ISSN 0280–5316
ISRN LUTFD2/TFRT--5537--SE

Extremal Seeking Control

A.K. Urquhart

Department of Automatic Control
Lund Institute of Technology
September 1995

Department of Automatic Control **Lund Institute of Technology** P.O. Box 118 S-221 00 Lund Sweden	*Document name* MASTER THESIS
	Date of issue September 1995
	Document Number ISRN LUTFD2/TFRT--5537--SE
Author(s) Alexander K. Urquhart of Imperial College of Science, Technology & Medicine	*Supervisor* Björn Wittenmark
	Sponsoring organisation The Engineering and Physical Sciences Research Council (EPSRC)

Title and subtitle

Extremal Seeking Control

Abstract

In this thesis, extremal control of a simple nonlinear stochastic system is analyzed. The process is modeled as a generalized Hammerstein model. Different problem formulations are discussed and the performances of the resulting nonlinear adaptive controllers are investigated. The adaptive controllers are based on on-line estimation of the parameters of the process and the certainty equivalence principle.

Key words

Adaptive Control, Nonlinear Systems and Extremal Control

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280–5316		*ISBN*
Language English	*Number of pages* 87	*Recipient's notes*
Security classification		

Acknowledgments

All the work presented in this project, stemmed from original ideas by Professor Björn Wittenmark. It was also he who supervised me throughout the project and it is to him, that I would like to express my sincere thanks.

I would also like to thank the Engineering and Physical Sciences Research Council (EPSRC), for giving me the financial support which allowed me to work on this project.

Contents

Introduction

In almost all human endevour the main objective is to maximize or minimize some criteria. Whether it be an athlete trying to run that little bit faster so that the sprint time is minimized or a plant operator trying to get exactly the right conditions to produce the maximum amount of product, the main objective is the same, striving to reach an extremum.

This project is concerned with not only reaching extremum points but also to track the extremum as the system evolves. Tracking an extremum is called Extremal Seeking Control. For a system to have an extremum then it must be nonlinear by definition. Hence to be able to find and track an extremum a nonlinear model and controller must be used.

A successful application of Extremal Seeking Control has been achieved in the automotive industry, see [WZ91]. The need for extremal seeking control arises in this area because in a spark ignition engine the variations in engine torque are a nonlinear function of variables such as spark angle, air/fuel ratio, engine speed and load. Hence, as for a given constant speed and load, the engine performance will vary with the spark angle and air/fuel ratio, there will be specific values of spark angle and air/fuel ratio, where the performance of the engine will be optimal. Thus in this case the Extremal Seeking Controller will measure the varying speed and load and find the corresponding extremum point of the performance criteria, in terms of the spark angle and air/fuel ratio.

This project also adds to the complexity of the problem by assuming that there is colored noise corrupting the measurable output. To compensate for the noise in the output the goal of the extremal controller is now not just to find the extremum, but also to control around the extremum in a way that minimizes the variance from the extremum. For stochastic linear dynamical systems there has been strong theoretical foundations available for many years, but for stochastic nonlinear systems there has not even been a foundation developed yet. Minimum variance extremal control is discussed in [Wit93] and this project is a rework and extension of the ideas presented therein. The paper by Wittenmark also contains an extensive literature review of the areas which Extremal Seeking control touch upon.

The first chapter in this report gives a formal definition of the problem, by clearly stating the type of model used and control objectives. The second and third chapters start the analysis of extremal control with various controllers being proposed and the relative merits of each being discussed. These two chapters also highlight the main problems with trying to achieve minimum variance control at an extremum.

To find controllers that have better and better performance is an endless task, if the best achievable performance is not known. The fourth chapter looks at finding a bound on the best achievable performance of the system and then assesses to what level the controllers in chapters two and three achieved this bound.

The final chapter looks at changing the analyzed problem from the known parameter case to the unknown parameter case. This effectively changes the controllers from being nonlinear minimum variance controllers to Adaptive Extremal Seeking controllers. The performance of the new Extremal Seeking controllers are then analyzed, with the strengths and weaknesses of each being discussed.

The main results from this project appear in the paper [WU95], by Wittenmark and Urquhart, which will be presented at the CDC 1995 in New Orleans.

Chapter 1

Problem Formulation

This chapter discusses and presents the problem that this project is concerned with. First the nonlinear model structure is selected and the example system that is used through out the project is stated. The objectives of the controller are also given along with the admissible control signals.

1.1 Model Structure

Extremum control problems are nonlinear in their very nature, as for a linear system the only possible extrema that could exist are $\pm\infty$. Hence some means of representing the system nature is required. At this point in time, there exist many methods of representing system nonlinearities, the various methods vary widely in complexity and structure. The problem that arises in the model selection procedure is that a compromise has to be made, so that the need to use a model that is general enough to represent most physical systems, does not make the model so overwhelmingly complex that system analysis becomes impossible.

Almost all of the different models have one feature in common, they use a polynomial expansion to represent the system nonlinearity. Out of these models the NARMAX (Nonlinear AutoRegressive Moving Average model with eXogenous inputs), see [LB85], is the one that is most general and hence can represent the most systems. This model structure was not chosen because it is more complex than is required for the preceding investigation. The model that has the right balance of complexity and generality, for this application, is the Hammerstein model. An overview of the basic model structure is shown in Figure 1.1.

The $F(\bullet)$ nonlinearity in Figure 1.1 is approximated by a polynomial expansion and the transfer function $H(z^{-1})$ is a ARMAX model. Hence the

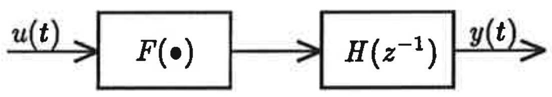


Figure 1.1: Cascaded block structure of the Hammerstein Model with non-linearity $F(\bullet)$ and transfer function $H(z^{-1})$.

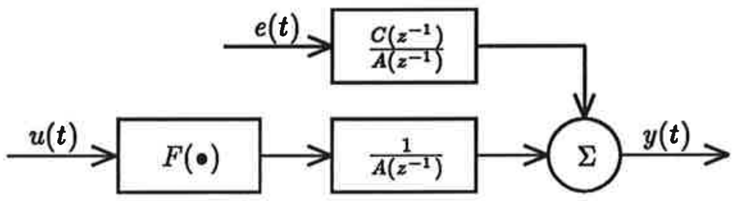


Figure 1.2: Block diagram of the Hammerstein Model.

system model equation is

$$\begin{aligned} A(z^{-1})y(t) =& B_o + B_1(z^{-1})u(t-1) + B_2(z^{-1})u^2(t-1) + \dots \\ & + B_n(z^{-1})u^n(t-1) + C(z^{-1})e(t) \end{aligned} \tag{1.1}$$

Where $A(z^{-1})$, $B_n(z^{-1})$ and $C(z^{-1})$ are polynomials in z^{-1} and $e(t)$ is Gaussian White Noise.

The block diagram of the particular Hammerstein system described by (1.1), is shown in Figure 1.2.

One of the main features of the Hammerstein model (and the NARMAX) is that it is linear in the parameters. This means that normal system identification techniques can be used to identify the parameters and hence adaptive controllers can be implemented.

The actual example system that is used throughout the project is as follows

$$A(z^{-1})y(t) = B_o + B_1(z^{-1})u(t-1) + B_2(z^{-1})u^2(t-1) + C(z^{-1})e(t) \tag{1.2}$$

where

$$\begin{aligned} A(z^{-1}) &= 1 + az^{-1} = 1 - 0.95z^{-1} \\ B_o &= b_o = 0.25 \\ B_1(z^{-1}) &= b_{10} + b_{11}z^{-1} = 0.5 - 0.25z^{-1} \\ B_2(z^{-1}) &= b_{20} = 0.5 \\ C(z^{-1}) &= 1 + cz^{-1} = 1 - 0.5z^{-1} \\ e(t) &= \text{Zero Mean Gaussian White Noise, Variance} = 0.04 \end{aligned}$$

At this point it might seem that the above example is very unambitious, as it is only a quadratic Hammerstein model, which at first glance might seem trivial to control in an extremal way. The next section shows that finding an acceptable controller for this simple model is far from trivial. Also once an extremal controller has been found for the simple quadratic case, it will only be an extension of the result to cope with a more general example as all functions are quadratic like near an extremum.

1.2 Control Objectives

To differentiate between all the different controllers that will be tested, some means of assessing their relative merits is required. To achieve this differentiation, the following control objective is defined.

$$V(\mathcal{U}) = \frac{1}{N}\sum_{t=1}^{N}(y(t) - y_o)^2 \tag{1.3}$$

where $\mathcal{U}$ is the chosen control strategy.

Hence in words, the control objective is to minimize the variance about the desired set point y_o. In the first three chapters that proceed y_o is assumed to be given, with it taking values that are near or at the extremum. In the Adaptive Extremal Seeking Control chapter, y_o is assumed to be the extremum and not to be given, hence the extremum must be identified before being given as the desired set point.

1.3 Admissible Control Signals

The control signals are restricted to being a member of the set $\mathcal{Y}_t$ i.e. $u(t) \in \mathcal{Y}_t$. The set $\mathcal{Y}_t$ is defined as follows

$$\mathcal{Y}_t = [y(t), y(t-1), \ldots ; u(t-1), u(t-2), \ldots] \tag{1.4}$$

where

$$y(t), y(t-1), \ldots ; u(t-1), u(t-2), \ldots \in \Re$$

Hence with $\mathcal{Y}_t$ defined as in (1.4) the control signals are restricted to being causal and real.

Chapter 2

Expectation & Variance Control

This chapter starts the analysis of the quadratic Hammerstein model. The approach that this chapter takes is the normal expectation and minimum variance approach but converted for the Hammerstein model case. Various methods of controlling the system are proposed with comparisons being made and conclusions drawn as to the best policy to implement using the expectation and variance ideas.

2.1 Expectation Extremum Control

This section starts off the investigation into the extremum problem by looking at the simplist control strategy, i.e. the control signal only depends on the reference input, y_o, and the static system characteristics.

The expression for the control signal is found by taking the expectation of the static Hammerstein model, $z^{-1} = 1$ and noise terms removed, given in (1.1). Thus the control equation in terms of the constant control, u_o, is as follows

$$E\{y(t)\} = \frac{B_o + B_1(1)u_o + B_2(1)u_o^2 + \ldots + B_n(1)u_o^n}{A(1)} = y_o \tag{2.1}$$

Solving (2.1) for the quadratic case gives

$$u_o = -\frac{B_1(1)}{2B_2(1)} \pm \sqrt{\left[\left(\frac{B_1(1)}{2B_2(1)}\right)^2 + \frac{A(1)y_o - B_o}{B_2(1)}\right]} \tag{2.2}$$

y_o	Expectation Controller		
	Variance	Loss	Mean
4.375	0.1298	0.1306	4.4029
5.0	0.1298	0.1306	5.0279
7.0	0.1298	0.1306	7.0278
10.0	0.1298	0.1306	10.0278

Table 2.1: Simulation results for the Expectation controller.

The solution given in (2.2) only exists if

$$y_o \geq \frac{B_o}{A(1)} - \frac{B_1(1)^2}{4A(1)B_2(1)} \tag{2.3}$$

By substituting in the parameters of the example system, expression (1.2), the minimum value of y_o, the extremum, is found to be 4.375.

To analyze the system performance with the control defined by expression (2.2), the system was implemented in Matlab-4.2c using the program min_exp.m which is given as given on page 61 of appendix A. Using the program the results in Table 2.1 were obtained. Note that all the proceeding simulations are 5000 steps in length with the first 100 steps being omitted from the calculations so that the initial transient effects are suppressed.

The sample variance and loss function are as defined below

$$\text{Variance of } y(t) = \frac{1}{N}\sum_{k=1}^{N}(y(k) - \overline{y})^2 \tag{2.4}$$

$$\text{Loss} = \frac{1}{N}\sum_{k=1}^{N}(y(k) - y_o)^2 \tag{2.5}$$

where

$$\begin{aligned}\overline{y} &= \text{The sample Mean of } y(t) \\ &= \frac{1}{N}\sum_{k=1}^{N}(y(k))\end{aligned} \tag{2.6}$$

Note the difference between the definition of the variance and the loss, in that if $y(t)$ does not have the desired mean value, y_o, then the variance and the loss functions can have very different values. In Table 2.1 the values are the same because the initial transients have been suppressed by not including the first 100 steps in the calculations and also the controller, by its definition, will make the output have the desired reference value. The variance of $y(t)$ shown in Table 2.1 is the open loop variance of the system,

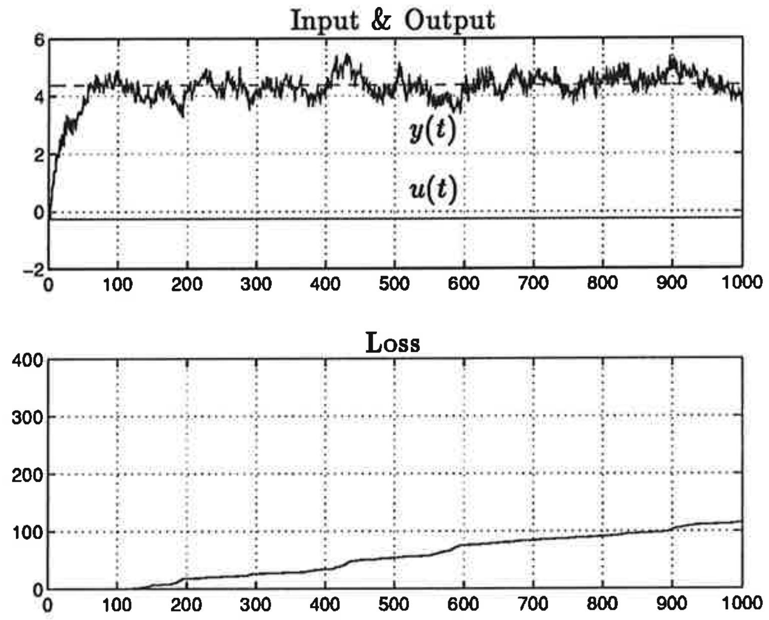


Figure 2.1: Simulation using the Expectation controller with $y_o = 4.375$. Dashed line equals the extremum, 4.375.

which can be found, via the method given in [ÅW90], to be 0.1231. The slight difference between the theoretical value for the open loop variance of the output signal and the values given in Table 2.1, is just due to the fact that only a finite length simulation was used to generate the results in the table.

The plots of the output, input and loss are shown in Figure 2.1. The loss function behaves as expected, linearly increases with time, as the output is a gaussian random sequence with mean y_o, hence there will always be an error from the desired output, y_o.

The performance of the expectation controller is good for what it is, but it makes no attempt to reduce the loss below the open loop variance. The next section looks at the problem of trying to achieve "minimum variance".

2.2 Minimum Variance Extremum Control

The problem with the expectation controller is that it does not attempt to reduce the variance of the output signal and hence the loss. A one step ahead minimum variance type controller is now presented with the objective of reducing the output variance.

The cost function that the minimum variance controller should minimize with respect to $u(t)$ is as follows.

$$J = E\left\{[\hat{y}(t+1) - y_o]^2 \,|\mathcal{Y}_t\right\} \tag{2.7}$$

The standard minimum variance approach is now adopted with the polynomial partition given below, being used to express the prediction of $y(t+1)$ in two uncorrelated parts, the one step ahead prediction and the one step ahead prediction error.

$$C(z^{-1}) = A(z^{-1})F(z^{-1}) + z^{-d}G(z^{-1}) \tag{2.8}$$

where the degrees of $F(z^{-1})$ and $G(z^{-1})$ are zero and $max(n_a - 1, n_c - 1)$ respectively.

Combining (1.2) with (2.8), noting that in this case $F(z^{-1}) = 1$ and $d = 1$, and suppressing the (z^{-1}) terminology for simplicity, gives

$$\begin{aligned}(C - z^{-1}G)y(t) &= B_o + B_1u(t-1) + B_2u^2(t-1) + Ce(t)\\ y(t+1) &= \frac{1}{C}\left(B_o + B_1u(t) + B_2u^2(t) + Gy(t)\right) + e(t+1)\end{aligned} \tag{2.9}$$

hence the one step ahead prediction of $y(t)$ is

$$\hat{y}(t+1 \mid t) = \frac{1}{C}\left(B_o + B_1u(t) + B_2u^2(t) + Gy(t)\right) \tag{2.10}$$

which for the example system, (1.2), becomes

$$\hat{y}(t+1 \mid t) = \frac{1}{C}\left(b_o + b_{10}u(t) + b_{11}u(t-1) + b_{20}u^2(t) + (c-a)y(t)\right) \tag{2.11}$$

Now by substituting in the actual values of the example system into (2.11) and combining it with the cost function, (2.7), gives

$$\begin{aligned}J = \Big(&0.25 + 0.5u(t) - 0.25u(t-1) + 0.5u^2(t) + 0.45y(t)\\ &+ 0.5\hat{y}(t \mid t-1) - y_o\Big)^2\end{aligned} \tag{2.12}$$

To find the minimum of this cost function, the partial derivative with respect to $u(t)$ has to be taken and is as follows

$$\begin{aligned}\frac{\partial J}{\partial u(t)} = u^3(t) &+ 1.5u^2(t) + \Big(1 + \hat{y}(t \mid t-1) - 0.5u(t-1)\\ &+ 0.9y(t) - 2y_o\Big)u(t) + 0.25 + 0.5\hat{y}(t \mid t-1)\\ &- 0.25u(t-1) + 0.45y(t) - y_o\end{aligned} \tag{2.13}$$

y_o	One Step controller		
	Variance	Loss	Mean
4.375	0.1300	0.5542	5.0262
5.0	0.0727	0.0956	5.1513
7.0	0.0406	0.0407	7.0089
10.0	0.0401	0.0401	10.004

Table 2.2: Simulation results for the One Step controller.

Solving (2.13) for its roots, gives the control which minimizes the cost function and is hence the one step ahead minimum variance control. The expressions for the minimal control are as shown below, note that there is the possibility that two of the roots will go complex if $\Upsilon < 0$.

$$u(t) = \begin{cases} -0.5 & \text{if } \Upsilon < 0, \\ -0.5 + 0.2236\sqrt{\Upsilon} & \text{if } \Upsilon > 0, \\ -0.5 - 0.2236\sqrt{\Upsilon} & \text{if } \Upsilon > 0. \end{cases} \tag{2.14}$$

where

$$\Upsilon = 10u(t-1) - 18y(t) + 40y_o - 20\hat{y}(t \mid t-1) - 5$$

Note that when Υ is less than zero, the last two roots in (2.14) become complex and, as a practical control signal is required, the only solution that remains is the first, $u(t) = -0.5$ root. This leads to a constant control which when applied to the system, the variance of the output will be the open loop variance. This suggests that the last two roots correspond to a minimum variance control and the first root corresponds to a constant, open loop variance control.

To simulate the system, some means of root selection is required. The roots are checked to make sure that they are not complex and then the cost function is evaluated for each root with the root which has the lowest cost being chosen. The program that does this is One_Step.m which is given on page 62 of appendix A. Using this program the results shown in Table 2.2 are obtained.

The first thing that is noticed from Table 2.2 is that the loss at the extremum is very high and in fact is 320% higher than the Expectation controller. This is in contrast with the loss away from the extremum, as a 70% reduction is achieved, when compared with the Expectation controller for $y_o = 10$. This vast difference in loss is due to the variance and mean values being far from what they should be when the system approaches the extremum. The variance at the extremum is the open loop variance, hence the One Step ahead minimum variance controller has failed completely in

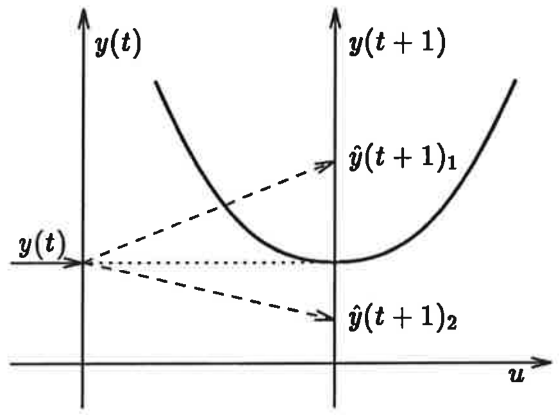


Figure 2.2: Visualization of the constraint the input nonlinearity places on the one step ahead minimum variance controller.

its objective to reduce the variance. Again this is contrasted with the case away from the extremum, as the intrinsic variance of the system is achieved for $y_o = 10$, thus the controller in this case actually produces a "minimum varaince" control. The mean value behaves in a similar way to the variance, in that it is far from what it should be at the extremum, but is almost exactly correct at $y_0 = 10$.

The reason for this poor performance at the extremum, is highlighted in Figure 2.2 which gives a visualization of the system where it is assumed, for simplicity, that there are no delays in the system.

When the system output is at $y(t)$, in Figure 2.2, and the predicted value $\hat{y}(t+1)_2$ is below the extremum, the controller has no problem in creating a control to push the system up to the extremum point. The problem comes when the noise induces the predicted value to be above the extremum, $\hat{y}(t+1)_1$. This creates the situation where the controller is very restricted by the input nonlinearity, in that the control can not go below the extremal point and hence can not pull $\hat{y}(t+1)_1$ down to the extremum. Thus the One Step controller can not act on the predicted value, when the system is near the extremum and the predicted value is above the extremum. Hence when $y(t)$ is near the extremum and $\hat{y}(t+1)_1$ is above it, the system is almost uncontrollable and becomes reliant on the noise in the system to carry it down to the extremum. This explains why the output variance goes to the open loop variance as y_o approaches the extremum.

