1995

Fuzzy control design based on genetic algorithms

Zibo Zhang
University of Wollongong

FUZZY CONTROL DESIGN BASED ON GENETIC ALGORITHMS

A thesis submitted in fulfilment of the requirements for the Award of the Degree

Master of Engineering (Honours)

from

THE UNIVERSITY OF WOLLONGONG

by

Zibo Zhang

M.E. (Honours), HUNAN UNIVERSITY, 1986
B.E. (Honours), HUNAN UNIVERSITY, 1982

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

1995

Declaration

This is to certify that the work presented in this thesis was carried out by the author in the Department of Electrical and Computer Engineering, the University of Wollongong, and has not been submitted to any other university or institute.

.....

Zibo Zhang

Acknowledgments

I would like to thank my supervisor Dr. Fazel Naghdy for his continuos guidance and assistance during the project. Without his help this work could not have been concluded. I would also like to acknowledge the contributions made by Mr P. P. Ciufo, Dr Li Zheng and Ms Bronwyn Evnas towards my experimental work. Also a warm thank to my wife and daughter who have patiently supported me throughout the course in China.

Zibo Zhang
March 1995

FUZZY CONTROL DESIGN BASED ON GENETIC ALGORITHMS

by

Zibo Zhang

Submitted to the Department of Electrical and Computer Engineering in June 1995, in fulfilment of the requirements for the award of the degree

Master of Engineering (Honours)

Abstract

A new methodology for design of fuzzy controllers based on Genetic Algorithms has been proposed. The developed design tool initially identifies an approximate model for a system based on a small set of input/output data of the plant. The system identification is also performed based on Genetic Algorithms. The model is then used in the design procedure. The tuning can be carried out either on the membership functions or the fuzzy rules of the fuzzy controller. The method can be applied to linear and non linear SISO and MIMO systems with time delay and unknown structure. Experimental observations show that the developed methodology performs well and is superior to other considered in this study.

Thesis Supervisor: Dr. F. Naghdy, Senior Lecturer,

Department of Electrical and Computer Engineering, the University of Wollongong.

Contents:

Chapter 1

Introduction

1.1 Problem Statement

Fuzzy control was first introduced by Mamdani in early 1970s [1] and since then it has been successfully applied to a large number of real industrial processes. Fuzzy controllers can easily handle non linear problems, exhibit robust behaviour and are cost effective because human knowledge can be easily captured using rules that contain fuzzy linguistic terms.

In spite of this simplicity, defining the membership functions, determining the number of rules and rule-consequent parameters are generally difficult tasks. This process is referred to as tuning a fuzzy controller to reproduce a desired behaviour at an adequate accuracy.

There has been a significant number of approaches proposed in the literature for tuning fuzzy controllers. This work also addresses this problem by proposing a tuning method developed based on genetic algorithms.

1.2 Aim and Objectives of the Work

The primary aim of this work has been to study the design and tuning procedures of FLC based on genetic algorithms. During the work, the following objectives have been pursued:

I. Study of the work conducted by the authors in the design of fuzzy controllers.
II. Development of techniques for identification of a system as a pre-requisite to design based on genetic algorithms.

III. Development of methodologies for tuning the rule-base and data-base of a FLC using genetic algorithms.

IV. Validation of the techniques developed in the work.

1.3 Significance of Fuzzy Controllers

In spite of considerable progress made in the field of control over the last four decades, PID controllers are still the most popular method implemented in industry. The PID controllers are simple to design and implement. A great deal of knowledge and experience in tuning them for different industrial applications has been also accumulated. In addition, PID controllers can be tuned based on indirect assumptions and a priori knowledge about the system without an explicit mathematical model of the plant.

There are also a number of shortcomings associated with PID controllers. It is rather difficult to achieve high performance control using PIDs. A change in the dynamic parameters of the plant can quickly degrade system performance and even make the plant unstable. In addition if a PID controller is tuned for a particular reference input, a satisfactory performance of the plant cannot be guaranteed if the reference input is changed.

Linear controllers such as state space feedback control systems, optimal control systems, and predictive control systems have also been successfully applied to some industrial processes. The performance characteristics, stability, and robustness of linear controllers can be analytically studied and accurately quantified. The linearity of the plant is, however, the fundamental assumption made in such methods. In the majority of industrial applications, such an assumption does not hold true. Hence a desired performance can not be achieved

if a linear model for a system may not be obtained or the derived linear model does not always describe the dynamic behaviour of the system.

FLC has been introduced as an alternative approach in control. FLC has been cast by many investigators as an expert system paradigm where human-like intelligence may be adopted and applied to a complex and non linear system. Hence FLCs can solve the control problems that PID and linear controller cannot handle due to the complexity of the dynamics of the plant.

Controllers designed based on FLC have been considered as a class of static (memoryless) non linear controllers[1, 2]. The input-output mapping in a FLC is achieved through three steps:

(a) Fuzzification or input fuzzy sets activation from the crisp values of the input sensors.

(b) Output fuzzy sets selection from the input fuzzy sets and the fuzzy rule base.

(c) Defuzzification or generation of crisp output values based on the output fuzzy sets

The fuzzy control methods have been commonly accepted and widely applied to various processes ranging from household appliances to industrial process control systems, and even production planing and scheduling [3]. FLCs have the following advantages:

(i) They are simple to implement and allow bypassing constraining mathematical analysis by a pure algorithmic process.

(ii) They are quite robust with respect to undesirable perturbations occurring within the process itself. This is because the fuzzy controllers can generalise and extrapolate from referential situations.

(iii) They are also robust with respect to outside disturbances affecting the control systems.

(iv) The design of a FLC for a system is not dependent on an accurate model of the plant.

(v) Fuzzy systems lend themselves to hardware realisation (fuzzy chips). This simplifies the implementation of a fuzzy controller.

The design of a FLC has been the main challenge in the utilisation of this method. The rule-base and data-base of a fuzzy controller play a dominant role in its performance. It is, therefore, necessary to tune them for a particular application. The main difficulty in this process is the large number of parameters that should be optimised. This requires an efficient and generic method. Genetic algorithms have been studied in this work for this purpose.

1.4 Genetic Algorithms

Genetic algorithms have been widely used in many fields to produce a global optimal solution. A genetic algorithm is a probabilitically-guided optimisation technique simulating genetic evolution. Due to random mutation procedure, the algorithms can always escape from local minima. Unlike other classical optimisation algorithms, genetic algorithms do not need to compute local derivatives to guide the search process. Through encoding the variables, the algorithms identify the populations with stronger fitness in the whole universe of discource and removes populations with weaker fitness in order to reproduce better offsprings.

Genetic algorithms, therefore, have great potential for identifying a mathematical model of a dynamic system. The search process used can determine the globally optimal parameters of a system.

The population of solutions is explored in parallel in genetic algorithms [2]. The search process is similar to the natural evolution. With the advantages genetic algorithms have been widely used in control systems. [3, 4, 5, 6].

1.5 Structure of Thesis

The details and results of the study conducted in the work are reported in this thesis through seven chapters. The problem studied, the aim, and objectives of the work are introduced in chapter one. A brief description of the nature and significance of FLC and genetic algorithms are also provided in this chapter.

Chapter 2 is mainly dedicated to study of the previous work. A review of the major methods developed in the design and tuning of FLC is conducted. The significance of each work and its outcome are briefly described.

An overview of the design approach developed in this work is provided in chapter 3. In addition FLC and genetic algorithms are described in more details. The methodologies developed to enhance the operation of genetic algorithms for this particular application are also introduced in this chapter.

Chapter 4 describes the first stage of the design, i.e., the identification of a mathematical model of the system. This is achieved through genetic algorithms(GAs). Contrary to conventional system identification methods, this approach can produce a satisfactory result with a small set of input/output data and can identify the structure and order of a system, as well as its time delay.

The optimisation process developed in this work is described in chapter 5. The procedure of automatic tuning of the membership functions and fuzzy rules are explained. The fuzzification and defuzzification processes used in the fuzzy logic controller are presented The encoding and search processes of the optimisation algorithm using GAs are finally addressed.

The validation of the methodology developed in this work is presented in chapter 6 through computer simulation and experimental work. In the simulation, the fuzzy membership functions and rules for a non linear system are optimised. In the experimental work the optimised controller tuned for a DC motor is applied in real time and the results are compared with a PD controller.

Chapter 7 provides an overview of the work carried out and summarises the results. Some conclusions are also drawn highlighting the significance of the work and its drawbacks.

Chapter 2

Background

2.1 Introduction

Fuzzy control has been successfully applied to a large number of real industrial processes. Fuzzy controllers can easily handle non linear control problems, exhibit robust behaviour and are very cost effective. This is because the human knowledge can be easily captured using rules that contain fuzzy linguistic terms.

In spite of this simplicity, defining the membership functions and determining the number of rules and rule-consequent parameters are generally difficult tasks. This process is referred to as tuning of a fuzzy controller to reproduce a desired behaviour with an adequate accuracy.

There have been a significant number of approaches proposed in the literature for tuning fuzzy controllers. They vary from trivial method of "trial and error" to automatic methods handling different stages of design. These methods, though radically different in concept and approach, can be categorised into the following groups:

(i) Trial and Error methods

(ii) Knowledge-base approaches

(iii) Computer-aided techniques

(iv) Design based on conventional control concepts

(v) Fuzzy PID methods

(vi) Neural network methods

(vii) Genetic Algorithms approach

The objective of this chapter is to provide a review on the most important works conducted on the design and tuning of the fuzzy controllers over the last two decades. In the end, the relationship between the reviewed methods and the approach developed in this work will be studied and a summary will be provided.

2.2 Trial and Error Method

This approach often employs the correlation-minimum inference method [7, 8] to transform inputs to outputs. The inputs are the errors between the required responses of the system to be controlled and the real responses of the system. The outputs are the control actions applied to the system. In this method the crisp input variables are initially transformed to fuzzy variables that can be described by a set of linguistic terms. The fuzzy linguistic terms representing the inputs correspond to a set of linguistic terms of the outputs. The performance of the control action is improved using correlation-minimum inference method.

The method is primarily based on the intuition and deduction ability of the designer and cannot be considered as a systematic method for the design of fuzzy controllers. The approach is generally inefficient, particularly if a large number of parameters are tuned.

The trial and error tuning method has been applied to the control of a two-tank system by Thomas and Sebastian [9]. Initially a crude mathematical model for the plant dynamics is

assumed. Then eleven fuzzy rules are derived from the mathematical model. Using trial and error, 32 parameters of the fuzzy rules are tuned for time-optimal control of the system.

2.3 Knowledge-Base Approach

In this method, an operator skilled to operate a system manually is interviewed and the knowledge obtained is used to model the required control strategy in the form of a *look - up table* and the necessary fuzzy rules. The designer may adjust the rules intuitively based on the characteristics of the physical system. Hence the design of the fuzzy controller depends entirely on the knowledge and experience of the expert and intuition of the designer. This approach, therefore, is far from a systematic and reliable method [11] and is effective when the experts can accurately express their knowledge in terms of fuzzy rules.

In the work conducted by M. Sugeno [11], the fuzzy controller was applied to park a model car based on if-then fuzzy rules obtained from a human operator. By tuning the fuzzy rules, the fuzzy controller can successfully park the model car. This approach has also been applied to the control of a cement kiln operation [12] by P. J. King and E. H. Mamdani.

In the work conducted by Floor Van Der Rhee, Hans R. Van Nauta Lemke, and Jaap G. Dijkman [13], the design procedures are divided into two phases. During the first phase the relationship between the input and output of a system is obtained and the knowledge structure is constructed. This is referred to as the learning phase. In the other phase, the knowledge structure is applied to control the process. Both linear and non linear systems can be controlled by this method.

A novel approach referred to as automated fuzzy controller design station (AFCDS) is reported in [14]. An automatic knowledge generator generates the fuzzy rules and semantic intervals. The rules and semantic intervals are then used by a fuzzy tuner to create and optimise the membership functions together with any other parameters. The tuner takes into account design parameters such as allowable percentage of the overshoot, the desired rise time and the maximum desired settling time. The optimised knowledge base is then interfaced to the user in the form of fuzzy rules and membership functions.

2.4 Computer-Aided techniques

Computer-aided tuning approaches for fuzzy controllers have been popular methods and reported in the literature by a number of authors. The work conducted by L. Zheng [15, 16, 17] is based on the knowledge of the process and the input-output data pairs obtained from the process. A max-min fuzzy inference engine and tuning technique by gradient analysis has been introduced. This method cannot automatically complete all the procedures necessary for the design of a fuzzy controller and may become trapped in local optima. Overall the method is not a systematic design technique.

In the work conducted by Boscolo and Drius [18], a crude model of the process to be controlled is assumed to be available from the input/output data. The identification algorithm applied to identify the system model is Instrumental variable (IV). The structure of the mathematical model is assumed to be a linear system of ARMAX type. A gradient optimal method is then employed to tune the membership functions of the fuzzy controller.

This approach has been applied to tune the fuzzy controller of a DC motor. The outcome has been satisfactory for a single set point. This is because the gradient optimal method does not guarantee an optimal performance for all possible set points. The identification

algorithm used to obtain the system model cannot estimate the parameters of a general industrial process with time delay.

A multi-objective design of controllers with fuzzy logic using control engineering computation environment is presented by H. D. Joos and M. Schlothane[21]. This method mainly uses Analysis & Design Control System (ANDECS) and Multi-Objective Programming System (MOPS) to optimise the fuzzy controller. For generating a data object which contains all necessary information about a fuzzy-logic controller, a specific editor module, called FZEDIT, has been developed and integrated in ANDECS. Each 'fuzzy-controller' data object consists of a rule base, a mathematical description of the linguistic variables, and information about the evaluation methods used for fuzzification, inference, and defuzzification. The hierarchical data structure is designed for an arbitrary number of input and output variables, and for rules of arbitrary length.

The multi-objective programming algorithm is employed to optimise the fuzzy rules. In this way if the mathematical model of the real system is available, then the fuzzy controller can be optimised and designed. The work, however, does not address how the mathematical model of the real plant for the simulation of the fuzzy controller is obtained.

2.5 Design Based on Conventional Control Concepts

Fuzzy dynamic programming has been proposed [22, 23, 24]. Since dynamic programming is basically a feedforward control, its application to real physical problems may be difficult. This method may also get trapped within the local optima. Once the targets are changed the fuzzy dynamic programming must be used again to optimise a set of new fuzzy logic rules for the same fuzzy controller. Moreover, According to Hojo, [22] a necessary condition for applying fuzzy dynamic programming is that the state variables can be obtained from the real system. This may not alway be possible.

Tanaka and Sano [25] proposed design methods for fuzzy phase-lead compensators based on frequency and transient characteristics. The methods attempt to find parameters that effectively compensate phase characteristics in the control systems. The work proposes two important theorems. One is to judge whether or not a fuzzy phase-compensator can be used. The second theorem provides a methodology to obtain the compensator. The method has been successfully applied to linear and non linear controlled objects through computer simulation.

2.6 Fuzzy PID Methods

Fuzzy PID controllers have been widely used in the control of real-time systems. The references [26, 27, 28] present several methods for tuning fuzzy PID controllers. Brehm and Rattan [27] propose a reduced fuzzy rule strategy and a hybrid PID controller. The reduced fuzzy rule strategy is used to simplify the design of the fuzzy controller based on crisp PI control theory. The reduced rule tables are illustrated in Figures (2-6-1) and (2-6-2). The fuzzy language terms are listed in table (2-6-1):

Number	Fuzzy Language Term	Symbol for input	Symbol for control action
1	"Negative big"	nb	NB
2	"negative middle"	nm	NM
3	"negative small"	ns	NS
4	"Zero"	0	ZO
5	"positive small"	ps	PS
6	"positive middle"	pm	PM
7	"positive big"	pb	PB

Table (2-6-1) Fuzzy language terms

The actual crisp control actions will be worked out by the defuzzification algorithms as described in **Chapter 5**.

Error / Σerror	nb	nm	ns	0	ps	pm	pb
nb	NB	NB	NB	NB	NM	NS	ZO
nm	NB	NB	NB	NM	NS	ZO	PS
ns	NB	NB	NM	NS	ZO	PS	PM
0	NB	NM	NS	ZO	PS	PM	PB
ps	NM	MS	ZO	PS	PM	PB	PB
pm	NS	ZO	PS	PM	PB	PB	PB
pb	ZO	PS	PM	PB	PB	PB	PB

Figure (2-6-1) PI control rule matrix

The error(k) is the error between the desired target and the real output of the system at the current sampling interval. The "error(k)-error(k-1)" is "change in error" and Σerror(k) expresses the accumulation of the error during the control period.

