nearly all Simulink models can be described by S-functions. S-functions are incorporated into Simulink models by using the S-Function block in the Nonlinear Block sublibrary. We can use the S-Function block's dialog box to specify the name of the underlying S-function, as illustrated in the figure 4.

3.1.2 When to Use an S-function

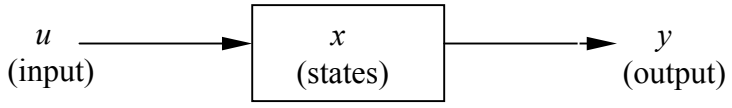
The most common use of S-functions is to create custom Simulink blocks. We can use S-functions for a variety of applications, including:

- Adding new general purpose blocks to Simulink
- Incorporating existing C code into a simulation
- Describing a system as a mathematical set of equations
- Using graphical animations (see the inverted pendulum demo, *penddemo*)

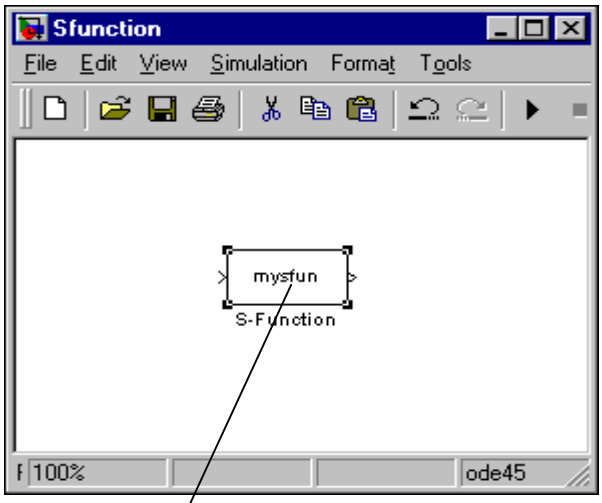
An advantage of using S-functions is that we can build a general purpose block that we can use many times in a model, varying parameters with each instance of the block and integrating with our own analysis and simulation routines.

3.1.3 How S-functions Work

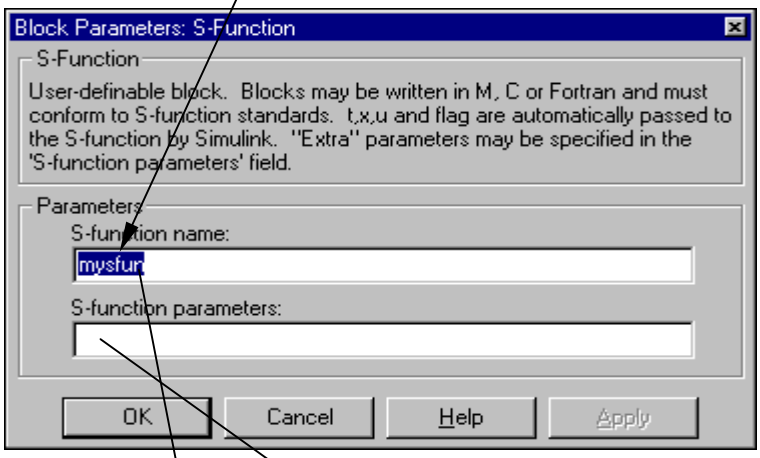
Each block within a Simulink model has the following general characteristrics: a vector of intputs**,** u, a vector of outputs, y, and a vector of states, x, as shown below:



A model that includes a S-function block



S-function Dialog Box



M File

```
%% filename: mysfun.m

function [sys, x0]=mysfun(t, x, u, flag, Ts, Para_ini )

if flag==0
•
•
•
```

Figure 4: A S-function Block, Its Dialog Box and the Source M-file

The state vector may consist of continuous states, discrete states, or a combination of both. The mathematical relationships between the inputs, outputs, and the states are expressed by the following equations:

$$y = f_o(t, x, u) \qquad \text{(output)}$$

$$\dot{x}_c = f_d(t, x, u) \qquad \text{(derivative)}$$

$$x_{d_{k+1}} = f_u(t, x, u) \qquad \text{(update)}$$

where $x = x_c + x_d$

In M-file S-functions, Simulink partitions the state vector into two parts: the continuous states and the dircrete states. The continuous states occupy the first part of the state vector, and the discrete states occupy the second part. For blocks with no states, x is an empty vector. In MEX-file S-functions, there are two separate state vectors for the contnuous and discrete states.

Simulink makes repeated calls during specific stages of simulation to each block in the model, directing it to perform tasks such as computing its outputs, updating its discrete states, or computing its derivatives. Additional calls are made at the beginning and end of a simulation to perform initialization and termination tasks. The figure 5 illustrates how Simulink performs a simulation. First, Simulink initializes the model; this includes initializing each block, and each S-functions. Then Simulink enters the simulation loop, where each pass through the loop is referred to as a simulation step. During each simulation step, Simulink executes the S-function block. This continues until the simulation is complete. Simulink makes repeated calls to S-functions in the model. During these calls, Simulink calls S-

function routines (also called *methods),* which perform tasks required at each stage. These tasks include:

- Initialization — Prior to the first simulation loop, Simulink initializes the S-function. During this stage, Simulink:

⇒ Initializes the SimStruct, a simulation structure that contains information about the S-function.

⇒ Sets the number and size of input and output ports.

⇒ Sets the block sample time(s).

⇒ Allocates storage areas and the sizes array.

- Calculation of next sample hit — If a variable step integration routine is selected, this stage calculates the time of the next variable hit, that is, it calculates the next stepsize.
- Calculation of outputs in the major time step — After this call is complete, all the output ports of the blocks are valid for the current time step.
- Update discrete states in the major time step —In this call, all blocks should perform once-per-time-step activities such as updating discrete states for next time around the simulation loop.
- Integration — This applies to models with continuous states and/or nonsampled zero crossings. If your S-function has continuous states, Simulink calls the output and derivative portions of your S-function at minor time steps. This is so Simulink can compute the state(s) for your S-function. If your S-function (C MEX only) has nonsampled zero crossings, then Simulink will call the output

and zero crossings portion of your S-function at minor time steps, so that it can locate the zero crossings.

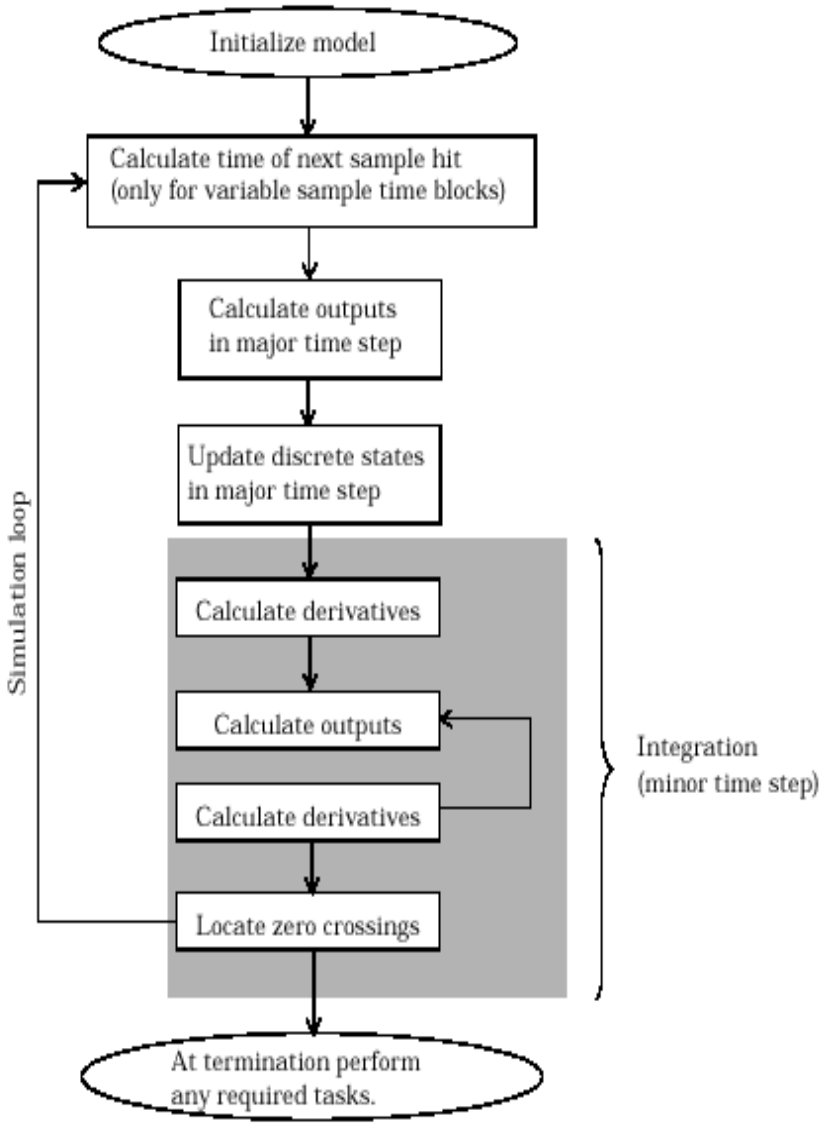


Figure 5: How Simulink Performs Simulation

3.1.4 A Simple Example of S-function

Consider a single-input, two-output set of state-space equations

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

where

$$A = \begin{bmatrix} -0.3 & 0 & 0 \\ 2.9 & -0.62 & -2.3 \\ 0 & 2.3 & 0 \end{bmatrix};\quad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix};\quad C = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -3 & 1 \end{bmatrix};\quad D = \begin{bmatrix} 0 \\ 1 \end{bmatrix};\quad x0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

This can be represented both as an Simulink model including an state-space block and a model including an S-function block, repectively in the next figure.

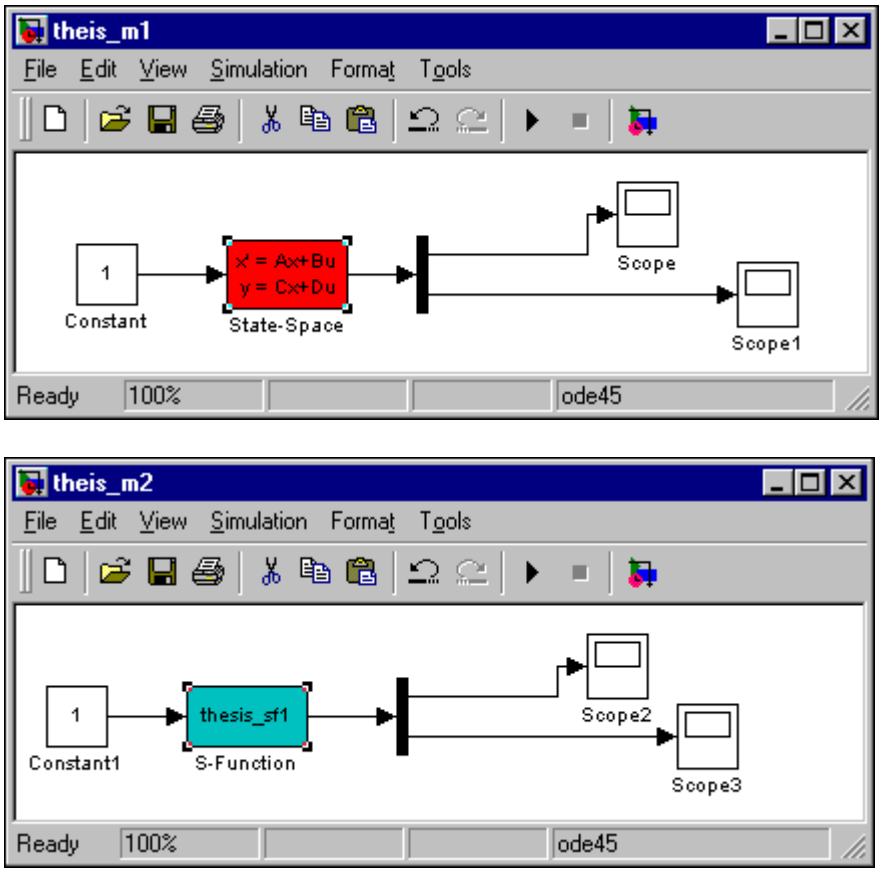


Figure 6: 2 Equivalent Simulink Models

The simulation results demostrate that the lower model in Figure 6, which includes a s-function block, work as well as the upper Simulink model in Figure 5.