The output, input and loss function of the system are plotted in Figure 2.3. The input and output plots illustrate the lack of controllability when the output is near the extremum, in that when the output is near the extremum, the controller can only apply a constant control, but when the output goes below the extremum, the controller can apply the minimum

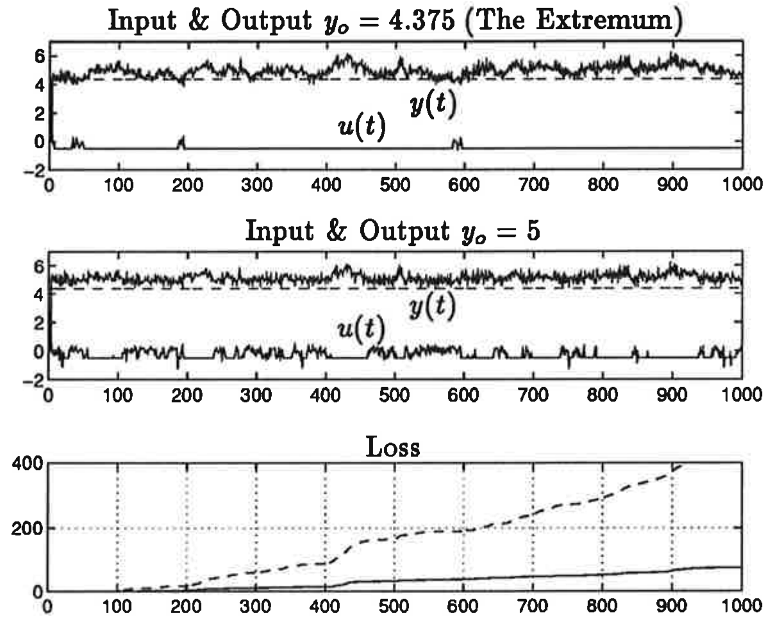


Figure 2.3: Simulation using the One Step controller. In the input and output plots the dashed line equals the extremum. In the Loss plot, the solid line equals $y_o = 5$ and the dashed line equals $y_o = 4.375$.

variance control. By comparing plots of the input and output, for different y_o values, it can be seen that by backing off from the extremum, going from $y_o = 4.375$ to 5.0, minimum variance controls can be applied more often. This is as expected as by backing off from the minimum the controller has a greater range to operate in and hence the probability of a minimum variance control being implementable increases.

The above discussion agrees with the theory in that by examining the expression for Υ, it can be seen that when y_o is near the extremum, the probability of Υ being greater than zero is very slim, as all the other terms are negative. Hence as y_o approaches the extremum the minimum variance roots will probably be complex, as Υ will probably be negative, hence the only root that can be applied is the constant control. By comparing the plots of the control signals with the corresponding output, the previous discussion is again supported by the control jumping to the -0.5 root when the output is just above the extremum, and appling a minimum variance control when the output goes below the extremum.

From the plot of the loss, the effect of the set point, y_o, is apparent by the loss for $y_o = 4.375$ increasing at a consistently higher rate than the $y_o = 5$ plot. The dramatic change in loss comes not only from the variance being higher, for the reasons discussed above, but also from the mean value not being equal to y_o, as Table 2.2 shows. The error in the mean value near the extremum is due to the controller being unable to push the output down, when the noise pushes it up. Hence the output floats above the set point and, in doing so, generates a considerably greater loss.

2.3 Hybrid Extremum Control

The previous section showed that the One Step controller gives poor results, with the closed loop variance going to the open loop variance and the mean value drifting far from what it should be, when the reference input approaches the extremum. This section looks to improve the performance of the One Step controller by solving for the control in a different way and also by combining it with some aspects of the expectation controller.

The objective of reformulating the minimum variance problem is to try and get a controller that will increase the probability of Υ being positive and hence, increase the probability of being able to apply minimum variance controls.

The cost function given in expression (2.7) can be rewritten as follows

$$\begin{aligned} J &= E\left\{[\hat{y}(t+1) - y_o]^2 \,|\mathcal{Y}_t\right\} \\ J &= E\left\{p(t)^2|\mathcal{Y}_t\right\} + \sigma^2 \end{aligned} \tag{2.15}$$

where σ is the standard deviation of the noise sequence.

Minimizing the above cost function is the same as minimizing $p(t)^2$, hence the following derivation looks to minimize $p(t)^2$. The first step in minimizing $p(t)^2$ is to separate out the parts in $p(t)$ that depend on $u(t)$. To achieve this, the following polynomial partitions are used.

$$C = 1 - z^{-1}\bar{C} \tag{2.16}$$

$$B_1 = b_{10} - z^{-1}\bar{B}_1 \tag{2.17}$$

$$B_2 = b_{20} - z^{-1}\bar{B}_2 \tag{2.18}$$

By performing some manipulations on (2.16), the following result is obtained.

$$\begin{aligned} \frac{1}{C} &= \frac{1}{1 - z^{-1}\bar{C}} = \left(\frac{1 - z^{-1}\bar{C} + z^{-1}\bar{C}}{1 - z^{-1}\bar{C}}\right) = \left(1 + \frac{z^{-1}\bar{C}}{1 - z^{-1}\bar{C}}\right) \\ &= \left(1 + \frac{z^{-1}\bar{C}}{C}\right) \end{aligned}$$

Now by substituting the above result into (2.10), the expression for $p(t)$ becomes

$$p(t) = \frac{B_0}{C} + B_1\left(1 + \frac{z^{-1}\bar{C}}{C}\right)u(t) + B_2\left(\frac{1}{1 - z^{-1}\bar{C}}\right)u^2(t) + \frac{G}{C}y(t) - y_o$$

and now by substituting in (2.17) and (2.18)

$$\begin{aligned}
p(t) &= \frac{B_0}{C} + b_{10}u(t) - \bar{B}_1 u(t-1) + \frac{B_1\bar{C}}{C}u(t-1) \\
&\quad + b_{20}u^2(t) - \bar{B}_2u^2(t-1) + \frac{B_2\bar{C}}{C} + \frac{G}{C}y(t) - y_o \\
&= b_{10}u(t) + b_{20}u^2(t) + \frac{B_0}{C} + \frac{B_1\bar{C} - \bar{B}_1C}{C}u(t-1) \\
&\quad + \frac{B_2\bar{C} - \bar{B}_2C}{C}u^2(t-1) + \frac{G}{C}y(t) - y_o \\
&= b_{10}u(t) + b_{20}u^2(t) + w(t) \qquad (2.19)
\end{aligned}$$

where

$$\begin{aligned}
w(t) &= \frac{B_0}{C} + \frac{B_1\bar{C} - \bar{B}_1C}{C}u(t-1) + \frac{B_2\bar{C} - \bar{B}_2C}{C}u^2(t-1) \\
&\quad + \frac{G}{C}y(t) - y_o \qquad (2.20)
\end{aligned}$$

With $p(t)$ in this from the cost function becomes

$$J = \left[b_{10}u(t) + b_{20}u^2(t) + w(t)\right]^2 + \sigma^2 \qquad (2.21)$$

To find the extremum points of (2.21), the differential is taken and set to zero as follows.

$$\frac{\partial J}{\partial u(t)} = 2\left[b_{10}u(t) + b_{20}u^2(t) + w(t)\right]\left[b_{10} + 2b_{20}u(t)\right] = 0 \qquad (2.22)$$

Solving the above equation, assuming $b_{20} > 0$, gives

$$u(t) = \begin{cases} -\frac{b_{10}}{2b_{20}} \pm \sqrt{\left(\frac{b_{10}}{2b_{20}}\right)^2 - \frac{w(t)}{b_{20}}} & w(t) < \frac{b_{10}^2}{4b_{20}} \\ -\frac{b_{10}}{2b_{20}} & w(t) \geq \frac{b_{10}^2}{4b_{20}} \end{cases} \qquad (2.23)$$

By substituting in the parameters of the example system it can be found that the above control is exactly the same as the control produced by the One Step controller, (2.14). Also the origin of the -0.5 root of the One Step controller becomes apparent, in that it is obtained by setting the right hand bracket in (2.22) to zero. Setting this bracket to zero, corresponds to minimizing the one step ahead expectation cost function, shown below.

$$J = E\left\{[\hat{y}(t+1) - y_o]\,|\mathcal{Y}_t\right\} \qquad (2.24)$$

The plot of the variance cost function, (2.21), using the example system parameters is given in Figure 2.4. The plot shows that the variance cost

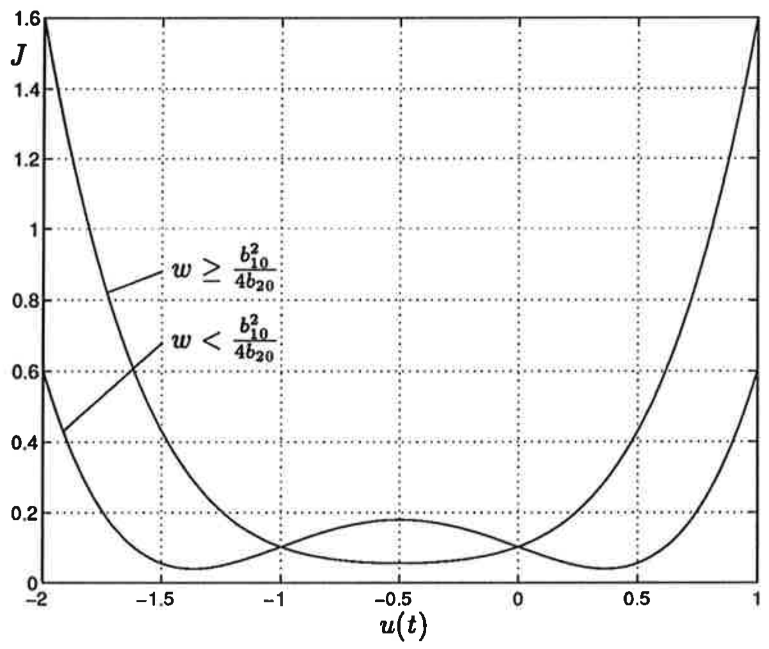


Figure 2.4: Visualization of the variance cost function, (2.21). Parabolic curve $w(t) = 0.25$, fourth order curve $w(t) = -0.25$.

function basically consists of two different cost functions, for the two different conditions on $w(t)$. The parabola plot is the one step ahead minimum expectation cost function and the fourth order polynomial curve is the one step ahead minimum variance cost function.

The advantage of reformulating the system in the above way, comes from first noting that the probability of being able to use the minimum variance control would be increased, if the probability of $w(t)$ being small could be increased. One way to increase the probability of $w(t)$ being small would be to obtain a one step ahead prediction of $w(t)$, substitute into it both the minimum variance controls, when they can be used, and choose the control that gave the smallest prediction of $w(t)$. The prediction of $w(t)$, $\hat{w}(t+1 \mid t)$ is obtained by again using the C partition, (2.16), along with the polynomial partition of G, as follows.

$$G = g_0 - z^{-1}\bar{G} \tag{2.25}$$

Combining the partitions with the expression for $w(t)$ given in (2.20), gives

$$\begin{aligned}\hat{w}(t+1 \mid t) = \frac{B_0}{C} + \frac{B_1\bar{C} - \bar{B}_1 C}{C}u(t) + \frac{B_2\bar{C} - \bar{B}_2 C}{C}u^2(t) \\ + g_0\hat{y}(t+1 \mid t) + \frac{G\bar{C} - \bar{G}C}{C}y(t) - y_o\end{aligned} \tag{2.26}$$

y_o	One Step w controller		
	Variance	Loss	Mean
4.375	0.1300	0.5542	5.0262
5.0	0.0696	0.0877	5.1345
7.0	0.04	0.04	7.0028
10.0	0.04	0.04	10.0027

Table 2.3: Simulation results for the One Step controller with predicted $w(t)$ minimization

which for the example system becomes

$$\hat{w}(t+1 \mid t) = \frac{1}{C}\Big(b_o - (1+c)y_o + (b_{11} - cb_{10})u(t) - cb_{20}u^2(t) + (c-a)\hat{y}(t+1 \mid t)\Big) \tag{2.27}$$

Note that by substituting in the actual system parameters, the $(b_{11}-cb_{10})$ term becomes zero. Hence the only term that affects the minimization of $\hat{w}(t+1 \mid t)$ is $cb_{20}u^2(t)$, thus the minimum variance control that has the lowest magnitude should be chosen. By examining (2.23) it can be found that the control generated by the positive sign will always have the lowest magnitude. At this point it might seem that the prediction, $\hat{w}(t+1 \mid t)$, does not need to be calculated during the simulation as the positive control should always be applied. This is true for the known parameter case but in the unknown parameter case, to be discussed later, the $(b_{11} - cb_{10})$ term will almost never be exactly zero. Hence in the proceeding programs the minimization of $\hat{w}(t+1 \mid t)$ is always carried out.

The Matlab program that implements the $\hat{w}(t+1 \mid t)$ minimization is called one_step_w.m and is as given on page 63 of appendix A. Using this program the results given in Table 2.3 can be obtained. The table shows that increasing the probability of $\hat{w}(t+1 \mid t)$ being small has improved the results, although not markably. The results at the extremum have not appreciably changed as a minimum variance control can almost never be applied, hence the minimization of $\hat{w}(t+1 \mid t)$ can not be achieved. In contrast to this, when y_o is increased from the extremum and the minimization of $\hat{w}(t+1 \mid t)$ can be applied much more often, the results are slightly better. This is of course due to the minimization of $\hat{w}(t+1 \mid t)$ increasing the amount of times the minimum variance control can be applied.

Although the minimization of $\hat{w}(t+1)$ improved the performance of the controller when the y_o is removed from the extremum, it had almost no effect when y_o was at the extremum. Hence some means of now improving the results when the system is at the extremum is required.

To improve the performance at the extremum, it should first be noted

y_o	Hybrid controller		
	Variance	Loss	Mean
4.375	0.0745	0.1100	4.5635
5.0	0.0747	0.0980	5.1525
7.0	0.0432	0.0432	7.0094
10.0	0.04	0.04	10.0027

Table 2.4: Simulation results for the Hybrid controller

that the high loss at the extremum is mainly due to the mean value of the output being far from what it should be. The discrepancy is due to the constant control being equal to the minimum of the one step ahead expectation cost function, (2.24). Hence the constant control only minimizes the short term loss, which would be fine if it was only applied for one step, but when y_o is at the extremum it is applied almost all the time. One solution to this problem is to use the expectation controller, (2.2), in place of the $u(t) = -0.5$, constant control, i.e. when a minimum variance control can not be applied, the One Step controller should switch to the Expectation controller. This effectively changes the horizon of the constant control from one to infinity. The new Hybrid controller can be expressed as in (2.28), with its implementation being achieved by the Matlab program hybrid.m, given on page 65 of appendix A. This program was used to produce the results in Table 2.4.

$$u(t) = \begin{cases} -\frac{b_{10}}{2b_{20}} \pm \sqrt{\left(\frac{b_{10}}{2b_{20}}\right)^2 - \frac{w(t)}{b_{20}}} & w(t) < \frac{b_{10}^2}{4b_{20}} \\ -\frac{B_1(1)}{2B_2(1)} \pm \sqrt{\left[\left(\frac{B_1(1)}{2B_2(1)}\right)^2 + \frac{A(1)y_o - B_o}{B_2(1)}\right]} & w(t) \geq \frac{b_{10}^2}{4b_{20}} \end{cases} \tag{2.28}$$

Table 2.4 shows that the hybrid controller gives a much better result at the extremum, but the performance for set points just of the extremum, it does not perform as well as the One Step controller with $w(t)$ minimization. The reason for this loss in performance, is due to the fact that to increase the probability of being able to apply the minimum variance control, the input of the system should be set to the extremum value, regardless of the set point, when minimum variance control is not possible. Hence as the expectation controller controls to make the output equal to the set point and not the extremum, the controllers performance is not that good away from the extremum.

To correct the deficiency in the Hybrid controller, the Hybrid Extremum controller sets the control to the extremum, -0.25 for the example system, when the minimum variance control can not be applied. The implementation of this controller is achieved by the program hybrid_extremum.m, given on page 65 of appendix A and was used to generate the results in Table 2.5.

y_o	Hybrid Extremum controller		
	Variance	Loss	Mean
4.375	0.0745	0.1100	4.5636
5.0	0.0463	0.0479	5.0399
7.0	0.04	0.04	7.0029
10.0	0.04	0.04	10.0027

Table 2.5: Simulation results for the Hybrid Extremum controller

From Table 2.5 it can be seen that the hybrid extremum controller gives good results. The variance at the extremum has been reduced by 43% and the loss has been reduced by 16% when compared with the expectation controller. The reduction in the loss is smaller than the reduction in the variance because the mean value created by the hybrid extremum controller is slightly higher than the mean value of the expectation controller. The off-set in the mean value is due to minimum variance controls always correcting any output value that is below the minimum, but due to the input constraint it can not correct the output values that are above the extremum. Hence the output mean value will always be slightly higher than the extremum.

The expression for the Hybrid Extremum controller is as follows.

$$u(t) = \begin{cases} -\frac{b_{10}}{2b_{20}} \pm \sqrt{\left(\frac{b_{10}}{2b_{20}}\right)^2 - \frac{w(t)}{b_{20}}} & w(t) < \frac{b_{10}^2}{4b_{20}} \\ -\frac{B_1(1)}{2B_2(1)} & w(t) \geq \frac{b_{10}^2}{4b_{20}} \end{cases} \tag{2.29}$$

The input, output and loss plots of the Hybrid Extremal controller are shown in Figure 2.5.

From Figure 2.5 it can be observed that the number of times the minimum variance control is applied, is more than any other controller and that the mean value of the output, is the closer to the extremum than any other controller. Hence, as the loss plot shows, the Hybrid Extremum controller has the best performance out of all the previously discussed controllers.

A good way to visualize the Hybrid Extremum controller is shown in Figure 2.6 where the control value is plotted against $w(t)$.

2.4 Conclusion

This chapter has looked at various ways of reducing the variance about a given set point, of the quadratic Hammerstein model. The main controllers that were looked at are as follows.

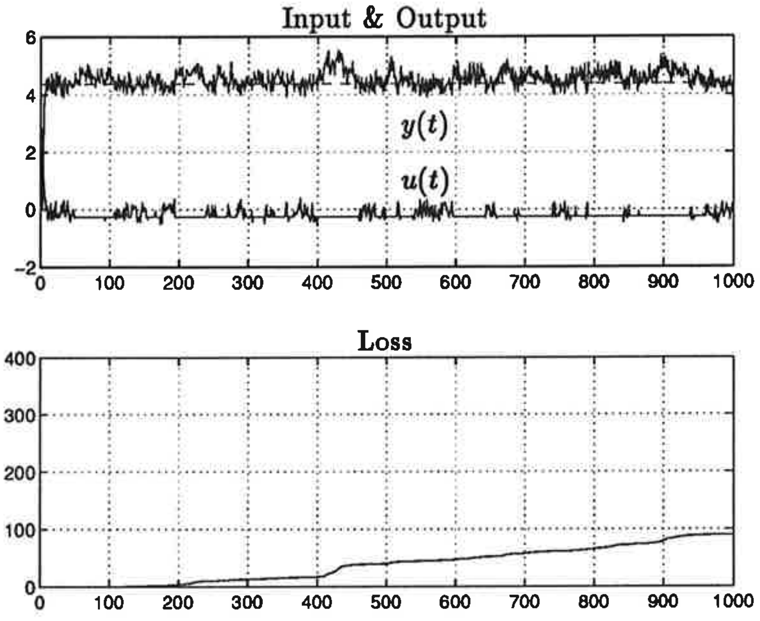


Figure 2.5: Simulation using the Hybrid Extremum controller with y_o = 4.375. Dashed line equals the extremum, 4.375.

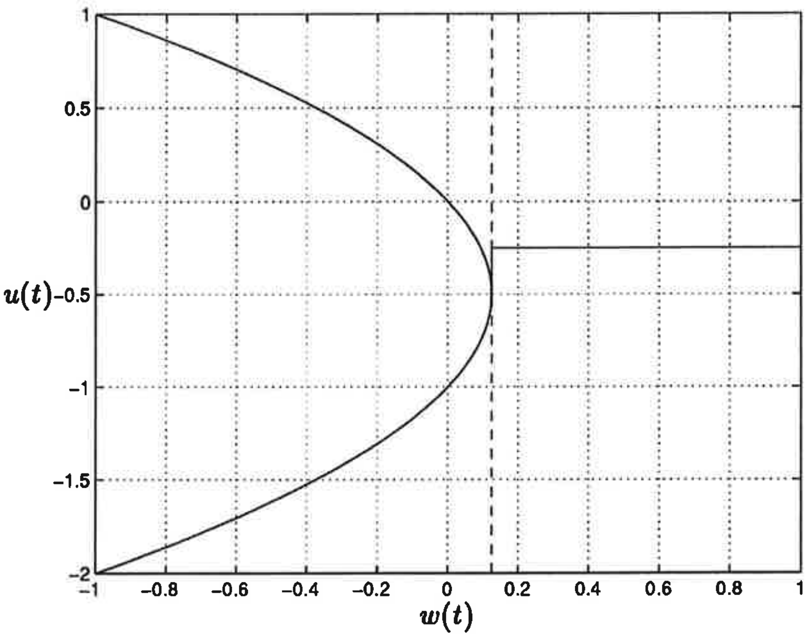


Figure 2.6: Visualization of the Hybrid Extremum controller. Dashed line equals the extremum in terms of $w(t)$.

Expectation controller

$$u_o = -\frac{B_1(1)}{2B_2(1)} \pm \sqrt{\left[\left(\frac{B_1(1)}{2B_2(1)}\right)^2 + \frac{A(1)y_o - B_o}{B_2(1)}\right]}$$

One Step controller

$$u(t) = \begin{cases} -0.5 & \text{if } \Upsilon < 0, \\ -0.5 + 0.2236\sqrt{\Upsilon} & \text{if } \Upsilon > 0, \\ -0.5 - 0.2236\sqrt{\Upsilon} & \text{if } \Upsilon > 0. \end{cases}$$

Hybrid controller

$$u(t) = \begin{cases} -\frac{b_{10}}{2b_{20}} \pm \sqrt{\left(\frac{b_{10}}{2b_{20}}\right)^2 - \frac{w(t)}{b_{20}}} & w(t) < \frac{b_{10}^2}{4b_{20}} \\ -\frac{B_1(1)}{2B_2(1)} \pm \sqrt{\left[\left(\frac{B_1(1)}{2B_2(1)}\right)^2 + \frac{A(1)y_o - B_o}{B_2(1)}\right]} & w(t) \geq \frac{b_{10}^2}{4b_{20}} \end{cases}$$

Hybrid Extremum controller

$$u(t) = \begin{cases} -\frac{b_{10}}{2b_{20}} \pm \sqrt{\left(\frac{b_{10}}{2b_{20}}\right)^2 - \frac{w(t)}{b_{20}}} & w(t) < \frac{b_{10}^2}{4b_{20}} \\ -\frac{B_1(1)}{2B_2(1)} & w(t) \geq \frac{b_{10}^2}{4b_{20}} \end{cases}$$

Figure 2.7 shows the loss, at the extremum, of each of the above controllers. Note that the loss plot at the extremum for the Hybrid controller and the Hybrid Extremum controller are identical, as they both have the same constant control at the extremum.