The PID controller is divided into two separate control actions: (1) PD controller for fastest response and (2) PI to remove the steady state error. Since the zero subset is defined as a range it is possible to have a small error when the control action is zero. In order to reduce the small error once the output of the system reaches the range, the PD controller is replaced with a PI controller. The PI control action eliminates the steady-state error. The other idea proposed by the paper is that when the output of the system is far from the set point the proportional gain will be decreased. On the contrary, when the output of the system is close to the set point the proportional gain is increased.

ΔError / error	nb	nm	ns	0	ps	pm	pb
nb	NB	NB	NB	NB	NM	NS	ZO
nm	NB	NB	NB	NM	NS	ZO	PS
ns	NB	NB	NM	NS	ZO	PS	PM
0	NB	NM	NS	ZO	PS	PM	PB
ps	NM	MS	ZO	PS	PM	PB	PB
pm	NS	ZO	PS	PM	PB	PB	PB
pb	ZO	PS	PM	PB	PB	PB	PB

Figure (2-6-2) PD control rule matrix

Theoretically, the controller designed based on this strategy would behave satisfactory if the fuzzy rules can be well tuned. The paper reports some excellent results obtained from computer simulation.

It should be noted that the reduced fuzzy rule method does not completely solve the problem of design of a fuzzy controller but reduces the number of fuzzy logic rules. When the system to be controlled is changed the designer must retune all the fuzzy rules. Although the total number of fuzzy rules are reduced there are still many more fuzzy rules to be tuned than the number of parameters of a conventional PID controller.

According to the method proposed in [26] a fuzzy controller can be derived from a PD controller using proportional and derivative gains. This is a simple and direct design method. The fuzzy controller designed by this method can reach a much better control performance than the original PD controller. The method, however, does not produce a fully optimal controller. The membership functions are still tuned by trial and error.

In [28] the authors propose a simulation method for determining fuzzy rules in fuzzy control of tracking systems. The method is mainly based on minimising the mean squared error for a combination of the fuzzy rules in the original position control system.

Using this method the fuzzy rules are reduced from 343 to 22. The parameters of the fuzzy rules, however, should be then tuned by 'Trial and Error' method.

2.7 Neural Network Methods

The research reported by W. C. Daugherity [29] involves a new combination of neural and fuzzy systems in which fuzzy meta-rules are used to produce a series of neural networks for longer input sequences. The method has improved the inference capability of the fuzzy system and the learning capability of the neural networks. It can not, however, solve the problem of design of fuzzy controllers.

The method developed by Khan and Venkatapuram [30] presents an elegant scheme to combine the neural networks and fuzzy logic. In this work a multilayered feedforward neural network is applied to learn system's input-output behaviour by using system's input-output data. The other multilayered back propagation neural network, which is called 'Fuzzy Rules Generator' (FRG), directly maps weights of different layers into fuzzy rules and fuzzy membership functions. Figure (2-7-1) shows the procedures that are applied to generate the fuzzy rules and membership functions.

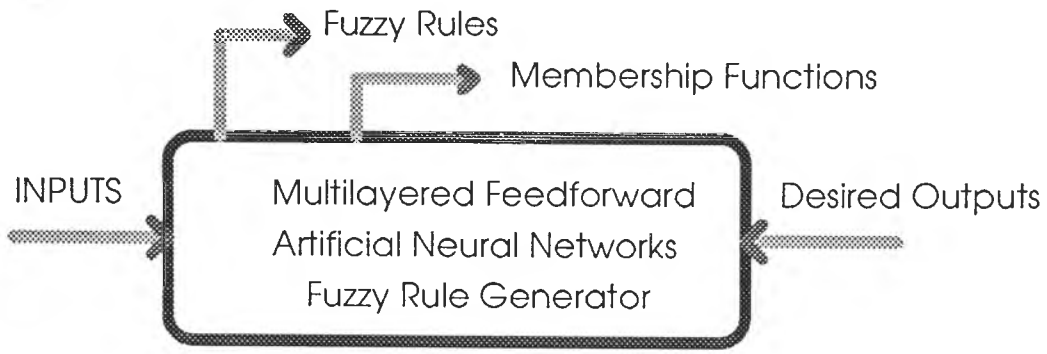


Figure (2-7-1) Creating fuzzy rules and membership functions based on neural network learning

This method uses a three-layered neural network. The first layer is the fuzzification stage whose task is to match the values of the input variables against the labels used in the fuzzy control rule. The first-layer neurons and the weights between layer 1 and layer 2 are also used to define the input membership functions. The middle-layer neurons represents the rule base. When the neural network has learned the knowledge from the input-output data of the real systems, the fuzzy rules and membership functions are generated and tuned. This Neufuz system can only be used to mimic a system model but not to design the fuzzy controller.

In [31] Y. Y. Chen presented a cell-to-cell design method. This method is further developed by L. Fortuna [32]. It requires the state variables of each cell for the design of the fuzzy controller. The plant dynamic is modelled by a set of multilayer neural networks and the fuzzy rules are optimised by annealing approach. Figure (2-7-2) shows the process of design of fuzzy controller, where $F(\bullet, U)$ is the cost function.

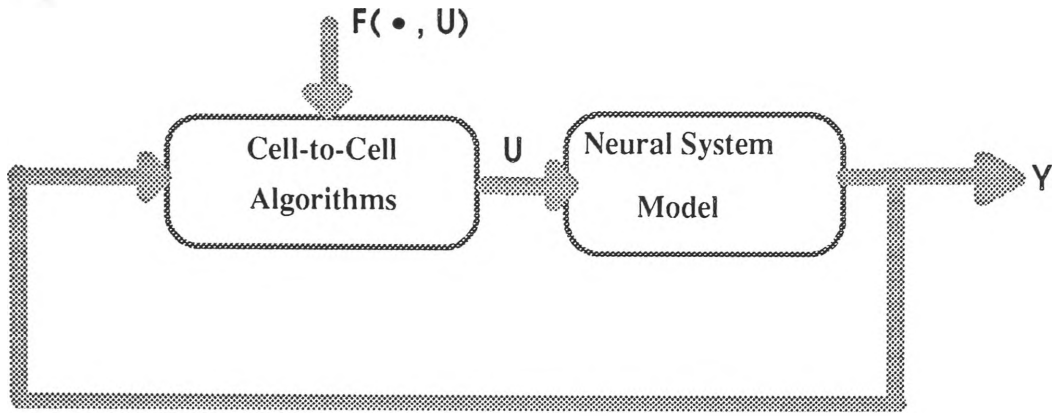


Figure (2-7-2) Cell-to-Cell design process

The proposed method provides a satisfactory system model and overcomes entrapment in local minima. The simulated annealing procedure is, however, very time consuming, as a large set of input/output data is needed to train the neural networks. The cost function $F(\bullet, U)$ used in the optimisation of the fuzzy rules is

$$F(\bullet, U) = \int_0^t [\text{setpoint} - Y(t)]^2 \, dt \qquad (2\text{-}7\text{-}1)$$

In this method the optimisation also takes place for only a specific set point.

A new Neufuz controller has been investigated by M. Balazinski and E. Czogala [10] (1993). This controller has been used to control metal cutting process. In their research work a conventional fuzzy controller is changed into a neural fuzzy controller. The difference between the two controllers is that the knowledge base and decision making of the conventional fuzzy controller are replaced with a neural network which is able to accept sampled fuzzy information in addition to crisp data. The inputs to the neural networks are error and change in error. The presented neural network-fuzzy controller is trained off-line by quantitative measurements obtained during the observation of the process. This method also requires a large set of input/output data. The training period of the neural networks can be also very lengthy.

Another design procedure for fuzzy controllers has been reported by Cheng-Liang. Chen and Wen-Chih. Chen [33]. The method employs a three-layer neural network. The controller calculates the control action according to the input error and change in error. First, a piece wise linear fuzzy controller (PLFC) is designed and transformed to a gaussian potential function network controller (GPFNC). Then the crude GPFNC is optimised. During optimisation, a mathematical model or a well-trained plant emulator is needed to estimate the plant response to the network controller parameters at the current operating point. This plant emulator is not, however, always available.

There are other tuning methods based on neural networks which automatically optimise different stages of design particularly the membership functions. They are referred to in the literature as self-learning or self-organising fuzzy controllers. Some examples are the work conducted by Takagi and Hayashi [34] using neural networks as membership values generator and the work by Nomura et al. [35] that handles fuzzy systems as networks and adjust the membership functions using a back-propagation technique.

2.8 Genetic Algorithms

Genetic algorithms (GAs) are probabilistically-guided optimisation techniques based on the genetic evolution. Because the algorithms are able to optimise a large number of variables simultaneously, they have been developed to tune fuzzy controllers. The five references reviewed in this thesis [36, 37, 38, 3, 4] focus also on processes which will be controlled by fuzzy controllers.

In the work carried out by D. Park [36], the genetic algorithms have been employed to design fuzzy controllers and a new fuzzy reasoning model is proposed. The available domain knowledge is exploited by the genetic algorithm (GA) to produce a better performance for the fuzzy controller. the GA is used to optimise the fuzzy relation matrices

defining the degree of dependency between fuzzy input and fuzzy output variables. The method has produced satisfactory results. It should be noted that the prior knowledge required by the method is not always available.

In [37], the GAs are successfully applied to optimise the objective functions for classification problems including finding the minimum number of fuzzy rules. The method has proven that GAs can be employed to complete the difficult task of design fuzzy controllers for control of industrial processes. Nevertheless, the work does not address the problem of tuning the fuzzy rules.

Fuzzy net controllers (FNC) is a new model of fuzzy controller [39]. In this model a fuzzy controller is converted to a set of multi-layer artificial neural networks for parallel computation of fuzzification and defuzzification. The GAs are used to optimise membership functions of the fuzzy controller. The method has been applied to control a temperature control system and an inverted pendulum. The work does not address how the approximate model of the plant can be obtained for an industrial process.

In the work conducted by S. Isaka and A. V. Sebald [38], GAs are applied to tune the membership functions of a fuzzy controller which is used to control blood pressure. The triangular-shaped membership functions are defined by formula (2-8-1). The main task of the genetic algorithm is to optimise the parameters A, B, and C of the membership functions. The quadratic cost function used is given in equation (2-8-2)

$$\begin{aligned} \mu(x) &= (x-A)/(B-A) && \text{if } A < x < B \\ \mu(x) &= (x-C)/(B-C) && \text{if } B < x < C \\ \mu(x) &= 0 && \text{if } x < A \text{ or } x > C \end{aligned} \qquad (2\text{-}8\text{-}1)$$

$$J = \sum_{t=1}^{T} \{|Y_d - Y(t)| + |u(t) - u(t-1)| \qquad (2\text{-}8\text{-}2)$$

Where Y_d is the target blood pressure, $Y(t)$ is the mean arterial pressure at time t, $u(t)$ is the drug infusion at time t and T is the simulation length.

Through tuning the membership functions, the fuzzy controller can be designed well using the GAs under the condition that the mathematical model is available.

Genetic algorithms have been used to determine the number of fuzzy rules[4, 5], and membership functions [4]. In Karr's method [4], a genetic algorithm is first used to the determine the number of rules according to a predefined rule. A genetic algorithm is used again to tune the membership functions. Lee and Takagi [6] have developed a similar technique with the difference being that two stages are conducted simultaneously with a penalty strategy favouring systems with fewer rules.

2.9 Conclusion

During the course of this chapter a wide range of works addressing the issue of designing a fuzzy controller were reviewed. It was realised that the heuristic methods are time consuming and often unreliable. The majority of the computer-aided design tools presented only optimise the fuzzy controllers for a single set point and often use the gradient method to tune the fuzzy rules. In contrast to these methods, genetic algorithms are able to deal with many tuning parameters simultaneously. The search process can seek for a set of global optimal fuzzy rules based on some criteria that express the design specifications directly.

In this work genetic algorithms will be studied for automatic design and tuning of fuzzy controllers. Genetic algorithms will be initially used to estimate the mathematical model of an industrial process. Then it will be applied to tune the fuzzy membership functions and optimise the fuzzy rules.

Chapter 3

Design Overview

3.1 Introduction

In this chapter the design of a fuzzy logic controller (FLC) in general and the methods developed in this work will be introduced. Initially, the principles behind the operation of FLC will be covered. The structure of an FLC and parameters considered in its design will be then addressed. An overview of the design method developed in this work will be presented. Since this method relies on genetic algorithms at different stages to tune the design, the remainder of the chapter will introduce the genetic algorithms and describe the methodologies developed to enhance its operation for this particular application.

3.2 Fuzzy Set and Fuzzy Logic Control

Fuzzy sets have been developed as an extension to crisp sets. In a crisp set, an object is either a member or not a member of a set. In fuzzy sets, partial membership is possible. Mathematically, the membership to a crisp set C is described by

$$\mu_C(x) = \begin{cases} 1 & \text{if } x \in C \\ 0 & \text{if } x \notin C \end{cases}$$

Whereas for a fuzzy set F the membership is defined as

$$\mu_F : U \rightarrow [0,1]$$

where U is the universal set defined for a specific problem. In fuzzy sets, membership is usually defined in terms of a membership function. As an example the membership function illustrated in Figure (3-2-1) indicates the following membership definition:

$$\mu(x) = \begin{cases} 1 & \text{if} \quad x = 0 \\ \dfrac{x+2}{2} & \text{if} \quad -2 < x < 0 \\ 0 & \text{otherwise} \end{cases},$$

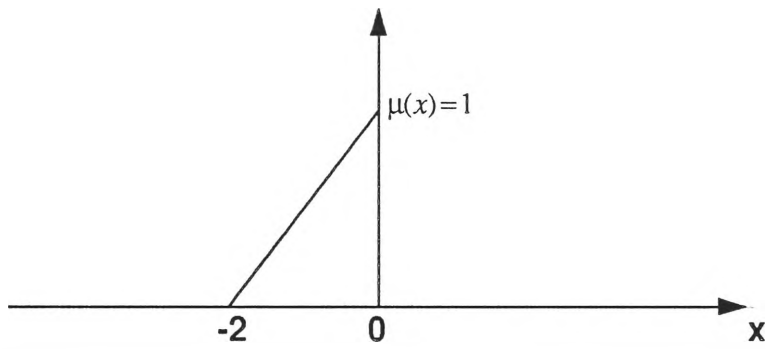


Figure (3-2-1) Example of fuzzy membership function

There are different types of membership functions used in fuzzy logic systems. The most common types are:

(i) Monotonic

(ii) Triangular

(iii) Trapezoidal

(iv) Bell-shaped

Fuzzy logic controllers (FLC) are developed based on fuzzy set theory. It lies in the general stream of a new paradigm of control theory known as expert control [7]. In this approach symbolic computing is used in the design process of the control algorithm. This method provides the opportunity to design simple as well as multivariable controllers with complex control laws.

One of the contributing factors towards development of FLC has been the studies conducted on the characteristics of a human operator as a controller [7]. It has been observed that human beings acts as a non linear controller with time-varying parameters. FLC has been developed to simulate similar behaviour through artificial intelligence.

A fuzzy controller can be illustrated by the block diagram shown in Figure (3-2-2) Similar to a conventional control system, the information about a system is read from the sensors attached to it. In an FLC, the sensor measurements are converted to linguistic labels in the Fuzzifier. There are a number of techniques available for this operation such as conversion into a fuzzy singleton, which is in fact a precise value, adopting a fuzzy operator to convert the probabilistic values contaminated with noise into fuzzy data, or using a hybrid method that involves both the uncertainty (fuzzy numbers) and randomness.