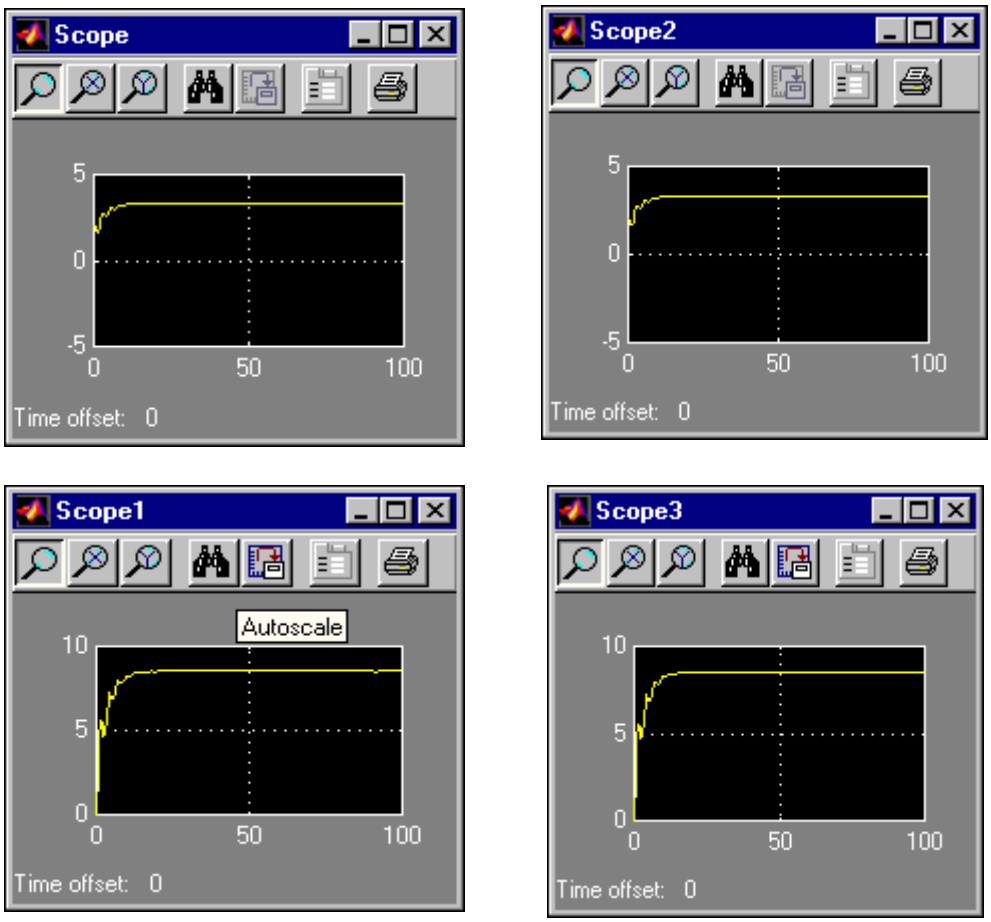


Figure 7: State-space Model and its equivalent S-function model

As the first example, we list the M-file code in the following. (other codes will be attached in the Appendix.)

```
function [sys,x0,str,ts]=thesis1(t,x,u,flag,A,B,C,D)
% Simulink requires that the output parameters, sys,x0, str and ts be placed in the
% order given. sys is a generic return argument, and its value could be the sizes of
```

% parameters, the state derivatives or the S-function output, depending on the *flag*

% options. For example, for *flag*=3, *sys* contains the S-function outputs.

% *x0* is the initial state value (an empty vector if there are no states in the system). *x0*

% is neglected, except when *flag*=0.

% *str* is reserved for future use. S-functions must set this to the empty matrix, [].

% *ts* is a two column matrix containing the sample times and offsets of the block.

% thesis1 is the S-function name.

% The first four inputs parameters, which Simulink passes to the S-function, must be

% the variables *t, x, u* and *flag*. *t*, *x* and *u* are the current time, current state vector

% and current input vector respectively. *flag* is the parameter that controls the

% S-function routines at each simulation stage.

% *A, B, C* and *D* are the additional input parameters of the S-function. They could be

% inputted in the dialog box of S-function block as shown in Figure 3.

switch flag,

% *flag* could have value of 0, 1, 2, 3, 4, and 9. Different values determine distinct

% routines of S-function at each simulation stage. The *flag* options available in

% Simulink are listed in the table 1.

case 0

[m,n]=size(D);

% m is the number of outputs, n is the number of inputs;

sys=[length(A),0,m,n,0,any(D~=0)];

% For a flag=0 call, sys contains the following information vital to simulation.

S-function Routine	Description
flag=1	Calculates the derivatives of the continuous state variables.
flag=2	Updates discrete states.
flag=3	Calculates the outputs of the S-function.
flag=4	Calculates the next sample hit for a discrete update.
flag=9	Performs any necessary end of simulation tasks.

Table 1: S-function Routines and Descriptions

sys(1) *sys*(2)	Number of Continuous States. Number of Discrete States.
sys(3)	Number of Inputs.
sys(4)	Number of Outputs.
sys(5)	Number of Sample times.
sys(6)	Flag for direct feedthrough.

```
  x0=[1;1;1];
```

% The initial state value.

```
case 1

  sys=A*x+B*u;
```

% Return the states derivatives, xDOT.

```
case 3

  sys=C*x+D*u;
```

% Return system output, y.

```
otherwise
  sys=[];
% In this example, no need to return anything, since this is a continuous system. It
% does not apply to other cases.
end
```

3.2 Simulation of RLS Estimator

The RLS estimator presented in Section 2.2.2 is simulated by using S-function under Simulink here. We include an S-function block defined by an M-file S-function code into an Simulink model, and excite the plant to be estimated with 1 Hz square wave. A couple of typical examples illustrate how we program the code and set up the additional input parameters of the S-function in the dialog box, and demonstrate the validity of the RLS estimation algorithm.

3.2.1 Plant Model and Estimation Algorithm

The plant to be estimated is in the general form as below

$$G_p(z^{-1}) = \frac{z^{-d}N(z^{-1})}{D(z^{-1})} \tag{3.1}$$

where

d is the time delay, $n \geq m + d$.

$$N(z^{-1}) = \beta_0 + \beta_1 z^{-1} + \ldots + \beta_m z^{-m}, \quad \deg[N(z^{-1})] = m .$$

$$D(z^{-1}) = 1 + \alpha_1 z^{-1} + \ldots + \alpha_n z^{-n}, \quad \deg[D(z^{-1})] = n$$

The estimation block diagram is shown in the following figure.

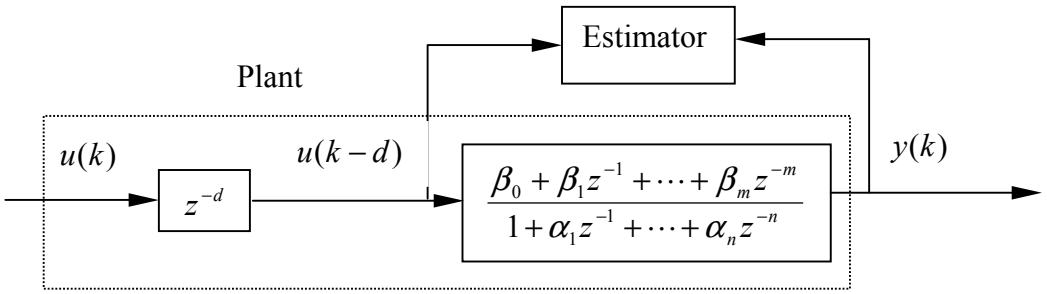


Figure 8: Estimation Block Diagram

Assuming that $u(k)$ and $y(k)$ are the input and the output of the plant, respectively we can write the plant model as below

$$y(k) = \beta_0 u(k-d) + \cdots + \beta_m u(k-d-m) - \alpha_1 y(k-1) - \cdots - \alpha_n y(k-n) \tag{3.2}$$

or in the form of vector

$$y(k) = \varphi^T(k)\theta(k) \tag{3.3}$$

where

$$\varphi(k) = [u(k-d), u(k-d-1), \cdots u(k-d-m), -y(k-1), -y(k-2), \cdots - y(k-n)]^T \in R^{(n+m+1)\times 1}$$

$$\theta(k) = [\beta_0, \beta_1, \cdots \beta_m, \alpha_1, \alpha_2, \cdots \alpha_n]^T \in R^{(n+m+1)\times 1}$$

The estimation algorithm is the same as the estimator presented in section 2.2.2.

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)[y(k) - \varphi^T(k)\hat{\theta}(k-1)]$$

$$K(k) = P(k-1)\varphi(k)(I + \varphi^T(k)P(k-1)\varphi(k))^{-1}$$

$$P(k) = P(k-1) - P(k-1)\varphi(k)[I + \varphi^T(k)P(k-1)\varphi(k)]^{-1}\varphi(k)P(k-1)$$

3.2.2 Simulation Experiments

In simulation, we use a data structure of matrix form as shown below. (Referring to the MATLAB code in Appendix.)

$$\begin{array}{c|ccc|c}
\beta_0(t) & p_{11}(t) & \cdots & p_{1(n+m+1)}(t) & u(t-d) \\
\beta_1(t) & p_{21}(t) & \cdots & p_{2(n+m+1)}(t) & u(t-d-1) \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\beta_m(t) & \vdots & \ddots & \vdots & u(t-d-m) \\
\alpha_1(t) & \vdots & \ddots & \vdots & -y(t-1) \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_n(t) & p_{(n+m+1)1}(t) & \cdots & p_{(n+m+1)(n+m+1)}(t) & -y(t-2) \\
\hline
\theta(t) & & P(t) & & \varphi(t)
\end{array}$$

Only three parameters d, m and n are needed to estimate the unknown parameters.

1. $G_p(z) = \dfrac{0.1065z + 0.0902}{z^2 - 1.6065z + 0.6065}$;

$G_p(z)$ could be rewritten as $G_p(z) = G_p(z^{-1}) = z^{-1}\dfrac{0.1065 + 0.0902z^{-1}}{1 - 1.6065z^{-1} + 0.6065z^{-2}}$. Then $d = 1$, $m = 1$ and $n = 2$. As the first example in this section, we show the Simulink block diagram in Figure 9, the S-function dialog box in Figure 10 and the experiment results displayed in the scopes of Figure 11..

2. $G_p(z^{-1}) = \dfrac{1}{1 - 0.6z^{-1} - 0.81z^{-2} + 0.67z^{-3} - 0.12z^{-4}}$

It is easy to see that $d = 0$, $m = 0$ and $n = 4$. Totally 5 unknown parameters need to be estimated.

Parameters	β_0	α_1	α_2	α_3	α_4
True Value	**1**	**-0.6**	**-0.81**	**0.67**	**-0.12**
Time=3s	1	-0.6	-0.81	0.67	-0.12