From Figure 2.7 it can be seen that the Hybrid Extremum controller has the best performance. The figure also shows that although the Expectation controller is the most primitive out of the controllers analyzed, it has very good performance when compared with the One Step controller. This is maily due to the One Step controller, jumping to the one step ahead minimum expectation control, when the input constraint in the Hammerstein model, makes it impossible to apply a minimum variance control. The Hybrid controller gets round this problem, by reformulating the problem is such a way that the constant control of the One Step controller, can be replaced with the Expectation controller. Hence the performance of the Hybrid controller at the extremum is a great improvement on the One Step controller and is a significant improvement on the Expectation controller. Also in the reformulation of the One Step problem it was found that the probability of being able to apply the minimum variance control on the next step, could be increased by minimizing the predicted value of w.

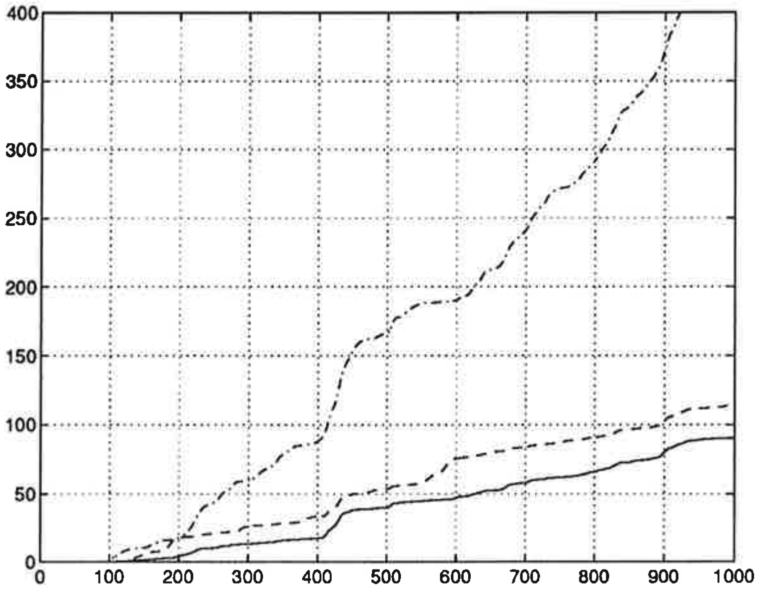


Figure 2.7: Comparison of the loss at the extremum for the following controllers. The solid line equals the Hybrid Extremum, the dashed line equals the Expectation and the dash-dot line equals the One Step.

In the analysis of the Hybrid controller, it was found that its performance was not as good as the One Step controller, when the set point was away from the extremum. It was found that better results could be obtained from keeping the constant control set to produce the extremum, independently of the set point. Hence as the Hybrid controller changes its constant control to follow the set point, its performance away from the extremum is poor. The Hybrid Extremum controller implemented the idea of keeping the constant control set to achieve the extremum at the output.

Thus the main conclusion of this chapter is that the Hybrid Extremum controller, one step ahead minimum variance control with constant root set to produce the extremum, is the best controller out of the ones considered.

Chapter 3

Generalized Predictive Control

In the analysis of the One Step controller in the preceding chapter, it was found that when the minimum variance roots could not be applied, the controller jumped to a constant control. It was then found that the value of this constant control was not the actual minimising value, as it was only the solution to the one step ahead expectation cost function. To try and improve the performance of the One Step controller, the control horizon is now increased from one step to many steps, hence Generalized Predictive Control (GPC).

3.1 Hammerstein Generalized Predictive Control

The One Step controller minimized the following cost function.

$$J = E\left\{[\hat{y}(t+1) - y_o]^2 \mid \mathcal{Y}_t\right\} \tag{3.1}$$

As this gave poor results due to (3.1) only looking one step ahead, the following cost function is now introduced to try and correct the problem.

$$J(N) = E\left\{\sum_{i=1}^{N} [\hat{y}(t+i) - y_o]^2 \mid \mathcal{Y}_t\right\} \tag{3.2}$$

Note that the example system is not the standard system model used for generalized predictive control in that it has no integral term, see [ÅW95]. This was removed from the model because in this case the assumption is made that the only disturbance is the noise, $e(t)$, which has a mean of zero.

To derive the controller that minimizes (3.2) with the system model being of the Hammerstein type, two polynomial partitions are required. The first partition undertakes the same role as in normal minimum variance control, in that it allows $y(t+1)$ to be expressed as the sum of two uncorrelated terms, the prediction at the i^{th} step and the prediction error. To achieve the first partition the following polynomial expression is used, where F_i and G_i are polynomials in z^{-1} with degrees $i-1$ and $max(n_a-1, n_c-i)$ respectively.

$$C = AF_i + z^{-i}G_i \tag{3.3}$$

The above partition should be performed for $i = 1, 2, \ldots, N$ so that for $N > 1$, two sequences of polynomials, $F_1, F_2, \ldots, F_N$ and $G_1, G_2, \ldots, G_N$, are generated.

Now by combining the system model, (1.2), and (3.3), $y(t+i)$ can be expressed in the following form.

$$\begin{aligned}(C - z^{-i}G_i)y(t) &= B_oF_i + B_1F_iu(t-1) + B_2F_iu^2(t-1) + CF_ie(t)\\ y(t+i) &= \frac{1}{C}\Big(B_oF_i + B_1F_iu(t+i-1) + B_2F_iu^2(t+i-1)\\ &\quad + G_iy(t)\Big) + F_ie(t+i)\end{aligned} \tag{3.4}$$

With $y(t+i)$ in the above form, the second polynomial partition, which separates the optimization variables $u(t+N-1), \ldots, u(t)$, $u^2(t+N-1), \ldots, u^2(t)$ from the data $u(t-1), u(t-2), \ldots$, $u^2(t-1), u^2(t-2), \ldots$, can be performed. The polynomial expression that achieves this partition is as follows, where E_{ij} and Γ_{ij} are polynomials in z^{-1} with degrees $i-1$ and $max(n_b-1, n_c-1)$ respectively.

$$B_jF_i = CE_{ij} + z^{-i}\Gamma_{ij} \qquad j = 1, 2 \tag{3.5}$$

Note that it is at this point that the derivation using the Hammerstein model diverges from the normal linear generalized predictive control derivation, see [WZ91]. The divergence comes from the powers of the input terms requiring extra E's and Γ's, hence for the quadratic Hammerstein model, there will be four sequences of polynomials $E_{11}, \ldots, E_{N1}, E_{12}, \ldots, E_{N2}$ and $\Gamma_{11}, \ldots, \Gamma_{N1}, \Gamma_{12}, \ldots, \Gamma_{N2}$. By substituting (3.5) into (3.4) the derivation proceeds as follows.

$$\begin{aligned}y(t+i) &= \frac{1}{C}\Big(B_oF_i + (CE_{i1} + z^{-i}\Gamma_{i1})u(t+i-1)\\ &\quad + (CE_{i2} + z^{-i}\Gamma_{i2})u^2(t+i-1)\\ &\quad + G_iy(t)\Big) + F_ie(t+i)\end{aligned}$$

$$y(t+i) = \frac{1}{C}\Big(B_oF_i + CE_{i1}u(t+i-1) + CE_{i2}u^2(t+i-1) + \Gamma_{i1}u(t-1) + \Gamma_{i2}u^2(t-1) + G_iy(t)\Big) + F_ie(t+i)$$

Thus the prediction of $y(t+i)$ given data up to t is

$$\hat{y}(t+i \mid t) = \frac{1}{C}\Big(B_oF_i + CE_{i1}u(t+i-1) + CE_{i2}u^2(t+i-1) + \Gamma_{i1}u(t-1) + \Gamma_{i2}u^2(t-1) + G_iy(t)\Big) \tag{3.6}$$

Hence with expression (3.6) the cost function for the GPC, (3.2), can be evaluated. The following two sections look at minimizing (3.2), for a control horizon of two, with first setting the C polynomial in the example system equal to one and then with C as normal.

3.2 The C Equal to One Case

As the C polynomial makes things much more complicated, it will first be assumed to be equal to one. By substituting in the example system parameters, with $C = 1$, and using the polynomial partitions, the following F, G, E and Γ polynomials are obtained, for a control horizon of two.

$F_1 = 1$	$E_{11} = 0.5$	$E_{12} = 0.5$
$F_2 = 1 + 0.95z^{-1}$	$E_{21} = 0.5 + 0.225z^{-1}$	$E_{22} = 0.5 + 0.475z^{-1}$
$G_1 = 0.95$	$\Gamma_{11} = -0.25$	$\Gamma_{12} = 0$
$G_2 = 0.9025$	$\Gamma_{21} = -0.2375$	$\Gamma_{22} = 0$

Using the above polynomials in (3.6) the following expressions can be obtained.

$$\hat{y}(t+1 \mid t) = 0.25 + 0.5u(t) - 0.25u(t-1) + 0.5u^2(t) + 0.95y(t) \tag{3.7}$$

$$\hat{y}(t+2 \mid t) = 0.4875 + 0.5u(t+1) + 0.225u(t) + 0.5u^2(t+1) + 0.475u^2(t) - 0.2375u(t-1) + 0.9025y(t) \tag{3.8}$$

By using (3.7) and (3.8), the cost function (3.2), with the control horizon equal to two, can be expressed as follows

$$J(2) = \Big[(\hat{y}(t+1 \mid t) - y_o)^2 + (\hat{y}(t+2 \mid t) - y_o)^2\Big] \tag{3.9}$$

Before derivatives are taken, it should be noted that $\hat{y}(t+2 \mid t)$ contains both $u(t)$ and $u(t+1)$, hence there will not be any neat expression for the derivative of the cost function with respect to u(t). This means that an analogous solution to the hybrid controllers, (2.22), is not possible as the derivative with respect to $u(t)$ can not be factored. The derivatives of (3.9) with respect $u(t)$ and $u(t+1)$ are as follows. Note that the Matlab Symbolic Toolbox program, two_steps_calc_C_1.m given on page 65 of appendix A, was used to take the differentials and collect terms.

$$\begin{aligned}\frac{\partial J(2)}{\partial u(t)} = {} & 1.9025u^3(t) + 2.14125u^2(t) + \big(2.0275 - 3.9y_o - 0.95125u(t-1) \\ & + 3.61475y(t) + 0.95u(t+1) + 0.95u^2(t+1)\big)u(t) + 0.469375 \\ & - 0.356875u(t-1) + 1.356125y(t) + 0.225u(t+1) \\ & + 0.225u^2(t+1) - 1.45y_o \end{aligned} \tag{3.10}$$

$$\begin{aligned}\frac{\partial J(2)}{\partial u(t+1)} = {} & u^3(t+1) + 1.5u^2(t+1) + \big(1.475 - 0.475u(t-1) + 0.45u(t) \\ & - 2y_o + 0.95u^2(t) + 1.805y(t)\big)u(t+1) + 0.4875 \\ & - 0.2375u(t-1) + 0.225u(t) - y_o + 0.475u^2(t) \\ & + 0.9025y(t) \end{aligned} \tag{3.11}$$

To find the minimizing $u(t)$, both of the above equations have to be solved simultaneously for $u(t)$ and $u(t+1)$. Note that although $u(t+1)$ is found, it is only $u(t)$ that is applied to the system. This type of control is called receding horizon control and in this case the horizon is two steps. Using two_steps_calc_C_1 again to do the algebra, the following three results are obtained. Note that x is just a dummy variable.

$$\begin{aligned} u(t)_i &= RootOf\Big(10x^2 + 10x + 5 + 19y(t) - 5u(t-1) - 20y_o\Big) \\ u(t+1) &= RootOf\Big(10x^2 + 10x + 5 - 5u(t)_i - y_o\Big) \qquad i = 1,2 \end{aligned} \tag{3.12}$$

$$\begin{aligned} u(t) &= RootOf\Big(15220x^3 + 17130x^2 + \big(14320 + 28918y(t) \\ & \quad - 7610u(t-1) - 31200y_o\big)x + 3305 + 10849y(t) \\ & \quad - 2855u(t-1) - 11600y_o\Big) \\ u(t+1) &= -0.5 \end{aligned} \tag{3.13}$$

$$\begin{aligned} u(t) &= -0.5 \\ u(t+1) &= RootOf\Big(400x^2 + 400x + 395 - 190u(t-1) \\ & \quad + 722y(t) - 800y_o\Big) \end{aligned} \tag{3.14}$$

y_o	One Step		Two Steps	
	Variance	Loss	Variance	Loss
4.375	0.3211	0.9082	0.2208	0.4420
5.0	0.1897	0.3326	0.1246	0.1767
7.5	0.0502	0.0518	0.0465	0.0472
10.0	0.0406	0.0407	0.0405	0.0405

Table 3.1: Simulation results for the One Step and Two Steps Controllers, with $C = 1$.

The above three solutions give nine different roots. As there is more than one valid solution to the cost function, some means of deciding what solution to use is required. The most obvious and simple way to check for minimality is to substitute the possible $u(t)$'s and $u(t+1)$'s into the cost function and then select the non-complex root that gives the lowest cost. During the simulations it was found that sometimes a few of the nine roots give the same cost of zero, but with different controls. It was also found that if, out of the roots that have the same minimal cost, the root that is most similar to the previous control is selected, a slightly lower loss is achieved. Selecting this root gives a lower loss because it prevents the system from jumping from one state to another. This control strategy along with the root selection scheme is implemented in the program two_steps_C_1.m shown on page 66 of appendix A. Table 3.1 compares the One Step controller against the new Two Steps controller. Note as in the last section, 5000 steps are simulated with the first 100 being omitted from the calculations.

From Table 3.1 it can be observed that the loss at the extremum has been reduced by 50% with the two steps controller.

With the success of the Two Steps controller, it is now apt to look at the more general and more complicated case of $C \neq 1$.

3.3 The C as in the Example System Case

By using the example system parameters and the polynomial partitions, with C now as in the example system, the following F, G, E and Γ polynomials are obtained.

$$\begin{array}{lll} F_1 = 1 & E_{11} = 0.5 & E_{12} = 0.5 \\ F_2 = 1 + 0.45z^{-1} & E_{21} = 0.5 + 0.225z^{-1} & E_{22} = 0.5 + 0.475z^{-1} \\ G_1 = 0.45 & \Gamma_{11} = 0 & \Gamma_{12} = 0.25 \\ G_2 = 0.4275 & \Gamma_{21} = 0 & \Gamma_{22} = 0.2375 \end{array}$$

Using the above polynomials in (3.6) the following expressions can be obtained.

$$\begin{aligned}\hat{y}(t+1 \mid t) &= 0.25 + 0.5u(t) - 0.25u(t-1) + 0.5u^2(t) + 0.45y(t) \\ &\quad + 0.5\hat{y}(t \mid t-1) && (3.15)\\ \hat{y}(t+2 \mid t) &= 0.3625 + 0.5u(t+1) - 0.025u(t) - 0.1125u(t-1) \\ &\quad + 0.5u^2(t+1) + 0.225u^2(t) + 0.4275y(t) \\ &\quad + 0.5\hat{y}(t+1 \mid t-1) && (3.16)\end{aligned}$$

It is at this point that the complication of $C \neq 1$ becomes apparent in that (3.16) can not be differentiated in its present form because $\hat{y}(t+1 \mid t-1)$ depends on $u(t)$. Hence $\hat{y}(t+1 \mid t-1)$ must be replaced by the expression for $\hat{y}(t+2 \mid t)$ but delayed by one step. Thus

$$\begin{aligned}\hat{y}(t+2 \mid t) &= 0.3625 + 0.5u(t+1) - 0.025u(t) - 0.1125u(t-1) \\ &\quad + 0.5u^2(t+1) + 0.225u^2(t) + 0.4275y(t) + 0.5\Big(0.3625 \\ &\quad + 0.5u(t) - 0.025u(t-1) - 0.1125u(t-2) + 0.5u^2(t) \\ &\quad + 0.225u^2(t-1) + 0.4275y(t-1) + 0.5\hat{y}(t \mid t-2)\Big) \\ &= 0.54375 + 0.5u(t+1) + 0.225u(t) - 0.125u(t-1) \\ &\quad - 0.05625u(t-2) + 0.5u^2(t+1) + 0.475u^2(t) \\ &\quad + 0.1125u^2(t-1) + 0.4275y(t) + 0.21375y(t-1) \\ &\quad + 0.25\hat{y}(t \mid t-2) && (3.17)\end{aligned}$$

Expression (3.17) can now be differentiated along with (3.15). Hence by following the same procedure as in the $C = 1$ case, the two expressions are substituted in to the cost function, (3.9), and the differentials are taken, using the program two_steps_calc.m on page 68 of appendix A, to give

$$\begin{aligned}\frac{\partial J(2)}{\partial u(t)} &= 1.9025u^3(t) + 2.14125u^2(t) + \big(2.134375 + \hat{y}(t \mid t-1) \\ &\quad - 0.7375u(t-1) + 1.71225y(t) - 3.900y_o + 0.95u(t+1) \\ &\quad + 0.475\hat{y}(t \mid t-2) - 0.106875u(t-2) + 0.95u^2(t+1) \\ &\quad + 0.21375u^2(t-1) + 0.406125y(t-1)\big)u(t) + 0.4946875 \\ &\quad + 0.5\hat{y}(t \mid t-1) - 0.30625u(t-1) + 0.642375y(t) \\ &\quad - 1.45y_o + 0.225u(t+1) + 0.1125\hat{y}(t \mid t-2) - 0.0253u(t-2) \\ &\quad + 0.225u^2(t+1) + 0.0506u^2(t-1) + 0.0962y(t-1) && (3.18)\end{aligned}$$

$$\begin{aligned}\frac{\partial J(2)}{\partial u(t+1)} = {} & u^3(t+1) + 1.5u^2(t+1) + \big(1.5875 + 0.4275y(t-1) \\ & + 0.45u(t) - 0.25u(t-1) - 0.1125u(t-2) + 0.95u^2(t) \\ & + 0.22500u^2(t-1) + 0.855y(t) + 0.5\hat{y}(t \mid t-2) - 2y_o\big)u(t+1) \\ & + 0.54375 + 0.21375y(t-1) + 0.225u(t) - 0.125u(t-1) \\ & - 0.0563u(t-2) + 0.475u^2(t) + 0.1125u^2(t-1) \\ & + 0.4275y(t) + 0.25\hat{y}(t \mid t-2) - y_o \qquad (3.19)\end{aligned}$$

To find the minimizing $u(t)$ both of the above equations are solved in an analogous way to the $C = 1$ case. The results are as follows

$$\begin{aligned} u(t)_i &= RootOf\Big(10x^2 + 10x + 9y(t) + 10\hat{y}(t \mid t-1) \\ &\quad - 5u(t-1) - 20y_o + 5\Big) \\ u(t+1) &= RootOf\Big(400x^2 + 400x + 245 - 200u(t)_i + 90u(t-1) \\ &\quad - 45u(t-2) + 90u(t-1)^2 + 171y(t-1) + 200\hat{y}(t \mid t-2) \\ &\quad - 40y_o - 380\hat{y}(t \mid t-1)\Big) \qquad i = 1,2 \qquad (3.20)\end{aligned}$$

$$\begin{aligned} u(t) &= RootOf\Big(30440x^3 + 34260x^2 + \big(6498y(t-1) - 1710u(t-2) \\ &\quad + 30350 - 62400y_o - 11800u(t-1) + 27396y(t) + 3420u^2(t-1) \\ &\quad + 7600\hat{y}(t \mid t-2) + 16000\hat{y}(t \mid t-1)\big)x + 10278y(t) \\ &\quad + 8000\hat{y}(t \mid t-1) - 405u(t-2) - 4900u(t-1) + 810u^2(t-1) \\ &\quad + 1539y(t-1) - 23200y_o + 1800\hat{y}(t \mid t-2) + 7015\Big) \\ u(t+1) &= -0.5 \qquad (3.21)\end{aligned}$$

$$\begin{aligned} u(t) &= -0.5 \\ u(t+1) &= RootOf\Big(400x^2 + 400x + 440 - 100u(t-1) - 45u(t-2) \\ &\quad + 90u^2(t-1) + 342y(t) + 171y(t-1) \\ &\quad + 200\hat{y}(t \mid t-2) - 800y_o\Big) \qquad (3.22)\end{aligned}$$

This control strategy along with the root selection scheme, discussed in the last section, are implemented in the program two_steps.m shown on page 69 of appendix A. Table 3.2 gives the results of the Two Steps controller along with the One Step and Hybrid Extremum controllers.

From Table 3.2 it can be seen that the Two Steps controller performs much better than the One Step, by it reducing the loss by about 70%. The

y_o	One Step		Hybrid Extremum		Two Steps	
	Variance	Loss	Variance	Loss	Variance	Loss
4.375	0.1266	0.5572	0.0745	0.1100	0.0897	0.1571
5.0	0.0741	0.1098	0.0463	0.0479	0.0487	0.0513
7.0	0.0422	0.0426	0.04	0.04	0.04	0.04
10.0	0.0401	0.0401	0.04	0.04	0.04	0.04

Table 3.2: Simulation results for the Two Steps controller.

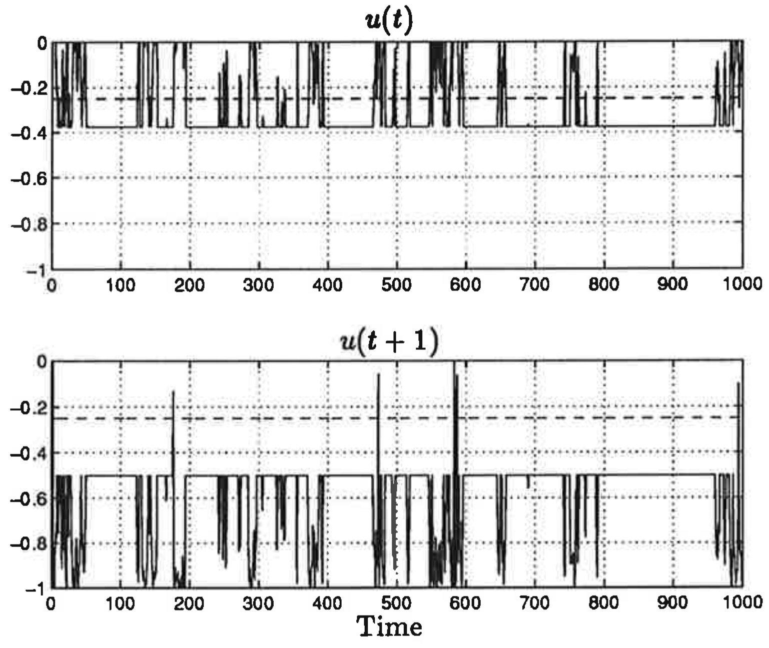


Figure 3.1: The Control signals for the Two Steps Controller. The dashed line equals the extremal control, -0.25.

reason for this reduction in the loss can be explained via Figure 3.1 which shows the $u(t)$ and $u(t+1)$ values.