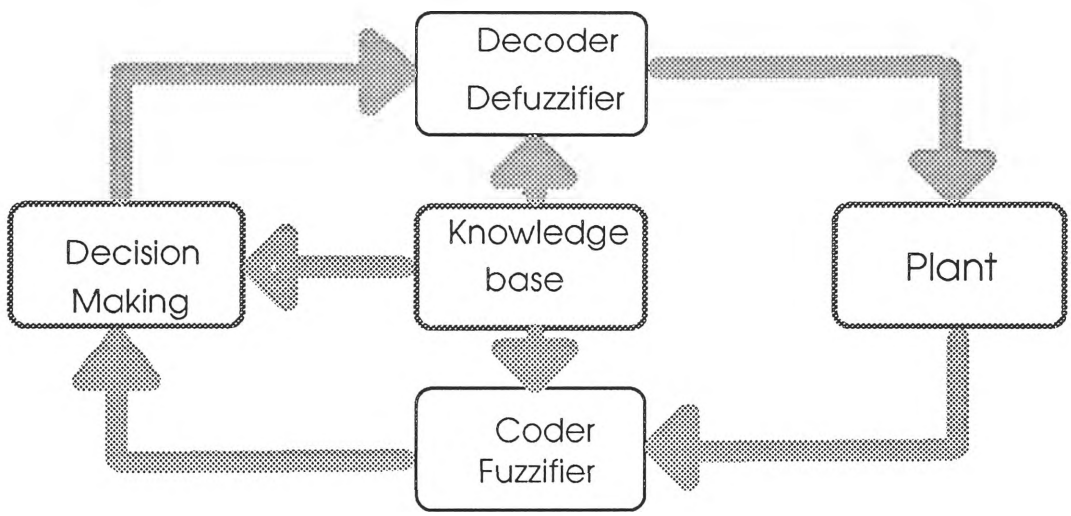


Figure (3-2-2) Simple architecture of FLC

The decision-making logic operates based on the knowledge-base. The knowledge-base (KB) consists of a database and a fuzzy control rule base. The database characterises the fuzzy control rules and fuzzy data manipulation. It defines, particularly, the membership functions. The rule-base is the collection of fuzzy conditional statements based on the process state and

control variables. Various approaches have been used to set up the data-base including the analytical knowledge obtained from the system, study of a human expert, or development of the fuzzy model of the process which is the linguistic description of the system dynamics. In the final stage, a defuzzifier converts the fuzzy values to crisp values in order to drive the system.

In design of a fuzzy controller the most important thing is to optimise these fuzzy rules. A fuzzy controller is a non-linear controller. It can, therefore, control some poorly-understood processes or complex systems if the fuzzy rules are chosen correctly for a particular system. Furthermore, optimising all the fuzzy rules to reach a global optimal solution is not a simple task.

3.3 Design Procedure

]The main strategy used in the design of a fuzzy controller is to identify the main control parameters and to determine a term set that is at the right level of granularity for describing the value of each linguistic variable. A simple example is a term set including linguistic values such as {very small, small, medium, large, very large}.

In this work, a systematic method for the design of FLCs in general has been developed. The design is based on the input/output relationship obtained from a system without any pre-knowledge about its behaviour or knowledge from an expert. Genetic algorithms are used primarily to tune the design parameters for an optimum performance.

The various stages of the design are illustrated in figure (3-3-1). Initially a small set of input and output data is obtained from system. The data set is then normalised to numbers between zero and one in order to be suitable for the design package to a variety of real processes.

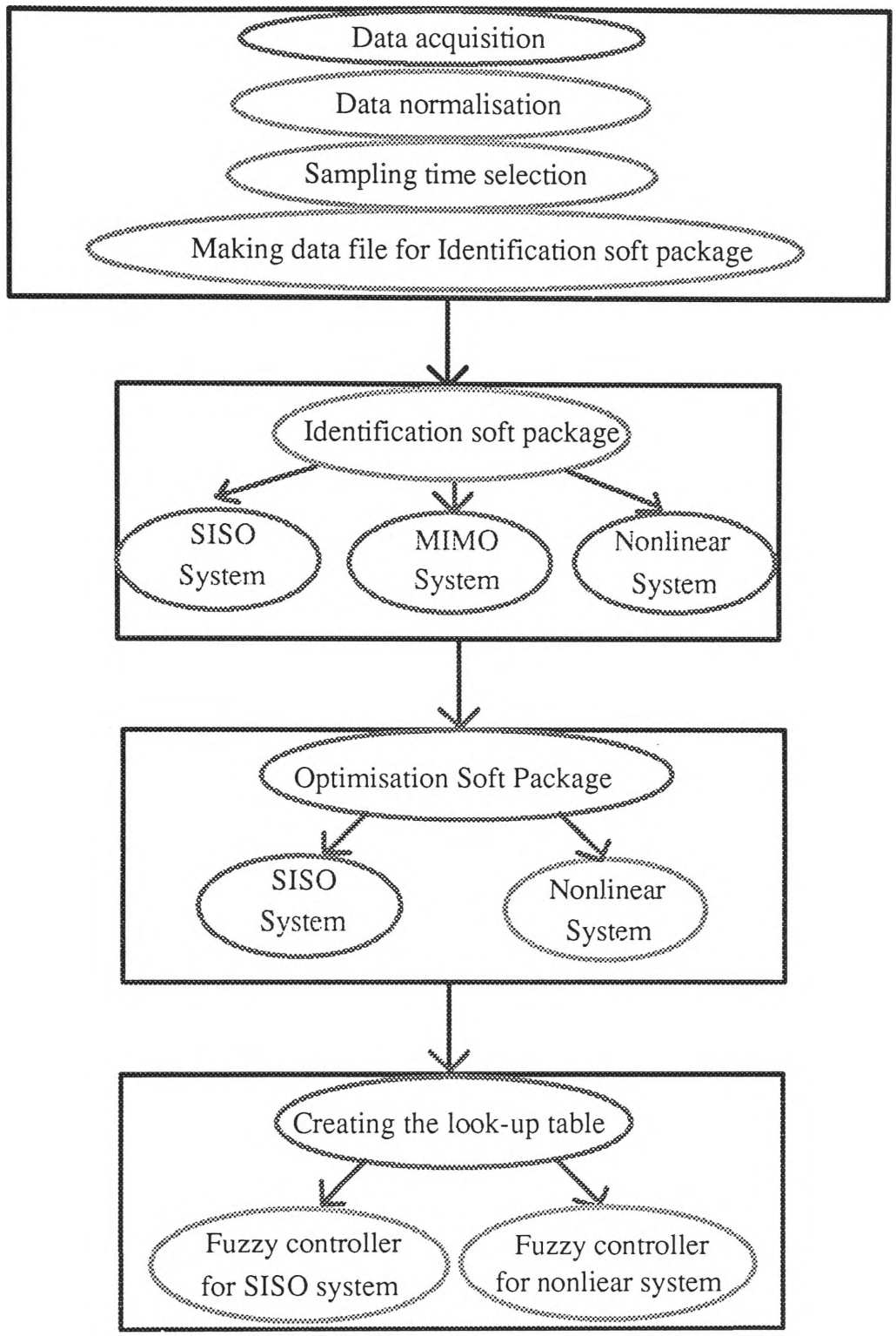


Figure (3-3-1) Overview of Design

The design tool is then able to process the sampled data for a variety of system with different ranges of parameters.

The next stage is to identify a crude model for the system. A specific method based on genetic algorithms (GAs) has been developed which can estimate the parameters of various types of physical systems including Single-Input Single-Output (SISO), Multiple-Input Multiple-Output (MIMO) and non linear of Hammerstien type. The estimated parameters include the coefficients of the system transfer function and its time delay. The structure of a SISO system can be also identified through this method. It is well recognised that conventional algorithms cannot easily identify the system structure and time delay.

When the crude mathematical model is obtained the GAs are employed again to tune the parameters of the fuzzy controller. The designer has the option to tune either the fuzzy rules or the membership functions. These two methods will be discussed in more details in chapter 5.

Once the optimised fuzzy rules or the fuzzy membership functions are obtained a look-up table can be generated and the design process is complete.

3.4 Characteristics of Genetic Algorithms

In this section an overview of GAs and novel approaches developed for GAs used in this work will be provided.

3.4.1 Introduction to Genetic Algorithms

Genetic algorithms (GAs) offer a solution for parameter optimisation problems based on the principle of natural evolution. The solution offered is based on this theory that evolution is an efficient process that implicitly identifies optimal solutions to aspects of events exhibited by the environment [2]. A population of solutions are explored in parallel by genetic algorithms [44].

In GAs, a problem is modelled by defining a candidate solution called an *organism,* which in most applications consists of a single *chromosome.* An iterative procedure is applied to a fixed-size population of organisms. Each iteration step produces a new *generation* of the population by evaluating and modifying the structure of the previous generation.

A chromosome of length n is a vector of the form [49]

$$<x_1, x_2, \ldots, x_n>$$

The component x_i is known as a *gene* and is chosen from a domain of values called the *alphabet.* The most commonly used alphabet is the binary digits [0,1], although sometimes real numbers are used.

The first generation of a population *P(0)* is chosen heuristically or randomly. Using a randomised procedure, the structures of *P(t+1)* is derived from *P(t)* after applying a *genetic recombination operators* to the new population [44]. The most popular recombination operator is *crossover.* Another operator called *mutation* is also applied to a population to introduce new genetic materials into it. A mutation causes a random change of a gene from one alphabet to another. It also helps to reinject any information that may have been lost in the previous generations [44].

3.4.2 Basic Parameters of Genetic Algorithms

There are several parameters in genetic algorithms:

(a) *Population Size* - This is a free parameter which trades off coverage of the search space against the time required to compute the next generation.

(b) *Length of chromosomes* - The length of a chromosome is generally equal to the product of the number of variables to be optimised and the length of every variable encoded in binary. A chromosome represents a combination of its genes. For example,

using genetic algorithms, the time delay d and the parameters of the system $a_1,a_2...a_n$, $b_0,b_1...b_m$ are encoded as10-bit binary strings .

(c) *Length of variables coded in binary*. This is the length of each variable in the chromosome.

(d) *Crossover probability* - This probability controls the frequency at which the crossover occurs for every chromosome in the search process. This is a number between (0,1) which is determined according to the sensitivity of the variables of the search process. For every crossover operation, a random function generates a random number which is compared with the crossover probability. If it is less than or equal to the crossover probability then the crossover operation takes place.

(e) *Mutation probability* - This probability controls the frequency at which mutation occurs for every gene of a chromosome in the search process. The mutation operations will be determined by a random function which generates a number between 0 and 1. If the random number is less than or equal to mutation probability then the mutations operation occurs. The selection of the mutation probability is dependent on the sensitivity of the objective function to the variables. The mutation determines the number of bits on which mutation will be carried out.

(f) *Objective Function* - This is the main evaluation function based on which the fitness of each member of the new generation is determined. The members identified as 'fit' survive and enter a mating pool to reproduce the next generation. The objective function chosen for this process is

$$\text{objective function} = \sum_{i=0}^{N} [y(k) - \hat{y}(k)]^2 / \{Max[y(k)]\}^2 \qquad (3\text{-}4\text{-}1)$$

Where N is the number of sampled input and output

Max[$y(k)$] is the maximum of $y(k)$ $\{k=1,...,N\}$

$y(k)$ is the measured output of the actual system to be identified.

3.4.3 Operation of Genetic Algorithms

The operation of a simple genetic algorithm is performed according to the following steps:

(i) Initialise the population consisting of chromosomes at random,

(ii) Calculate the fitness of each chromosome in the population,

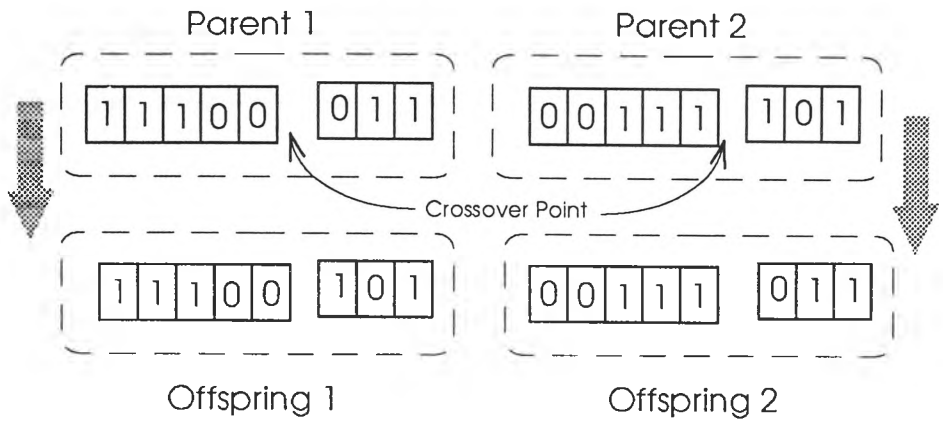
(iii) Reproduce new chromosomes by mating current chromosomes and applying crossover and mutation according to the crossover and mutation probability.

(iv) Delete members of the population to make room for the new chromosomes.

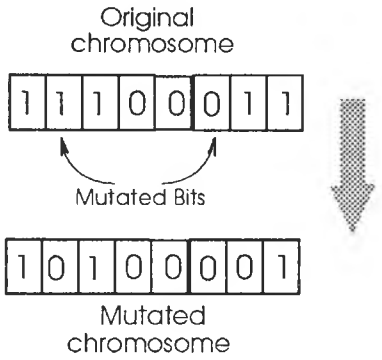
(v) Evaluate the new chromosomes and insert them into the population.

(vi) If the available time has expired, stop and find the best result. If not go to step 2.

The implementation of crossover and mutation is shown in Figure (3-4-1)a and Figure (3-4-1)b respectively. After selecting two parents, crossover is performed according to crossover probability. If crossover is performed then the portions of genes of offspring are constructed by copying portions of parent genes which depend on the crossover point determined by random selection. The other portions of genes of the two offspring are produced by exchanging the other portions of the two parents in the other side of the crossover point. Mutation is believed to help to reinject any information that may have been lost in previous generations[2]. More detailed discussion of genetic algorithms can be found in [2], [44].



(a) crossover



(b) Mutation

Figure (3-4-1) - Genetic Operations and Chromosome with binary encoding

The Genetic Algorithms can be divided into two groups depending on whether binary encoding or real number encoding is used. The first method employs a binary string to encode variables and to form a chromosome. The other employs real numbers that often vary from -1.0 to +1.0, of course one can use the range of the real number he wants. The search processes used is, however, mostly the same for both methods and modelled after natural evolution.

A) Binary encoding

The chromosomes of the genetic algorithms with binary encoding is shown in figure (3-4-1). In this case a chromosome can be encoded as many binary strings each of which represents a variable to be optimised.

B) Real number encoding

In real number encoding, every variable becomes a gene in the chromosome. As an example, assume that the variables [a,b,c,d, and e] should be optimised. The population [a,b,c,d, and e] is [a_1,b_1,c_1,d_1,e_1], [a_2,b_2,c_2,d_2,e_2] and [a_n,b_n,c_n,d_n,e_n] when n is the population size. The five variables express five genes in chromosome.

1. Operation of crossover: After crossover the offspring of [a_1,b_1,c_1,d_1,e_1] and [a_2,b_2,c_2,d_2,e_2] will be, [a_2,b_2,c_2,d_1,e_1] and [a_1,b_1,c_1,d_2,e_2] if the crossover point selected at random is located between c and d.
2) Operation of mutation: Mutation is the occasional alteration of some gene values in a chromosome. Every gene in each chromosome is a candidate for mutation, and its selection is determined by the mutation probability. The current value of the selected gene is then either added by a random number or substracted by a random number in my code.

The other operations and parameters are the same as the genetic algorithms with binary encoding as described above.

3.4.4 Two-Point Crossover

The crossover operator has been modified to improve the performance of the GAs. Multi-crossover points and two crossover points [44(1989), 2(1991)] are some examples to mention. In this work a new approach for two crossover points has been developed which has proved to improve the efficiency of the crossover in the exchange of information. It operates as follows (Figure (3-4-2)

(i) Select points j_1, j_2 randomly.

(ii) Employ a random selection number between 0.0 and 1.0. If the random number is greater than 0.5 swap the genes of the parents to the left of j_1 to produce the first stage offspring.

(iii) If the random number is less than or equal to 0.5 then swap the genes to the right of j_2 in the first stage offsprings to produce the next generation.

It will be demonstrated in Chapter 6 that the method proposed here converges more rapidly than the simple two crossover points suggested in [49] .