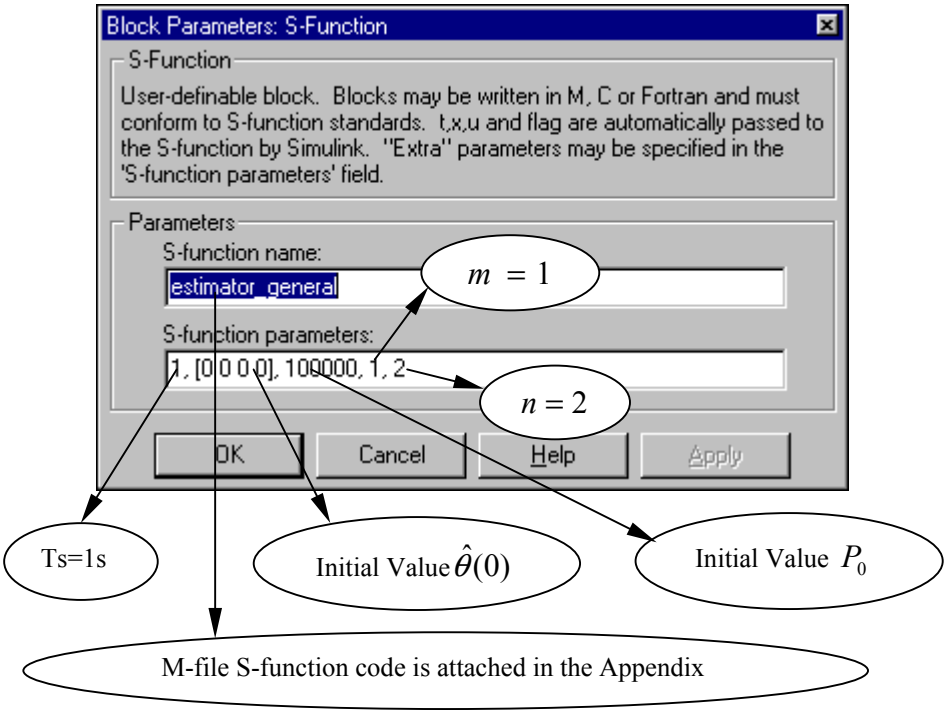


Figure 9: S-function Dialog Box of Example 1

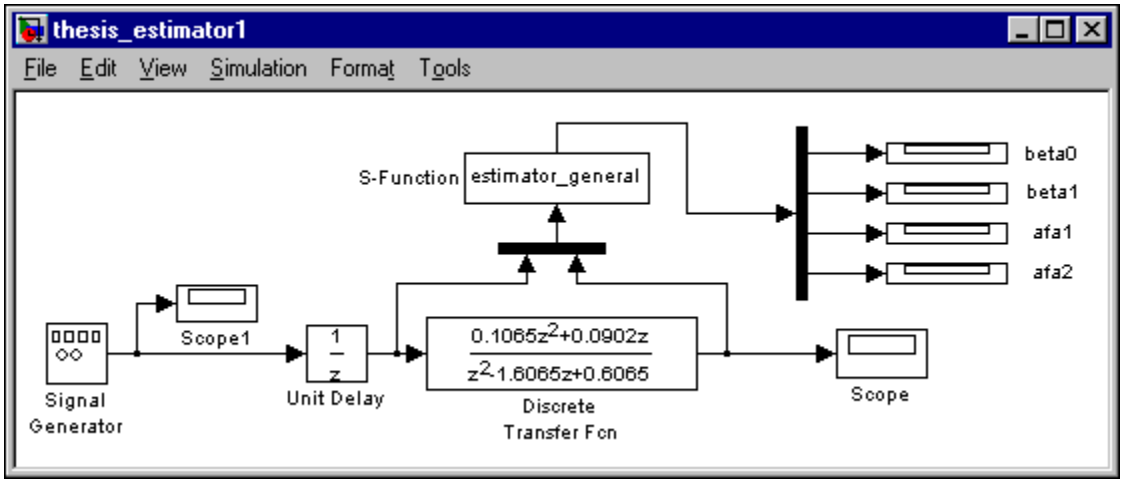


Figure 10: Simulink Block Diagram of Example 1

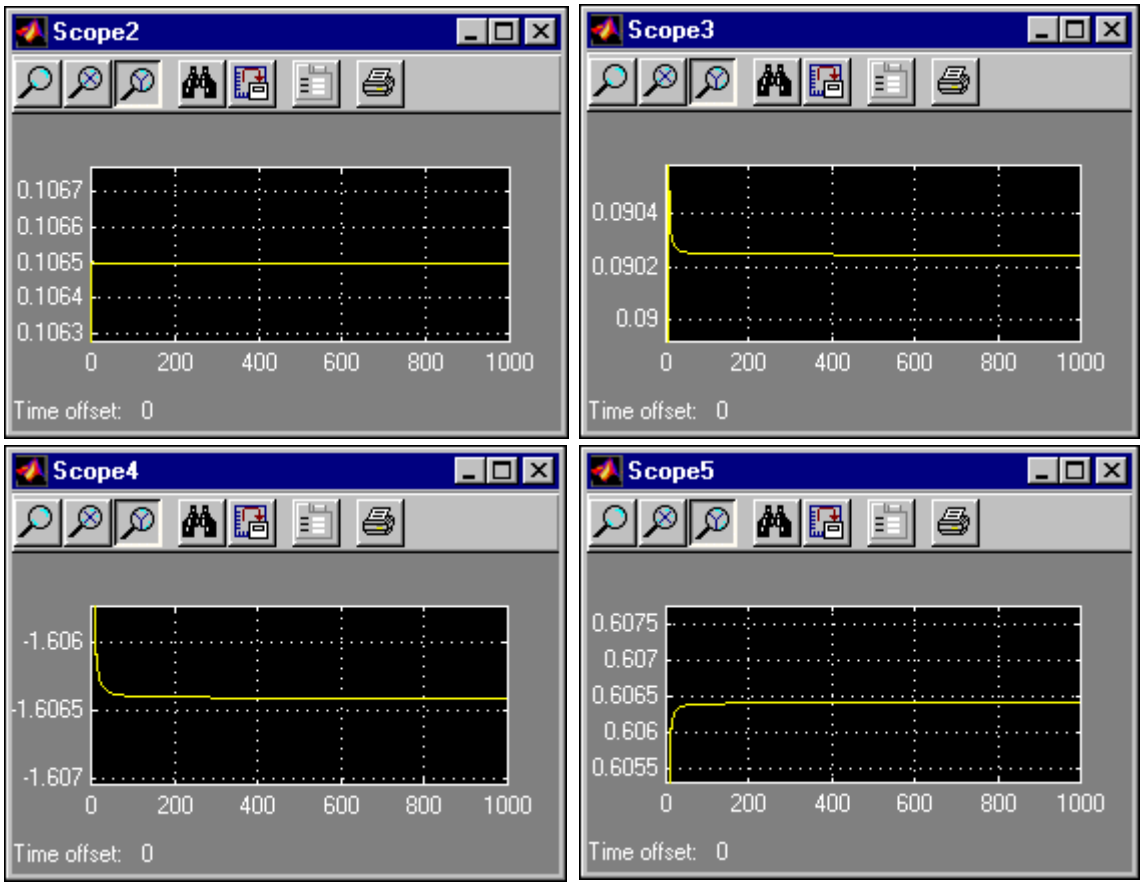


Figure 11: Simulation Results of Example

3. $G_p(z) = \dfrac{z^2 + 1.2z + 0.27}{z^3 - 1.1z^2 + 0.09z + 0.445}$;

$G_p(z)$ could be rewritten as $G_p(z) = G_p(z^{-1}) = z^{-1} \dfrac{1 + 1.2z^{-1} + 0.27z^{-2}}{1 - 1.1z^{-1} + 0.09z^{-2} + 0.445z^{-3}}$.

It holds that $d = 1$, $m = 2$ and $n = 3$. Totally 6 unknown parameters need to be estimated.

Parameters	β_0	β_1	β_2	α_1	α_2	α_3
True Value	**1**	**1.2**	**0.27**	**-1.1**	**0.09**	**0.445**
Time=50s	1	1.991	0.2694	-1.1009	0.0915	0.4442
Time=500s	1	1.991	0.2694	-1.1009	0.0915	0.4442

4. $G_p(z) = \dfrac{0.3769z + 0.2642}{z^2 - 1.3679z + 0.3679}$

$G_p(z)$ could be rewritten as $G_p(z) = G_p(z^{-1}) = z^{-1}\dfrac{0.3679 + 0.2649z^{-1}}{1 - 1.3679z^{-1} + 0.3679z^{-2}}$.

Similarly we find that $d = 1$, $m = 1$ and $n = 2$ and totally 4 unknown parameters need to be estimated. In this example, however, we use the projection algorithm to estimate the parameters.

Parameters	β_0	β_1	α_1	α_2
True Value	**0.3679**	**0.2642**	**-1.3679**	**0.3679**
Time=100s	0.3678	0.2675	-1.3645	0.3645
Time=500s	0.3678	0.2675	-1.3648	0.3648

3.3 Simulation of MDPP Controller

The MDPP control law presented in 2.3.4 is simulated by using S-function under the Simulink in this section. We program 3 S-functions in M-file to estimate the unknow process parameters, to calculate the controller parameters and to implement the control law. The S-functions are programed in an open way so that it applies to a general process model. Given the degree of the polynomials of the process model and the reference model parameters, the system will be simulated automatically. We only need pay attention to the selection of some intial values of the unknown parameters. A second order and third order process are chosen to illustrate the simulation procedure. The method to solve the Diophantine equation is also discussed.

3.3.1 Simulation Steps

Data: Give the reference model in the form of a desired closed-loop pulse transfer operator B_m / A_m and a desired polynomial A_o.

Step 1: Estimate the coefficients of the polynomials A and B in equation (2.1) using the RLS method given in 2.2.2**.**

Step 2: Using the polynomials A and B estimated in step 1, apply the MDPP method presented in 2.3.4 . The polynomials R, S and T of the controller are then obtained by solving the Diophantine equation (2.7).

Step 3: Compute the control action from equation (2.4), that is

$$Ru(t) = Tu_c(t) - Sy(t)$$

Repeat steps and 3 at each sampling period.

3.3.2 Solving the Diophantine Equation with Euclid's Algorithm

In order to compute the control law, we need to solve the following Diophantine equation

$$AR + BS = A_c \tag{3.4}$$

The equation is linear in the polynomial of R and S. A solution to the equation existes if A and B are coprime. However, the equation has many solutions. For example, assuming that R^* and S^* are solutions, then $R = R^* + BW$ and $S = S^* - AW$ are also solutions, where W is an arbitrary polynomial. A specified solution can be achieved by imposing some constraints on the general solutions. Since a controller must be causal, the constraint condition $\deg S \le \deg R$ must hold. The condition will restrict the number of solutions significantly. Here, we adopt Euclid's algorithm to solve the equation.

This algorithm finds the greatest common divisor D of two polynomials A and B. If one of the polynomials, say , is zero, then is equal to A. If this is not the case, the algorithm follows. Let $A_0 = A$ and $B_0 = B$ and iterate the equations

$$A_{n+1} = B_n$$

$$B_{n+1} = A_n \bmod B_n$$

until $B_{n+1} = 0$. The greatest common divisor is then $D = B_n$. Similar to the case that when A and B are numbers, $A \bmod B$ means the reminder when A is divided by B when A and B are polynomials. Bachtracking, we find that D can be expressed as

$$AX + BY = D \tag{3.5}$$

where the polynomials X and Y can be found by keeping track of $A_n \operatorname{div} B_n$ in Euclid's algorithm. This establishes the link between Euclid's algorithm and the Diophantine equation. The extended Euclidean algorithm gives a convenient way to determine X and Y as well as the minimum-degree solutions U and V to

$$AU + BV = 0 \tag{3.6}$$

we can rewrite equation and as

$$F\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} X & Y \\ U & V \end{bmatrix}\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} D \\ 0 \end{bmatrix} \tag{3.7}$$

The matrix F can thus be viewed as the matrix, which performs row operations on $\begin{bmatrix} A & B \end{bmatrix}^T$ to give $\begin{bmatrix} D & 0 \end{bmatrix}^T$. A convenient way to find is to observe that

$$\begin{bmatrix} X & Y \\ U & V \end{bmatrix}\begin{bmatrix} A & 1 & 0 \\ B & 0 & 1 \end{bmatrix} = \begin{bmatrix} D & X & Y \\ 0 & U & V \end{bmatrix}$$

The extended Euclidean algorithm can be expressed as follows: start with the matrix

$$M = \begin{bmatrix} A & 1 & 0 \\ B & 0 & 1 \end{bmatrix}$$

If we assume that $\deg A \geq \deg B$, then calculate $Q = A \operatorname{div} B$, multiply the second row of M by Q , an subtract from the first row. Then apply the same procedure to the second row and repeat until the following matrix is obtained.

$$\begin{bmatrix} D & X & Y \\ 0 & U & V \end{bmatrix}$$

By using the extended Euclidean algorithm it is now straightforward to solve the Diophantine equation (3.4) $AR + BS = A_c$.

This is done as follows: Determine the greatest common divisor D and the associated polynomials X, Y, U and V using the extremed Euclidean algorithm. To have a solution to equation (3.4), D must divide A_c. A pariticular solution is given by

$$R^* = XA_c \operatorname{div} D$$

$$S^* = YA_c \operatorname{div} D$$

and the general solution is

$$R = R^* + WU$$

$$S = S^* + WV$$

where W is an arbitrary polynomial. The minimum-degree solution is obtained by choosing $W = -S^* \operatorname{div} V$. This implies that $S = S^* \bmod V$.

By equating coefficients of equal order , the Diophantine equation given by equation (3.4) can be written as a set of linear equations:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & b_1 & 0 & \cdots & 0 \\ a_1 & 1 & \ddots & \vdots & b_2 & b_1 & \ddots & \vdots \\ a_2 & a_1 & \ddots & 0 & b_3 & b_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & 1 & \vdots & \vdots & \ddots & 0 \\ a_{n-1} & \vdots & \ddots & a_1 & b_n & \vdots & \ddots & b_1 \\ a_n & a_{n-1} & \ddots & a_2 & 0 & b_n & \ddots & b_2 \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_n & 0 & \cdots & 0 & b_n \end{bmatrix} \begin{bmatrix} r_1 \\ \vdots \\ r_{n-1} \\ s_0 \\ \vdots \\ s_{n-1} \end{bmatrix} = \begin{bmatrix} a_{c,1} - a_1 \\ \vdots \\ a_{c,n} - a_n \\ a_{c,n+1} \\ \vdots \\ a_{c,2n-1} \end{bmatrix}$$

If the time delay of the plant is d, then $b_0 = b_1 = b_{d-1} = 0$. The matrix on the left-hand side is called the Sylverster matrix. It occurs frequently in applied mathematics.

It has the property that it is nonsingular if and only if the polynomials A and B do not have any common factors.

3.3.3 Two Simulation Examples

For the model of Example 4 in section 3.2.2 $G_p(z^{-1}) = \dfrac{0.3769z^{-1} + 0.2642z^{-2}}{1 - 1.3679z^{-1} + 0.3679z^{-2}}$.

We specify the reference model as B_m / A_m, where

$$A_m = z^2 - 1.3205z + 0.4966;$$

$$\beta = \frac{1 + a_{m1} + a_{m2}}{b_0 + b_1} = \frac{1 - 1.3205 + 0.4966}{0.3769 + 0.2642} = 0.2786;$$

$$B_m = \beta B = 0.2786 * (0.3769z + 0.2642);$$

$$A_o = z + 0.8;$$

Following the simulation steps in section 3.3.1, we solve the Diophantine

$$(z^2 - 1.3679z + 0.3679)R + (0.3769z + 0.2642)S = (z + 0.8)(z^2 - 1.3205z + 0.4966)$$

and thus obtain the polynomials R, S and T as follows

$$R(z) = z + 0.8042$$

$$S(z) = 0.1173z + 0.3842$$

$$T(z) = 0.2786z + 0.2229$$

Finally, we obtain the control law

$$u(t) = -0.8042u(t-1) + 0.2786u_c(t) + 0.2229u_c(t-1) - 0.1173y(t) - 0.3842y(t-1)$$

The simulation process is illustrated by the following block diagram Figure 11. In this diagram, the S-function block "estimator" estimates the process parameters, i.e. Step 1; the S-function block "contr_calc" is to solve the Diophantine equation and to

get the polynomials R, S and T, i.e. Step 2; the S-function block "controller" computes the control law.