Figure 3.1 shows that, when a minimum variance control can not be applied, the constant control that is applied has a value of -0.3752 and not -0.5 as the One Step controller would apply. The analysis of the Hybrid Extremum controller showed that, by increasing the control from -0.5 to -0.25, a lower loss can be achieved. Hence this explains why the loss of the Two Step controller is between the One Step and Hybrid Extremum controllers losses.

The reason why the constant control value has been increased from -0.5 to -0.3752 by the Two Steps controller, is that as the control horizon is now at two, the controller is much more considerate of the long term objectives, which have been showed to be satisfied more satisfactorily by increasing the constant control from -0.5 to -0.25. Also as the loss is measured over the entire 5000 point simulation, the objective of the Two Steps controller is closer to the objective of minimizing the loss than the One Step controller. Hence it should be expected that the Two Steps controller should have a lower loss than the One Step controller.

As the cost function is two dimensional, a good visualization of the cost function can be obtained for both, when the minimum variance and the constant control can be applied. The two cases of the cost function are shown in Figure 3.2

Figure 3.2 shows the two different forms the cost function can take for GPC with a horizon of two. When y_o is at the extremum, the cost function almost always looks as in the figure, with only one root being available for control as the others are complex. The root that is available for control corresponds to the constant control root. When $y_o = 10$ the converse occurs, with the cost function almost always being as shown in the figure, which shows that a minimum variance control can be applied as all the roots are real. Note that if a cross-section is taken across either axis of one of the two three dimensional plots, then a similar plot to that obtained for the one step cost function, see Figure 2.4, is obtained.

3.4 Extending the Control Horizon

The results for the Two Steps controller are promising in that the loss has been reduced by a significant amount, although not as much as the Hybrid Extremum controller. To try and get as good or even better results than the Hybrid Extremum controller, using the GPC technic, the control horizon is now increased from two to three.

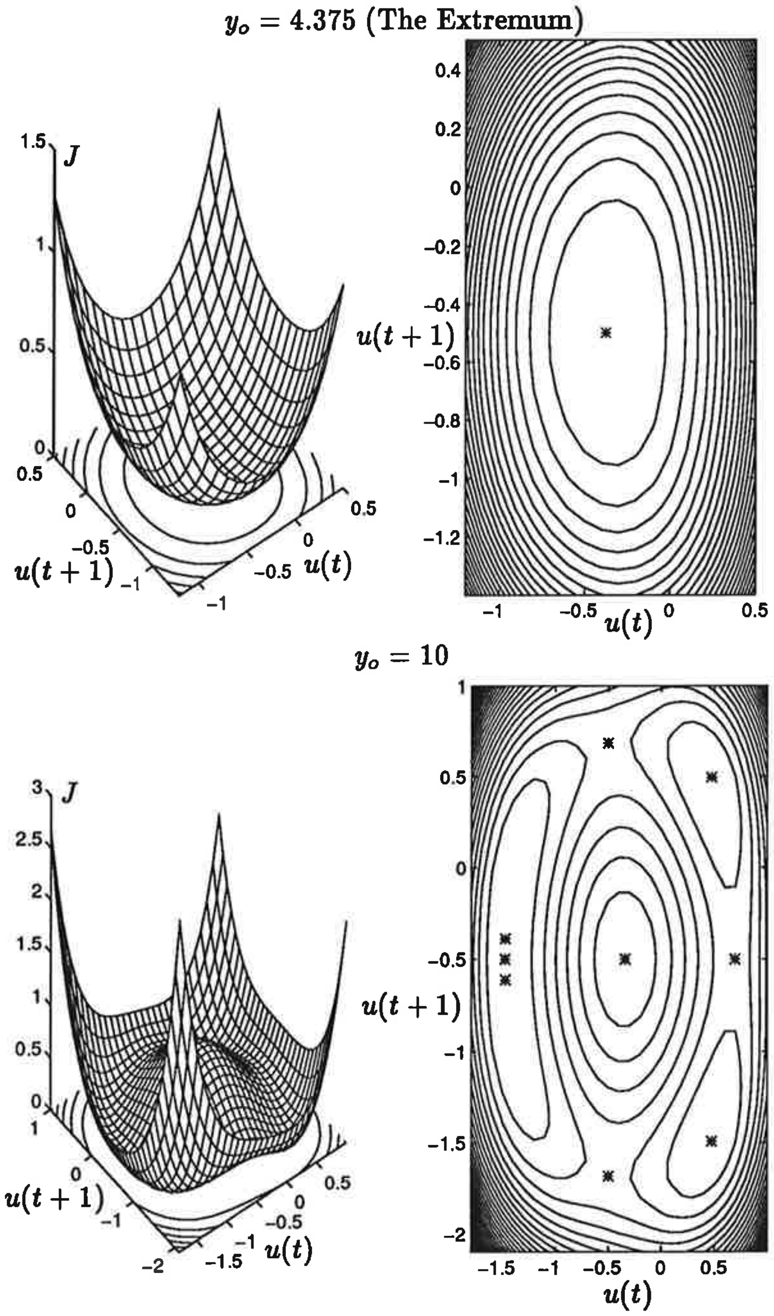


Figure 3.2: Visualization of the Cost Function at the 500th step of the simulation, $*$ shows root locations.

For the Three Steps controller the cost function, (3.2), becomes

$$J(3) = \left[(\hat{y}(t+1 \mid t) - y_o)^2 + (\hat{y}(t+2 \mid t) - y_o)^2 + (\hat{y}(t+3 \mid t) - y_o)^2\right] \tag{3.23}$$

To obtain $\hat{y}(t+3 \mid t)$, the polynomial partitions for the control horizon of two, are extended for a horizon of three and the following extra partitions are required.

$$\begin{aligned} F_3 &= 1 + 0.45z^{-1} + 0.4275z^{-2} & G_3 &= 0.406125 \\ E_{31} &= 0.5 + 0.225z^{-1} + 0.21375z^{-2} & E_{32} &= 0.5 + 0.475z^{-1} + 0.45125z^{-2} \\ \Gamma_{31} &= 0 & \Gamma_{32} &= 0.225625 \end{aligned}$$

Hence by using (3.6), the three steps ahead prediction becomes

$$\begin{aligned} \hat{y}(t+3 \mid t) = {} & 0.469375 + 0.5u(t+2) - 0.025u(t+1) + 0.10125u(t) \\ & - 0.106875u(t-1) + 0.5u^2(t+2) + 0.225u^2(t+1) \\ & + 0.21375u^2(t) + 0.406125y(t) \\ & + 0.5\hat{y}(t+2 \mid t-1) \end{aligned} \tag{3.24}$$

At this point in the Two Steps derivation both the derivatives of the cost function were taken, one with respect to $u(t+1)$ and one with respect to $u(t)$ and then these derivatives were set to zero and solved simultaneously. For the three steps case one extra derivative, with respect to $u(t+2)$, has to be taken and then all three differentials have to be set to zero and solved simultaneously. Theoretically there is no problem with this method of obtaining the Three Steps controller but in practice, trying to solve three third order polynomials simultaneously, which would give 27 different roots, is a very involved calculation, which most symbolic mathematical software packages (Matlab Symbolic Toolbox, Maple) can not handle. Hence as an alternative to obtaining a closed form solution, numerical optimization techniques can be used to obtain the control. This way of calculating the controls has the obvious disadvantage that the optimization procedure might converge to a local minima and not a global one. In this case this is not too much of a problem as the minima of the cost function, for the One Step and Two Step controllers have the same cost, hence it is reasonable to assume that the Three Steps will be the same. The optimization algorithm used, is from the Matlab Optimization Toolbox, which uses the Broyden, Fletcher, Goldfarb, Shanno (BFGS) formula, with a mixed quadratic and cubic polynomial interpolation and extrapolation line search algorithm, see [Fle80].

The algorithm for the Three Steps controller, three_steps.m, is as given on page 71 of appendix A. The main idea behind the program, is to find the controls using the BFGS formula, then use this control as the start point

y_o	Hybrid Extremum		Two Steps		Three Steps	
	Variance	Loss	Variance	Loss	Variance	Loss
4.375	0.0745	0.1100	0.0897	0.1571	0.0781	0.1414
5.0	0.0463	0.0479	0.0487	0.0513	0.0555	0.0689
7.0	0.04	0.04	0.04	0.04	0.0403	0.0405
10.0	0.04	0.04	0.04	0.04	0.04	0.04

Table 3.3: Simulation results for the Three Steps controller.

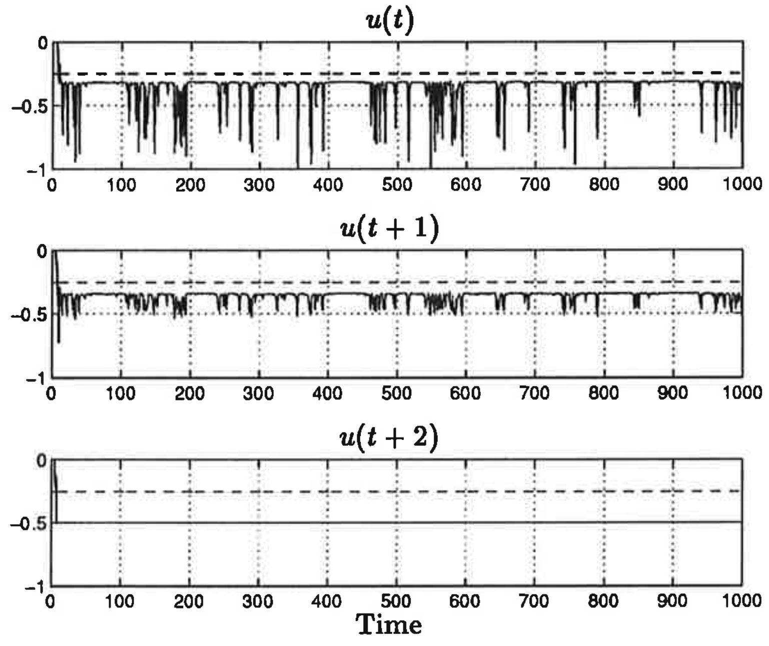


Figure 3.3: The Control signals for the Three Steps Controller. The dashed line equals the extremal control, -0.25.

of the optimization in the next step, when the cost function has changed with the new data. The results of the Three Steps controller are shown in Table 3.3.

Table 3.3 shows that the Three Steps controller gives a slightly better loss than the Two Steps but is still along way from the Hybrid Extremum controller. The slight improvement is due the mean value of $u(t)$, -0.3141, being slightly closer to -0.25 than the Two Steps. This can also be observed in Figure 3.3.

Table 3.3 also shows that the loss of the Three Steps controller is higher for $y_o = 5$. This increase in loss is probably due to the prediction, $\hat{y}(t+3 \mid t)$, being inaccurate due to the increased number of steps it has to predict into the future. This highlights the problem that will be faced if the control horizon is increased any further ahead than three, in that the prediction

$\hat{y}(t+i \mid t)$ will get more and more uncertain as i increases. Also as the cost function gives equal weighting to the $(\hat{y}(t+i \mid t) - y_o)^2$ terms, the performance of the controller based on this cost function will also become uncertain as i is increased. Hence it is more likely that a Four Steps controller would give a higher loss than the Three Steps controller.

3.5 Conclusion

The objective of using the generalized predictive control technique, was to improve on the performance of the One Step controller. The Two Steps controller did indeed improve on the results of the One step controller but it did not surpass the performance of the Hybrid Extremum controller. When the control horizon was then increased to three, hence the Three Steps controller, the improvement from the Two Steps controller was only very slight at the extremum and non existent away from the extremum. The failure of the Three Steps controller to reduce the loss was probably due to the third step ahead prediction of the output being inaccurate. The inaccuracy of the predictions would become more acute if the control horizon is extended any more, hence the limitation on the GPC idea becomes apparent in this case. Note that some weighting strategy on each prediction term in the cost function, could be employed to minimize the effect of the prediction inaccuracies i.e. multiply the first prediction by a large number and then multiply the predictions that look further into the future by smaller numbers. However this would be defeating the whole point of GPC, as by putting a low weight on the predictions that look further into the future, the effect of the GPC would on the constant control would not increase as the control horizon increased.

The main over riding flaw in the GPC, is that is does not give better results that the simple Expectation controller, even though it has a much more complex algorithm. Also it would be almost impossible to implement an adaptive version of even the Two Steps controller, as the control would have to be parameterized in terms of the system parameters, which would lead to infeasibly large equations. This effectively means that it would be impossible to convert the generalized predictive controllers, into Extremal Seeking controllers.

The main thing that can be taken from the GPC analysis, is that it supports the result that the One Step constant control, is the minimization of the one step ahead expectation cost function. It does this, by showing that as the control horizon is increased, in at least the first few steps, the constant control tends towards the Expectation, infinite horizon, controller.

Chapter 4

The Optimal Control Policy

In the previous chapters various controllers have been looked at, with some achieving lower losses than others. The search for controllers that give lower and lower loss is an endless problem unless a lower bound on the loss is known. The controller that achieves this lower bound on the loss, is then known to be an optimal controller. This chapter looks to obtain the lower bound on the loss and make conclusions on how optimal the previously discussed controllers actually are.

4.1 Dynamic Programming

Taking into account that the Hammerstein model is nonlinear and stochastic, only leaves one general route to find the lower bound on the loss, the dynamic programming route.

The problem that dynamic programming seeks to solve can be expressed as follows. Given an initial state y_0, an admissible control sequence, $u_t \in \mathcal{Y}_t$, must be found such that the following cost function is minimized.

$$J_{\mathcal{Y}_t}(y_0) = \underset{e_t}{E}\left\{ g_N(y_N) + \sum_{t=0}^{N-1} g_t(y_t, u_t, e_t) \right\} \tag{4.1}$$

subject to the system constraint

$$y_{t+1} = f_t(y_t, u_t, e_t) \tag{4.2}$$

where $g_t(y_t, u_t, e_t)$ is some value function.

The dynamic programming solution to the above problem is as follows. Let $J^*(y_0)$ be the optimal value of the cost function. Then

$$J^*(y_0) = J_0(y_0) \tag{4.3}$$

where the function J_0 is given by the last step of the following dynamic programming algorithm, which proceeds backwards time from period $N-1$ to period 0, in the usual dynamic programming way.

$$J_N(y_N) = g_N(y_N)$$
$$J_t(y_t) = \min_{u_t \in \mathcal{Y}_t} \left\{ \underset{e_t}{E} \left\{ g_t(y_t, u_t, e_t) + J_{t+1}(f_t(y_t, u_t, e_t)) \right\} \right\} \tag{4.4}$$

where $t = N-1\ldots0$.

Rather than rigorously prove the above dynamic programming result, see [Ber76], an intuitive verification of the result is given as follows.

The derivation of any dynamic programming algorithm hinges on one key fact, that is called the "principal of optimality". The principal of optimality can be expressed in the following way. Suppose that $\{u_0^*, u_1^*, \ldots, u_{N-1}^*\}$ is the optimal control for the problem given in (4.1). Consider the subproblem whereby we are at state y_i at time i and the goal is to minimize the "cost to go" from time i to time N, i.e.

$$J_{\mathcal{Y}_t}(y_0) = E \left\{ g_N(y_N) + \sum_{t=i}^{N-1} g_t(y_t, u_t, e_t) \right\} \tag{4.5}$$

Then the control $\{u_i^*, u_{i+1}^*, \ldots, u_{N-1}^*\}$ is also optimal for this subproblem. Hence the strategy in dynamic programming is to start at the end, y_N, solve for the optimal control u_{N-1}, calculate the minimum cost and then go one step back. The process is now repeated but with the optimal cost calculated in the last step being added to the optimal cost in the new step. When the start time is reached the cost obtained is the optimal cost and the sequence of calculated controls are an optimal set.

In the next two sections the dynamic programming algorithm is developed, for first the C polynomial in the example system equal to one and then for C unchanged. The reason for this is that in the C equal to one case, the calculations are simpler, hence a good feel for what the dynamic programming algorithm looks like, can be obtained before the more complex case of C not equal to one is tackled.

4.2 The C Equal to One Case

This section deals with deriving and implementing the dynamic programming algorithm to find out what the actual optimal control is and in doing so, determine a lower bound on the loss.

Before the dynamic programming algorithm can be applied to the example system, the value function, g_t, has to be defined, as follows.

$$g_t(y_t, u_t, e_t) = (y(t) - y_o)^2 \tag{4.6}$$

Note that the y_o in (4.6) is the set point and not the initial y, y_0, as in (4.3). Using the definition of the value function, (4.1) becomes

$$J_{y_t}(y_0) = \underset{e_t}{E}\left\{(y(N) - y_o)^2 + \sum_{t=0}^{N-1}(y(k) - y_o)^2\right\} \tag{4.7}$$

In the first step, the dynamic programming algorithm, (4.4), reduces to

$$J_N(y_N) = \min_{u(N-1)}\left\{\underset{e_N}{E}\left\{(y(N) - y_o)^2\right\}\right\} \tag{4.8}$$

From the definition of the system model (with $C = 1$), (1.2), an expression for $y(N)$ can be obtained as follows.

$$\begin{aligned} y(N) &= b_{10}u(N-1) + b_{20}u^2(N-1) \\ &\quad + \overbrace{b_0 + b_{11}u(N-2) - ay(N-1) - y_o}^{w(N-1)} + y_o + e(N) \end{aligned} \tag{4.9}$$

Now by substituting this expression into (4.8), (4.8) reduces to

$$\begin{aligned} J_N(y_N) = \min_{u(N-1)}\Big\{\underset{e_N}{E}\Big\{\big(&b_{10}u(N-1) + b_{20}u^2(N-1) + w(N-1) \\ &+ e(N)\big)^2\Big\}\Big\} \end{aligned} \tag{4.10}$$

By noting that $e(N)$ is uncorrelated with all the other terms in (4.10), and using the standard property of the expectation operator given in (4.11), (4.12) can be obtained.

$$E\left\{\sum_{j=1}^{M} g_j(X)\right\} = \sum_{j=1}^{M} E\{g_j(X)\} \tag{4.11}$$

where X is any random variable

$$\begin{aligned} J_N(y_N) = \min_{u(N-1)}\Big\{&\big(b_{10}u(N-1) + b_{20}u^2(N-1) + w(N-1)\big)^2 \\ &+ \underset{e_N}{E}\left\{e(N)^2\right\}\Big\} \end{aligned} \tag{4.12}$$

Note that the $\underset{e_N}{E}\{e(N)^2\}$ term is only a constant and will have no effect on the minimization, hence it is omitted from the proceeding derivation.

At this stage, as it is the first step in the dynamic algorithm, some means of deciding how to choose the control, given values of $w(t)$, is required. This has to be done for the first step because the cost of the last step is not known, as there has not been a last step yet. The method of choosing the control for given $w(t)$ will be to use the One Step controller, (2.23).

The first step is also peculiar in that a closed form solution can be used to obtain the control. This is not the case in the steps that follow hence a discretization of both the w and u variables is required. Hence on each step of the dynamic algorithm two tables need to be generated. One is a table of optimal controls for every $w(t)$ and the other is a table of cost values for every $w(t)$. The table of optimal controls has to be stored for every step of the dynamic algorithm, as it is these tables that will be used to produce the optimal control in any proceeding simulation. The table of costs is only required for the next step, as will be shown in the derivation for the second step in the dynamic algorithm.

The cost function for the second step in the dynamic programming algorithm is

$$J_{N-1}(y_{N-1}) = \min_{u(N-1),u(N-2)} \left\{ \underset{e_{N-1}}{E} \left\{ (y(N-1) - y_o)^2 + (y(N) - y_o)^2 \right\} \right\}$$

which, by the principal of optimality, becomes

$$J_{N-1}(y_{N-1}) = \min_{u(N-2)} \left\{ \underset{e_{N-1}}{E} \left\{ (y(N-1) - y_o)^2 + J_N(w(N-1)) \right\} \right\}$$

and now by using the results found in the first step

$$J_{N-1}(y_{N-1}) = \min_{u(N-2)} \Big\{ \left(b_{10}u(N-2) + b_{20}u^2(N-2) + w(N-2) \right)^2 + \underset{e_{N-1}}{E} \left\{ J_N(w(N-1)) \right\} \Big\} \quad (4.13)$$

In (4.13) the $w(N-1)$ term, using (4.9), can be expressed as

$$w(N-1) = b_0 + b_{11}u(N-2) - ay(N-1) - y_o$$

and by using (4.9) but delayed one step

$$y(N-1) = b_{10}u(N-2) + b_{20}u^2(N-2) + w(N-2) + y_o + e(N-1)$$

gives

$$w(N-1) = b_0 - (a+1)y_o + (b_{11} - ab_{10})u(N-2) - ab_{20}u^2(N-2) - aw(N-2) - ae(N-1) \quad (4.14)$$

From (4.14) it can be seen that $w(N-1)$ is a function of $u(N-2)$, $w(N-2)$ and $e(N-1)$ and as $u(N-2)$ and $w(N-2)$ are the variables that will be discretized, for each value discrete value of $u(N-2)$ and $w(N-2)$, $w(N-1)$

can be thought as a function of $e(N-1)$. Hence $w(N-1)$ can be expressed as

$$w(N-1) = f_{u(N-2),w(N-2)}(e(N-1)) \tag{4.15}$$

Now by using the above expression for $w(N-1)$ and the definition of the expectation operator when it operates on function of a random variable, below, the expectation term in (4.13) can be expressed as in (4.17).

$$E[Y] = \int_{-\infty}^{\infty} g(x) f_X(x) dx \tag{4.16}$$

where $Y = g(X)$ is any function of X, $f_X(x)$ is the probability density of X and X is any random variable.

$$\begin{aligned} &\underset{e_{N-1}}{E} \{J_N(w(N-1))\} \\ &\quad = \int_{-\infty}^{\infty} J_N(f_{u(N-2),w(N-2)}) f_{e(N-1)}(e(N-1)) de(N-1) \end{aligned} \tag{4.17}$$

Note here that as $e(N-1)$ is normal with zero mean, $f_{e(N-1)}$ is the usual normal density function and is defined as follows, where σ is the standard deviation of $e(N-1)$.

$$f_{e(N-1)} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \left(\frac{e(N-1)^2}{2\sigma^2}\right) \tag{4.18}$$

Now by substituting in (4.17), into (4.13) the second step in the dynamic programming algorithm becomes

$$\begin{aligned} J_{N-1}(y_{N-1}) = \min_{u(N-2)} \Bigg\{ &(b_{10}u(N-2) + b_{20}u^2(N-2) + w(N-2))^2 \\ &+ \int_{-\infty}^{\infty} J_N(f_{u(N-2),w(N-2)}) f_{e(N-1)}(e(N-1)) de(N-1) \Bigg\} \end{aligned} \tag{4.19}$$

From the second step of the dynamic algorithm, it can easily be seen that the third, fourth, ... steps are exactly the same as the second, but with the indices changed. The second step expression shows that for each w and u value, their will have to be one numerical integration over e.