3.4.5 Adaptive Computation of Fitness

The fitness of the member of a new generation is determined by minimising the objective function. In system identification and design of fuzzy controllers, the problem of minimising the objective function must be transferred to maximising fitness which can be defined as

$$Fitness = PM - Objective\ Function$$

or

$$Fitness = PM - \sum_{i=0}^{N} [y(k) - \hat{y}(k)]^2 / \{Max[y(k)]\}^2$$

where PM is a positive number for scaling the fitness.

During the search process, if PM is a constant value, the fitness will gradually lose its distinction as the objective function is reduced in value. This will produce a fairly constant fitness for members of a new generation. As the result, identifying the individual members of the population with distinct fitnesses to enter the mating pool will be difficult and consequently the process will evolve slowly.

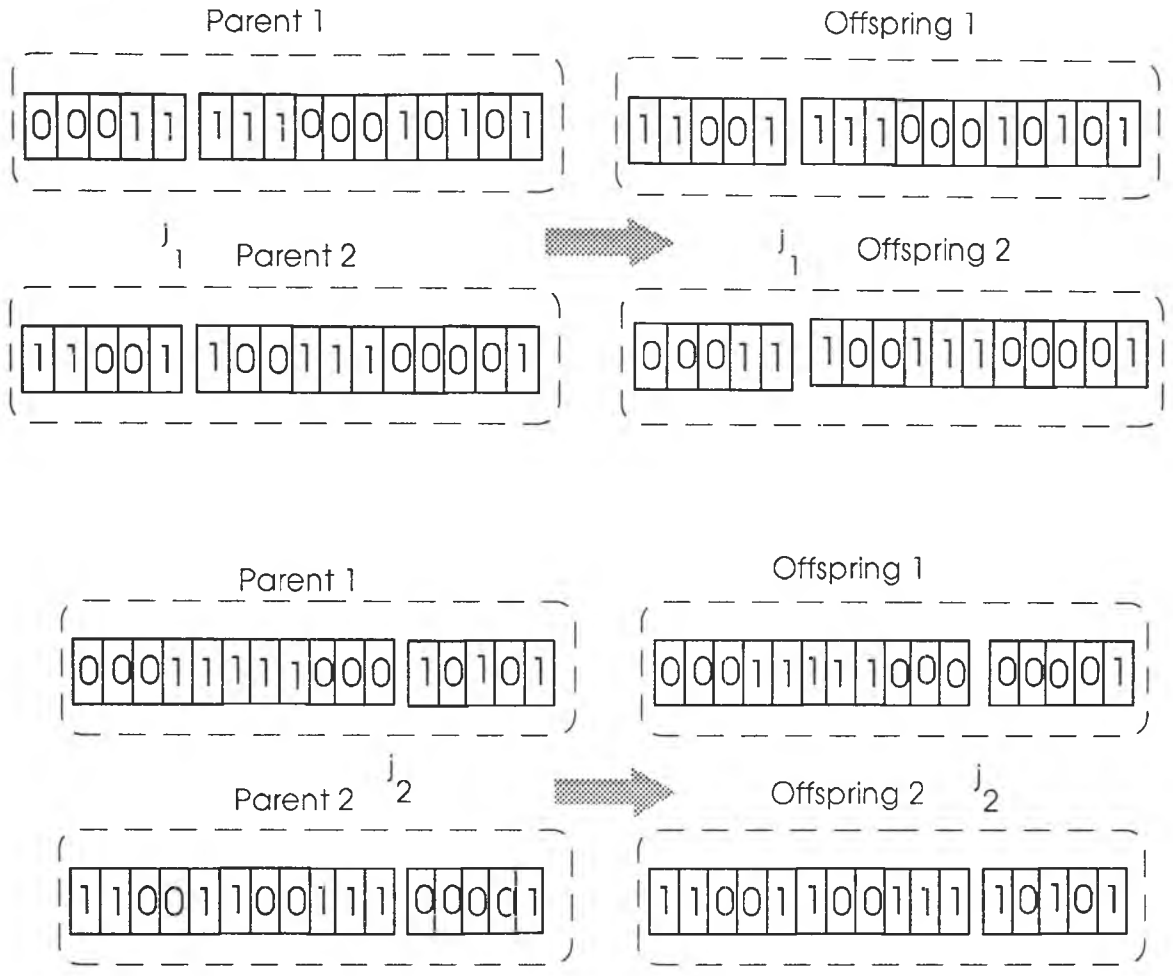


Figure (3-4-2) - The new procedure of two crossover points

On the other hand PM can not be set to a small value as the Objective Function varies during the search from a large value to a small positive number and hence Fitness may become negative at some stage.

In this work an adaptive algorithm based on GAs for choosing the best PM for each iteration of search is developed to overcome this problem. Using this algorithm, for every new generation of the population, an appropriate value for PM is found before the fitnesses are calculated. Assume that *k*th generation of population with size of *ps* is produced. The adaptive algorithm to calculate PM progresses as follows:

(i) Sort the population of generation K according to the ascending order of the objective functions of the population.

(ii) Choose the 75% top elements of the sorted population as candidates to reproduce the next generation. Set the fitnesses of the other 25 percent of the population to zero.

(iii) Set PM for generation K to the maximum of the objective function and calculate the fitnesses.

(iv) Use the method of remainder random rample without replacement to change the structure of population [44].

(v) Operate crossover and mutation to reproduce the next generation,

3.5 Summary

In this chapter the structure and procedures of a design tool developed in this work to tune a FLC was introduced. Genetic algorithms have been the main methodology used to identify a crude model of the system and tune the parameters of the controller. During this chapter GAs and the method of their employment have been explored. In addition two new strategies of search in GAs developed in this work were also described. These new strategies help the search process to converge faster towards an optimal solution compared to conventional GAs search routines.

Chapter 4

System Identification Using GAs

4.1 Introduction

The first stage of the design methodology developed in the this work for fuzzy controllers is to identify the model of a system. The application of genetic algorithms in the identification process is illustrated in this chapter. The method is applied to SISO, MIMO and non linear systems with time delay according to the procedures described in section 3.4. The models used in each case and the results of the identification have been extensively described.

4.2 Background

In spite of their suitability, genetic algorithms have not been widely applied to identification of MIMO system and nonlinear systems. The works reviewed in this chapter [50, 51] focus only on SISO linear systems with parameters of zeros and poles or SIMO [52] linear systems with state space parameters. They do not address the issues of determining the time delay and model structure of a system.

In the work carried out by Kristinsson [50] a PRBS is used to identify the poles and zeros of a system. The fitness function is chosen as

$$F(t) = \sum_{i=0}^{\omega} M - (\eta(t-i))^2 \qquad (4\text{-}2\text{-}1)$$

where M is a bias term needed to ensure a positive fitness, ω is the window size or the number of time steps over which the fitness is accumulated and $\eta(t) = y(t) - \hat{y}(t)$. The variable

$\hat{y}(t)$ represents the output of a deterministic system driven by the actual input $U(t)$. The authors have not addressed how the positive number M can be selected to ensure the fitnesses of the individuals are significantly different and hence better offspring are reproduced. The results obtained from computer simulation have shown that GAs are potentially useful tool for identification of dynamic systems.

In the work reported by [20], a recursive adaptive filter has been designed using Genetic algorithms. This paper also does not address how the structure and time delay of a dynamic system can be estimated using this method.

In the work reported by Andersen [52], the identification process of a SIMO (Single Input Multiple Output) system with state space parameters has been investigated. A random variable with a uniform distribution is employed as the input signal to drive the process. The cost function ($\hat{E}$) is defined as

$$\hat{E}=\frac{1}{n}\tilde{y}^T R\tilde{y} \qquad (4\text{-}2\text{-}2)$$

where

$$\tilde{y} = y - \hat{y} \qquad (4\text{-}2\text{-}3)$$

y is the observation vector of order n, n is the dimension of the error vector, $\tilde{y}$ is the estimate vector of dimension n, $R=\Omega^T\Omega$, and Ω is the diagonal standard deviation matrix related to the noise characteristics of the sensor used in the process. This work has not utilised the encoding technique available in the application of GA's to estimate the time delay.

4.3 SISO Linear System Identification Using GA's

The system considered in this study is assumed to have an ARMAX model the parameters of which are obtained using search process of the genetic algorithms. The developed algorithm

estimates all the parameters of the system including zeros, poles, and time-delay of the transfer function. The adaptive method presented in section 3.4.5 will be used to calculate the fitness of the population. In addition the new two crossover point strategy introduced in section 3.4.4 will be employed to speed up the search process.

In the majority of the identification methods used for SISO system including LMS, reduced order parameter estimation [53], [54], GRLS [55] and testing model structure [56], the system model is assumed to be ARMAX :

$$\begin{aligned} y(k) = & -a_1 y(k-1) - a_2 y(k-2) - \ldots - a_n y(k-n) + \\ & + b_1 u(k-d-1) + b_2 u(k-d-2) + \ldots + b_m u(k-d-m) \end{aligned} \qquad (4\text{-}3\text{-}1)$$

Where coefficients $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2, \ldots, b_m\}$ determine the poles and zeros of the transfer function of the dynamic system. These coefficients are estimated during the identification process.

Existence of a time delay in the system dynamics increases the number of parameters to be estimated for the ARMAX model. For example if the time delay is equal to eight sampling intervals, the parameters $b_1, b_2, \ldots, b_m$ will increase to $b_1, b_2, \ldots, b_m, \ldots, b_{m+8}$. This will not only increase the amount of computation required but reduce the accuracy of the identification process. Genetic algorithms can also be applied to estimate a system's time delay without much difficulty since the delay time can be encoded as a parameter to be identified. Thus we do not need a high order model to be identified which is used when employing other methods such as IV to identify a system with time delay.

Estimation of the system structure and time delay is generally difficult particularly for iterative algorithms such as LMS, IV, IVR and others.

4.3.1 Encoding Zeros and Poles

Initially a stable and minimum phase SISO system with the ARMAX model is assumed. The zeros and poles of this type of system are inside the unit circle. A single zero is encoded as a binary string with 10 bits whose most significant bit denotes the sign (±) and the other 9 bits will express the value of the zero ranging from 0.0 to 1.0 after being divided by 512.

If *m* is odd then the system will have at least a real zero. The numerator polynomial of the transfer function can be expressed by equation (4-3-2) with **complex conjugate zeros**:

$$B(q^{-1}) = q^{-d}(1-\beta_1 q^{-1})[1-(\beta_2 + j\beta_3)q^{-1}][1-(\beta_2 - j\beta_3)q^{-1}]... \qquad (4\text{-}3\text{-}2)$$

or equation (4-3-3) without **complex conjugate zeros**:

$$B(q^{-1}) = q^{-d}(1-\beta_1 q^{-1})[1-\beta_2 q^{-1}][1-\beta_3 q^{-1}]... \qquad (4\text{-}3\text{-}3)$$

On the other hand, if m is even then the numerator polynomial of the transfer function is expressed by equation (4-3-4) with **complex conjugate zeros**:

$$B(q^{-1}) = q^{-d}[1-(\beta_2 + j\beta_3)q^{-1}][1-(\beta_2 - j\beta_3)q^{-1}]... \qquad (4\text{-}3\text{-}4)$$

or (4-3-5) without **complex conjugate zeros**:

$$B(q^{-1}) = q^{-d}[1-\beta_1)q^{-1}][1-\beta_2 q^{-1}]... \qquad (4\text{-}3\text{-}5)$$

In order to code a pair of complex conjugate poles and zeros as a pair of strings, the real part is again encoded as a binary string with 10 bits. For example string 1100000000 is equivalent to -0.5. The imaginary part is encoded as a binary string with 10 bits whose most significant

bit denotes whether the string is real or imaginary by setting it to 0 or 1 respectively. As an example the string 1100000000 and 0100000000 represent -j0.5 and 0.5 respectively.

The method explained can be also applied to encoding the poles of a transfer function. If n is odd then the system should include at least a single pole. Therefore the denominator polynomial of the transfer function can be expressed by equation (4-3-6) with **complex conjugate poles**:

$$A(q^{-1}) = (1-\alpha_1 q^{-1})[1-(\alpha_2 + j\alpha_3)q^{-1}][1-(\alpha_2 - j\alpha_3)q^{-1}]... \quad (4\text{-}3\text{-}6)$$

or (4-3-7) without **complex conjugate poles**:

$$A(q^{-1}) = (1-\alpha_1 q^{-1})[1-\alpha_2 q^{-1}][1-\alpha_3 q^{-1}]... \quad (4\text{-}3\text{-}7)$$

If m is even then the denominator polynomial of the transfer function can be expressed by equation (4-3-8) with **complex conjugate poles**:

$$A(q^{-1}) = [1-(\alpha_1 + j\alpha_2)q^{-1}][1-(\alpha_2 - j\alpha_2)q^{-1}]... \quad (4\text{-}3\text{-}8)$$

or equation (4-3-9) without **complex conjugate poles**:

$$A(q^{-1}) = [1-\alpha_1)q^{-1}][1-\alpha_2 q^{-1}]... \quad (4\text{-}3\text{-}9)$$

4.3.2 Encoding Time Delay

The time delay of a system is encoded by a binary string of length 8. The string represents a sampling interval of 0 to 255. For example the binary string 00110011 indicates that the time delay of the system is equal to 51 sampling intervals.

4.3.3 Identification of Parameters and System Structure

The identification process progresses as follows:

(i) Select a small positive number ε.

(ii) Begin with the order of $m = 1$ and $n = 1$.

(iii) Encode d, a_i, b_j, to identify poles and zeros in the complex plane according to the procedures proposed by Kristinsson and Dumont [50]. For a stable minimum phase system the poles and zeros are inside the unit circle. Hence the real and imaginary parts of poles and zeros are less than or equal to 1. If the system to be identified is non-minimum phase, the maximum of real and imaginary parts of the poles and zeros can be selected according to *a prior* knowledge available on the system.

(iv) Use genetic algorithms to minimise the objective function or maximise the fitness.

(v) Check the fitness of every individual and the fitness sum. When the search process converges the fitness sum becomes very small. It implies that all fitnesses are approximately equal. In order to detect the convergence of this search process a small positive number ε (=0.001) is defined compared to which fitness sum measured. If this sum is less than the product of *Population size* $*\varepsilon$, the search process is terminated. Otherwise, compare the minimum objective function of the previous search stage with the current one. If the former is less than the latter, then output the results of the previous search stage and stop. Else continue

(vi) Increment m and n. Go to step (iv).

4.3.4 Simulation Results

In order to verify the identification method proposed in this chapter, two systems with the following model structures have been used in a computer simulation.

$$A(q^{-1})y(k) = q^{-d}B(q^{-1})u(k) + v(k) \quad (4\text{-}3\text{-}10)$$

Where

$$A(q^{-1}) = 1 + a_1 q^{-1} + a_2 q^{-2} + \ldots + a_n q^{-n}, \quad (4\text{-}3\text{-}11)$$

$$B(q^{-1}) = b_0 + b_1 q^{-1} + b_2 q^{-2} + \ldots + b_m q^{-m}, \quad (4\text{-}3\text{-}12)$$

$v(k)$ is white noise, $y(k)$ and $u(k)$ are the output and input of the system to be identified.

4.3.4.1 Simulation I

The first system has a model as:

$$y(k)-1.5y(k-1)+.54y(k-2)=3.2u(k-6)-0.96u(k-7) \qquad (4\text{-}3\text{-}13)$$

The conditions and results of the system identification for system 1 are given below. The parameters of the identified system and the real system are compared in Table (4-3-1). The responses of $y(k)$ and $\hat{y}(k)$ given the input $u(k)$ are illustrated in Figure (4-3-1). The time delay is identified after 60 generations.

Number of data =60;

Generations = 345;

Population = 50;

The length of chromosome =50 bits.