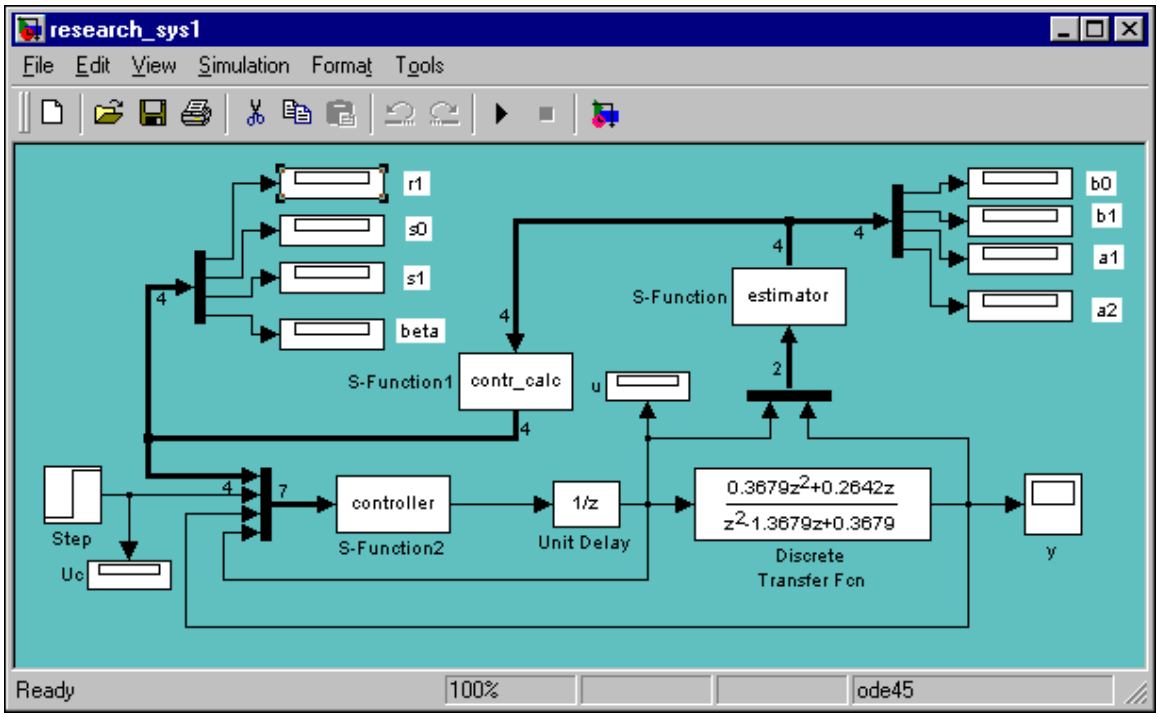


Figure 12: Simulation Block Diagram of A Second Order System

The output of the system is shown in the following Figure 13.

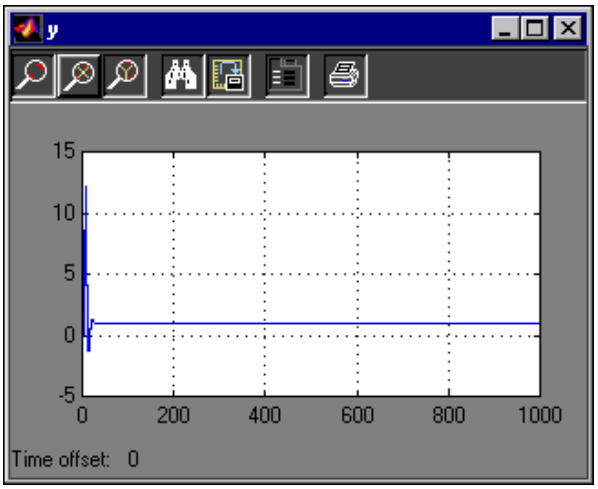


Figure 13: Output of A 2-ord System

$G_p(z^{-1}) = z^{-1}\dfrac{1+1.2z^{-1}+0.27z^{-2}}{1-1.1z^{-1}+0.09z^{-2}+0.445z^{-3}}$ is the model of Example 3 in section 3.2.2. We specify the reference model as B_m / A_m, where

$$A_m = [z-(0.6+0.4j)][z-(0.6-0.4j)](z+0.2)$$

$$= z^3 - z^2 + 0.28z + 0.104$$

$$\beta = \frac{A_m(1)}{B(1)} = \frac{1-1+0.28+0.104}{1+1.2+0.27} \approx 0.1555$$

$$B_m = \beta B = 0.1555z^2 + 0.1866z + 0.0420$$

$$A_o = (z+0.4)(z-0.8) = z^2 - 0.4z - 0.32$$

Following the simulation steps in section 3.3.1, we need to solve the Diophantine equation

$$(z^3 - 1.1z^2 + 0.09z + 0.445)R(z) + (z^2 + 1.2z + 0.27)S(z)$$

$$= (z^2 - 0.4z - 0.32)(z^3 - z^2 + 0.28z + 0.104)$$

and we get the polynomials R, S and T as follows

$$R(z) = z^2 - 0.6419z - 0.2460$$

$$S(z) = 0.3419z^2 - 0.6003z + 0.2822$$

$$T(z) = 0.1555z^2 - 0.0622z - 0.0498$$

Finally we get the control law in the vector form as

$$u(t) = [0.1555, -0.0622, -0.0498, 0.3419, -0.6003, 0.2822, -0.6419, -0.2460] \begin{bmatrix} u_c(t) \\ u_c(t-1) \\ u_c(t-2) \\ -y(t) \\ -y(t-1) \\ -y(t-2) \\ -u(t-1) \\ -u(t-2) \end{bmatrix}$$

The output of the simulated systemand the simulation diagram are shown in the following figures respectively.

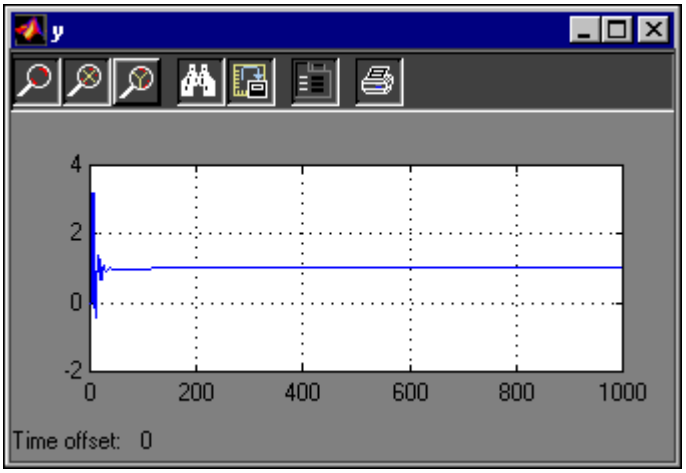


Figure 14: Output of A 3-ord System

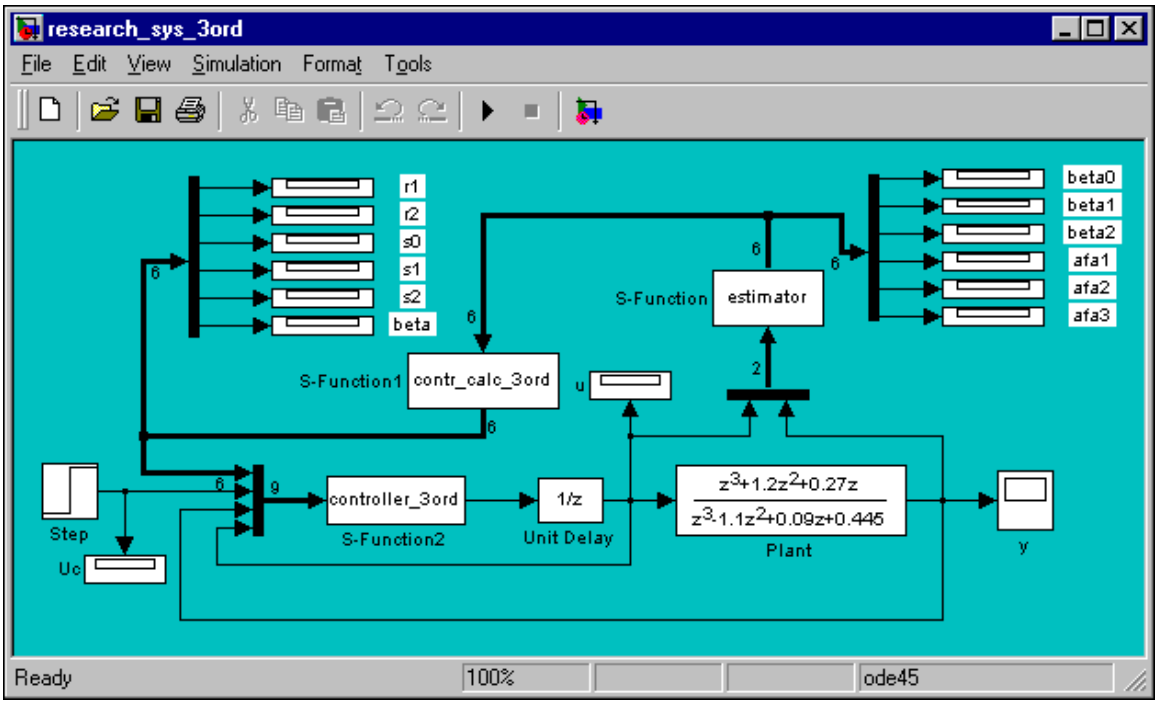


Figure 15: Simulation Block Diagram of A Third Order System

CHAPTER 4 ROBUSTNESS ANALYSIS AND SIMULATION

In previous sections, we mainly used S-function to implement MDPP adaptive algorithm. And we show that adaptive control can be very useful and can give good closed-loop performance. It is attributed to the adpative behavior of the controller that it changes its paramtets, not the structer, according to the changing dynmacis of the system. However, that does not mean that adaptive control is the universal tool that should always be used. How about its robustness property? Many papers examine the robustness of existing adaptive algorithms to unmodeled dynamics and disturbance. The adpative controller itself is able to adjust its parameters to the varying environments adaptively. In this sense, the adaptive controller has the robustness to some degree. But the design guideline of adaptive controller is extremely different from the idea of robust controller design. Charles E. Rohrs's paper [4] *robustness of continuous-time adaptive control algorithms in the presence of unmodeled dynamics* spurred much discussion in the adaptive control community in the past years. That is the motivation for us to study the robustness in this chapter.

4.1 Frequency Analysis of the Convergent Adaptive Controller

In this section, Bode plot technique is utilized to analyze the stability margin of the system. We introduce unmodeled dynamics with peak value at the crossover frequency of the system and unity magnitude in the low frequency. The simulation

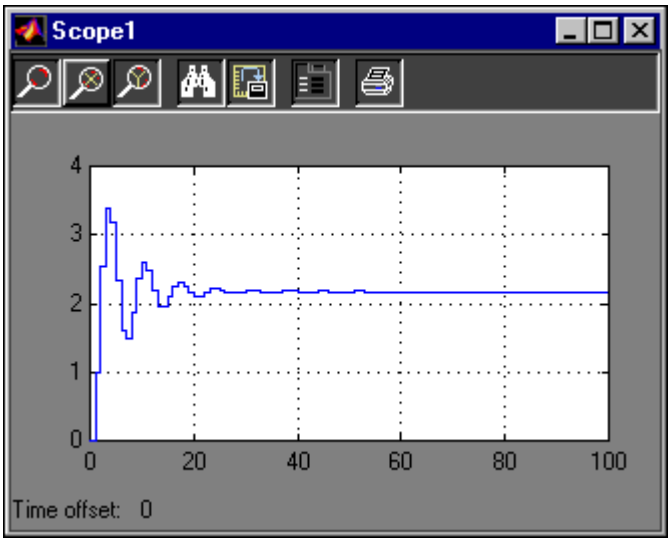
results show that the adpative controller failes to cope with some unmodeled dynamics.

4.1.1 Stability Margin

We choose the original plant model as

$$G_p(z) = \frac{z + 0.5}{z^2 - 1.0344z + 0.7241}$$

This model has a pair of complex poles at 0.5172+j0.6757 and 0.5172-j0.6757 and a zero at -0.5. It has slow response and large overshot as shown below.



In order for the feedback system to have zero tracking error for the step input, and have zero response at high frequency, we add one more pole at 1, and one more zero at –1. Thus, the overall plant model will be

$$G_p(z) = \frac{z^2 + 1.5z + 0.5}{z^3 - 2.0344z + 1.7585z - 0.7241} \tag{4.1}$$

For this composite plant model, we use MDPP algorithm to design an adaptive controller. The desired pole location for the closed-loop model is $0.2 + j0.4$,

$0.2 - j0.4$, -0.75, $-0.5172 + 0.2069j$ and $-0.5172 - 0.2069j$. This model have better step response (shorter setting time and smaller overshoot.). It is illustrated below.