To implement the dynamic programming algorithm in Matlab, some care should be taken as to how the program is written. Care is required because for a discretization of say 40 points in u, w and e, 1600 numerical integrations over e have to be performed, hence an efficient algorithm is required for the main loop in the program. This is achieved in Matlab, by coding the loops as functions and making sure that vector and matrix operations are always used instead of For loops where possible. The program that implements the dynamic algorithm is called min_var_dym_C_1.m which calls loop_C_1.m to do the main recursive looping. min_var_dym_C_1.m and loop_C_1.m are given on pages 72 and 73 of appendix A, respectively.

Another issue in implementing the dynamic algorithm is how to discretize the u, w and e variables. A linear discretization could be made but this would not take into account the quite natural desire to have greater definition in the mid range values of the variables, than at the extremities. Hence a discretization strategy that samples in the mid range at higher frequencies than at the extremities is required. The strategy that was chosen, is to sample the function values generated by (4.20), which would give the positive values, reflecting about the zero axis, to give the negative values and then an offset can be added as required.

$$f(x) = \exp\left(\frac{x^2}{\nu}\right) - 1 \tag{4.20}$$

Note increasing ν increases the frequency of discretization at the mean and reduces it at the extremities. Hence ν can be varied until the desired discretization is obtained. The discretization procedure is programmed in the discretize.m function, given on page 74 of appendix A, which the min_var_dym.m program calls to discretize the variables.

Using min_var_dym.m with 140 points used in the discretization, the results in Figure 4.1 were obtained. Note that the w discretization was set so that the high density of points was at 0.125, the value of w where the One Step controller switches to the constant value. Also the discretization of u is set to have a mean of -0.5, the most common control when the One Step controller is at the extremum.

Figures 4.1 shows that the dynamic programming tables converge towards the Hybrid Extremum controller table. This supports the conclusion of the second chapter which stated that the Hybrid Extremum controller has the best performance out of the controllers discussed.

Rather than analyze the C equal to one case in more detail, the C not equal to one case is introduced in the next section and then is analyzed in detail.

4.3 The C as in the Example System Case

This section looks at generalizing the results in the preceding section for the C polynomial as in the example system case. The procedure that is adopted is the same as in the last section, with the first step in the dynamic programming algorithm being analyzed first and then the second step.

The derivation for the first step in the dynamic algorithm, for C as in the example system, is the same as the previous section, up until equation

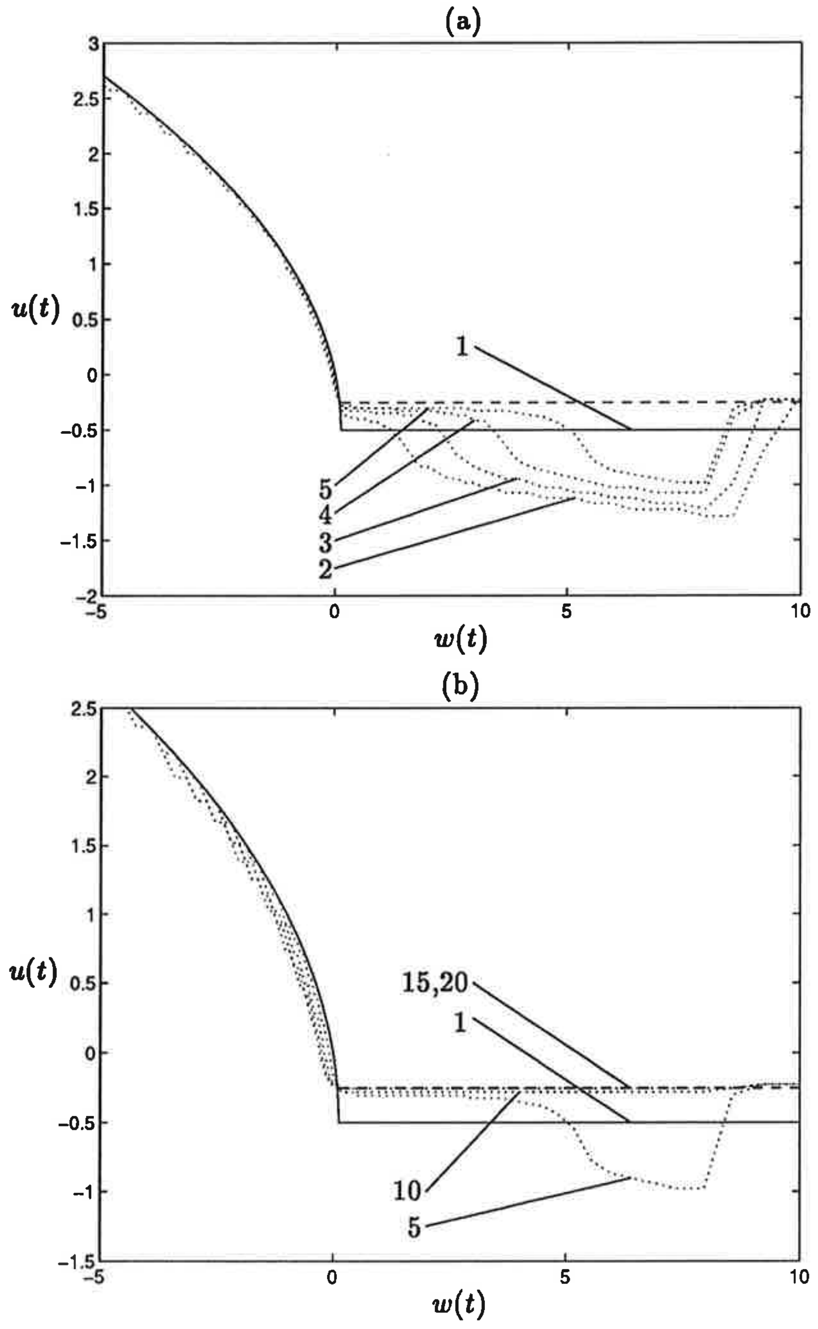


Figure 4.1: Control verses w tables produced by Dynamic Programming ($C = 1$), (a) The initial steps, 1, 2, 3, 4, 5, (b) The final steps, 10, 15, 20. Solid line corresponds to the One Step controller and the dashed line corresponds to the Hybrid Extremum controller.

(4.9), at which point it diverges and (4.9) becomes

$$y(N) = b_{10}u(N-1) + b_{20}u^2(N-1)$$
$$+ \overbrace{b_0 + b_{11}u(N-2) - ay(N-1) - y_o}^{w(N-1)} + y_o + e(N) + ce(n-1) \quad (4.21)$$

and hence (4.10) becomes

$$J_N(y_N) = \min_{u(N-1)} \left\{ \underset{e_N}{E} \left\{ \Big(\overbrace{b_{10}u(N-1) + b_{20}u^2(N-1) + w(N-1)}^{\Psi} + e(N) + ce(N-1) \Big)^2 \right\} \right\} \quad (4.22)$$

Now by multipling out the square, canceling out the terms that are uncorrelated, as the mean of e is zero, and noting that $\underset{e_N}{E}\{e(N)^2\} = \underset{e_N}{E}\{e(N-1)^2\}$ gives

$$J_N(y_N) = \min_{u(N-1)} \left\{ \underset{e_N}{E}\{\Psi^2\} + 2c\underset{e_N}{E}\{\Psi e(N-1)\} + (1+c^2)\underset{e_N}{E}\{e(N)^2\} \right\} \quad (4.23)$$

Also note that the above step would not be possible without the fact that independence is not required for (4.11) to hold.

Now by substituting the definition of Ψ back into the middle expectation of (4.23) and again canceling the uncorrelated terms gives

$$2c\underset{e_N}{E}\{\Psi e(N-1)\} = 4c\underset{e_N}{E}\{w(N-1)e(N-1)\}$$

substituting in $w(N-1)$ gives

$$= -8ac\underset{e_N}{E}\{y(N-1)e(N-1)\}$$

and substituting in $y(N-1)$ gives

$$= -8ac\underset{e_N}{E}\{e(N-1)e(N-1)\} \quad (4.24)$$

Hence as (4.24) turns out to be a constant value and the right hand term in (4.23) is also a constant, the minimization becomes

$$J_N(y_N) = \min_{u(N-1)} \left\{ \underset{e_N}{E}\left\{ \big(b_{10}u(N-1) + b_{20}u^2(N-1) + w(N-1)\big)^2 \right\} \right\} \quad (4.25)$$

which is exactly the same as the C equal to one case, (4.12).

For the second step in the dynamic algorithm with C as in the example system, the derivation is exactly the same as in the C equal to one case, up to expression (4.14), which becomes

$$\begin{aligned} w(N-1) = b_0 - (a+1)y_o + (b_{11} - ab_{10})u(N-2) - ab_{20}u^2(N-2) \\ - aw(N-2) - ae(N-1) - ace(N-2) \end{aligned} \tag{4.26}$$

As $w(N-1)$ now becomes a function of $e(N-1)$ and $e(N-2)$, expression (4.15) becomes

$$w(N-1) = f_{u(N-2),w(N-2)}(e(N-1), e(N-2)) \tag{4.27}$$

Hence expression (4.13) must be rewritten as follows with the expectation taking into account both $e(N-1)$ and $e(N-2)$.

$$\begin{aligned} J_{N-1}(y_{N-1}) = \min_{u(N-2)} \Big\{ \left(b_{10}u(N-2) + b_{20}u^2(N-2) + w(N-2)\right)^2 \\ + \underset{e_{N-1}e_{N-2}}{E} \{J_N(w(N-1))\} \Big\} \end{aligned} \tag{4.28}$$

To be able to take the expectation in the above equation, expression (4.16) must be extended for the function of two random variables case, as follows.

$$E[Z] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y) f_{XY}(x,y)dxdy \tag{4.29}$$

where $Z = g(X,Y)$ is any function of X and Y, $f_{XY}(x,y)$ is the joint probability density of X and Y with X and Y being any random variables. Hence the expectation in (4.28) becomes

$$\begin{aligned} \underset{e_{N-1}e_{N-2}}{E} \{J_N(w(N-1))\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} J_N(f_{u(N-2),w(N-2)}) \\ f_{e(N-1)e(N-2)}(e(N-1), e(N-2))de(N-1)de(N-2) \end{aligned} \tag{4.30}$$

Hence the second step of the dynamic algorithm becomes

$$\begin{aligned} J_{N-1}(y_{N-1}) = \min_{u(N-2)} \Bigg\{ \left(b_{10}u(N-2) + b_{20}u^2(N-2) + w(N-2)\right)^2 \\ + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} J_N(f_{u(N-2),w(N-2)}) \\ f_{e(N-1)e(N-2)}(e(N-1), e(N-2))de(N-1)de(N-2) \Bigg\} \end{aligned} \tag{4.31}$$

The second step in the dynamic programming algorithm, for the C not equal to one case is similar to the C equal to one case, except that the joint probability density function, given below, has to be used, making the integration a double integration.

$$f_{e(N-1)e(N-2)} = \frac{1}{2\pi\sigma^2} \exp - \left(\frac{e(N-1)^2+e(N-2)^2}{2\sigma^2}\right) \tag{4.32}$$

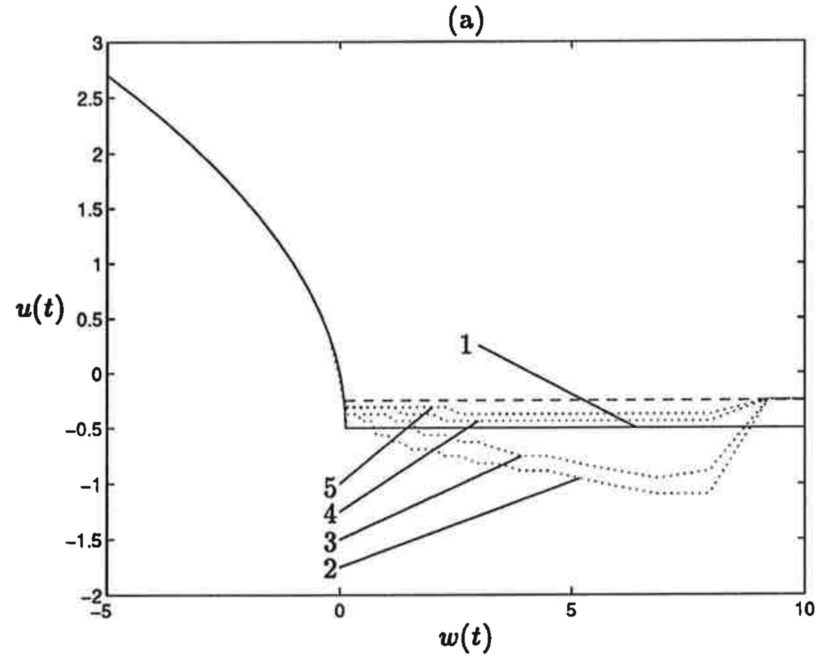
It is at this point that Bellman's "Curse of Dimensionality" becomes apparent, in that the change from a single intergration to a double integration, effectively adds another dimension to the problem and consequently increases the amount of computations required by a power.

To overcome the curse of dimensionality, more thought is needed in the choice of discrete points so that the points are used as efficiently as possible. To do this, the results in the last section can be used because the control table is independent of C, C only effects the actual w values during a simulation. As results in the last section showed that the dynamic programming tables converged to the Hybrid Extremum controller, instead of independently discretizing u, the discrete values of u are generated using the Hybrid Extremum controller and the discrete values of w. This change in the discretization of u should help the dynamic algorithm to converge quickly, as the values of u that it should converge to are available for selection. The discretization function that implements the new strategy of discretizing u is called discretize_un.m and is as shown on page 77 of appendix A. Also because the e points are put into the normal function, (4.32), there is no need to include values that are far away from the mean, as the value of the normal function at these points will be negligible. The discretization of w is the same as the C equal to one case.

The extra dimension the double integration adds, means that the programs for the C not equal to one case, have to be written in a completely different way from the C equal to one case. The main difference is that there will be one extra loop in the loop_C_1.m function. Hence the C not equal to one case will take much longer to run. The min_var_dym.m program and loop.m function are as shown on pages 75 and 76 of appendix A, respectively.

Using min_var_dym.m with 71 points of discretization, the results in Figure 4.2 were obtained.

From Figure 4.2 it is observed that dynamic programming gives the same results as in the $C = 1$ case, this is as expected because changing the C polynomial does not change the actual relationship between w and u, it just changes the values that w is likely to take. This can be verified by examination of expressions (2.19) and (2.20). Note that Figure 4.2 supports the result that the positive minimum variance roots should always be applied, so that the prediction $\hat{w}(t+1 \mid t)$ is minimized and the probability of being able to apply a minimum variance control on the next step is increased.



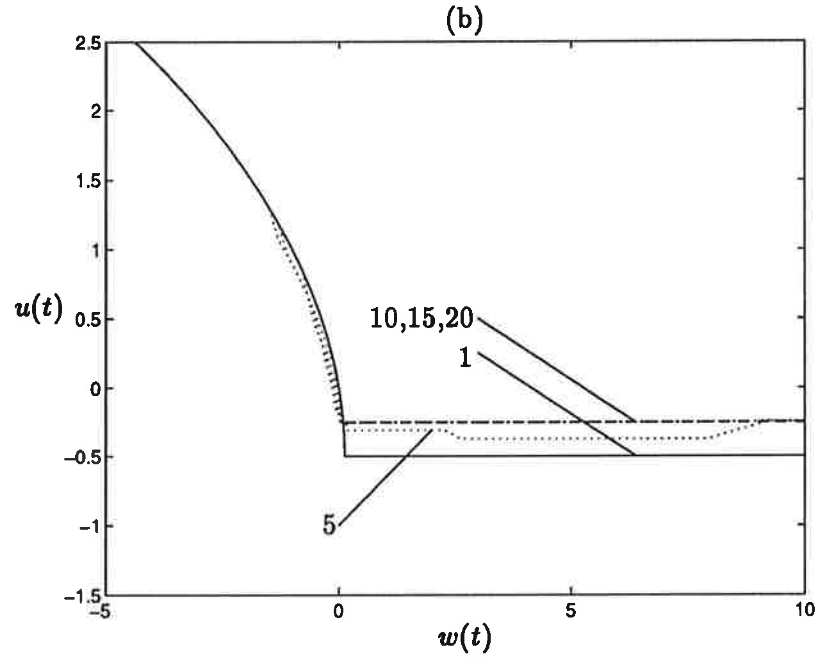


Figure 4.2: Control verses w tables produced by Dynamic Programming ($C \neq 1$). (a) The initial steps, 1, 2, 3, 4, 5, (b) The final steps, 10, 15, 20. Solid line corresponds to the One Step controller and the dashed line corresponds to the Hybrid Extremum controller.

y_o	Hybrid Extremum		Dynamic Programming	
	Variance	Loss	Variance	Loss
4.375	0.0745	0.1100	0.0787	0.1075
5.0	0.0463	0.0479	0.0496	0.0506
7.0	0.04	0.04	0.0465	0.0671
10.0	0.04	0.04	0.0519	0.0753

Table 4.1: Simulation results for the Dynamic programming table 20.

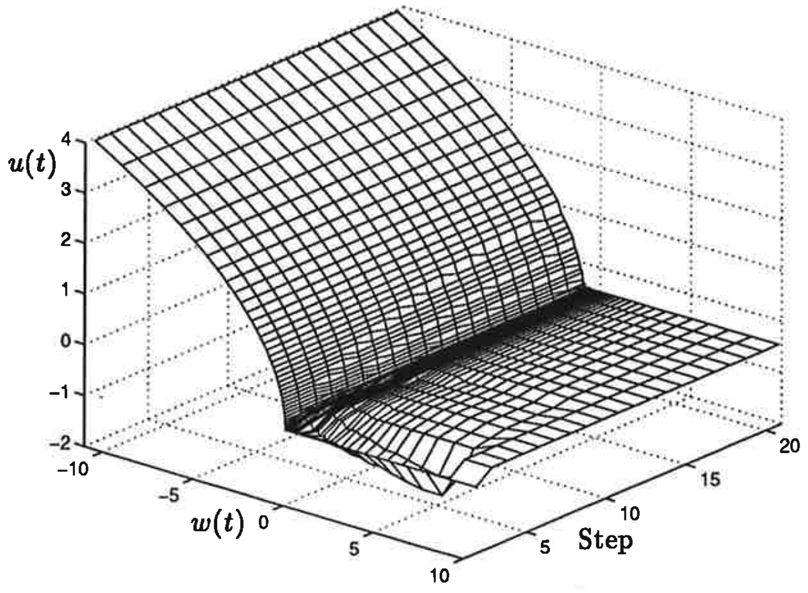


Figure 4.3: Three dimensional visualization of all 20 tables of u and w produced by dynamic programming.

With the tables of u versus w generated by dynamic programming, the example system can now be simulated. Out of the 20 tables generated, the last table, table 20, will be used to simulate the system as the dynamic algorithm converged to table 20, with no appreciable difference between the last few tables, see Figure 4.3. Using table 20 in program dym_sim.m, given on page 78 of appendix A, with linear interpolation being used to generate the control values, the results in Table 4.1 were obtained.

The results show that at the extremum, table 20 produces a lower loss than the Hybrid Extremum controller. The difference in loss is 2%, which may seem to be trivial but in some processes a small difference in loss can mean a huge difference in profit. The reason why table 20 produces a lower loss than the Hybrid Extremum controller can be found by comparing Figure 4.2 (b), which shows table 20, with Figure 2.6, which shows the u against

w plot for the Hybrid Extremum controller. The main difference between the two plots is that the table 20 plot does not go all the way down to $u(t) = -0.5$ before it changes to the constant control, but actually goes straight to the constant control when $u(t) = -0.25$. To verify that this is not just due to a lack of points in the discretization, the Hybrid Extremum controller was changed to mimic table 20, by changing the value at which the controller switches to the constant control. It was found empirically that a loss of 0.1075, the same as table 20, could be obtained with the following modified Hybrid Extremum controller switching condition.

$$w(t) < 0.0115 \tag{4.33}$$

The reduction in the loss probably comes from eliminating the uncertainty introduced by the $\hat{y}(t+1 \mid t)$ prediction. In that when the value of $\hat{y}(t+1 \mid t)$ allows a minimum variance control to be used, it is actually a better strategy to apply the constant extremum control, than rely on the information contained in the prediction. This is especially true when the difference between $\hat{y}(t+1 \mid t)$ and the extremum, is less than the intrinsic variance of the system, because then the $\hat{y}(t+1 \mid t)$ prediction is basically just noise and does not contain any information. Also the penalty for trusting an erroneous prediction would be quite high, as the end result would be to push the system even further away from the extremum then it already was. This would then mean that a minimum variance control could not be applied until the noise floated the system below the extremum. As this could take a significant number of steps, the extra loss incurred could be quite high. In contrast when $\hat{y}(t+1 \mid t)$ is far from the extremum the prediction is much more reliable as the intrinsic variance of the system is a much smaller fraction of the difference between the extremum and the prediction.