Probability of Crossover =0.9

Probability of Mutation = 0.1

Total Error:

$$\sum_{i=1}^{N}[y(k)-\hat{y}(k)]^2/\{Max[y(k)]\}^2=137.2/(196.8*196.8)=0.0036;$$

Max[$y(k)$]=196.8,

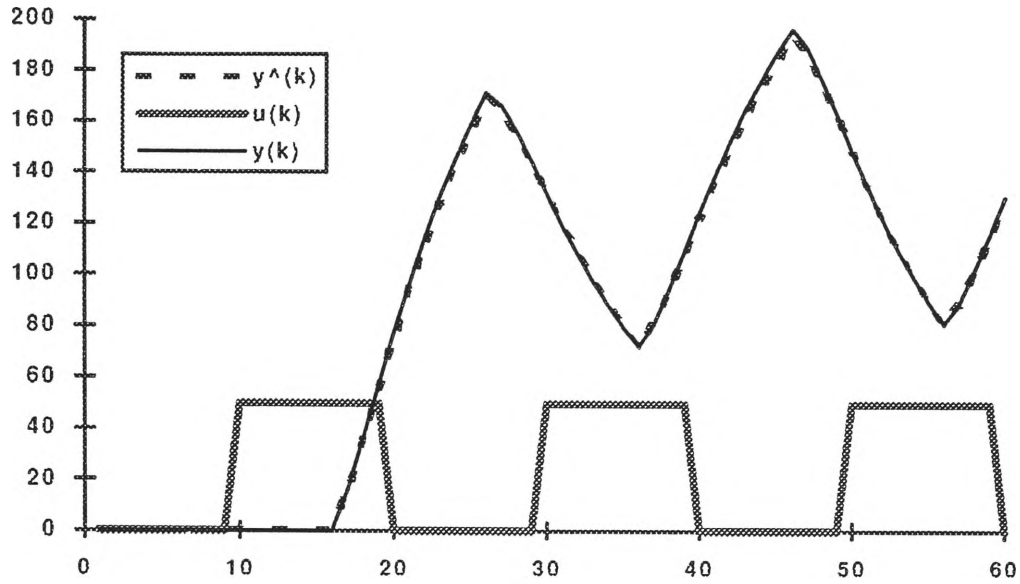


Figure (4-3-1) The response of y(k) and $\hat{y}(k)$ to input $\hat{u}(k)=u(k)/10$, $u(k)$ is the real input value.

Identified system:

$$\hat{y}(k) - 1.43\hat{y}(k-1) + 0.477\hat{y}(k-2) = 3.22u(k-6) - 0.702u(k-7) \qquad (4\text{-}3\text{-}14)$$

	a_1	a_2	b_1	b_2	Delay Time
Original parameters	-1.5	0.54	3.2	-0.96	6
Identified parameters	-1.43	0.477	3.22	-0.702	6

Table (4-3-1) Parameters and Results of Simulation 1

4.3.4.2 Simulation II

In this simulation a standard system often used in the literature [50] is employed. A time delay of 7 times the sampling interval is added to the model. The model, as shown below, has a pair of **complex conjugate poles**.

$$y(k) - 1.5y(k-1) + .7y(k-2) = 1.0u(k-7) + 0.5u(k-8) + C(q^{-1})v(k) \qquad (4\text{-}3\text{-}15)$$

The poles and zeros of the model are:

$$poles = (0.75 \pm j0.37)$$
$$zeros = (-0.5)$$

A pseudo random binary signal (PRBS) with a period of 127 and a bit interval of one sampling interval is used as the test signal to drive the system. The delay time is identified after 90 generations of GA. In the test procedure 250 input and output data have been sampled. The conditions and results of the system identification are given below. The parameters of the identified system are compared with the original system in Table (4-3-2). The responses of the original and identified systems are illustrated in Figure (4-3-2).

N = 250 Population = 60

Probability of Crossover = 0.68, Probability of Mutation = 0.06

Time delay = 7 Sampling Intervals,

Estimated poles and zeros:

$$poles = (0.74 \pm j0.347)$$
$$zeros = (-0.583)$$
$$\text{Total error} = \sum_{i=1}^{number} [y(k) - \hat{y}(k)]^2 / \{Max[y(k)]\}^2 = 0.0953$$

The identified system:

$$\hat{y}(k) - 1.48\hat{y}(k-1) + 0.668\hat{y}(k-2) = 1.06u(k-7) + 0.618u(k-8) \qquad (4\text{-}3\text{-}16)$$

	a_1	a_2	b_1	b_2	Delay Time
Original parameters	-1.5	0.7	1.0	-0.5	7
Identified parameters	-1.48	0.668	1.19	-0.618	7

Table (4-3-2) Parameters and Result of Simulation II

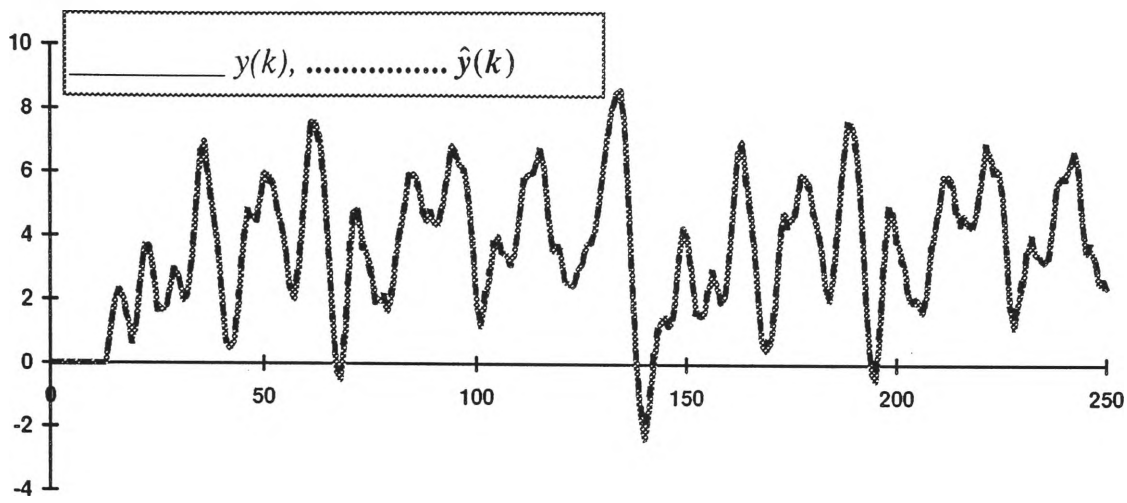


Figure (4-3-2) Response of y(k) and $\hat{y}(k)$ to PRBS.

4.3.4.3 Simulation III

In order to compare the simple two point crossover procedure with the new two point crossover procedure, system 3 is selected as a model for simulation with the time delay equal to 9 sampling intervals:

$$y(k)-1.8y(k-1)+.8075y(k-2)=1.4u(k-9)-0.7u(k-10) \qquad (4\text{-}3\text{-}17)$$

The results with the new two point crossover procedure are given below. The parameters are compared in Table (4-3-3) and simulation results are shown in Figure (4-3-3).

$$\hat{y}(k)-1.75\hat{y}(k-1)+.759\hat{y}(k-2)=1.38u(k-9)-0.61u(k-10) \qquad (4\text{-}3\text{-}18)$$

	a_1	a_2	b_1	b_2	Delay Time
Real parameters	-1.8	0.8075	1.4	-0.7	9
identified parameters	-1.75	0.759	1.38	-0.61	9

Table (4-3-3) Comparison of Parameters using new two-point crossover

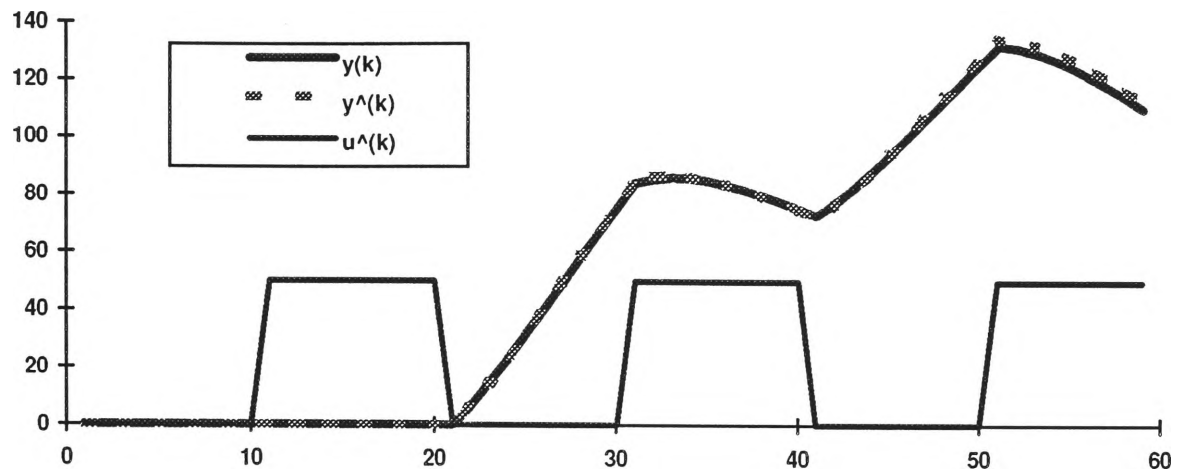


Figure (4-3-3) Results with new two point crossover u^(k)=u(k)/10

Number of data =60 Generations = 525

Probability of Crossover =0.8 Mutation = 0.08

Total Error:

$$\sum_{i=1}^{number} [y(k)-\hat{y}(k)]^2/\{Max[y(k)]\}^2 = 96.98/(132.39*132.39) = 0.005514;$$

Max[$y(k)$]=132.39

The responses of $y(k)$ and $\hat{y}(k)$ under the input $u(k)$ are shown in Fig (4-3-3).

The results with simple two point crossover procedure are given in Table (4-3-4), and Figure (4-3-4). The real model is

$$y(k)-1.8y(k-1)+.8075y(k-2)=1.4u(k-9)-0.7u(k-10) \qquad (4.3.19)$$

The identified model is

$$\hat{y}(k)-1.721\hat{y}(k-1)+0.731\hat{y}(k-2)=1.19u(k-9)-0.618u(k-10) \qquad (4.3.20)$$

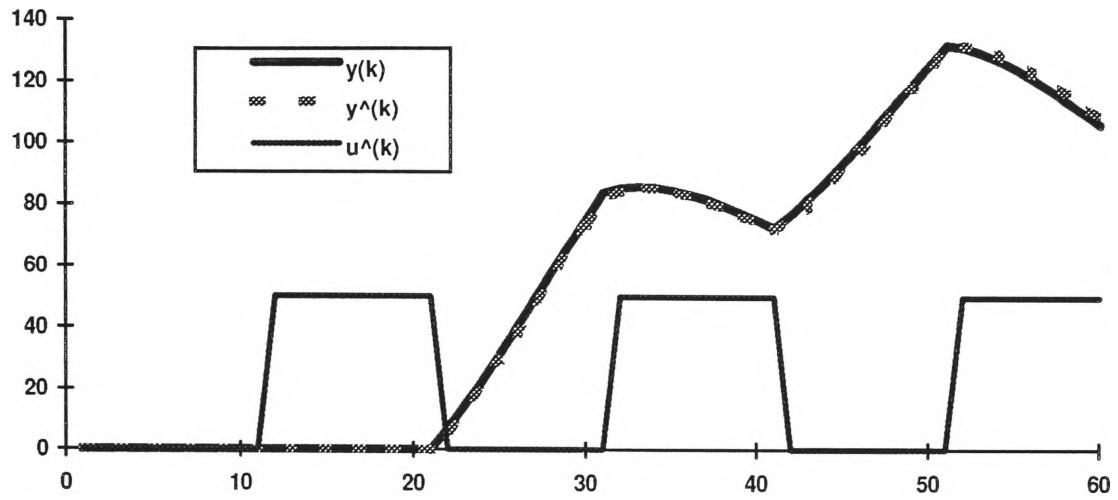


Figure (4-3-4) Results with simple two point crossover u^(k)=u(k)/10

The parameters of the original system are compared with the estimated system in Table(4-3-4).

	a_1	a_2	b_1	b_2	Time Delay
Real parameters	-1.8	0.8075	1.4	-0.79	9
identified parameters	-1.721	0.731	1.19	-0.618	9

Table (4-3-4) Comparison of Parameters using simple two point crossover

The rest of the results are :

Time Delay = 9 sample intervals

Number of data =60;

Generations = 3729; Probability of Crossover =0.85

$\sum_{i=1}^{number} [y(k) - \hat{y}(k)]^2 / Max\{[y(k)]\}^2$ 103.33.2/(132.9*132.39)=0.00585;

Probability of Mutation = 0.08

With the new two point crossover, the search process is converged after 525 generations compared to simple two point crossover which takes 3729 generations to converge. Therefore, it can be obviously seen from the results that the new two point crossover procedure has produced a result much faster than the simple method.

4.3.5 Validation of the Method

In order to validate the method, the instrumental variable identification method (IV) has been employed to compare its result with GAs. In this experiment the model of the system to be identified and input/output data are assumed to be the same as Simulation III:

The parameters estimated by IV are compared with GA's method in Table (4-3-5)

	a_1	a_2	b_1	b_2	b_3
Identified parameters	-1.233	0.250	0.0436	0.1554	-0.073
Original parameters	-1.80	0.8075	0.0	0.0	0.0
	b_4	b_5	b_6	b_7	b_8
Identified parameters	-0.058	-0.046	-0.0363	-0.0276	-0.020
Original parameters	0.0	0.0	0.0	0.0	0.0
	b_9	b_{10}	b_{11}		
Identified parameters	-0.014	0.0	1.443		
Original parameters	0.0	0.70	1.40		

Table (4-3-5) Comparison of system parameters of the original

Comparison of Figures (4-3-4) and (4-3-5) shows that the GAs has produced a performance closer to the actual system for the same input due to using a lower order model in the identification process.

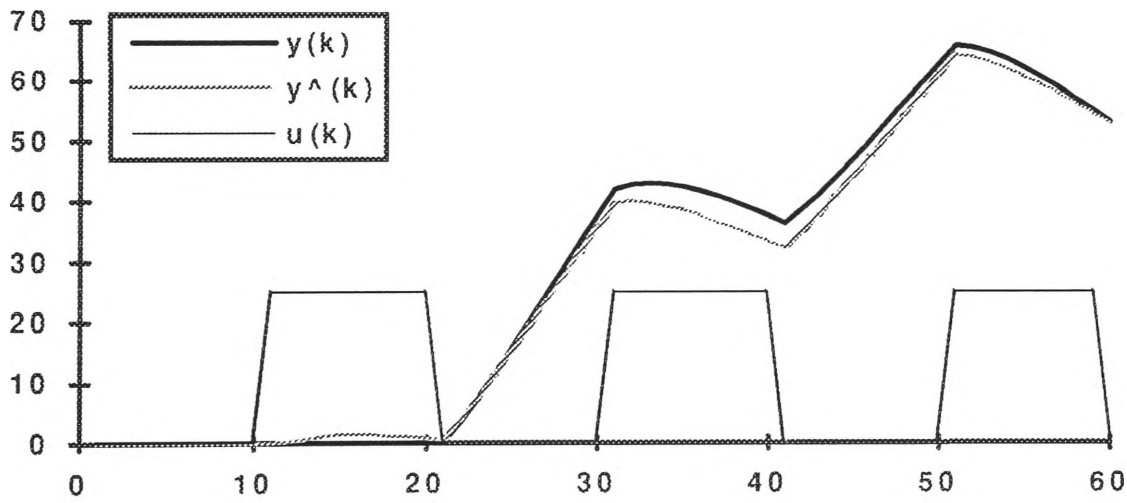


Figure (4-3-5) $\hat{y}(k)$ **and y(k) for** $\hat{u}(k)$ **using IV4 (Instrumental Variable)**

The IV method cannot estimate well the parameters of the system when time-delay is unknown and a small set of input and output data is available.