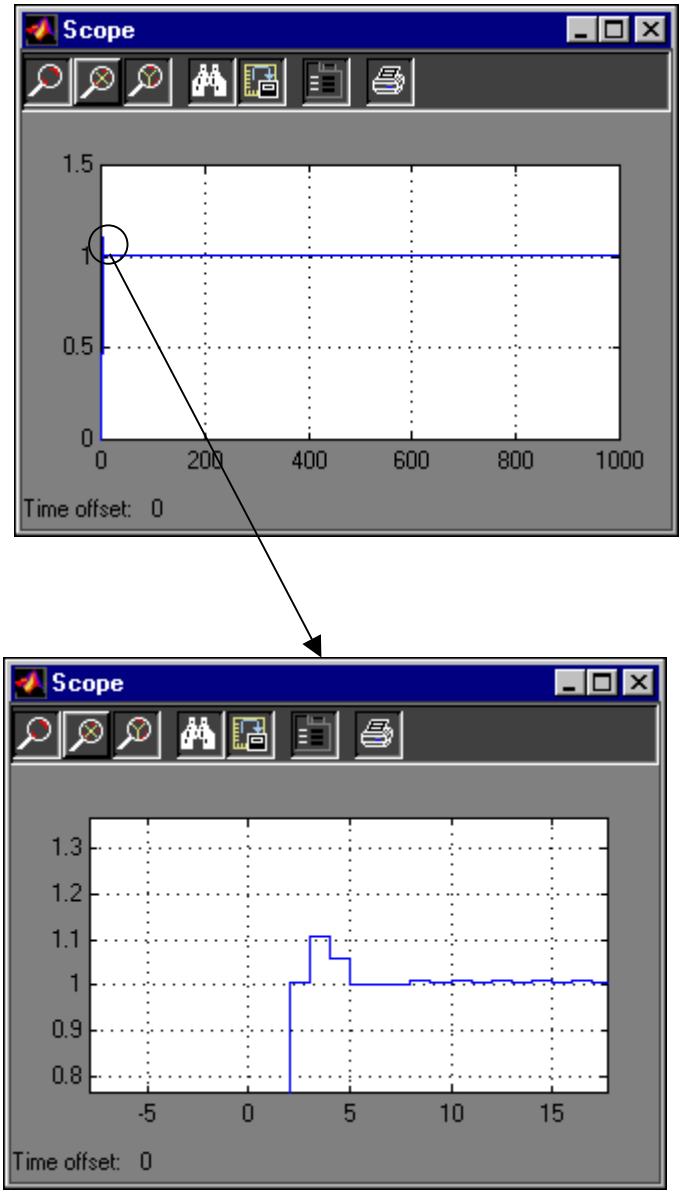


Figure 16: Step Response of Adaptive Control System

The adaptive control system block diagram is shown in the following figure. When the controller parameters converge to their normal value, we obtain the following controller from the output of the block "contr_calc_3ordT" in the Figure 17.

$$(z^2 + 1.4537z + 0.4737)u(t) = (0.4667z^2 + 0.4828z + 0.1448)u_c(t)$$
$$- (1.9561z^2 - 1.65z + 0.7792)y(t) \qquad (4.2)$$

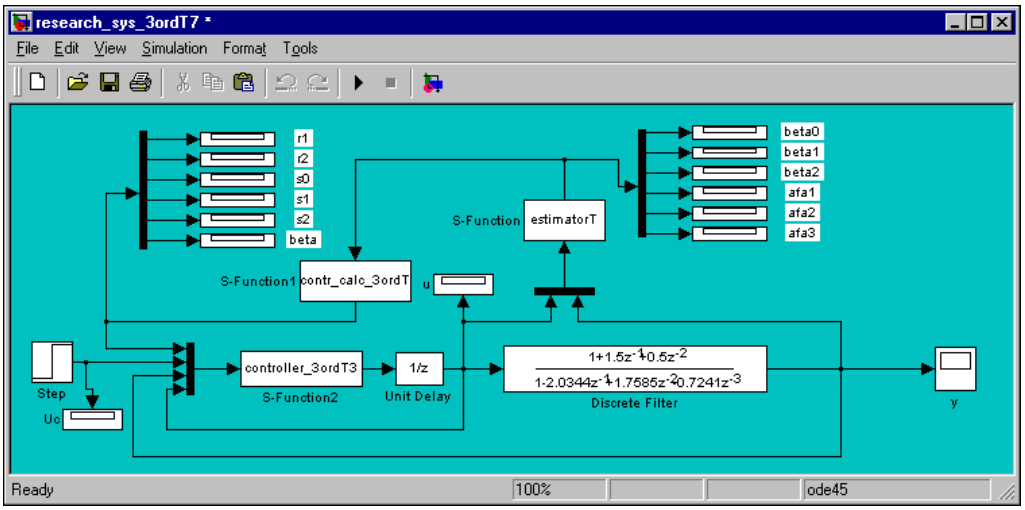


Figure 17: Adaptive Control System

According to the control law (4.2), we have the equivalent controller block diagram shown in the Figure 18.

For the convenience of analysis, we omit the feedforward part in the above figure. Then we get the open-loop transfer function $\frac{B(z)S(z)}{A(z)R(z)}$. By using the MATLAB command "dbode", we can obtain the bode plot in Figure 19.

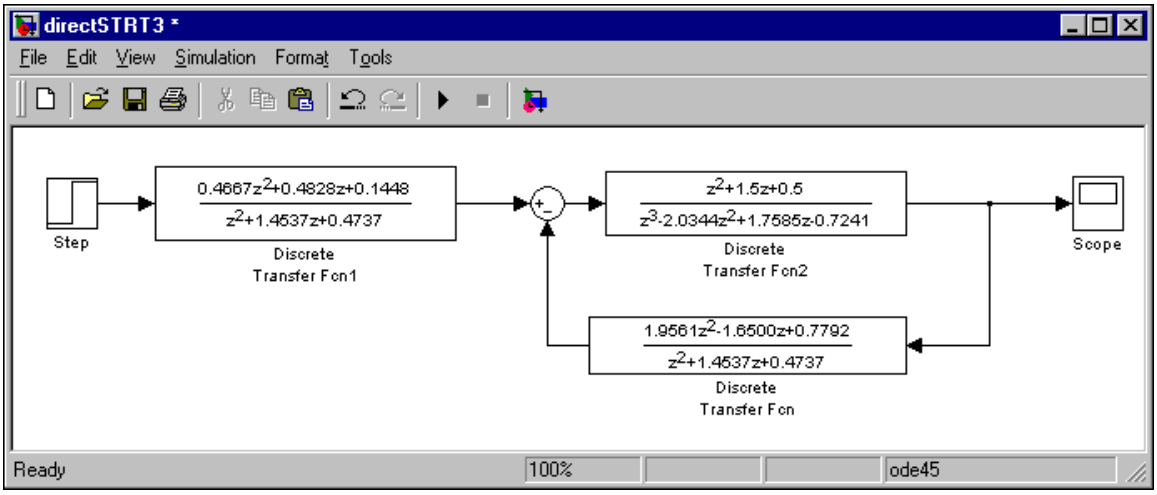


Figure 18: Adaptive Controller

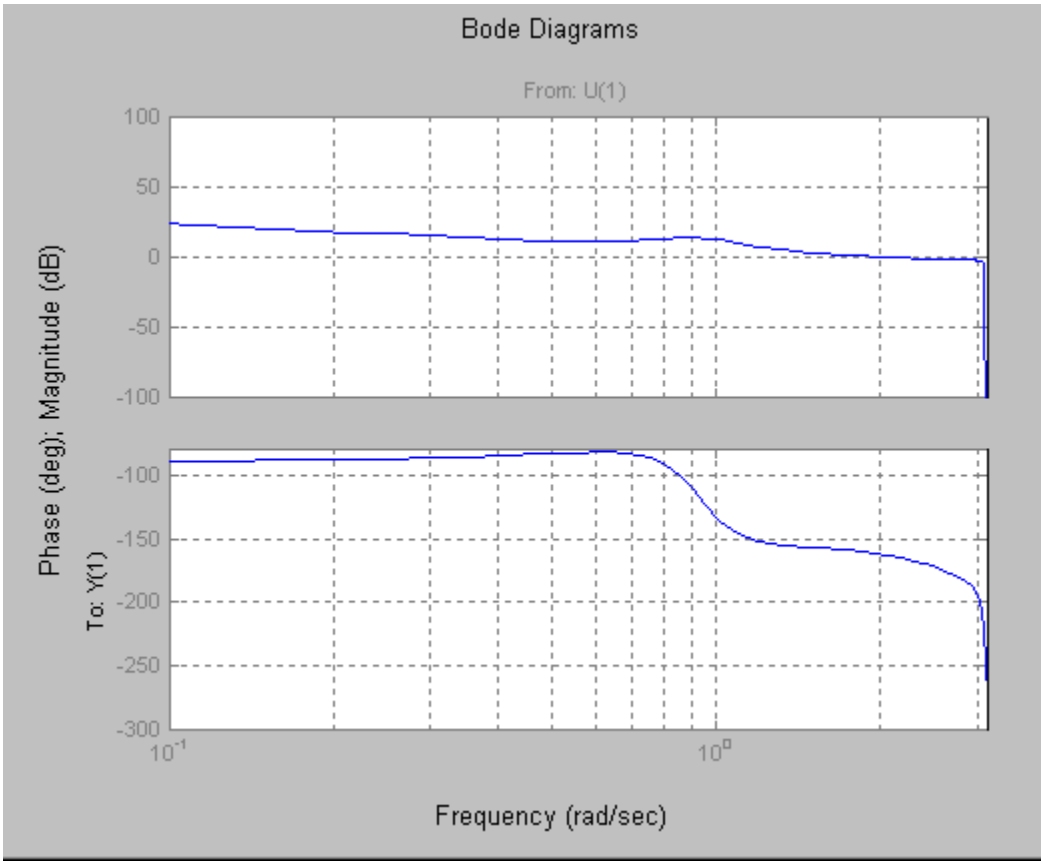


Figure 19: Bode Plot of the Adaptive Control System

From Figure 19, we can find that the gain margin and the phase margin are about 5dB and 10° respectively. And the crossover frequency is around 2 rad/sec. With such a small stability margin, the adpative control system unlikely to maintain stability in presence of unmodeled dynamics at the crossover frequency range.

4.1.2 Adaptive Controller Under Unmodeled Dynamics

In order to check the robustness of the adaptive controller, we intentionally introduce some unmodeled dynamics into the plant. We choose the plant as model (4.1)

$$G_p(z) = \frac{z^2 + 1.5z + 0.5}{z^3 - 2.0344z + 1.7585z - 0.7241}$$

and the unmodeled dynamics as

$$G_{um}(z) = 1 + k\frac{-0.5z^2 + 0.8375z - 0.2739}{z^2 - 0.1179z + 0.6674}$$

where k takes different values.

The unmodelled dynamics can effect the control performance, even destablize the system. The following experiments demostrate how the unmodelled dynamics introduced make the system unstable. It implies that the adaptive control scheme has relatively small stability margin, compared with robust control mehtods. The adaptive behavior itself does not suffice to guarantee its stability under some unmodelled dynamics or noise with much energy.

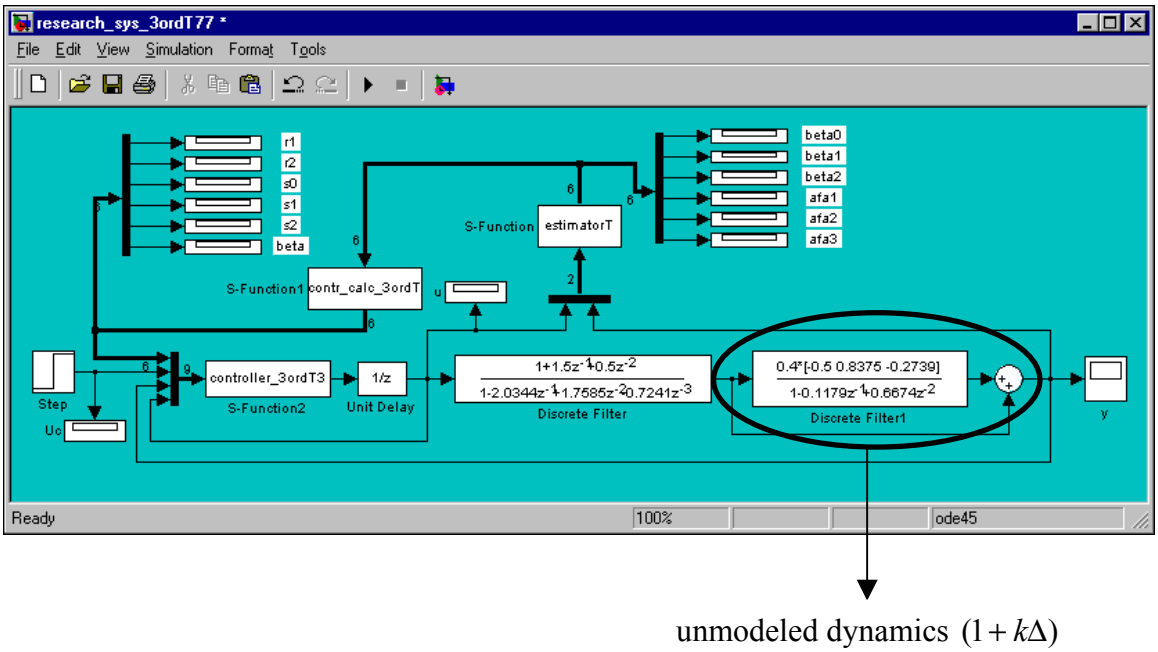


Figure 20: Adaptive Controller Under Unmodeled Dynamics

1) $k = 0.5$. The unmodeled dynamics destablize the system.