Away from the extremum the results of the dynamic programming table 20 get slightly worse than the Hybrid Extremum controller. This is due to the discretization of the w and u variables getting more and more sparse as the set point is moved away from the extremum.

The plot of the input and output generated by table 20 and a comparison against the loss of the unchanged Hybrid Extremum controller is given in Figure 4.4

The loss plot in Figure 4.4 highlights that table 20 is a slightly better controller than the Hybrid Extremum controller, by showing that the plots diverge slightly with table 20 having the lower loss.

In the second chapter it was found that it is a much better strategy to switch to an extremum constant control rather than the Expectation control when a minimum variance control could not be found. To now confirm that this is indeed the case, the min_var_dym.m was run under the same conditions as before but with the set point, y_o, set to 10. The control table that ensued, again table 20, is shown in Figure 4.5.

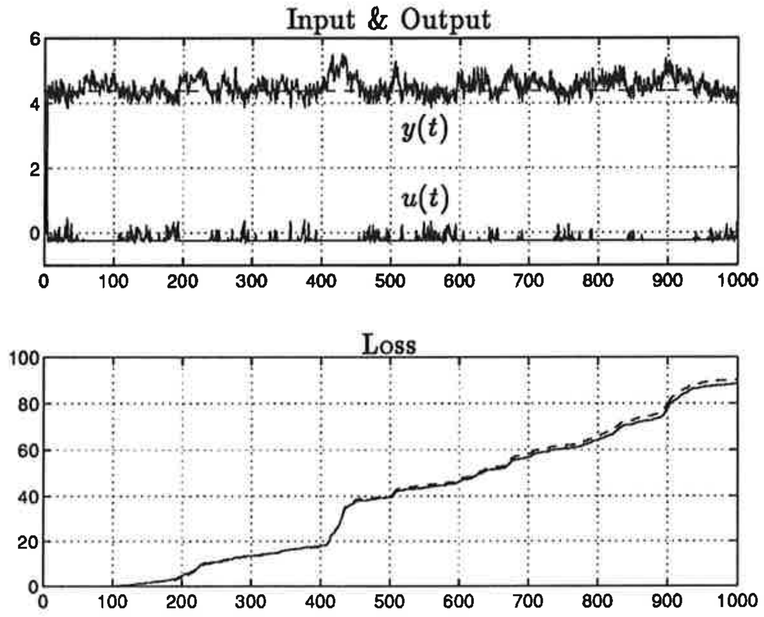


Figure 4.4: Simulation at the extremum using Table 20. In the output plot the dashed line equals the extremum. In the Loss plot, the solid line equals Table 20 and the dashed line equals the Hybrid Extremum controller.

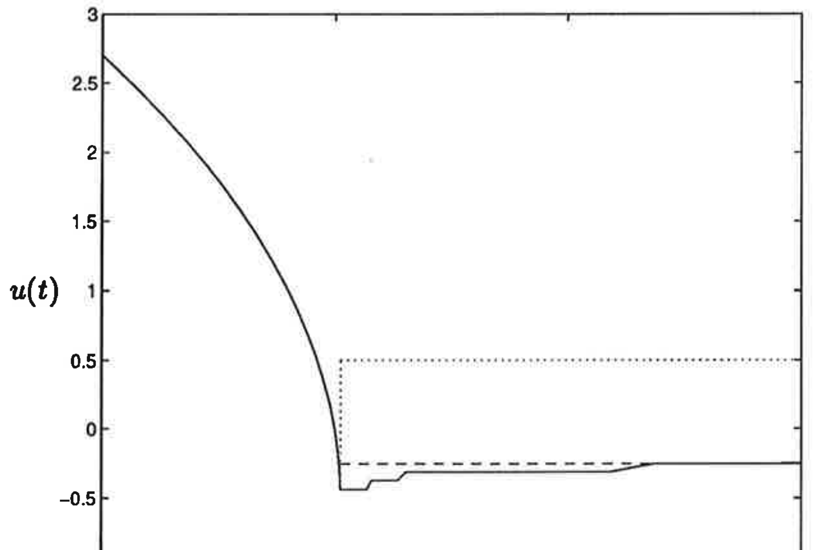


Figure 4.5: Control verses w table produced by Dynamic Programming, $y_o = 10$. The solid line equals equals table 20, the dashed line equals the Hybrid Extremum controller and the dotted line equals the Expectation controller.

From Figure 4.5 it can observed that the Expectation control is far away from the dynamic programming table 20, but the Hybrid Extremum controller is not far away at all. The reason why table 20 does not exactly converge to the Hybrid Extremum controller is due to the dynamic algorithm being stopped after a finite time, three days on a Sun Sparc 20 computer. To show that the difference between table 20 and the Hybrid Extremum controller is just due to the slow convergence of the dynamic algorithm and not to table 20 highlighting some enhanced performance characteristic, table 20 was simulated using the dym_sim.m program with the set point equal to 10. The resulting loss was 0.0498 which is worse than the Hybrid Extremum controller, which has a loss of 0.04 for a set point of 10. Hence the difference between the constant extremum controller and the dynamic algorithm is in fact due only a finite length simulation being performed. Thus dynamic programming shows that applying the constant extremum control, when a minimum variance control can not be found, is a much better strategy than using the Expectation control.

4.4 Conclusion

The goal of this chapter was to try and find a lower bound on the achievable loss so that the proposed controllers would have a target to strive for. The problem with using dynamic programming to do this, is that to actually achieve the lower bound on the loss, the optimal loss, the dynamic algorithm would have to be run for an infinite amount of time, with infinite precision calculations being performed. As these simulation lengths and precision requirements are not feasible, it is more realistic to give a lower bound on the loss along with the number of decimal places at which it is valid. Hence the lower bound, the optimal loss, is 0.1075 at the extremum to within four decimal places.

With this lower bound in the loss, it can be said that the Hybrid Extremal controller has a loss that is within 2.273% of the optimal loss. Also if the point at which the Hybrid Extremal controller switches to the constant control, is changed to (4.33), then the controller achieves the optimal loss and hence is then an optimal controller.

This a good result, in that a relatively simple controller can get within 2.273% of the optimal loss and with some empirical optimizing can actually achieve the optimal loss. Also in the case of extremal seeking control, it is a good result because, as will be shown in the next chapter, the Hybrid Extremum controller can easily be converted from the known parameter case to the unknown parameter case.

Chapter 5

Adaptive Extremal Seeking Control

In the previous chapters the extremum of the system has always been known, but in extremal control problems the extremum is not known and therefore has to be found and then tracked. This chapter looks at finding the extremum and then using the results of the previous chapters, to control the system at the extremum in a way that minimizes the loss.

5.1 Finding the Extremum

One of the main reasons why the Hammerstein model was chosen to represent the example system, was that it is linear in the parameters. The importance of having linearity in the parameters becomes apparent when the extremum has to be found, because the main techniques in identification, have the restriction that the system must be linear in the parameters. As the example system has a C polynomial that has order greater than zero, the most appropriate identification method to use, is the extended least squares (ELS) method, see [LS83].

Once the parameters have been estimated using ELS, the certainty equivalence approach is taken, where the estimated parameters are used as if they were the actual parameters, see [ÅW95].

To estimate the parameters using ELS the Hammerstein example system should be expressed in the following way

$$y(t) = \varphi(t-1)^T\theta + e(t) \tag{5.1}$$

where

$$\begin{aligned}
\varphi^T(t-1) &= [-y(t-1)\; u(t-1)\; u(t-2)\; u^2(t-1)\; \hat{e}(t-1)\; 1\,] \\
\theta^T &= [a\; b_{10}\; b_{11}\; b_{20}\; c_1\; b_0\,] \\
\hat{e}(t) &= y(t) - \varphi^T(t-1)\hat{\theta}(t-1)
\end{aligned}$$

The estimates using ELS are then given by the following equations

$$\begin{aligned}
\hat{y}(t) &= \varphi^T(t-1)\hat{\theta}(t-1) \\
\hat{\theta}(t) &= \hat{\theta}(t-1) + K(t)\,(y(t) - \hat{y}(t)) \\
K(t) &= P(t)\varphi(t-1)\left(\lambda + \varphi^T(t-1)P(t)\varphi(t-1)\right)^{-1} \\
P(t) &= \left(P(t-1) - K(t-1)\varphi^T(t-2)P(t-1)\right)/\lambda
\end{aligned}$$

where λ is an exponential forgetting factor.

By using the above ELS formulas, the parameters of the Hammerstein model and hence the extremum, can be estimated. The next section looks at using these estimates in the previously analyzed controllers to produce adaptive extremal seeking controllers.

5.2 Extremal Seeking Controllers

To obtain extremal seeking controllers, the ELS formulas are combined with the expectation and variance controllers. This effectively means that the new controllers will try to find and then track the extremum.

Combining the ELS formulas with the Expectation controller set to operate at the extremum, the Adaptive Expectation controller is obtained and is implemented using the min_exp_adp.m program, given on page 79 of appendix A. The results obtained from the program are as shown in Figure 5.1. Note that in this case, a forgetting factor of $\lambda = 0.99$, was found to be a good trade off between parameter convergence and parameter noise amplification.

From Figure 5.1 it can be observed that the loss is much worse than the known parameter case, the Expectation controller, as the loss has increased by about 170%. This large increase in loss is solely due to the parameter estimates not converging to there correct values, which can easily be seen in Figure 5.1 plot (a). The lack in convergence is due to the input signal not being persistently exciting enough, which should be expected as the controller is designed to produce a constant input signal. The parameter convergence rate can not be increased by lowering λ, as this makes the parameter estimates very noisy and the loss much worse. Note that at about the 500^{th} step mark, a large jump in the noise induces the parameter estimates to converge, hence the control changes to the value in the known

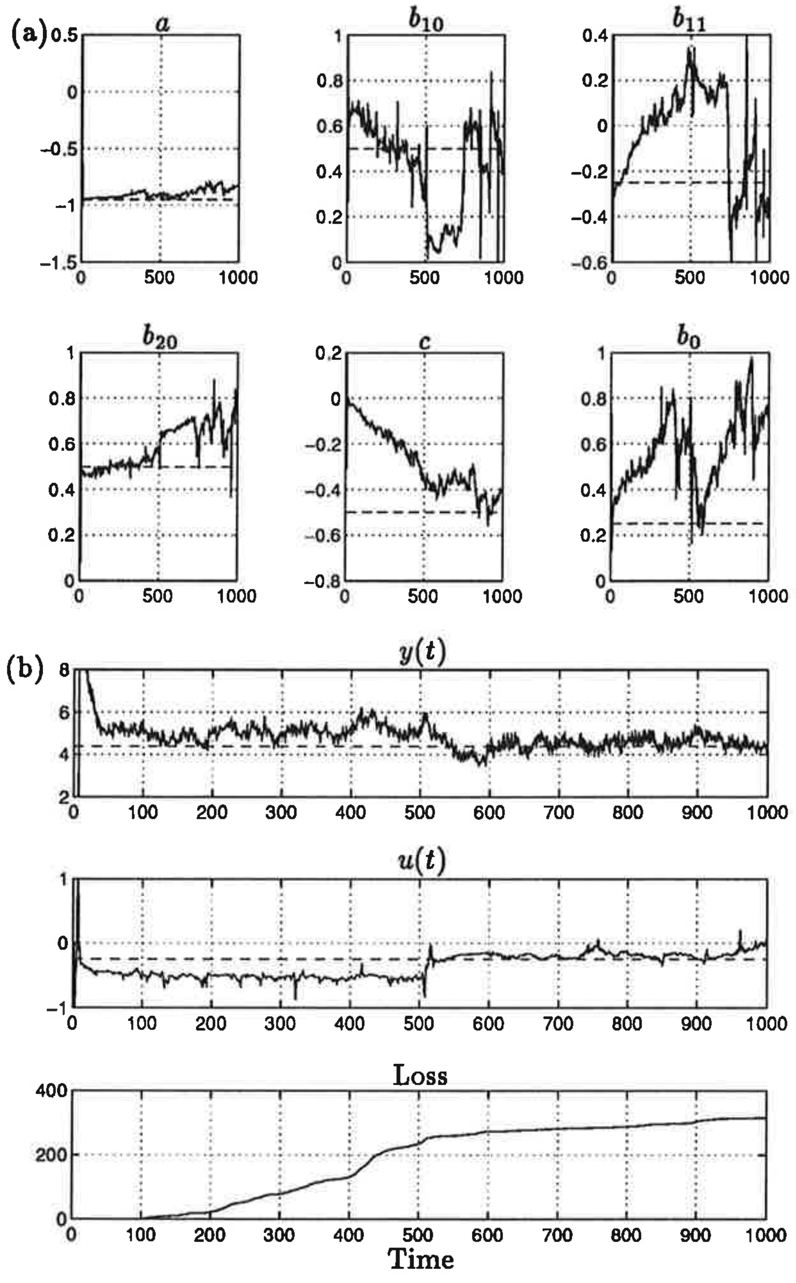


Figure 5.1: Adaptive Expectation controller. (a) Model parameter estimates, dashed lines equal the actual parameter values. (b) Input, Output and Loss, dashed lines equal extremum input and output values.

parameter case. This is also reflected in the gradient of the loss and output values becoming smaller.

Combining the ELS formulas with now the One Step controller, with w minimization and set to operate at the extremum, the Adaptive One Step controller is obtained and the program one_step_w_adp.m, as given on page 80 of appendix A, implements the new controller. The results obtained from the program are as shown in Figure 5.2. For the Adaptive One Step controller the forgetting factor value of $\lambda = 0.99$ was found to generate too much noise in the parameters and hence a high loss. Thus the forgetting factor was increased to 0.999.

The first thing that should be noted from Figure 5.2 plot (a), is that the convergence of the parameters is much better than the Adaptive Expectation controller case. This is due to the input signal being more exciting, which can be observed from Figure 5.2 plot (b). Although the parameter convergence is better, it is still not that good, especially for the c parameter. In a 5000 point simulation the loss of the controller is found to be 0.9777, which is, as expected, worse than the known parameter loss of 0.5542. In the input plot there is a large spike at the start of the simulation which induces a large spike in the output. The spikes are due to the initial parameter uncertainty giving inaccurate control values. This is not really a problem, as a limiting operation can be performed on the control input so that it does not go to unrealistic values.

Now by again combining the ELS formulas with the Hybrid Extremum controller set to operate at the extremum, the Adaptive Hybrid Extremum controller is obtained and is implemented using the hybrid_extremum_adp.m program, given on page 82 of appendix A. The results obtained from the program are as shown in Figure 5.3. As in the Adaptive One Step controller case, a forgetting factor of $\lambda = 0.99$ was found to be too low, hence $\lambda = 0.999$ was again used.

Figure 5.3 shows that the parameter estimates, in this case have converged slightly better than the Adaptive One Step controller. This is due to, as can be observed from Figure 5.3 (b), the input being more exciting. The increase in excitement of the input is due to the minimum variance control being applied more often, for the reasons discussed in chapter two. In a 5000 point simulation the controller had a loss of 0.1209, which is 9% worse than the known parameter case. This is again due to the parameter estimates not converging exactly to the system parameters.

5.3 Conclusion

A comparison of the known parameter case and the extremal seeking, unknown parameter case is given in Figure 5.4. The figure shows that the loss of the Expectation and One Step controllers gets much worse when they

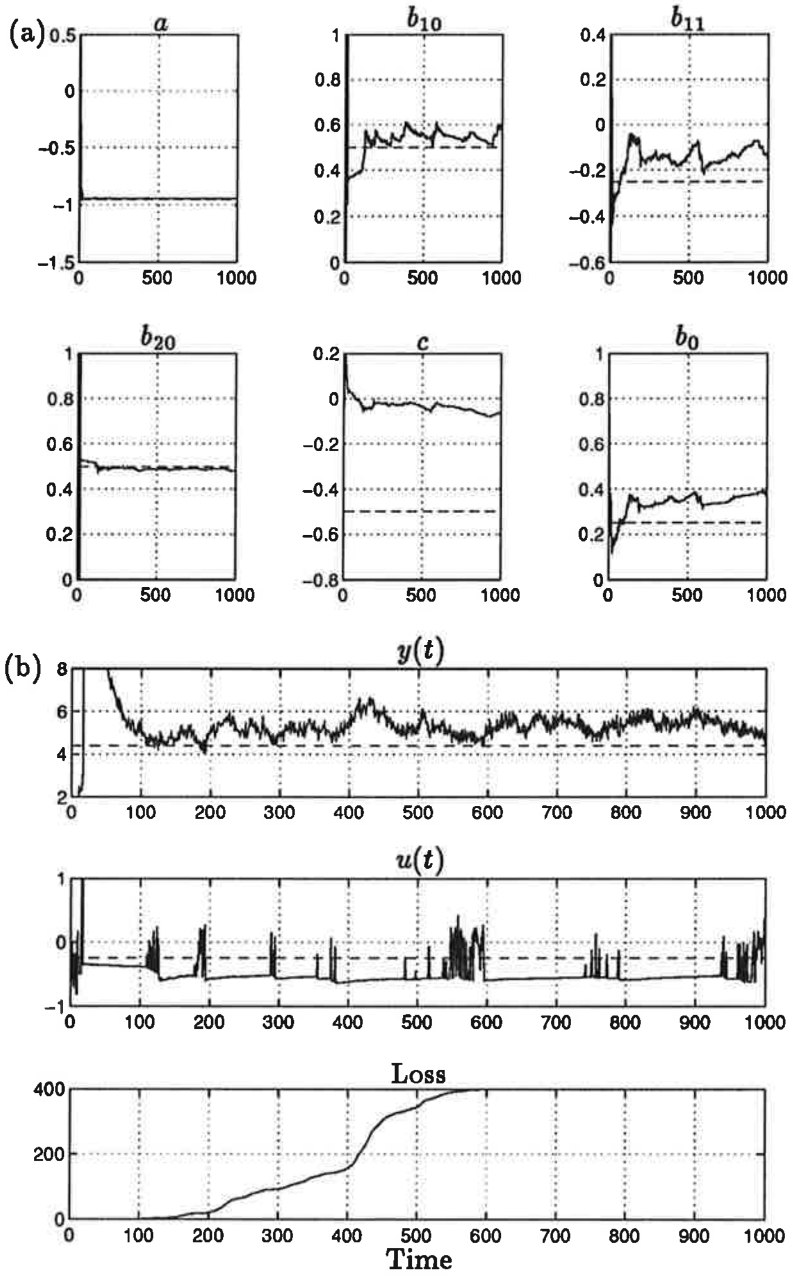
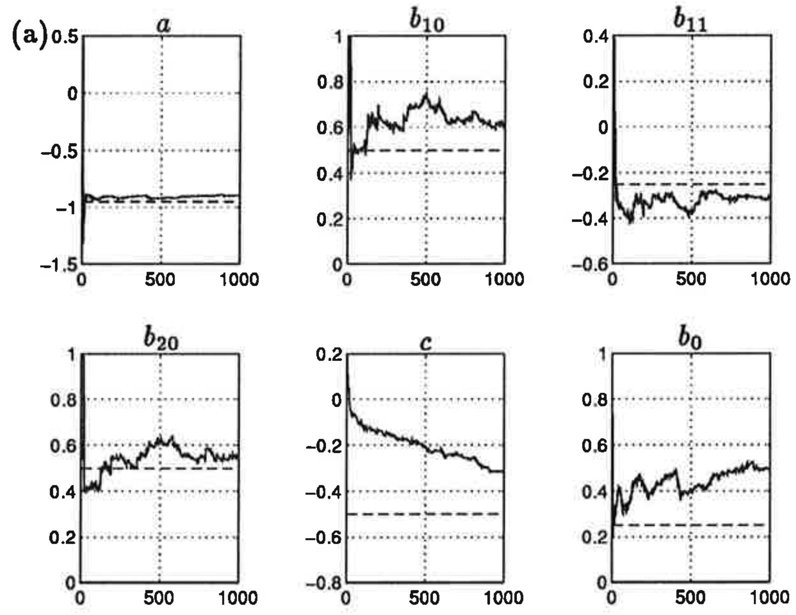


Figure 5.2: Adaptive One Step controller. (a) Model parameter estimates, dashed lines equal the actual parameter values. (b) Input, Output and Loss, dashed lines equal extremum input and output values.



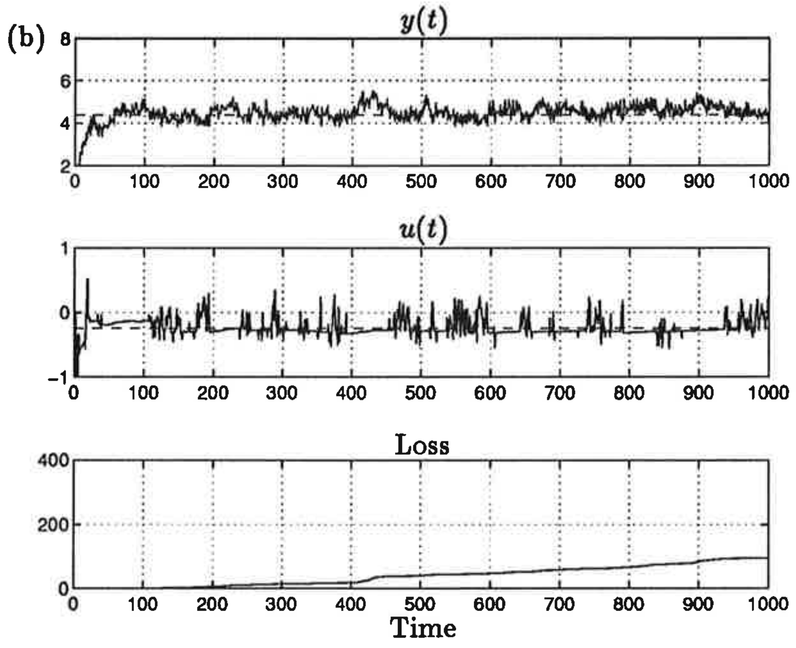


Figure 5.3: Adaptive Hybrid Extremum controller. (a) Model parameter estimates, dashed lines equal the actual parameter values. (b) Input, Output and Loss, dashed lines equal extremum input and output values.