4.4. MIMO System Identification Using GAs

The multi-variable system employed in this study is an ARMAX model

$$Y(K) = A(q^{-1})U(k) + v(k) \qquad (4\text{-}4\text{-}1)$$

Where $A(q^{-1})$ is a transfer function matrix in backward shift operator q^{-1}, i.e. $y(k-1) = q^{-1} y(k)$

$Y(k) = [y_1(k), y_2(k), \ldots y_n(k)]$ is the output vector,

$U(k) = [U_1(k), U_2(k), \ldots U_m(k)]$ is the input vector,

and v(k) is white noise vector. $Y(k)$ can be written as formula (4-4-2).

$$\begin{aligned} y_1(k) &= A_{11}(q^{-1})u_1(k) + A_{12}(q^{-1})u_2(k) + \ldots + A_{nm}(q^{-1})u_m(k) \\ y_2(k) &= A_{21}(q^{-1})u_1(k) + A_{22}(q^{-1})u_2(k) + \ldots + A_{2m}(q^{-1})u_m(k) \\ &\cdot \\ &\cdot \\ y_n(k) &= A_{n1}(q^{-1})u_1(k) + A_{n2}(q^{-1})u_2(k) + \ldots + A_{nm}(q^{-1})u_m(k) \end{aligned} \qquad (4\text{-}4\text{-}2)$$

In order to simplify the encoding procedure of the parameters in the identification process, $y_1, y_2 \ldots y_n$ are decomposed as formula (4-4-3)

$$\begin{aligned} y_1 &= y_{11} + y_{12} + \ldots + y_{1m} \\ y_2 &= y_{21} + y_{22} + \ldots + y_{2m} \qquad (4\text{-}4\text{-}3) \\ &\vdots \\ y_n &= y_{n1} + y_{n2} + \ldots + y_{nm} \qquad (4\text{-}4\text{-}4) \end{aligned}$$

where

$$y_{11} = A_{11}(q^{-1})u_1(k), y_{12} = A_{12}(q^{-1})u_2(k),\ldots, y_{1m} = A_{1m}(q^{-1})u_m(k)$$
$$y_{21} = A_{21}(q^{-1})u_1(k), y_{22} = A_{22}(q^{-1})u_2(k),\ldots, y_{2m} = A_{2m}(q^{-1})u_m(k) \quad (4\text{-}4\text{-}5)$$
$$\vdots$$
$$\vdots$$
$$y_{n1} = A_{n1}(q^{-1})u_1(k), y_{n2} = A_{n2}(q^{-1})u_2(k),\ldots, y_{2m} = A_{nm}(q^{-1})u_m(k)$$

The following parameters are also defined:

D_{11} the time delay of the transfer function $A_{11}(q^{-1})$,

D_{12} the time delay of the transfer function $A_{12}(q^{-1})$,

$\vdots$

$\vdots$

D_{1m} the time delay of the transfer function $A_{1m}(q^{-1})$,

D_{21} the time delay of the transfer function $A_{21}(q^{-1})$,

D_{22} the time delay of the transfer function $A_{22}(q^{-1})$,

$\vdots$

$\vdots$

D_{nm} the time delay of the transfer function $A_{nm}(q^{-1})$,

This approach will also simplify the calculation of the objective function.

If the input *u(k)* is a PRBS, the parameters of the transfer matrix $A(q^{-1})$ can be calculated theoretically. Nevertheless in most of the real systems, a PRBS cannot be applied to the systems.

The objective function is the same as SISO system but defined for every output:

$$Obj_1 = \sum_{i=1}^{N} [y_1(k) - \hat{y}_1(k)]^2 / [Max\{y_1(k)\}]^2$$

$$Obj_2 = \sum_{i=1}^{N} [y_2(k) - \hat{y}_2(k)]^2 / [Max\{y_2(k)\}]^2 \qquad (4\text{-}4\text{-}6)$$

:

:

$$Obj_n = \sum_{i=1}^{N} [y_n(k) - \hat{y}_n(k)]^2 / [Max\{y_n(k)\}]^2$$

Similar to a SISO system $y_j(k)_{\{j=1,2,...,n\}}$ are the outputs of the real system, and $Max\{y_j(k)\}_{(j=1,2,...,n)}$ are constant values equal to the maximum of $y_j(k)\{k = 1,2,...,N\}$ respectively. The coefficients of $A_{i1}(q^{-1}), A_{i2}(q^{-1}),..., A_{im}(q^{-1})$ are estimated by genetic algorithms using $Obj_{j\{j=1,2,...,n\}}$ as objective functions.

The encoding method for MIMO system is the same as SISO system described in section 4.3.2. However there are more parameters to be identified in an MIMO system

4.4.1 Experimental Work

A series of experiments were conducted through computer simulation to study the operation and accuracy of GAs in the identification of a MIMO system. For this purpose a system with the following model was used:

$$Y(k) = A(q^{-1})U(k) \quad \{n = 2, m = 2\}$$

where

$$A_{11}(q^{-1}) = \frac{1.2q^{-8}(1-0.73q^{-1})}{1-1.6q^{-1}+0.63q^{-2}}, \qquad A_{12}(q^{-1}) = \frac{0.38q^{-1}}{1-0.95q^{-1}}$$

$$A_{21}(q^{-1}) = \frac{0.25q^{-1}}{1-0.88q^{-1}}, \qquad A_{22}(q^{-1}) = \frac{0.8q^{-6}(1-0.78q^{-1})}{1-1.75q^{-1}+0.762q^{-2}}$$

In implementing the GAs the time delay of $A_{i,j}(q^{-1})$, i.e., $D_{11}, D_{21},..., D_{m1}$, $D_{12}, D_{22},..., D_{m2},... D_{1n}, D_{2n},..., D_{mn}$ were encoded as 8-bit strings on the right side of the chromosomes, the gain $K_{i,j}$ of $A_{i,j}(q^{-1})$ as 10-bit strings in the medium of the chromosomes

and the zeros and poles were encoded as 10-bit strings on the left of the chromosomes. The identification process has been based on 80 samples of input and output of the system.

4.4.1.1 Estimation of $A_{11}(q^{-1})$ and $A_{12}(q^{-1})$

In the estimation of $A_{11}(q^{-1})$ and $A_{12}(q^{-1})$, a probability of crossover of 0.6 is selected. This is to ensure a more stable search process when a time delay exists. Furthermore since the number of sampled data is small, the probability of mutation of 0.05 is chosen. This will help to reintroduce the information that has been lost in previous generations of the search process. The objective function calculated was:

$$Obj_1 = \sum_{i=1}^{number} [\hat{y}_1(k) - y_1(k)]^2 / [Max\{y_1(k)\}]^2 = 303.174/12689.8 = 0.0239,$$

The evolution of the algorithm in estimating the poles and zeros of $A_{11}(q^{-1})$ and $A_{12}(q^{-1})$ is illustrated in Figure (4-4-1). The dominant pole of $A_{11}(q^{-1})$ with a large time constant (pole1 of $A_{11}(q^{-1})$) has been estimated the fastest and reached a stable value of 0.92 after 700 generations. Since a small change in this pole will cause a big variation in the value of the objective functions, the accuracy of the estimation of the dominant pole of $A_{11}(q^{-1})$ has a great deal of influence on the objective function. The identified poles and zeros on the unit circle for every 400 generations are shown in Figure (4-4-2).

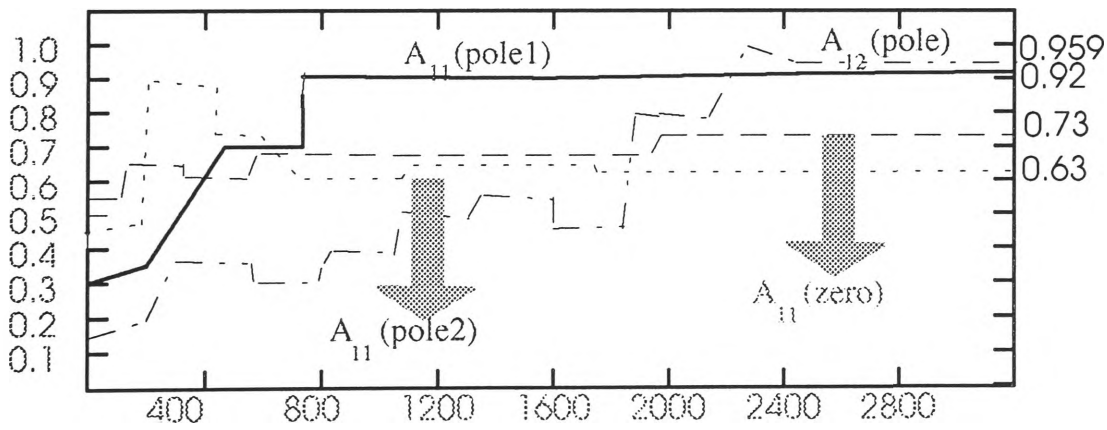


Figure (4-4-1) Generations of zeros and poles for A_{11} and A_{12}

The estimation process for the gains of $A_{11}(q^{-1})$ and $A_{12}(q^{-1})$ is illustrated in Figure (4-4-3). In this case the gain of the main channel from input $u_1(k)$ to $y_1(k)$ converges much faster than the channel from input $u_2(k)$ to $y_1(k)$. A similar pattern can be also observed in the estimation of the time delay as shown in Figure (4-4-4). It has taken about 1000 generations to converge to a time delay of 8 sampling intervals.

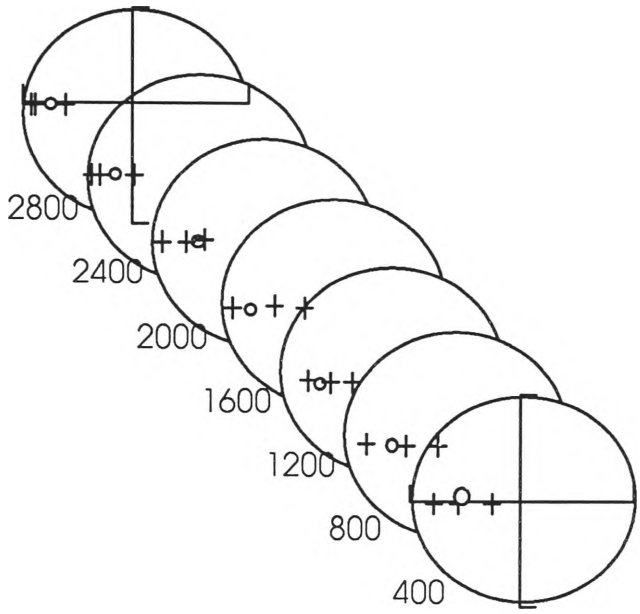


Figure (4-4-2) Identified Poles and zeros on the unit circle for $A_{11}(q^{-1})$ and $A_{12}(q^{-1})$. (+ and O stand for poles and zeros respectively)

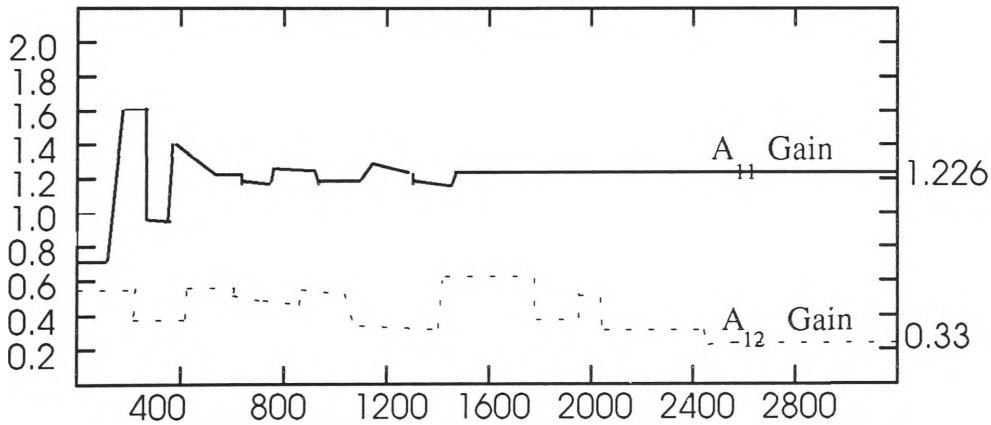


Figure (4-4-3) Generations of gains for $A_{11}(q^{-1})$ and $A_{12}(q^{-1})$

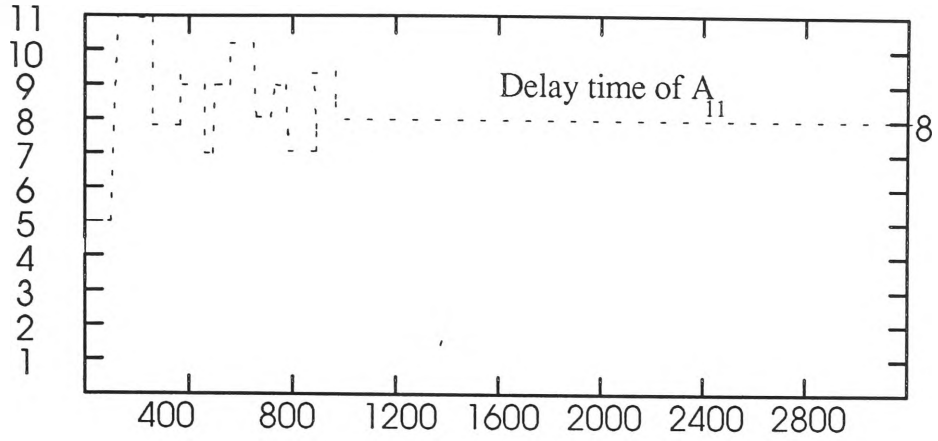


Figure (4-4-4) Generations of delay time for A_{11}

4.4.1.2 Estimation of $A_{22}(q^{-1})$ and $A_{21}(q^{-1})$

In the estimation of $A_{22}(q^{-1})$ and $A_{21}(q^{-1})$ a smaller value of 0.6 for probability of crossover is selected to inhibit some chromosomes from crossover operation and hence maintain a more stable system. Furthermore since the number of sampled data is not large the probability of mutation is chosen to be 0.038 to restore some of the information lost in the previous generations. The objective function is

$$Obj_2 = \sum_{i=1}^{number} [\hat{y}_2(k) - y_2(k)]^2 / [Max\{y_2(k)\}]^2 = 287.69/23545.3 = 0.0122$$

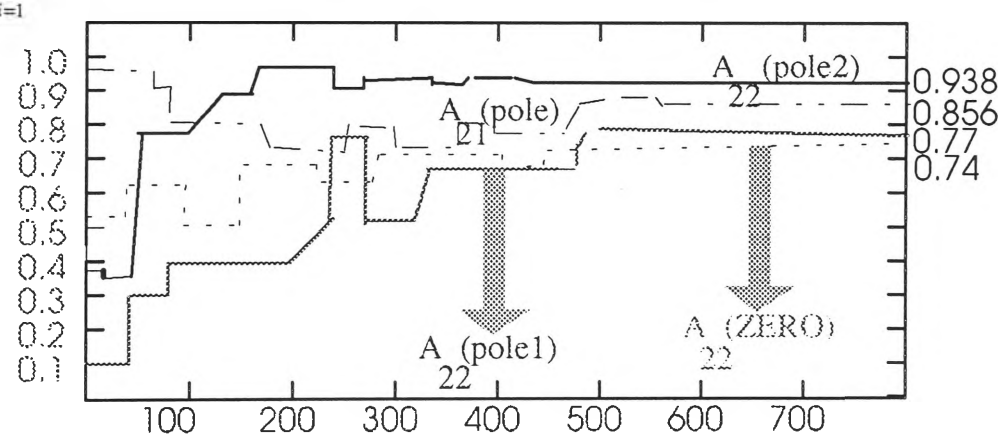


Figure (4-4-5) Convergence of poles and zeros for $A_{22}(q^{-1})$ and $A_{21}(q^{-1})$

Similar procedures have been applied to $A_{22}(q^{-1})$ and $A_{21}(q^{-1})$. The estimation of poles and zeros of $A_{22}(q^{-1})$ and $A_{21}(q^{-1})$ is illustrated in Figure (4-5-5). It can be seen that main pole of

$A_{22}(q^{-1})$ with a large time constant (pole2 of $A_{22}(q^{-1})$) converges after about 220 generations and reaches a stable value of 0.938. The identified poles and zeros on the unit circle for every 100 generations are shown in Figure (4-4-6).

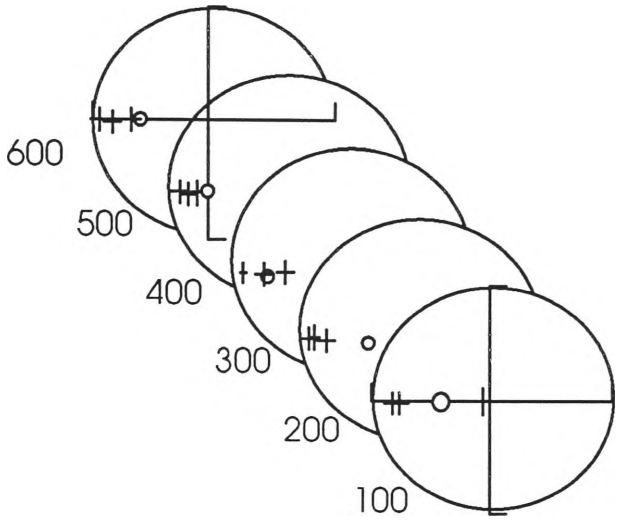


Figure (4-4-6) Identified Poles and zeros on the unit circle for $A_{22}(q^{-1})$ and $A_{21}(q^{-1})$. (+ and O for poles and zeros respectively)

The estimation of the gains of $A_{22}(q^{-1})$ and $A_{21}(q^{-1})$ and their convergence after 390 generations are shown in Figure (4-4-7). The gain of main channel from input $u_2(k)$ to $y_2(k)$ converges much faster than the other channel from input $u_1(k)$ to $y_2(k)$.The time delay has converged to a 6 sampling time after 250 generations Figure (4-4-8).