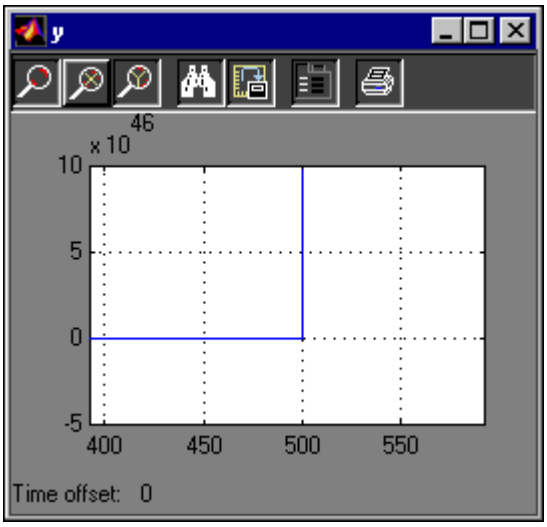


Figure 21: System Output

The frequency response of the unmodeled dynamics is shown below

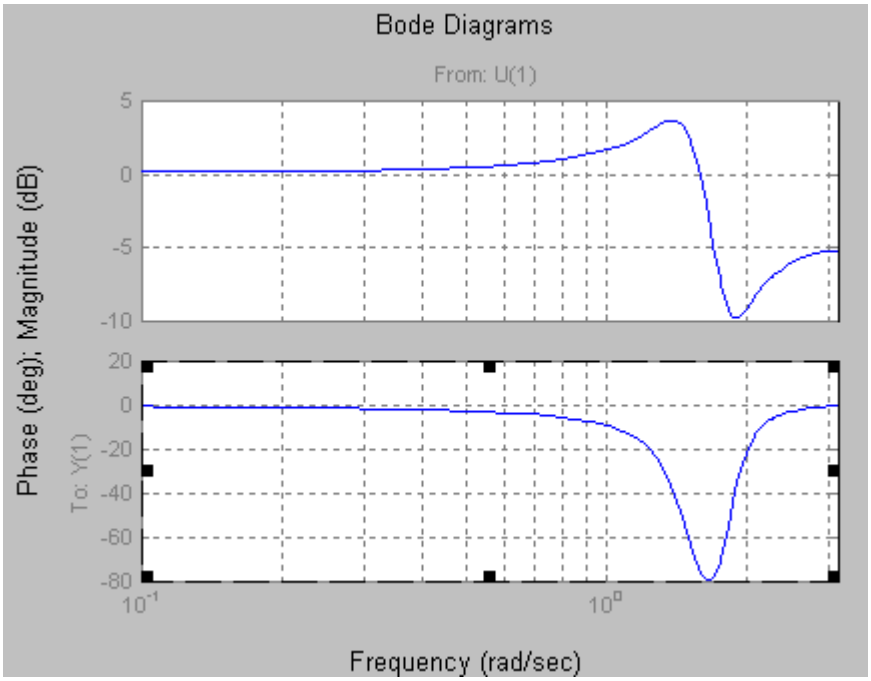


Figure 22: Frequency Response of Unmodeled Dynamics

2) $k = 0.48$. The system remains stable.

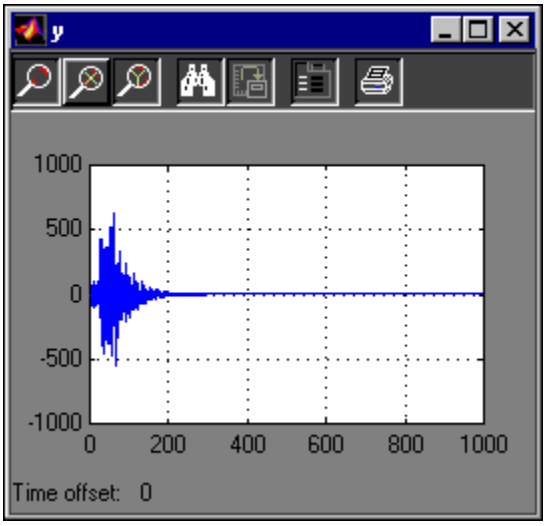


Figure 23: System Output

The frequency response of the unmodeled dynamics is shown below

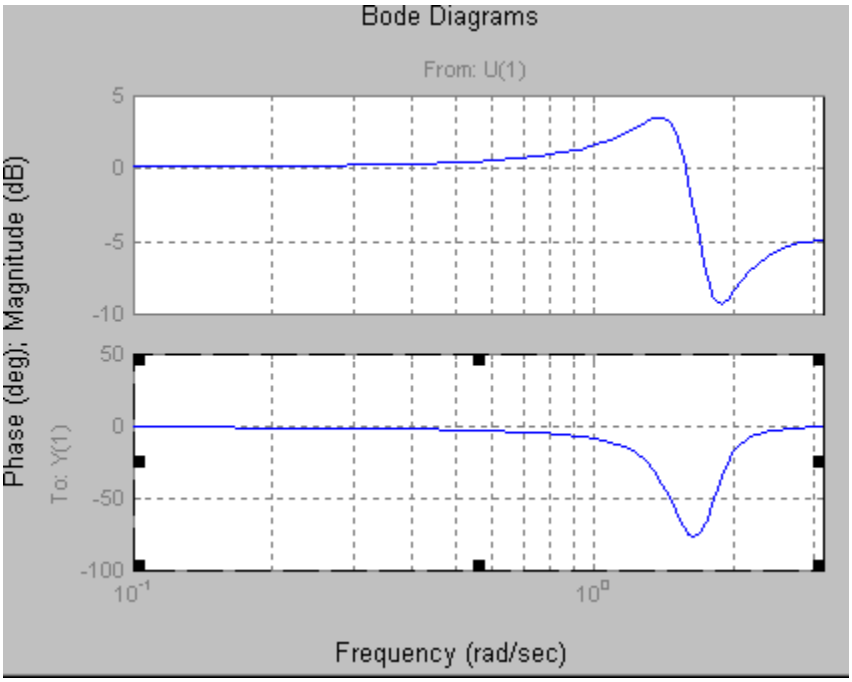


Figure 24: Frequency Response of Unmodeled Dynamics

4) $k = 0.45$. The system is stable,too. (in the next page)

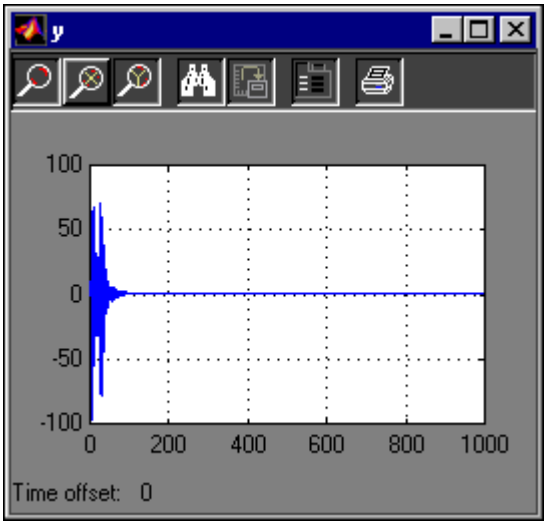


Figure 25: System Output

The frequency response of the unmodeled dynamics is shown below.

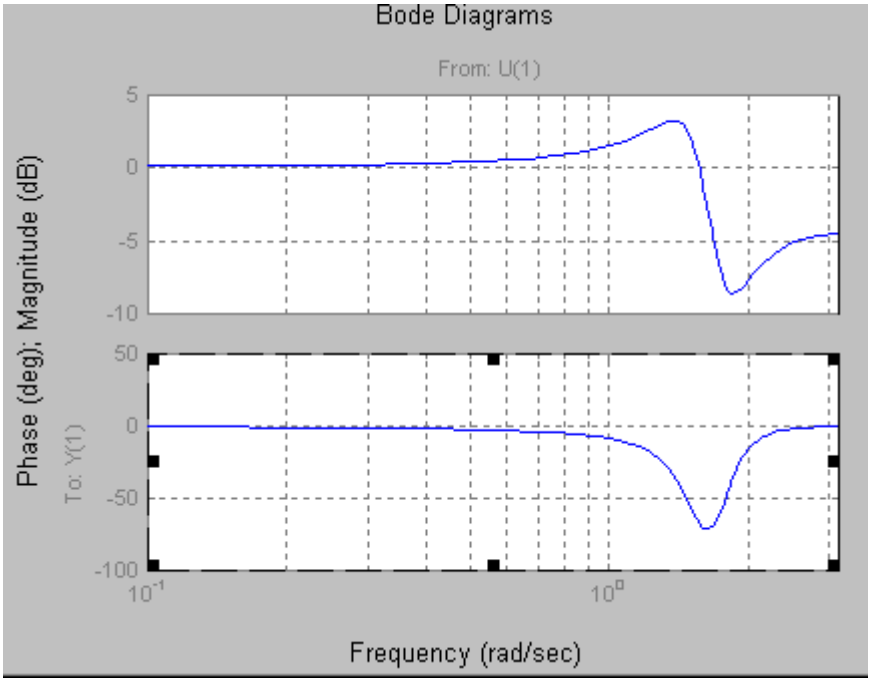


Figure 26: Frequency Response of Unmodeled Dynamics

4.1.3 Classic Feedback Controller

Compared with the adaptive controller in the above, the following simulation shows that classic feedback controller provides a better performance, even with a larger size of unmodeled dynamics.

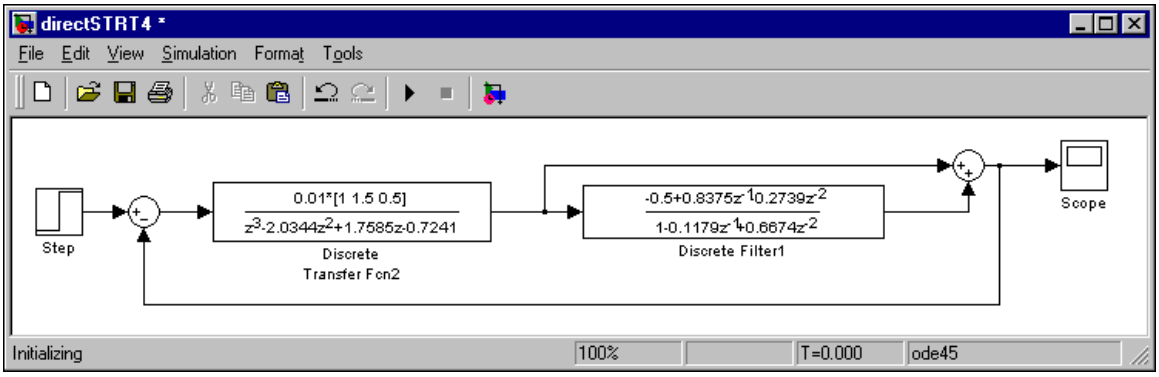


Figure 27: Feedback Controller Under Unmodeled Dynamics

1) $k = 1$. The system is stable and with good step response.

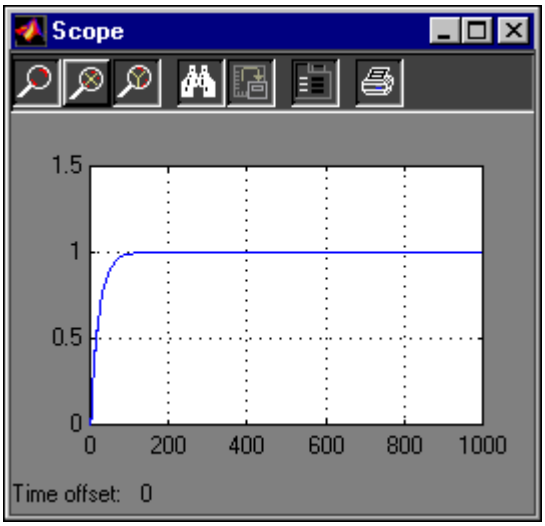


Figure 28: System Output

2) $k = 10$. The system is stable and with slow response.