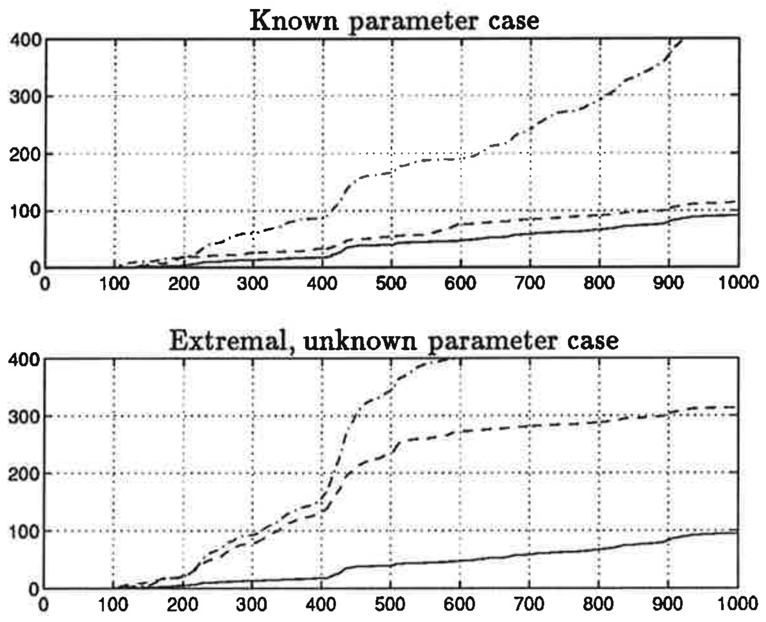


Figure 5.4: Loss plots for both the known and unknown parameter case, at the extremum. Solid line equals the Hybrid Extremum, dashed line equals the Expectation and the dot-dash line equals the One Step controllers.

are converted to extremal seeking controllers. This is due to the fact that at the extremum, the one step and obviously the Expectation controllers, produce an input signal that is mainly a constant, which means that the system will not be persistently excited enough, to ensure that the system has good parameter convergence. Without good parameter estimates it is an impossible task for the controllers to produce good results.

The Adaptive Hybrid Extremum controller does not suffer so much from the problem of slow parameter estimate convergence, because at the extremum it produces an input signal that is more exciting than the other controllers. Hence the controller has a much better performance than the other controllers, with the loss only increasing by 9% from the known parameter case.

Thus this chapter has shown that, as in the known parameter case, the Hybrid Extremum controller has the best performance out of all the controllers. As the dynamic programming algorithm did not look at the unknown parameter case, it can not be said that the Adaptive Hybrid Extremum controller is in any way optimal. If some dynamic programming algorithms were run, taking into account the parameter estimation, it might show that the performance could be improved by adding some noise to the input signal, when the need for parameter convergence is great, i.e. when the control is constant.

Conclusion

The object of this project was to find an extremal control strategy, that found and then tracked the extremum of a quadratic Hammerstein system. Before an extremal control strategy could be developed, the known parameter case had first to be analyzed.

In the known parameter case it was found that a simple Expectation controller, that just applys a static control signal, gave good results, but it made no attempt to reduce the variance. To reduce the variance, the normal one step ahead variance cost function was introduced and then minimized. This lead to the One Step controller, which produced good results away from the extremum by achieving the intrinsic variance of the system. However this was contrasted with the results near the extremum, which were much worse than the Expectation controller. The reason for the poor performance at the extremum was due to: minimum variance controls not being able to be applied, due to the input nonlinearity constraining the possible controls and the constant control that was applied instead of the minimum variance control, being very short sighted and not taking into account the long term objectives.

The problem with not being able to apply the minimum variance control can not be solved, as it is an inherit aspect of the system, although the situation can be improved upon, by maximizing the probability of being able to apply a minimum variance control on the next step. The maximization of this probability did indeed improve the results, but only slightly and its effect decreased as the set point approached the extremum.

The problem with the constant control on the other hand can be corrected, by instead of using the original One Step constant control, which was found to be the minimum of the one step ahead expectation cost function, the Expectation controller was used to generate the control. This effectively changes the number of steps, the constant control cost function looks ahead, from one to infinity. This controller gave good results, with the loss at the extremum being better than the Expectation controller. However the loss away from the extremum was worse than the One Step controller. The results got worse as the set point moved away from the extremum, because it is a much better strategy to set the constant control at a value that will give the extremum, as this will increase the chance of being able to apply the minimum variance control on the next step. The Hybrid Extremum controller implemented the above idea and it achieved the best loss out of all the known parameter controllers.

The effect of increasing the control horizon was also investigated via Generalized Predictive Control, because the main problem with the One Step controller was that it only looked one step ahead. For a control horizon of two, better results than the One Step controller were obtained, but they were not better than the Hybrid Extremum controller. With the control horizon at three, it was found that performance only slightly increased and it is at this point that it becomes clear that extending the control horizon, would not give better results than had already been obtained. The reason for this, is that as the control horizon is increased, the i^{th} step ahead prediction, which the control is based on, gets more and more unrealable. Thus GPC did not improve on the results obtained with the Hybrid Extremum controller and even if it did, the controller would be too complex to convert to an extremal seeking controller.

The Hybrid Extremum controller achieves a good loss, but until the minimum achievable loss is known, a measure of how good it is can not be obtained. The optimal policy chapter, via dynamic programming, found that the Hybrid Extremum controller does indeed have a good loss and with some empirical testing of the point at which the constant control should be applied, the optimal loss can be achieved. This is a pleasing result because the Hybrid Extremum controller is very simple in design and can easily be converted to an extremal seeking controller.

The Expectation, One Step and Hybrid Extremum controllers were converted to adaptive extremal seeking controllers by adding in an extended least squares algorithm, to identify the parameters. It was found that the Expectation and One Step adaptive controllers did not produce good results. The reason for this is that, as both controllers apply a constant control for the majority of the time, the system is not persistently excited enough for the ELS algorithm to identify the parameters. Hence the applied control will be very inaccurate. The Adaptive Hybrid Extremum controller does not suffer so much from this problem as it does not apply the constant control as many times as the other controllers. The performance of the Adaptive Hybrid Extremum controller is good, with it being only 9% worse than the known parameter case.

Although the Adaptive Hybrid Extremum controller achieves a good loss, it is probable that if dynamic programming were used to analyze the unknown parameter case, it would find that it was not an optimal controller. The reason for this, is that it might be more advantageous to apply a constant control with some noise attached, to speed up the convergence of the parameters. Also an optimal controller for the unknown parameter case, would have to take into account the uncertainties in the parameter estimates.

The next step in the analysis of extremal seeking systems would be to see if the generally accepted fact that, if an optimization technique works well for a quadratic function, then it will work well for any smooth nonlinear function, actually holds. If it did, then the Hybrid Extremum Extremal controller would be very useful, as it could be used to find and then track the extremum of any smooth nonlinear system.

This project achieved its goal of finding a good extremal controller, by finding a controller that is almost optimal for the quadratic Hammerstein model and with more work, it could probably be shown that it generalizes to many more nonlinear systems.

Appendix A

Matlab Programs

The programs and functions presented in this appendix are all written in the Matlab-4.2c programming code. A table of the programs and functions is as follows.

Program Name	Page Number
min_exp.m	61
one_step.m	62
one_step_w.m	63
hybrid.m	65
hybrid_extremum.m	65
two_steps_calc_C_1.m	65
two_steps_C_1.m	66
two_steps_calc.m	68
two_steps.m	69
three_steps.m	71
cost_three_steps.m	72
min_var_dym_C_1.m	72
loop_C_1.m	73
discretize.m	74
min_var_dym.m	75
loop.m	76
discretize_un.m	77
dym_sim.m	78
min_exp_adp.m	79
one_step_w_adp.m	80
hybrid_extremum_adp.m	82

min_exp.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the Expectation controller, min_exp.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
y=zeros(1,length);
lossp=zeros(1,length-100);
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;

for k=3:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
  y(k)=thi_act'*theta_act;

  %% Calculate the Expectation control.
  u(k)=-(B1)/(2*B2)+sqrt((B1/(2*B2))^2+((A1*yo-b0)/B2));

  %% Calculate the loss.
  if k>100
  lossp(k-99)=lossp(k-100)+(y(k)-yo)^2;
  end;
end;

%% Calculate the variance, loss and mean.
var=cov(y(100:length))
loss=sum(((y(100:length)-yo).^2))/(length-100)
mean_y=mean(y(100:length))

k=1:length;
clg;

figure(1)
  clg
  subplot(2,1,1)
    plot(k,y(k),'-');hold
    plot(k,ye*ones(1,1000),'--')
    plot(k,u(k),'-');hold
    grid;
  subplot(2,1,2);
    plot(k,[zeros(1,99) lossp]);
    axis([0 1000 0 400])
    grid;
```

one_step.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the One Step controller, one_step.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
y=zeros(1,length);
yhp1_old=0; %% The prediction of y(t) given t-1.
lossp=zeros(1,length-100);
roots_array=zeros(3,2); %% Array of root values and cost
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;

for k=3:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
  y(k)=thi_act'*theta_act;

  %% Calculate the possible One Step controls and store in roots_array
  Upsilon=0.2236068*(-5-20*yhp1_old+10*u(k-1)-18*y(k)+40*yo)^(1/2);
  roots_array(1,1)=-0.5;
  roots_array(2,1)=-0.5+Upsilon;
  roots_array(3,1)=-0.5-Upsilon;

  ut=roots_array(:,1);
  %% Calculate the cost of each root and store value in roots_array.
  roots_array(:,2)=(0.25+0.5*ut-0.25*u(k-1)+0.5*ut.^2+0.45*y(k)+...
                   0.5*yhp1_old-yo).^2;

  %% Set cost to infinity if root is complex.
  for i=1:3
    if imag(roots_array(i,1))~=0
      roots_array(i,:)=NaN*ones(1,2);
      end
    end


    %% Find the lowest cost root.
    [x,p]=sort(roots_array(:,2));
    u(k)=roots_array(p(1),1);

    %% Calculate the prediction of y(t) given t-1.
    yhp1_old=0.25+0.5*u(k)-0.25*u(k-1)+0.5*u(k)^2+0.45*y(k)...
            +0.5*yhp1_old;

    %% Calculate the loss.
    if k>100
    lossp(k-99)=lossp(k-100)+(y(k)-yo)^2;
    end;
  end;

  %% Calculate the variance, loss and mean.
  var=cov(y(100:length))
  loss=sum(((y(100:length)-yo).^2))/(length-100)
  mean_y=mean(y(100:length))
```

one_step_w.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the One Step controller with predicted
% w(t) minimization, one_step_w.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
y=zeros(1,length);
w=zeros(1,length);
y_o=0;w_o=0;
lossp=zeros(1,length-100);
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;

for k=3:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
```

```
    y(k)=thi_act'*theta_act;

    %% Calculate the w(t) value.
    w(k)=-c*w(k-1)+b0-(1+c)*yo+(b11-c*b10)*u(k-1)-c*b20*u(k-1)^2+...
          (c-a)*y(k);

    %% If the minimum variance roots are not complex, calculate them.
    if w(k) < (b10^2/(4*b20))
      %% Calculate the prediction of y(t) and then w(t), positive
      %% quadratic sign.
      u_p=-(b10/(2*b20))+sqrt((b10/(2*b20))^2-(w(k)/b20));
      y_p=-c*y_o+b0+b10*u_p+b11*u(k-1)+b20*u_p^2+(c-a)*y(k);
      w_p=-c*w_o+b0-(1+c)*yo+(b11-c*b10)*u_p-c*b20*u_p^2+(c-a)*y_p;

      %% Calculate the prediction of y(t) and then w(t), negative
      %% quadratic sign.
      u_n=-(b10/(2*b20))-sqrt((b10/(2*b20))^2-(w(k)/b20));
      y_n=-c*y_o+b0+b10*u_n+b11*u(k-1)+b20*u_n^2+(c-a)*y(k);
      w_n=-c*w_o+b0-(1+c)*yo+(b11-c*b10)*u_n-c*b20*u_n^2+(c-a)*y_n;

      %% Check to see which minimum variance root gives the lowest
      %% prediction of w(t).
      if w_n<w_p
        u(k)=u_n;
        y_o=y_n; w_o=w_n;
      else
        u(k)=u_p;
        y_o=y_p; w_o=w_p;
      end;
    %% Calculate constant control if minimum variance control complex.
    else
      %% Calculate the one step ahead minimum expectation control.
      u(k)=-b10/(2*b20);
      %% Update the predictions of y(t) and w(t).
      y_p=-c*y_o+b0+b10*u(k)+b11*u(k-1)+b20*u(k)^2+(c-a)*y(k);
      w_p=-c*w_o+b0-(1+c)*yo+(b11-c*b10)*u(k)-c*b20*u(k)^2+(c-a)*y_p;
      y_o=y_p; w_o=w_p;
    end;

    %% Calculate the loss.
    if k>100
    lossp(k-99)=lossp(k-100)+(y(k)-yo)^2;
    end;

end;

%% Calculate the variance, loss and mean.
var=cov(y(100:length))
loss=sum(((y(100:length)-yo).^2))/(length-100)
mean_y=mean(y(100:length))

k=1:length;
clg;
```

```
subplot(2,1,1)
  plot(k,y(k),k,u(k))
  grid;

subplot(2,1,2);
  plot(k,[zeros(1,99) lossp]);
  grid;
```

hybrid.m The hybrid.m program is exactly the same as the one_step_w.m program, except that the expression for the constant control should be replaced with the following line of code.

```
%% Calculate the Expectation control.
u(k)=-(B1)/(2*B2)+sqrt((B1/(2*B2))^2+((A1*yo-b0)/B2));
```

hybrid_extremum.m The hybrid_extremum.m program is exactly the same as the one_step_w.m program, except that the expression for the constant control should be replaced with the following line of code.

```
%% Calculate the Extremum control.
u(k)=-(B1)/(2*B2);
```

two_steps_calc_C_1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to symbolically calculate the Two Steps control
% for the C=1 case, two_steps_calc_C_1.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

%% Define all symbolic variables.
u=sym('u');          %% u(t).
u2=sym('u^2');       %% u(t)^2.
u_1=sym('u_1');      %% u(t-1).
u_12=sym('u_1^2');   %% u(t-1)^2.
up1=sym('up1');      %% u(t+1).
up12=sym('up1^2');   %% u(t+1)^2.

y=sym('y');          %% y(t).
yo=sym('yo');        %% y_o.

%% Build, symbolically, the one step ahead prediction equation.
y1_1=symop('0.25','+','0.5','*',u,'+','0.5','*',u2,...
           '-','0.25','*',u_1);
y1_2=symop('0.95','*',y);
y1_3=symop(y1_1,'+',y1_2);
y1_4=symop(y1_3,'-',yo);

%% Build, symbolically, the two steps ahead prediction equation.
y2_1=symop('0.4875','+','0.5','*',up1,'+','0.225','*',u,....
           '+','0.5','*',up12);
y2_2=symop('0.475','*',u2,'-','0.2375','*',u_1);
y2_3=symop('0.9025','*',y);
y2_4=symop(y2_1,'+',y2_2,'+',y2_3,'-',yo);

%% Construct, symbolically, the cost function for GPC with
%% horizon two.
J=symop(y1_4,'*',y1_4,'+',y2_4,'*',y2_4);
cJ=simple(J)

%% Calculate the derivative with respect to u(t).
dJ_du = diff(J,'u');
%% Calculate the derivative with respect to u(t+1).
dJ_dup1 = diff(J,'up1');
%% Collect terms.
cdJ_du = collect(dJ_du,'u')
cdJ_dup1 = collect(dJ_dup1,'up1')

%% Solve the two polynomials simultaneously for u(t) and u(t+1).
solve(dJ_du,dJ_dup1,'u,up1')
```

two_steps_C_1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the Two Steps controller for the
% C equal to one case, two_steps_C_1.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
y=zeros(1,length);
yhp1_old=0; %% The one step ahead prediction of y given t-1.
lossp=zeros(1,length-100);
roots_array=zeros(9,3); %% Array of root values and cost
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=0; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;

for k=3:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
  y(k)=thi_act'*theta_act;
  %% Calculate the possible Two Steps controls and store
  %% in roots_array
  ua=roots([10 10 5+19*y(k)-5*u(k-1)-20*yo]);
  for i=1:2
    up1a=roots([10 10 5-5*ua(i)-yo]);
    if i==1
      roots_array(1:2,1:2)=[ua(i) up1a(1);ua(i) up1a(2)];
    else
      roots_array(3:4,1:2)=[ua(i) up1a(1);ua(i) up1a(2)];
    end
  end

  ub=roots([15220 17130 (14320+28918*y(k)-7610*u(k-1)-31200*yo)...
             3305+10849*y(k)-2855*u(k-1)-11600*yo]);
  up1b = -0.5;
  roots_array(5:7,1:2)=[ub(1) up1b;ub(2) up1b;ub(3) up1b];

  uc= -0.5;
  up1c=roots([400 400 395-190*u(k-1)+722*y(k)-800*yo]);
  roots_array(8:9,1:2)=[uc up1c(1);uc up1c(2)];

  %% Calculate the cost of each pair of roots and store value
  %% in roots_array.
  ut=roots_array(:,1); up1t=roots_array(:,2);
  roots_array(:,3)=...
    (0.25+0.5*ut+0.5*ut.^2-0.25*u(k-1)+0.95*y(k)-yo).^2+...
    (0.4875+0.5*up1t+0.225*ut+0.5*up1t.^2+0.475*ut.^2-0.2375*u(k-1)...
    +0.9025*y(k)-yo).^2;

  %% Set cost to infinity if root is complex.
   for i=1:9
    if ((imag(roots_array(i,1))~=0)|(imag(roots_array(i,2))~=0))
      roots_array(i,:)=NaN*ones(1,3);
    end
  end

  %% Find the lowest cost root.
  [x,p]=sort(roots_array(:,3));
  %% Find root, out of the roots that have the same minimal cost,
  %% which is closest to the last control.
  i=1;temp=0;
  while x(i)==x(1)
    temp(i,1)=p(i);temp(i,2)=u(k-1)-roots_array(p(i),1); i=i+1;
  end
  [xx,pp]=sort(temp(:,2));

  u(k)=roots_array(p(pp(1)),1);
  ukp1=roots_array(p(pp(1)),2);

end;

%% Calculate the variance, loss and mean.
var=cov(y(100:length))
loss=sum(((y(100:length)-yo).^2))/(length-100)
```

```
mean_y=mean(y(100:length))
```

two_steps_calc.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to symbolically calculate the Two Steps control
% for the C as in the example system case, two_steps_calc.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

%% Define all symbolic variables.
u=sym('u');           %% u(t).
u2=sym('u^2');        %% u(t)^2.
u_1=sym('u_1');       %% u(t-1).
u_2=sym('u_2');       %% u(t-2).
u_12=sym('u_1^2');    %% u(t-1)^2.
up1=sym('up1');       %% u(t+1).
up12=sym('up1^2');    %% u(t+1)^2.

y=sym('y');           %% y(t)
y_1=sym('y_1');       %% y(t-1)
yo=sym('yo');         %% y_o.
yhp1_old=sym('yhp1_old');        %% prediction of y(t) given t-1.
yhp2_oldold=sym('yhp2_oldold'); %% prediction of y(t) given t-2.

%% Build, symbolically, the one step ahead prediction equation.
y1_1=symop('0.25','+','0.5','*',u,'-','0.25','*',u_1,...
          '+','0.5','*',u2);
y1_2=symop('0.45','*',y,'+','0.5','*',yhp1_old);
y1_3=symop(y1_1,'+',y1_2);
y1_4=symop(y1_3,'-',yo);

%% Build, symbolically, the two steps ahead prediction equation.
y2_1=symop('0.54375','+','0.5','*',up1,'+','0.225','*',u,...
          '-','0.125','*',u_1);
y2_2=symop('-0.05625','*',u_2,'+','0.5','*',up12,...
          '+','0.475','*',u2,'+','0.1125','*',u_12);
y2_3=symop('0.4275','*',y,'+','0.21375','*',y_1,...
          '+','0.25','*',yhp2_oldold);
y2_4=symop(y2_1,'+',y2_2,'+',y2_3,'-',yo);

%% Construct, symbolically, the cost function for GPC with
%% horizon two.
J=symop(y1_4,'*',y1_4,'+',y2_4,'*',y2_4)

%% Calculate the derivative with respect to u(t).
dJ_du = diff(J,'u');
%% Calculate the derivative with respect to u(t+1).
dJ_dup1 = diff(J,'up1');
%% Collect terms.
cdJ_du = collect(dJ_du,'u')
cdJ_dup1 = collect(dJ_dup1,'up1')
```

```
%% Solve the two polynomials simultaneously for u(t) and u(t+1).
solve(dJ_du,dJ_dup1,'u,up1')
```

two_steps.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the Two Steps controller for the
% C in the example system case, two_steps.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
y=zeros(1,length);
yhp1_old=0;   %% The prediction of y(t) given t-1.
yhp2_oldold=0;%% The prediction of y(t) given t-2.
lossp=zeros(1,length-100);
roots_array=zeros(9,3); %% Array of root values and cost
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;

for k=3:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
  y(k)=thi_act'*theta_act;

  %% Calculate the possible Two Steps controls and store
  %% in roots_array
  ua=roots([10 10 9*y(k)+10*yhp1_old-5*u(k-1)-20*yo+5]);
  for i=1:2
    up1a=roots([400 400 245-200*ua(i)+90*u(k-1)-45*u(k-2)+...
      90*u(k-1)^2+171*y(k-1)+200*yhp2_oldold-40*yo-380*yhp1_old]);
    if i==1
      roots_array(1:2,1:2)=[ua(i) up1a(1);ua(i) up1a(2)];
      else
        roots_array(3:4,1:2)=[ua(i) up1a(1);ua(i) up1a(2)];
    end
  end

  ub=roots([30440 34260 (6498*y(k-1)-1710*u(k-2)+30350-62400*yo-...
       11800*u(k-1)+27396*y(k)+3420*u(k-1)^2+7600*yhp2_oldold+...
       16000*yhp1_old) 10278*y(k)+8000*yhp1_old-405*u(k-2)-...
       4900*u(k-1)+810*u(k-1)^2+1539*y(k-1)-23200*yo+...
```

```
          1800*yhp2_oldold+7015]);
  up1b = -0.5;
  roots_array(5:7,1:2)=[ub(1) up1b;ub(2) up1b;ub(3) up1b];

  uc= -0.5;
  up1c=roots([400 400 440-100*u(k-1)-45*u(k-2)+90*u(k-1)^2+...
          342*y(k)+171*y(k-1)+200*yhp2_oldold-800*yo]);
  roots_array(8:9,1:2)=[uc up1c(1);uc up1c(2)];