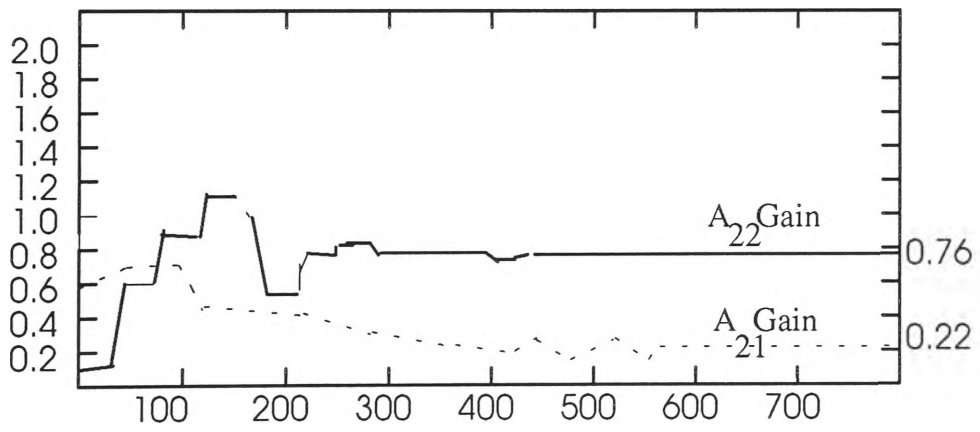


Figure (4-4-7) Convergence of the gains of $A_{22}(q^{-1})$ and $A_{21}(q^{-1})$

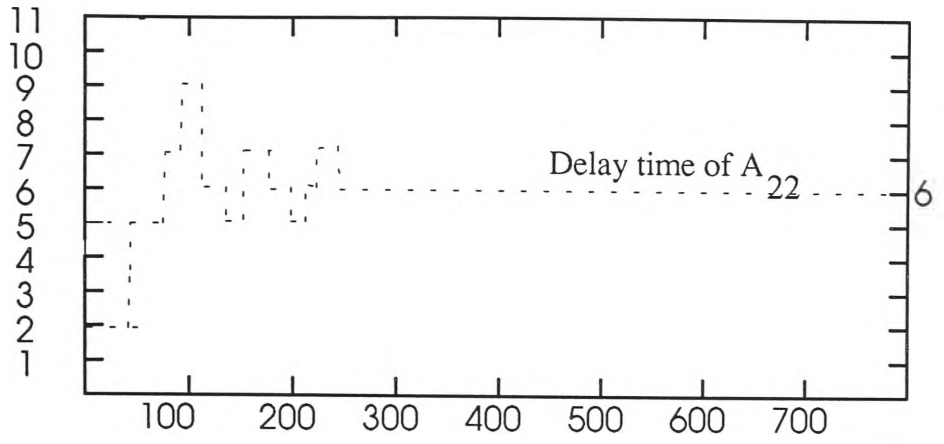


Figure (4-4-8) Convergence of delay time for $A_{22}(q^{-1})$

4.4.1.3 Parameters of Identified Model

The coefficients of the identified model based on GAs for a model structure of

$$\hat{Y}(k) = \hat{A}(q^{-1})U(k)\{\text{n} = 2, \text{m} = 2\}$$

is as follows:

$$\hat{A}_{11} = \frac{1.226q^{-8}(1-0.73q^{-1})}{1-1.55q^{-1}+0.579q^{-2}} \qquad \hat{A}_{12}(q^{-1}) = \frac{0.33q^{-1}}{1-0.959q^{-1}}$$

$$\hat{A}_{21}(q^{-1}) = \frac{0.22q^{-1}}{1-0.856q^{-1}} \qquad \hat{A}_{22}(q^{-1}) = \frac{0.76q^{-6}(1-0.744q^{-1})}{1-1.708q^{-1}+0.7226q^{-2}}$$

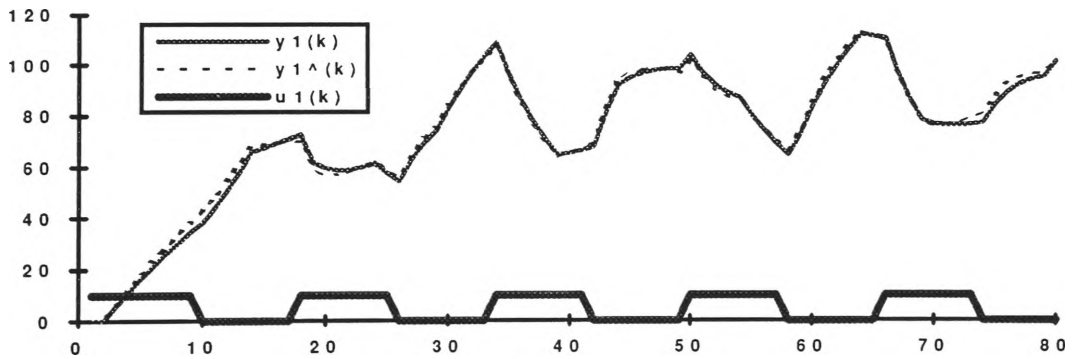


Figure (4-4-9) $y_1(k)$ and $\hat{y}_1(k)$ in repsonse to $u_1(k)$ and $u_2(k)$

(the chart of $u_2(k)$ is illustrated in the Figure (4-4-10)

The coefficients of this model in comparison with the actual model of the system is shown in Table (4-4-1). The poles, zeros and gains are also compared with the original system in Table (4-4-2). The response of the identified model to inputs u_1 and u_2 have been also compared with the original system in Figures (4-4-9) and (4-4-10).

		a_1	a_2	b_1	b_2	d
$A_{11}(q^{-1})$	Real	-1.6	0.63	1.2	-0.876	8
	Identified	-1.55	0.579	1.226	0.895	8
$A_{12}(q^{-1})$	Real	-0.95		0.38		
	Identified	-0.959		0.33		
$A_{22}(q^{-1})$	Identified	-1.75	0.762	0.8	-0.624	6
	Real	-1.708	0.722	0.76	-0.56	6
$A_{22}(q^{-1})$	Identified	-0.88		0.2		
	Real	-0.856		0.22		

Table (4-4-1) Comparison of the coefficients

		Pole 1	Gain	Pole 2	Zero 1	Gain
$A_{11}(q^{-1})$	Real	-0.7		-0.9	-0.73	1.2
	Identified	-.63		-0.92	-0.73	1.226
$A_{12}(q^{-1})$	Real	-0.95	0.38			
	Identified	-0.959	0.33			
$A_{22}(q^{-1})$	Identified	-0.81		-0.95	-0.78	0.8
	Real	-0.77		-0.938	-0.744	0.76
$A_{22}(q^{-1})$	Identified	-0.88	0.2			
	Real	-0.856	0.22			

Table (4-4-2) Comparison of the poles, zeros and gains

Furthermore the performance of GAs in the identification of the model has been compared with instrumental variable (IV) Method using MATLAB package. The input/output data from the same system used in the GAs was used for system identification. In the estimation of the coefficients using IV method the model of the system was assumed to be ARMAX with a known structure. The numerators of $A_{11}(q^{-1})$ and $A_{22}(q^{-1})$ were also assumed to be polynomials with 10 and 8 terms respectively. The method cannot estimate the time delay.

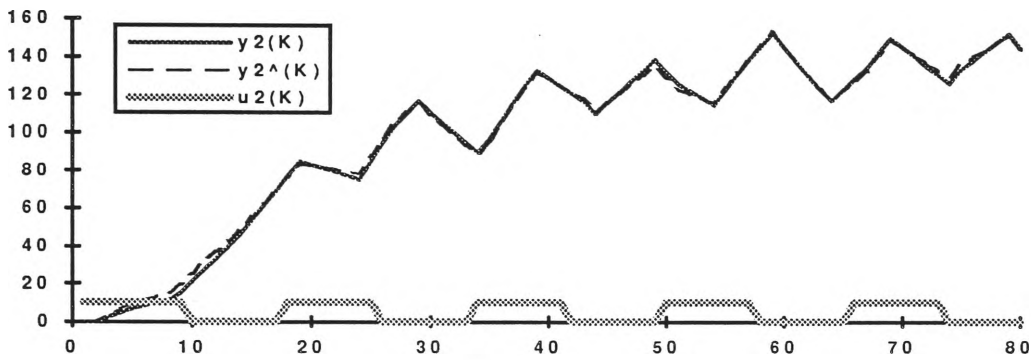


Figure (4-4-10) $y_2(k)$ **and** $\hat{y}_2(k)$ **in repsonse to** $u_1(k)$ **and** $u_2(k)$ **(the chart of** $u_1(k)$ **is illustrated in the Figure (4-9-9))**

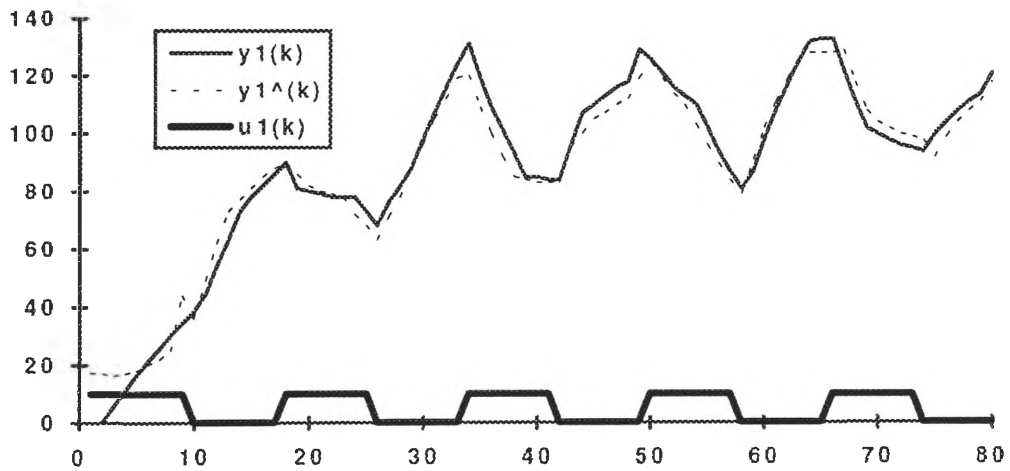


Figure (4-4-11) Response of $y_1(k)$ **and** $\hat{y}_1(k)$ **using IV method**

The coefficients of the identified system are compared with the original system in table (4-4-8). The responses of this system in comparison with the original system to inputs u_1 and u_2 are also illustrated in Figures (4-4-11) and (4-4-12). It is clear that the system identified by the

IV method is not as accurate as the one obtained using GAs. One important reason for this is the shortcoming of the IV method to estimate the time delay of the system. This particularly accounts for the large errors the original and estimated response in the early phases.

		a_1	a_2	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
$A_{11}(q^{-1})$	Real	-1.6	0.63	0	0	0	0	0	0	0	0	1.2	-
	Identified	-1.04	0.093	1.72	-2.39	0.144	0.149	0.08	0.0	0.0	0.43	-	-
$A_{12}(q^{-1})$	Real	-0.95	0	0.38									
	Identified	-	0.0	0.34									
$A_{22}(q^{-1})$	Identified	-1.75	0.762	0	0	0	0	0	0	0.8	-0.62		
	Real	-1.39	0.404	0.03	0.12	-	0.256	0.03	0.04	0.131	0.19		
$A_{22}(q^{-1})$	Identified	-0.88	0.0	0.2									
	Real	-	-	0.35									

Table(4-4-3) Comparison of the coefficient of the system identified by IV method with the original system

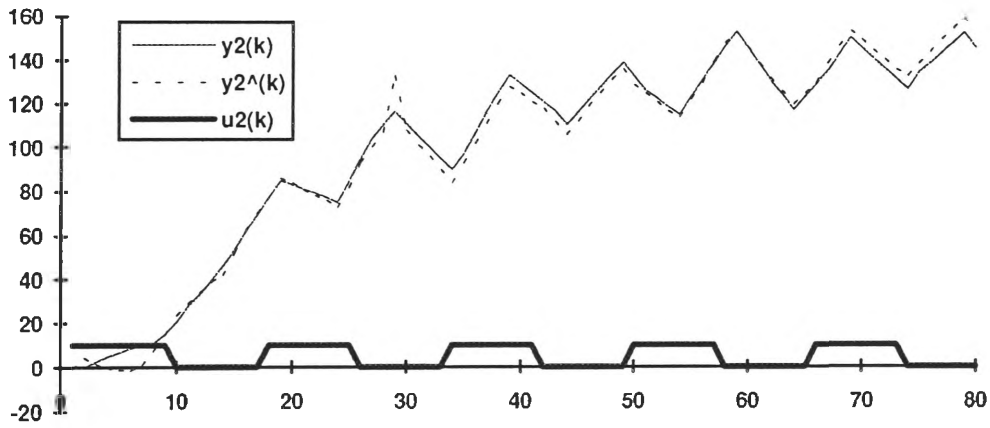


Figure (4-4-12) Responses of $y_2(k)$ and $\hat{y}_2(k)$ using IV method

Nevertheless the response using IV method can be improved if PRBS is used as the input and the time delay given as a prior knowledge to the identification algorithm.

4.5 Identification of SISO Hammerstein-type Nonlinear System

A SISO Hammerstein type non-linear system can be modelled as a cascade nonlinear memory-less subsystem and a linear dynamic subsystem as shown in Figure (4-5-1).

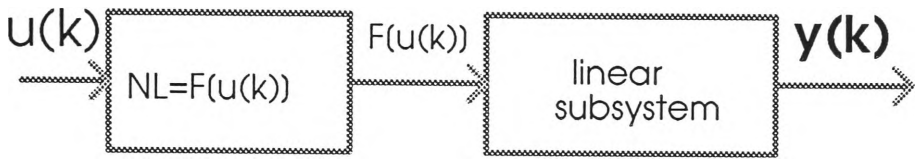


Figure (4-5-1) Model of plant characteristics, non linear and linear

A discrete time model of this system can be described by:

$$A(q^{-1})y(k) = q^{-d}B(q^{-1})F[u(k)] + v(k) \qquad (4\text{-}5\text{-}1)$$

Where

$A(q^{-1}) = 1 + a_1 q^{-1} + a_2 q^{-2} + \ldots + a_n q^{-n}$,

$B(q^{-1}) = b_0 + b_1 q^{-1} + b_2 q^{-2} + \ldots + b_m q^{-m}$.,

$v(k)$ is white noise.

It is also assumed that $F[u(k)]$ can be approximately represented by a polynomial of u(k)

$$F[u(k)] = f_0 + f_1 u(k) + f_2 u^2(k) + \ldots + f_j u^j(k) \qquad (4\text{-}5\text{-}2)$$

The main aim is not only to identify parameters of the linear subsystem but also to estimate the parameters of the non linear subsystem. The parameters to be identified are encoded as :

$$\hat{\Psi} = [a_1, a_2, \ldots, a_n,\ b_1, b_2, \ldots, b_m,\ f_0, f_1, \ldots, f_j, d] \qquad (4\text{-}5\text{-}3)$$

$a_1, a_2, \ldots, a_n$, $b_1, b_2, \ldots, b_m$ can be represented by the poles and zeros. If the linear subsystem is minimum phase, its zeros and poles are located inside a unit circle on the complex plane. Therefore these zeros and poles can be directly identified and encoded between [0, ±1] using genetic algorithms. For parameters $f_0, f_1, \ldots, f_j$ of the nonlinear subsystem, the range of coding is dependent on the amplitude of control signal *u(k)*. In this case a range between 0.0 to ±2*Max{$|u(K)|$} is chosen. The time delay *d* can be encoded within 50 sampling intervals. If this is not sufficient the range can be extended to 1023 sampling intervals.