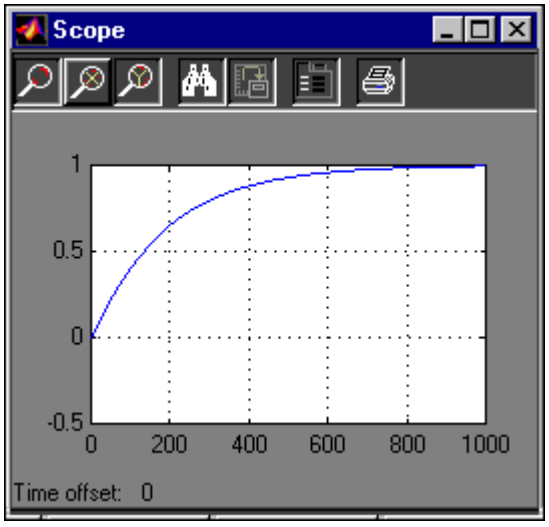


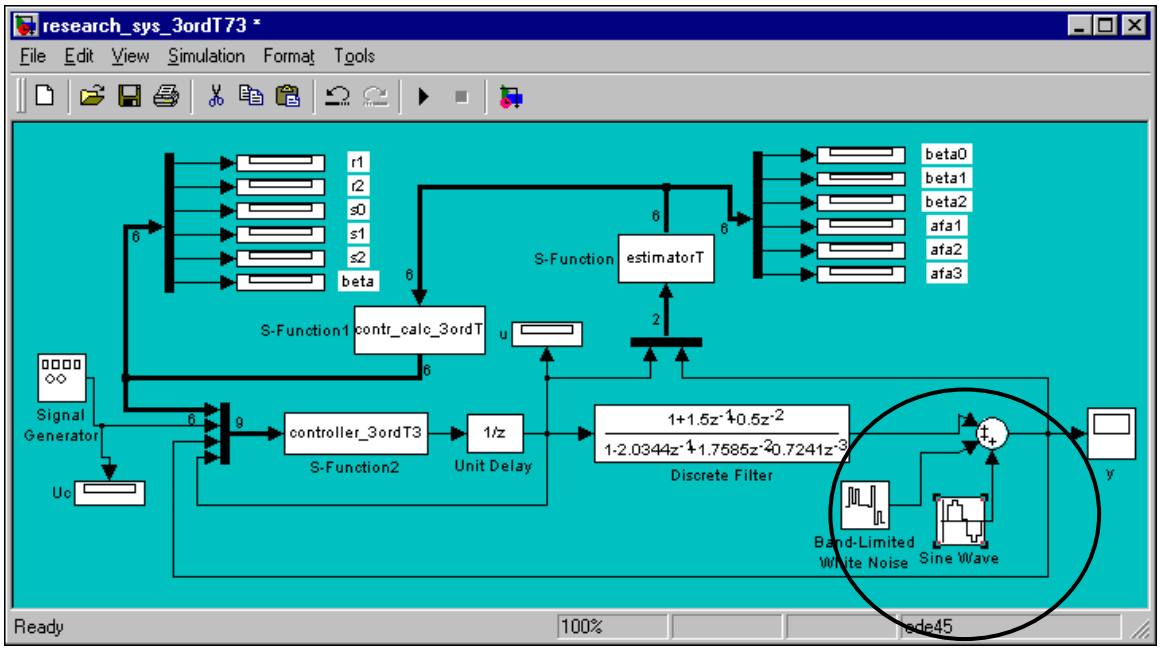
Figure 29: System Output

4.2 Noise Contamination

The basic idea of adaptive controller design is to estimate the plant on-line, so that the controller can change its parameters to adapt to the changing environments. Therefore a robust estimator in an adaptive control system plays a key role. Besides the unmodeled dynamics introduced to the system, we also add some noise signal to the estimator to test its estimation accuracy.

1) Adding noise to the plant output;

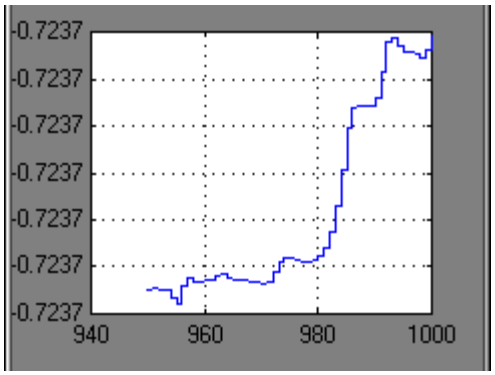
The magnitude of square wave from the signal generator is 1.0, and the band-limited noise power is 0.8, the sine wave magnitude is 0.8.The block diagram is as below.



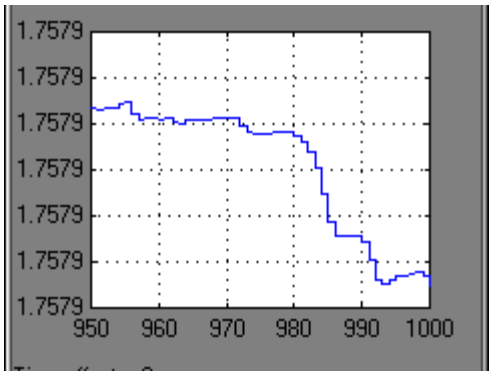
Noise Signal

Figure 30: Type 1 Noise Signal Contamination

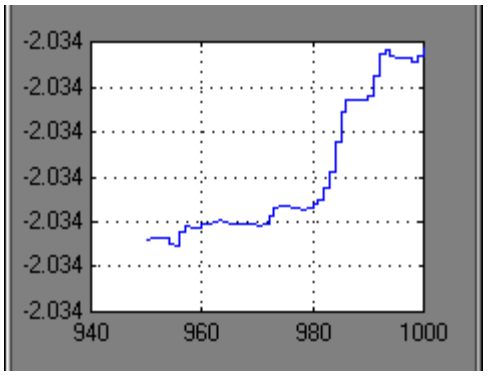
The estimated parameters are shown in the following scopes. We can find that only small error exist.



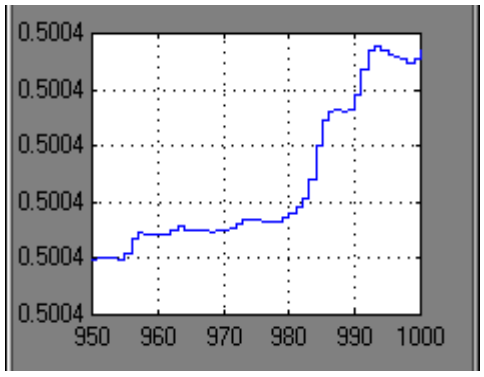
True Value is **–0.7241**



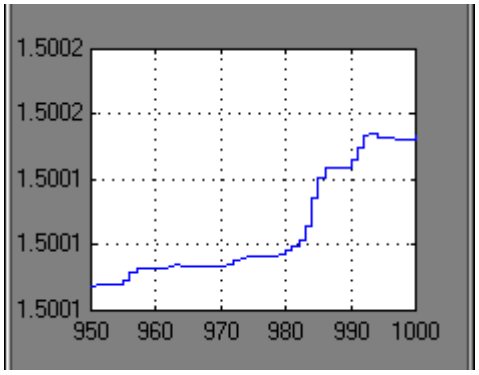
True Value is **1.7585**



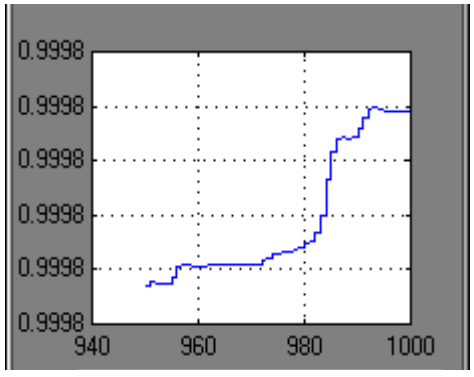
True Value is **–2.0344**



True Value is **0.5**



True Value is **1.5**



True Value is **1.0**

Figure 31: Estimation Under Noise Signal (1)

2) Adding noise to the estimator input

The magnitude of square wave from the signal generator is 1.0, and the band-limited noise power is 0.8, the sine wave magnitude is 0.5. The block diagram is as below.

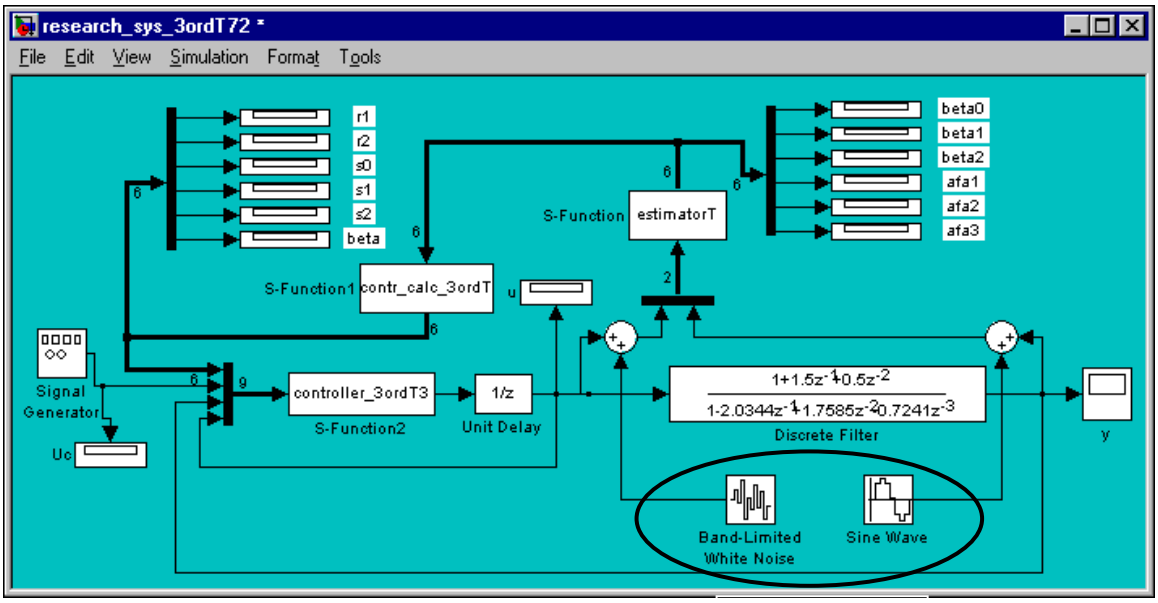
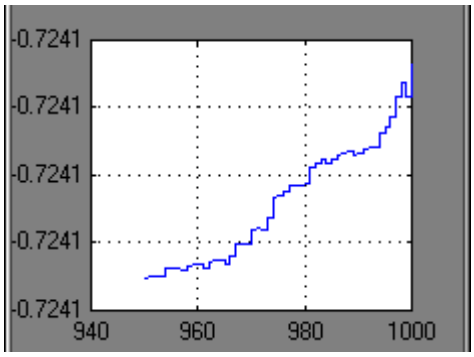


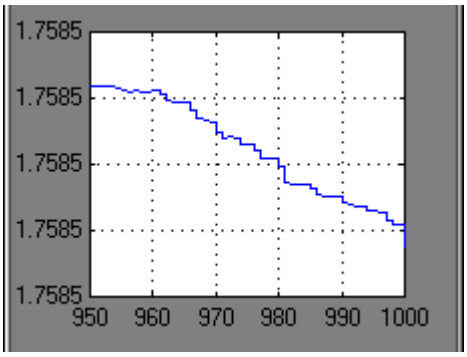
Figure 32: Type 2 Noise Signal Contamination

The estimated parameters are shown in the following figure. And we can find that there is almost no error.

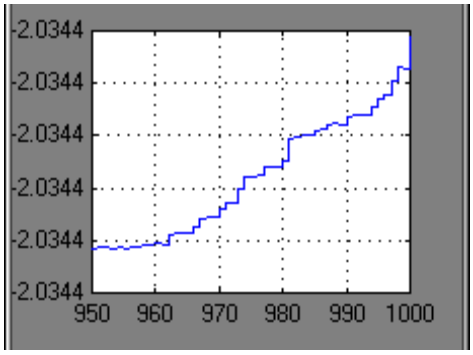
From above simulation experiments, we can conclude that the estimator is very robust, and the noise contamination does not influence the estimation accuracy at all.