  %% Calculate the cost of each pair of roots and store value
  %% in roots_array.
  ut=roots_array(:,1); up1t=roots_array(:,2);
  roots_array(:,3)=...
   (0.25+0.5*ut-0.25*u(k-1)+0.5*ut.^2+0.45*y(k)+...
    0.5*yhp1_old-yo).^2+(0.54375+0.5*up1t+0.225*ut-0.125*u(k-1)-...
    5.625e-2*u(k-2)+.5*up1t.^2+.475*ut.^2+.1125*u(k-1)^2+...
    .4275*y(k)+.21375*y(k-1)+.25*yhp2_oldold-yo).^2;

  %% Set cost to infinity if root is complex.
  for i=1:9
    if ((imag(roots_array(i,1))~=0)|(imag(roots_array(i,2))~=0))
      roots_array(i,:)=NaN*ones(1,3);
    end
  end

  %% Find the lowest cost root.
  [x,p]=sort(roots_array(:,3));
  %% Find root, out of the roots that have the same minimal cost,
  %% which is closest to the last control.
  i=1;temp=0;
  while x(i)==x(1)
    temp(i,1)=p(i);temp(i,2)=u(k-1)-roots_array(p(i),1); i=i+1;
  end
  [xx,pp]=sort(temp(:,2));

  u(k)=roots_array(p(pp(1)),1);
  ukp1(k)=roots_array(p(pp(1)),2);

  %% Calculate the prediction of y(t) given t-1.
  yhp1_old=0.25+0.5*u(k)-0.25*u(k-1)+0.5*u(k)^2+0.45*y(k)+...
           0.5*yhp1_old;

  %% Calculate the prediction of y(t) given t-2.
  yhp2_oldold=...
    0.3625+0.5*u(k)-0.025*u(k-1)-0.1125*u(k-2)+0.5*u(k)^2+...
    0.225*u(k-1)^2+0.4275*y(k-1)+0.5*yhp2_oldold;
end;

%% Calculate the variance, loss and mean.
var=cov(y(100:length))
loss=sum(((y(100:length)-yo).^2))/(length-100)
mean_y=mean(y(100:length))
```

three_steps.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the Three Steps controller,
% three_steps_sim.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
Ut=zeros(3,length); %% Starting point for optimization alg.
y=zeros(1,length);
yhp1_old=0; %% The prediction of y(t) given t-1.
yhp2_old=0; %% The prediction of y(t) given t-2.
yhp3_old=0; %% The prediction of y(t) given t-3.
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;

%% Set the options for the simulation, see Optimization
%% Toolbox manual.
opts=foptions; opts(1)=0; opts(2)=1e-4; opts(6)=0; opts(9)=0;
opts(14)=400;

for k=4:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
  y(k)=thi_act'*theta_act;

  %% Find the minimum of the cost function, see the
  %% cost_three_step.m function which follows.
  [Ut(:,k),options]=fminu('cost_three_steps',Ut(:,k-1),opts,[],...
                   u(k-1),u(k-2),u(k-3),y(k),y(k-1),y(k-2),yo,...
                   yhp1_old,yhp2_old,yhp3_old);

  u(k)=Ut(1,k);
  %% Calculate the prediction of y(t) given t-1.
  yhp1_old=0.25+0.5*Ut(1,k)-0.25*u(k-1)+0.5*Ut(1,k)^2+0.45*y(k)+...
          0.5*yhp1_old;

  %% Calculate the prediction of y(t) given t-2.
  yhp2_old=0.3625+0.5*Ut(2,k)-0.025*Ut(1,k)-0.1125*u(k-1)+...
          0.5*Ut(2,k)^2+0.225*Ut(1,k)^2+0.4275*y(k)+...
          0.5*yhp2_old;
  %% Calculate the prediction of y(t) given t-3.
  yhp3_old=0.469375+0.5*Ut(3,k)-0.025*Ut(2,k)+0.10125*Ut(1,k)-...
           0.106875*u(k-1)+0.5*Ut(3,k)^2+0.225*Ut(2,k)^2+...
           0.21375*Ut(1,k)^2+0.406125*y(k)+0.5*yhp3_old;
end;

%% Calculate the variance, loss and mean.
var=cov(y(100:length))
loss=sum(((y(100:length)-yo).^2))/(length-100)
mean_y=mean(y(100:length))
```

cost_three_steps.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function required by the fminu Optimization Toolbox function.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function V = cost_three_steps(u,u_1,u_2,u_3,y,y_1,y_2,yo,...
                              yhp1_old,yhp2_old,yhp3_old)

%% Calculate the cost for the given u(1), u(2) and u(3) values.

V=(0.25+0.5*u(1)-0.25*u_1+0.5*u(1)^2+0.45*y+0.5*yhp1_old-yo)^2+...
  (0.3625+0.5*u(2)-0.025*u(1)-0.1125*u_1+...
   0.5*u(2)^2+0.225*u(1)^2+0.4275*y+0.5*yhp2_old-yo)^2+...
  (0.469375+0.5*u(3)-0.025*u(2)+0.10125*u(1)-0.106875*u_1+...
   0.5*u(3)^2+0.225*u(2)^2+0.21375*u(1)^2+0.406125*y+...
   0.5*yhp3_old-yo)^2;
```

min_var_dym_C_1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to run the Dynamic Programming Algorithm for the
% C=1 case, min_var_dym_C_1.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

%% Initialize all the variables.
start=1000; %% Define the starting point of the simulation
endtime=990;%% Define the end point of the simulation
discrete_step=0.1;%% Define Discretization step length.
var=0.04;
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=0; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;

%% Call discretize.m to discretize w.
w=discretize(discrete_step,0.125); len_w=length(w);
%% Make sure that the 0.125 value is in the discretization
```

```
i=1;
while w(i)< 0.125
  i=i+1;
  end
w(i)=0.125;

%% Call discretize.m to discretize u.
un=discretize(discrete_step,-0.25); len_un=length(un);

%% Call discretize.m to discretize e.
en=discretize(discrete_step,0); len_en=length(en);

%% Define matrix to store the control tables.
U_w=zeros(len_w,(start-endtime));

%% Calculate the One step control for the first step.
for i=1:len_un
  if w(i) < (b10^2/(4*b20))
    u1(i)=-(b10/(2*b20))+sqrt((b10/(2*b20))^2-(w(i)/b20));
  else
    u1(i)=-b10/(2*b20);
  end
end

%% Calculate the cost, and create the cost table for the first step
v=(b10*u1+b20*u1.^2+w).^2;
U_w(:,1)=u1';
v_tab=[w;v]';
v_tab(1,:)=[-1e10 0];v_tab(len_w,:)=[1e10 1e+20];

%% Call loop_C_1.m to perform the main looping.
[U_w] = loop(w,en,un,len_w,len_un,len_en,start,endtime,...
            U_w,v_tab,var,a,b10,b11,b20,c,b0,yo);
```

loop_C_1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to do the main looping of the Dynamic Programming
% Algorithm for the C=1 case, loop_C_1.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [U_w] = loop(w,en,un,len_w,len_un,len_en,start,...
              endtime,U_w,v_tab,var,a,b10,b11,b20,c,b0,yo);

%% To optimize the efficiency of the code, a meshgrid is
%% created so that the calculation of f_uw can be performed
%% in one step.
[Un,En]=meshgrid(un,en);

%% Calculate the f_e matrix.
f_e=(1/sqrt(2*pi*var))*exp((-en.^2)/(2*var));
f_e=meshgrid(f_e,1:len_un)';

for k=(start-1):-1:endtime
```

```
    for i=1:len_w
      %% Calculate the f_uw function.
      f_uw=b0-(a+1)*yo+(b11-a*b10)*Un-a*b20*Un.^2-a*w(i)-a*En;

      %% Change the matrix f_uw into a vector Q so that the
      %% interpolation can be done in one operation.
      Q=reshape(f_uw,len_en*len_un,1);
      Q=table1(v_tab,Q);
      %% Change the vector Q into a matrix.
      Jn=reshape(Q,len_en,len_un);

      %% Calculate the product of Jn and f_e.
      Jn_f_e=Jn.*f_e;

      %% Calculate the cost of the step.
      Jn_1_f_e=(b10*un+b20*un.^2+w(i)).^2+trapz(en,Jn_f_e);

      %% Find the minimum cost.
      [Y,I]=sort(Jn_1_f_e);

      %% Store the minimum control in the U_w matrix and the
      %% corresponding cost in table v_tab, for the next step.
      U_w(i,(start-k)+1)=un(I(1));
      v_tab(i,2)=Y(1);
    end

    %% Make sure that the interpolation always has a value to
    %% go to.
    v_tab(1,2)=0; v_tab(len_w,2)=1e+20;
  end
```

discretize.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to perform the discretization for the Dynamic
% Programming Algorithm, discretize.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x] = discretize(discrete_step,mean_est);

%% Find the linear discretized values.
i=1;
xb=0:discrete_step:1;

%% Convert the linear discretization in to a exponential
%% discretization.
x=exp((xb.^2)/0.2)-1;
while x(i)<=10
  i=i+1;
end
x=x(1:i);
sx=length(x);
```

```
%% Create the negative values and add offset.
x=[-flipud(x(2:sx)')' x];
x=x+mean_est;
```

min_var_dym.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to run the Dynamic Programming Algorithm for the
% C as in the example system case, min_var_dym.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

%% Initialize all the variables.
start=1000; %% Define the starting point of the simulation
endtime=990;%% Define the end point of the simulation
discrete_step=0.3;%% Define Discretization step length.
var=0.04;
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;


%% Call discretize.m to discretize w.
w=discretize(discrete_step,0.125); len_w=length(w);
%% Make sure that the 0.125 value is in the discretization
i=1;
while w(i)< 0.125
  i=i+1;
  end
w(i)=0.125;

%% Call discretize_un.m to discretize u.
un=discretize_un(w,len_w,b10,b20,B1,B2); len_un=length(un);

%% Call discretize.m to discretize e.
en=discretize(discrete_step,0); len_en=length(en);
%% Suppress the extremities of en, as they are negligible.
%en=en(17:55); len_en=length(en);%0.02

%% Define matrix to store the control tables.
U_w=zeros(len_w,(start-endtime));

%% Calculate the One step control for the first step.
for i=1:len_un
  if w(i) < (b10^2/(4*b20))
    u1(i)=-(b10/(2*b20))+sqrt((b10/(2*b20))^2-(w(i)/b20));
  else
    u1(i)=-b10/(2*b20);
  end
```

```
end

%% Calculate the cost, and create the cost table for the first step
v=(b10*u1+b20*u1.^2+w).^2;
U_w(:,1)=u1';
v_tab=[w;v]';
v_tab(1,:)=[-1e10 0];v_tab(len_w,:)=[1e10 1e+20];

%% Call loop.m to perform the main looping.
[U_w] = loop(w,en,un,len_w,len_un,len_en,start,endtime,...
            U_w,v_tab,var,a,b10,b11,b20,c,b0,yo);
```

loop.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to do the main looping of the Dynamic Programming
% Algorithm for the C=1 case, loop_C_1.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [U_w] = loop(w,en,un,len_w,len_un,len_en,start,...
               endtime,U_w,v_tab,var,a,b10,b11,b20,c,b0,yo);

%% To optimize the efficiency of the code, a meshgrid is
%% created so that the calculation of f_uw can be performed
%% in one step.
[En1,En2]=meshgrid(en,en);

%% Calculate the f_e matrix.
f_e=(1/(2*pi*var))*exp(-(En1.^2+En2.^2)/(2*var));

%% Define the vector of cost values.
Jn_1_f_e=zeros(1,len_un);

for k=(start-1):-1:endtime

  for i=1:len_w

    %% Note that one extra loop is now required.
    for j=1:len_un
      %% Calculate the f_uw function.
      f_uw=b0-(a+1)*yo+(b11-a*b10)*un(j)-a*b20*un(j)^2-...
           a*w(i)-a*En1-a*c*En1;

      %% Change the matrix f_uw into a vector Q so that the
      %% interpolation can be done in one operation.
      Q=reshape(f_uw,len_en*len_en,1);
      Q=table1(v_tab,Q);
      %% Change the vector Q into a matrix.
      Jn=reshape(Q,len_en,len_en);

      %% Calculate the product of Jn and f_e.
      Jn_f_e=Jn.*f_e;

      %% Calculate the cost of the step.
      Jn_1_f_e(j)=(b10*un(j)+b20*un(j)^2+w(i))^2+...
                  trapz(en,trapz(en,Jn_f_e));
    end

    %% Find the minimum cost.
    [Y,I]=sort(Jn_1_f_e);

    %% Store the minimum control in the U_w matrix and the
    %% corresponding cost in table v_tab, for the next step.
    U_w(i,(start-k)+1)=un(I(1));
    v_tab(i,2)=Y(1);
  end

  %% Make sure that the interpolation always has a value to
  %% go to.
  v_tab(1,2)=0; v_tab(len_w,2)=1e+20;
end
```

discretize_un.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to perform the discretization for the Dynamic
% Programming Algorithm C not equal to one case, of the
% un variable, discretize_un.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [u] = discretize_un(w,len_w,b10,b20,B1,B2);

%% Use the Hybrid extremal controller to produce the un
%% values.
%% Calculate the controls using the positive roots.
i=1;
while w(i) < (b10^2/(4*b20))
  u(i)=-(b10/(2*b20))+sqrt((b10/(2*b20))^2-(w(i)/b20));
  i=i+1;
end

%% Insert the constant extremal control, -(B1)/(2*B2).
u(i)=-(B1)/(2*B2);

%% Calculate the controls using the negative roots.
i=1;
while w(i) < (b10^2/(4*b20))
  temp(i)=-(b10/(2*b20))-sqrt((b10/(2*b20))^2-(w(i)/b20));
  i=i+1;
end

%% Join both controls together.
u=[u fliplr(temp)];
```

dym_sim.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the Dynamically programmed table 20
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Load in the dynamic programming tables, and set U_w equal to
%% table 20.
load C_case_data_ye
U_w=U_w(:,20); w=w';

%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
y=zeros(1,length);
w_k=zeros(1,length);
lossp=zeros(1,length-100);
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;

for k=3:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
  y(k)=thi_act'*theta_act;


  %% Calculate the w value and interpolate to get u.
  w_k(k)=b0+b11*u(k-1)-a*y(k)-yo;
  u(k)=interp1(w,U_w,w_k(k),'linear');

  %% Calculate the loss function.
  if k>100
  lossp(k-99)=lossp(k-100)+(y(k)-yo)^2;
  end;
end

%% Calculate the variance, loss and mean.
var=cov(y(100:length))
loss=sum(((y(100:length)-yo).^2))/(length-100)
mean_y=mean(y(100:length))
```

min_exp_adp.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the Adaptive Expectation controller,
% min_exp_adp.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
y=zeros(1,length);
lossp=zeros(1,length-100);
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;
%% Initialize the ELS variables.
lamda=0.99;
ee=zeros(1,length);
P=10e4*eye(6);
theta_est=ones(6,length);

for k=3:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
  y(k)=thi_act'*theta_act;

  %% Perform the ELS identification.
  thi=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 ee(k-1) 1]';
  ee(k)=y(k)-thi'*theta_est(:,k-1);
  K=P*thi*inv(lamda+thi'*P*thi);
  theta_est(:,k)=theta_est(:,k-1)+K*ee(k);
  P=(P-K*thi'*P)/lamda;
  a=theta_est(1,k); b10=theta_est(2,k); b11=theta_est(3,k);
  b20=theta_est(4,k); c=theta_est(5,k); b0=theta_est(6,k);
  B1=b10+b11; B2=b20;

  %% Calculate the Expectation control based on the parameter
  %% estimates.
  u(k)=-(B1)/(2*B2);

  %% Calculate the loss.
  if k>100
    lossp(k-99)=lossp(k-100)+(y(k)-ye)^2;
  end;
```

```
end;

%% Calculate the variance, loss and mean.
var=cov(y(100:length))
loss=sum(((y(100:length)-ye).^2))/(length-100)
mean_y=mean(y(100:length))
```

one_step_w_adp.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program to simulate the Adaptive One Step controller,
% one_step_w_adp.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%% Define Simulation length.
length=1000;
%% Define start seed so that all e(t) sequences will be the
%% same.
seed_start=10;
randn('seed',seed_start); e=0.2*randn(1,length);
%% Initialize all the variables.
u=zeros(1,length);
y=zeros(1,length);
w=zeros(1,length);
y_o=0; w_o=0;
lossp=zeros(1,length-100);
a=-0.95; b10=0.5; b11=-0.25; b20=0.5; c=-0.5; b0=0.25;
A1=1+a; B1=b10+b11; B2=b20;
theta_act=[a b10 b11 b20 1 c b0]';
%% Calculate the extremum.
ye=(b0/A1)-(B1^2)/(4*A1*B2);
%% Set the set point.
yo=ye;
%% Initialize the ELS variables.
lamda=0.999;
ee=zeros(1,length);
P=10e4*eye(6);
theta_est=ones(6,length);

for k=3:length
  %% Calculate the actual y(t) value.
  thi_act=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 e(k) e(k-1) 1]';
  y(k)=thi_act'*theta_act;

  %% Perform the ELS identification.
  thi=[-y(k-1) u(k-1) u(k-2) u(k-1)^2 ee(k-1) 1]';
  ee(k)=y(k)-thi'*theta_est(:,k-1);
  K=P*thi*inv(lamda+thi'*P*thi);
  theta_est(:,k)=theta_est(:,k-1)+K*ee(k);
  P=(P-K*thi'*P)/lamda;
  a=theta_est(1,k); b10=theta_est(2,k); b11=theta_est(3,k);
  b20=theta_est(4,k); c=theta_est(5,k); b0=theta_est(6,k);
  A1=1+a; B1=b10+b11; B2=b20;

  %% Calculate the extremum based on the parameter estimates.
  yo=(b0/A1)-(B1^2)/(4*A1*B2);

  %% Calculate the w(t) value.
  w(k)=-c*w(k-1)+b0-(1+c)*yo+(b11-c*b10)*u(k-1)-c*b20*u(k-1)^2+...
        (c-a)*y(k);

  %% If the minimum variance roots are not complex, calculate them.
  %% Note that the condition on w has changed due to the possibility
  %% of b20 going negative.
  if (b10^2-4*b20*w(k))>0
    %% Calculate the prediction of y(t) and then w(t), positive
    %% quadratic sign.
    u(k)=-(b10/(2*b20))+sqrt((b10/(2*b20))^2-(w(k)/b20));
    y_p=-c*y_o+b0+b10*u(k)+b11*u(k-1)+b20*u(k)^2+(c-a)*y(k);
    w_p=-c*w_o+b0-(1+c)*yo+(b11-c*b10)*u(k)-c*b20*u(k)^2+(c-a)*y_p;

    %% Calculate the prediction of y(t) and then w(t), negative
    %% quadratic sign.
    u(k)=-(b10/(2*b20))-sqrt((b10/(2*b20))^2-(w(k)/b20));
    y_m=-c*y_o+b0+b10*u(k)+b11*u(k-1)+b20*u(k)^2+(c-a)*y(k);
    y_o=y_m;
    w_m=-c*w_o+b0-(1+c)*yo+(b11-c*b10)*u(k)-c*b20*u(k)^2+(c-a)*y_m;
    w_o=w_m;
    %% Check to see which minimum variance root gives the lowest
    %% prediction of w(t).
    if w_m>w_p
      u(k)=-(b10/(2*b20))+sqrt((b10/(2*b20))^2-(w(k)/b20));
      y_o=y_p; w_o=w_p;
    end;
  %% Calculate constant control if minimum variance control complex.
  else
    %% Calculate the one step ahead minimum expectation control.
    u(k)=-b10/(2*b20);
    %% Update the predictions of y(t) and w(t).
    y_p=-c*y_o+b0+b10*u(k)+b11*u(k-1)+b20*u(k)^2+(c-a)*y(k);
    w_p=-c*w_o+b0-(1+c)*yo+(b11-c*b10)*u(k)-c*b20*u(k)^2+(c-a)*y_p;
    y_o=y_p; w_o=w_p;
  end;

  %% Calculate the loss.
  if k>100
    lossp(k-99)=lossp(k-100)+(y(k)-ye)^2;
  end;
end;

%% Calculate the variance, loss and mean.
var=cov(y(100:length))
loss=sum(((y(100:length)-yo).^2))/(length-100)
mean_y=mean(y(100:length))
```

hybrid_extremum_adp.m The hybrid_extremum_adp.m program is exactly the same as the one_step_w_adp.m program, except that the expression for the constant control should be replaced with the following line of code.

```
%% Calculate the Extremum control.
u(k)=-(B1)/(2*B2);
```

Bibliography

[ÅW90] K. J. Åström and B. Wittenmark. *Computer Controlled Systems*, chapter 6. Prentice Hall, Englewood Cliffs, New Jersey, second edition, 1990.

[ÅW95] K. J. Åström and B. Wittenmark. *Adaptive Control*, chapter 4. Addison-Wesley, Reading, Massachusetts, second edition, 1995.

[Ber76] D.P. Bertsekas. *Dynamic Programming and Stochastic Control*, volume 125 of *Mathematics in Science and Engineering*. Academic Press, New York, 1976.

[Fle80] R. Fletcher. *Unconstrained Optimization*, volume 1, chapter 3. John Wiley & Sons Ltd. Chichester, England, 1980.

[LB85] I. J. Leontarisis and S. A. Billings. Input-output parameter models for nonlinear systems. Part 1: Deterministic nonlinear systems. Part 2: Stochastic nonlinear systems. *Int. J. of Control*, 41:303–344, 1985.

[LS83] L. Ljung and T. Söderström. *Theory and Practice of Recursive Identification*. MIT Press, Cambridge, Massachusetts, 1983.

[Wit93] B. Wittenmark. Adaptive control of a stochastic non-linear system: An example. *Int. J. Adaptive Control and Signal Processing*, 7:327–337, 1993.

[WU95] B. Wittenmark and A.K. Urquhart. Adaptive extremal control. In *The* 34^{th} *Conference on Decision and Control, New Orleans*, 1995. To appear.

[WZ91] P.E. Wellstead and M.B. Zarrop. *Self-Tuning Systems*, chapter 9 & 13. John Wiley & Sons Ltd. Chichester, England, 1991.