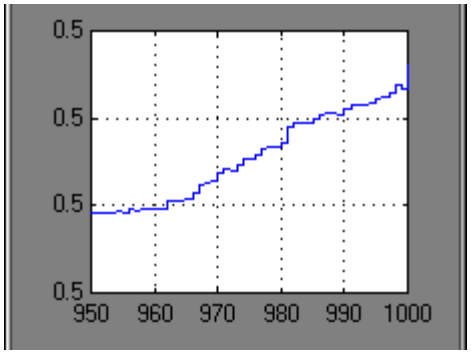
True Value = **–0.7241**



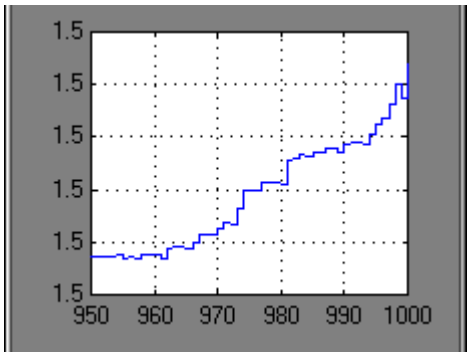
True Value = **1.7585**



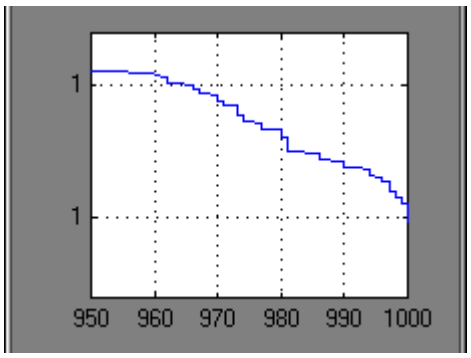
True Value = **–2.0344**



True Value = **0.5**



True Value = **1.5**



True Value = **1.0**

Figure 33: Estimation Under Noise Signal (2)

REFERENCES

[1] K.J. strom and B. Wittenmark, "*Adaptive Control*", Addison-Wesley, 1995

[2] Thomas Kailath, *Linear Systems*, Prentice-Hall, 1980

[3] G.C. oodwin and K.S. Sin, "*Adaptive Filtering, Prediction and Control*", Prentice-Hall, 1984

[4] C.E. Rohrs, L. alavani, M. thans and G. tein, "*Robustness of Continuous Time Adaptive Control Algorithms in the Presence of Unmodeled Dynamics*", IEEE Trans. Auto. Control, Vol. 30, pp881-889, 1985

[5] P.P. Khargonekar and R.Orgeta, "*Comments on the Robust Stability Analysis of Adaptive Controllers Using Normalizations*", IEEE Trans. Auto. Control, Vol. 34, pp478-479, 1989

[6] L. Praly, "*Robustness of Model Rreference Adaptive Control*", in Proc. III Yale Workshop on Adaptive Systems, pp224-226, 1983

[7] P.A. Ioannou and K.S. Tsakalis, "*A robust direct adatpive controller*", IEEE Trans. Auto. Contro*l*, Vol. 31, pp1033-1043, 1986

[8] G. ao and P. oannou, "*Robust Adaptive Control—A Modified Scheme*", Int. J. Control, Vol. 54, pp241-256, 1992

[9] P. oannou and J. un, "*Robust Adaptive Control*", Prentice-Hall, 1996

[10] P. oannou and J. un, "*Theory and Design of Robust Direct and Indirect Adaptive Control Schemes*", Int. J. Control, Vol. 47, pp775-813, 1988

[11] D.W. Clarke, E. osca and R. attolini*, "Robustness of An Adaptive Predictive Controller*", IEEE Trans. Auto. Control, Vol. 39, pp1052-1056, 1994

[12] Bellman, R*, "Adaptive Control----A Guided Tour*", Princeton University Press, 1961

[13] Tsypkin, Y.Z, "*Adaption and Learning in Automatic Systems*", Academic Press, New York, 1971

APPENDIX SIMULATION PROGRAMS

Program 1

```
% filename: estimatorT.m
%  this program is to estimate an unknown procee with Least-square method;
%  the plant to be estimated is in the form of
%  P[z^(-1)]=[z^(-d)*Num[z^(-1)]]/Den[z^(-1)];
%  Num[z^(-1)]=beta_0+beta_1*z^(-1)+beta_2*z^(-2)+...+beta_m*z^(-m);
%  deg(Num)=m;
%  Den[z^(-1)]=1+afa_1*z^(-1)+afa_2*z^(-2)...+afa_n*z^(-n); deg(Den)=n;
function [sys,x0]=estimatorT(t,x,u,flag,ts,Para_ini,p0,m,n)
% ts=sampling time;
% Para_ini=the initial value of the parameters to be estimated;
% p0=the initial value of the convariance matrix;
% m, n=the order of Num[z^(-1)] and Den[z^(-1)], respectively;
index1=n+m+1;
%  the number of the parameters to be estimated;
index2=index1^2;
%  the number of the elements of P(t) matrix;
index3=index1+index2;
p1=p0*eye(index1,index1);
%  the initial value of the P(t) matrix;
p2=zeros(index1,1);
```

```
%  the initial value of the regressor vector;
if flag==0
   sys=[0,2*index1+index2,index1,2,0,0];
%  0 continuous states; (2*index1+index2) discrete states; index1 outputs; 2 inputs;
%  0 dicrete roots; 0 direct feedthrough;
   x0=[Para_ini,p1(1:index2),p2'];
%  it is a row vector of all the initial discrete states;
elseif flag==1
   sys=[];
elseif flag==2
%  theta(t-1);
     theta1=x(1:index1);
%  P(t-1);
   for i=1:index1                                    % to form the P matrix;
      for j=1:index1
         P(j,i)=x(i*index1+j);
      end
   end
%  phi(t), the regressor;
   phi(1)=u(1);                                      % the first input variable is u(t-d);
   phi(2:m+1)=x(index3+1:index3+m);
%  update the parameters from 1st to (m+1)_th;
```

```
  phi(m+2:index1)=x(index3+m+2:index3+index1);
% keep parameters from (m+2)_th to index1_th unchanged;
  phi=phi';
% to estimate the plant parameters with Least-square method;
  temp1=P*phi;
  temp2=inv(1+temp1'*phi);
  P=P-temp1*temp2*temp1';
  theta=theta1+P*phi*(u(2)-phi'*theta1);
% to update the phi vector;
  phi(m+3:index1)=phi(m+2:index1-1);
% update parameters from (m+2)_th to index1_th;
  phi(m+2)=-u(2);
% to output sys for recursive calculation;
  sys=[theta',P(1:index2),phi'];
elseif flag==3
  sys=x(1:index1)';
elseif flag==4
  sys=ceil(t/ts+ts/1e8)*ts;
else sys=[];
end
```

Program 2

```
% filename: Contr_calc_3ordT.m
% this program is to calculate the 3_order controller parameters by use of STR
% algorithm;
% the reference model is represented by the s-function input parameters;
% no process zeros are canceled;
function [sys,x0]=contr_calc_3ord(t,x,u,flag,ts,para_ini,am1,am2,am3,ao1,ao2,n)
% ts=the sampling time;
% para_ini=the initial values of the controller parameters;
% am1,am2,am3,bm0,bm1,bm2=the reference model parameters;
% ao1,ao2=parameters of Ao;
% Ao=a factor of the closed-loop characteristic polynomial Ac=Ao*Am;
% n=the order of the reference model or the plant;
if flag==0
  sys=[0,6,6,6,0,0];
% 0 continuous states; 6 discrete states; 6 outpus; 6 inputs;
% 0 discretes roots; 0 direct feedthrough;
  x0=para_ini;    % 6*1 row vector;
elseif flag==1
  sys=[];
elseif flag==2
  b0=u(1); b1=u(2); b2=u(3); a1=u(4); a2=u(5); a3=u(6);
```

```
% parameters initialization;
   Coeff_B=[b0,b1,b2];
   Coeff_A=[a1,a2,a3];
   Coeff_AoAm=conv([1 am1 am2 am3],[1 ao1 ao2]);
%[R,S]=dio_solver(n,Coeff_A,Coeff_B,Coeff_AoAm);
   [R,S]=dio_solverT(n,1,Coeff_A,Coeff_B,Coeff_AoAm(2:6));
%  call a function to solve the Diophantine equations;
   x(1)=R(1);
   x(2)=R(2);
   x(3)=S(1);
   x(4)=S(2);
   x(5)=S(3);
   x(6)=(1+am1+am2+am3)/(b0+b1+b2);                    % beta
   sys=x';
elseif flag==3
   sys=x(1:6);
%  1st output=r1, R(z)=z^2+r1*z+r2;
%  2nd output=r2, R(z)=z^2+r1*z+r2;
%  3rd output=s0, S(z)=s0*z^2+s1*z+s2;
%  4th output=s1, S(z)=s0*z^2+s1*z+s2;
%  5th output=s2, S(z)=s0*z^2+s1*z+s2;
%  6th output=beta, T(z)=beta*Ao(z);
```

```
elseif flag==4
  sys=ceil(t/ts+ts/1e8)*ts;
else sys=[];
end
```

Program 3

```
% filename: controller_3ordT.m;
% this program is to implement the STR algorithm;
function [sys,x0]=controller_3ordT(t,x,u,flag,ts,ao1,ao2)
% ts=the sampling time;
% ao1,ao2=parameter of Ao, Ao is a factor of the closed-loop characteristic
% polynomial Ac=Ao*Am;
if flag==0
  sys=[0,5,1,9,0,6];
% 0 continuous states; 5 discrete states; 1 output; 9 inputs; 0 discrete roots; 3 direct
% feedthrough;
  x0=[1,0,0,0,0];
elseif flag==1
  sys=[];
elseif flag==2
  xk=x(1:5);
% xk=[Uc(t),Uc(t-1),Uc(t-2),-y(t),-y(t-1),-y(t-2)-u(t-1),-u(t-2)]
```

```
  xk(2)=xk(1);                    % update the xk vector;
  xk(1)=u(7);
  xk(4)=xk(3);
  xk(3)=-u(8);
  xk(5)=-u(9);
  sys=xk';
elseif flag==3
  xxp=[u(6)*ao1,u(6)*ao2,u(4),u(5),u(2)];
%  u(6)=beta, T(z)=beta*Ao(z);
%  u(5)=s2, S(z)=s0*z^2+s1*z+s2;
%  u(4)=s1, S(z)=s0*z^2+s1*z+s2;
%  u(3)=s0, S(z)=s0*z^2+s1*z+s2;
%  u(1)=r1, R(z)=z^2+r1*z+r2;
% v u(2)=r2, R(z)=z^2+r1*z+r2;
sys=xxp*x+u(6)*u(7)-u(3)*u(8)-u(1)*u(9);
elseif flag==4
  sys=ceil(t/ts+ts/1e8)*ts;
else sys=[];
end
```

Program 4

```
% filename: dio_solverT.m
% this program is to solve a Diophantine Equation;
% call form: [Coeff_R,Coeff_S]=dio_solverT(n,d,Coeff_A,Coeff_B,Coeff_AoAm)
% n=deg[A(z)];
% d=time delay, n-d=deg[B(z)];
% A(z)R(z)+B(z)S(z)=Ao(z)Am(z);
% A(z)=z^n+a1*z^(n-1)+a2*z^(n-2)+...+an, deg[A(z)]=n, MONIC POLYNOMIAL
% B(z)=b0*z^(n)+b1*z^(n-1)+b2*z^(n-2)+...+bn, b0=b1=b2=...=b_d-1=0,
% deg[B(z)]=n-d;
% R(z)=z^(n-1)+r_1*z^(n-2)+...+r_(n-1), deg[R(z)]=n-1, MONIC POLYNOMIAL;
% S(Z)=s_0*z^(n-1)+s_1*z^(n-2)+...+s_(n-1), deg[S(z)]=n-1 <==s_0~=0;
% Ao(z)Am(z)=z^(2n-1)+gama_1*z^(2n-2)...+gama_(2n-1)
% deg[Ao(z)Am(z)]=2n-1, MONIC POLYNOMIAL;
% Ao(z)=z^(n-1)+ao_1*z^(n-2)+...+ao_(n-1),
% deg[Ao(z)]=n-1, MONIC POLYNOMIAL;
% Am(z)=z^(n)+am_1*z^(n-1)+...+am_n, deg[Am(z)]=n, MONIC POLYNOMIAL;
% Ao(z)Am(z) can be specified by the verctor [gama_1, gama_2, ..., gama_(2n-1)]
% or by the two vectors
% [ao_1, ao_2, ..., ao_(n-1)] and [am_1, am_2, ..., am_n] respectively;
function [Coeff_R,Coeff_S]=dio_solverT(n,d,Coeff_A,Coeff_B,Coeff_AoAm)
% Coeff_A=[a1,a2,...an];
```

```
%  Coeff_B=[b0,b1,b2,...bn];
%  Coeff_AoAm=[gama_1,gama_2,...,gama_(2n-1)];
%  to format the matrix M composed of the coefficients of A(z) and B(z)
%  M is a (2n-1)*(2n-1) Matrix;
degA=length(Coeff_A);
degB=length(Coeff_B)-1;
if d~=(degA-degB) disp('Error in degrees of A and B!')
end
A=Coeff_A;
B=[zeros(1,d),Coeff_B];   % b0=b1=...=b_d-1=0;
Arolling=[1,A,zeros(1,n-2)];
Brolling=[B,zeros(1,n-2)];
M(:,1)=Arolling';
for i=2:n-1
  M(:,i)=[0;M(1:2*n-2,i-1)];
end
M(:,n)=[Brolling(2:2*n-1),0]';
M(:,n+1)=Brolling';
for i=n+2:2*n-1
  M(:,i)=[0;M(1:2*n-2,i-1)];
end
M;
```

```
if det(M)<1e-5
   disp('A(z) and B(z) are not coprime!')
   M;
end
Ac=Coeff_AoAm-[A,zeros(1,n-1)];
para=inv(M)*Ac';
Coeff_R=para(1:n-1);
Coeff_S=para(n:2*n-1);
```

VITA

Zhongshan Wu was born in China on October 21, 1974. He studied control and systems at Northeastern University and achieved the degree of Bachelor of Science in Eletrical Engineering on July 6, 1996. He then worked in the Research Center of Automation at Northeastern University. He entered the master's program in the Department of Electrical and Computer Engineering at Louisiana State University in the spring of 2000. Now he is a candidate for the degree of Master of Science in Electrical Engineering. In the meantime, he is studying toward a doctorate degree in electrical engineering at Louisiana State University.