4.5.1 Simulation Results I

In the simulation a "standard" model of a real system which has been employed often in the literatures [4, 44] to test different identification method is chosen:

$$A(q^{-1}) = 1.0 - 1.5q^{-1} + 0.7q^{-2}$$
$$B(q^{-1}) = 1.0 + 0.5q^{-1} \qquad u(k) \geq -0.5 \qquad \text{(4-5-4)}$$
$$F[u(k)] = [u(k) + 0.5]^{1/2} - 0.5^{1/2}$$

Input test signal is a PRBS.

Since *F[u(k)]* is a nonlinear subsystem the amplitude of the test PRBS must be changed in the search process. In this simulation PRBS is divided into eight amplitude ranges from 0 to 2.0. Each amplitude input is a PRBS with the period of 32 bit length and duration of 50 sampling intervals. There are 400 input and output data to be used in the search.

Responses of the "standard" system and the identified system are shown in Figure (4-5-2). The curves of the "standard" non linear subsystem *F[u(k)]* and identified non linear subsystem *F*[u(k)]* are shown in Figure (4-5-3). A comparison between identified parameters and "standard" system can be seen in table (4-5-1).

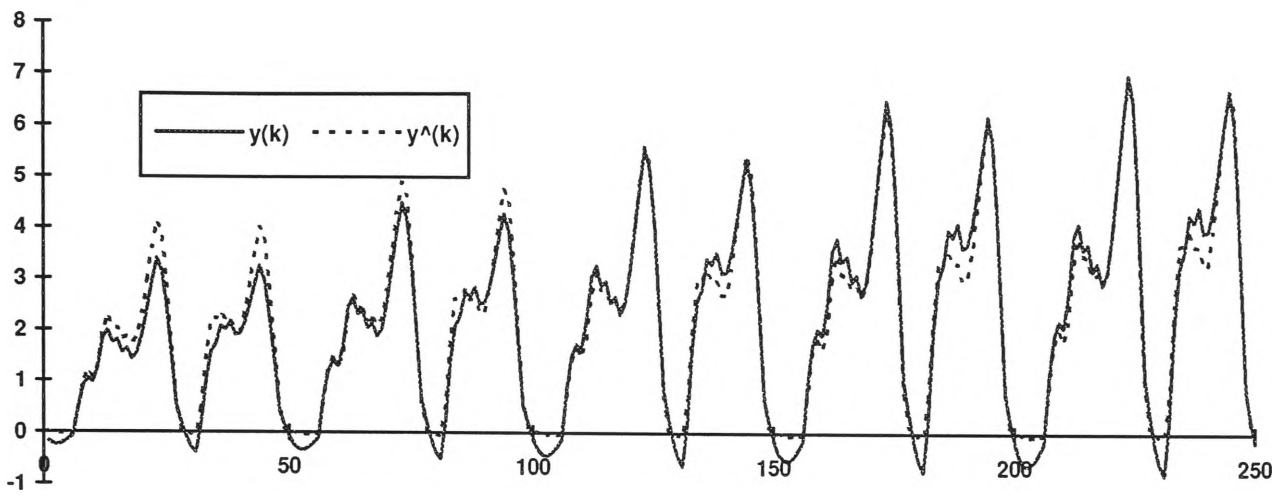


Figure (4-5-2) Responses of the real system y(k)and the identified system $\hat{y}(k)$

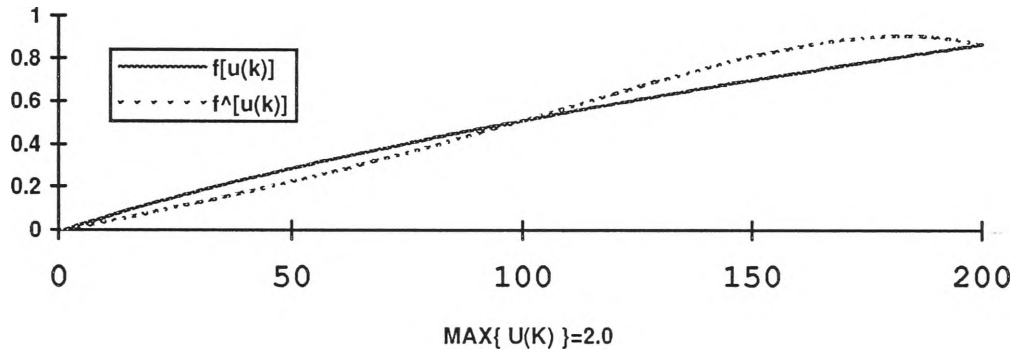


Figure (4-5-3) The identified $\hat{F}[u(k)]$ and the real *F[u(k)]*

	a_1	a_2	b_1	b_2
Real parameters	-1.5	0.7	1.0	0.5
Identified parameters	-1.472	0.674	0.924	0.294
	f_1	f_2	f_3	f_4
Identified parameters	0.23047	0.12333	0.15	0.04667
	f_5	f_6	f_7	f_8
Identified parameters	0.0581	-0.07191	0.00381	0.0.0009504

Table (4-5-1) Identified parameters

Generation = 361; Probability of mutation = 0.015; Probability of crossover = 0.9;

$$\sum_{i=0}^{i=number}[\hat{y}(k)-y(k)]^2 = 115.6$$

It is obvious from the results that the nonlinear systems with a Hammerstein model can be directly identified by genetic agorithms. This method provides good convergence and is successful in a probabilistically guided search process to estimate the parameters of the linear dynamic and nonlinear subsystems.

4.5.2 Simulation Results II

In order to test the convergence of the search process, the model is changed to the following

$$y(k+1) = 1.8y(k) - 0.9y(k-1) + 1.0F[u(k)] + 0.7F[u(k-1)]$$

$$F[u(k)] = \sin\{\frac{u(k) * \Pi}{4.0}\} \qquad u(k) \in [\ 0.0, 2.0\ ]$$

The results of simulation for the linear dynamic subsystem are listed in table (4-5-2)

	a_1	a_2	b_1	k
Real parameters	1.8	0.9	0.7	0.5
Identified parameters	1.788	0.892	0.554	0.532

Table (4-5-2) Identified parameters of *F[u(k)]*

$\sum_{i=0}^{i=number}[\hat{y}(k)-y(k)]^2 = 69.705$, the amplitude of PRBS is 2.0.

The responses of real system and identified system between the range of $U(K)$ = 0.0 to 2.0 are shown in Figure (4-5-4).

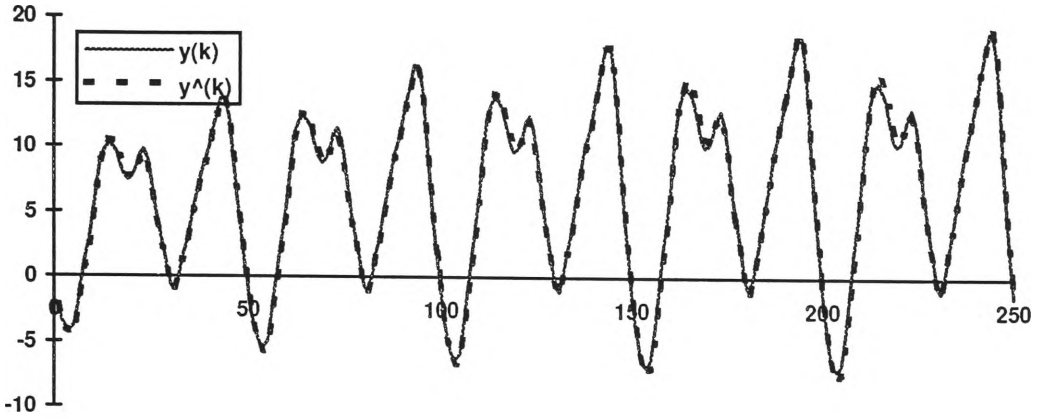


Figure (4-5-4) Responses of the real system $y(k)$and the identified system $\hat{y}(k)$

$$\sum_{i=0}^{i=number}[\hat{y}(k)-y(k)]^2 = 63.18$$

Generation = 1371; Probability of mutation = 0.008; Probability of crossover = 0.99;

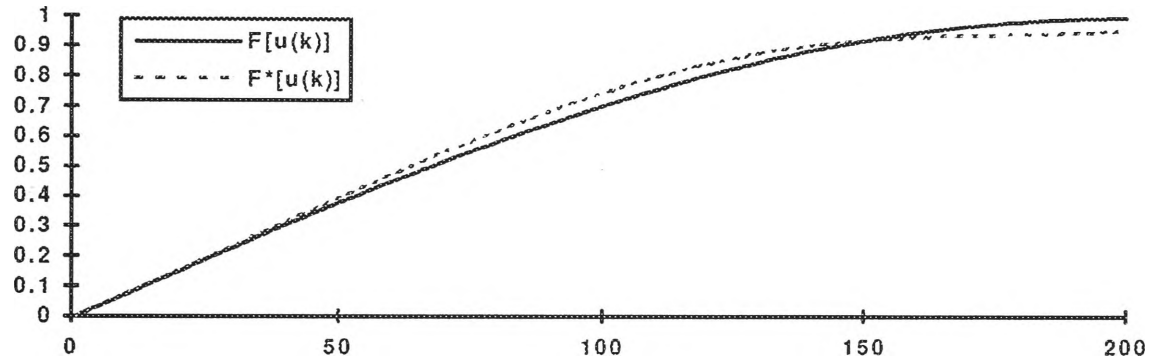


Figure (4-5-5) the identified $F^*[u(k)]$ and the real $F[u(k)]$

The parameters of $F^*[u(k)]$ are given in to table (4-5-3)

	f_1	f_2	f_3	f_4
Identified parameters	0.78	0.050	0.04	-0.15
	f_5	f_6	f_7	f_8
Identified parameters	0.02	0.01	0.0	0.0

Table (4-5-3) Identified parameters of $F[u(k)]$

It can be seen that in spite of some small bias in the estimations, the responses of the real system and the identified system are almost the same. Therefore the identified model is sufficient to be employed in the design of a controller.

4.6 Summary

The results of computer simulations carried out in this chapter demonstrate that genetic algorithms can accurately identify the time delay, order, structure and parameters of a system with fewer sampled input and output data of the system in comparison to conventional methods of system identification.

The developed methodologies will be used to identify the approximate model of a process and tune the parameters of the fuzzy controller designed for it.

Chapter 5

Optimisation of a Fuzzy Controller

5.1. Introduction

This chapter describes the approach developed in this work to optimise a fuzzy controller. This method uses genetic algorithms and is not dependent on the knowledge obtained from a skilled operator. In the process, the membership functions or fuzzy rules are automatically tuned to produce a better control performance. This is radically different from heuristic methods. The method requires only a small set of input and output data from the real system. The technique is developed based on the assumption that system to be controlled is a nonlinear system with a Hammerstein model. The discrete-time mathematical model of such system is given by

$$A(q^{-1})Y(k) = q^{-d}B(q^{-1})F[U(k)] \qquad (5\text{-}1\text{-}1)$$

Where $A(q^{-1}) = a_0 + a_1 q^{-1} + \ldots + a_n q^{-n}$

and $B(q^{-1}) = b_1 q^{-1} + b_2 q^{-2} + \ldots + b_m q^{-m}$,

F[U(k)] is a nonlinear memory less and continuous function related to *U(k)*.

The chapter starts by defining the normalisation procedures used in the approach. The fuzzification of the membership functions and fuzzy rules are then described. The

defuzzification of the output using two methods is presented. The encoding and search processes of the optimisation algorithm using GAs are finally addressed.

5.2 Normalisation of Input and Output Values

In order to design a fuzzy controller the variables are normalised to a value between -1 and +1. This will produce a more generic design algorithm.

The procedures of normalisation of input and output values are as follows:

a) The maximum range of output $Y(k)$ (based on the A/D converter) of the system to be controlled is ±1200, and the maximum given value of the target value $R(k)$ is ±1000. Hence the coefficient of normalisation should be 0.001.

b) $E(k) = R(k) - Y(k)$ (5-2-1)

It is assumed that $E(k)$ varies from +1.2 to -1.2. For values out of this range an overflow is considered.

c) $\Delta E(k) = E(k) - E(k-1)$ (5-2-2)

After normalisation the maximum value of $\Delta E(k)$ will vary between [-2.4, +2.4] based on the limits defined on $E(k)$ and $E(k-1)$. If we choose a fuzzy rule known as very big positive $\Delta E(k)$ when $\Delta E(k)$ is equal to or greater than 1.2, and another fuzzy rule known as very big negative $\Delta E(k)$ when the $\Delta E(k)$ is equal to or smaller than -1.2 the $\Delta E(k)$ will range between -1.2 and +1.2. Furthermore, if the constraints on the real controller output action is considered, a very big $\Delta E(k)$ cannot demand a very big control action due to the constraints on the real controller. Based on the above

conditions, the range of $\Delta E(k)$ can be reduced to be [-1.2, +1.2]. The maximum and minimum membership functions are shown in figure (5-2-1) corresponding to two fuzzy rules of Maximum and Minimum of $\Delta E(k)$

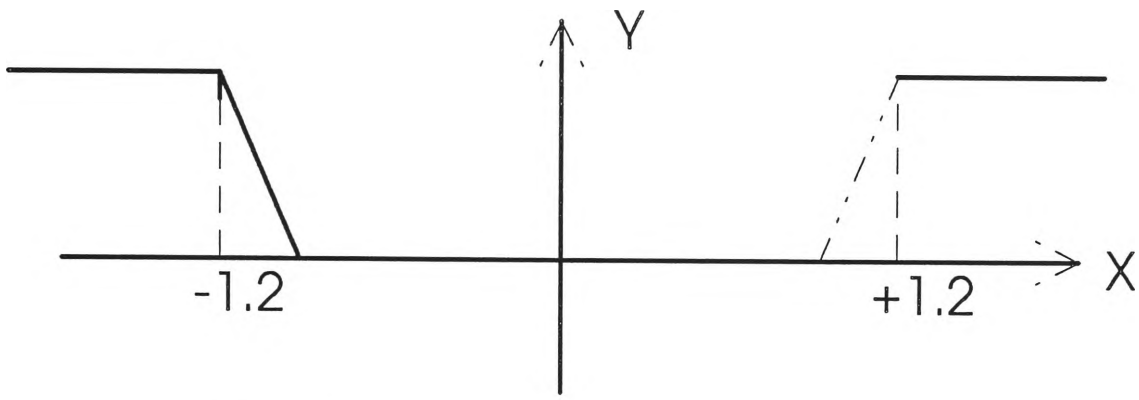


Figure (5-2-1) The maximum and minimum membership function

d) The output of the controller is a fuzzy combination of the membership functions of $E(k)$ and $\Delta E(k)$ such as $[F(E(k)) \vee F(\Delta E(k))]$, or $[F(E(k)) \wedge F(\Delta E(k))]$ where $F(.)$ is membership function. The X axis should be normalised according to the maximum output of the D/A Converter. If the D/A Converter has a 10-bit length then it covers values from 0 to 1023. The X axis will be in the range of -512 to +512. Based on the scaling principle the values -1 and +1 on X axis should provide an output of -500 and +500 respectively.

5.3 Fuzzification Process

In this section the fuzzification process applied to the input and output of the fuzzy controller will be described.

5.3.1 Membership Functions of $E(k)$ and $\Delta E(k)$

In order to get a better approximate fuzzy value of the membership function when it is in the vicinity of 0.0 or 1.0, the membership function of $E(k)$ is assumed to have a square function expressed by (5-3-1) where a and b are sections of the membership functions on X axis.

$$Y_{e(k)} = \frac{-4*(E-a)(E-b)}{(a-b)^2} \qquad \text{a, b} \in [-1.2, 1.2] \qquad (5\text{-}3\text{-}1)$$

If a_0 and b_0 are the values of a and b located near the coordinate system origin then it is assumed that $a_0 = -b_0$. This ensures that the membership function is a symmetric function relative to the Y axis. The span of the membership function should be $2*b_0$.

In general it is desirable to increase the resolution of the errors near the origin of the coordinate system. If the membership function is located within a small range of the origin of the coordinate system, the spans should be selected to be smaller than when it is far from the origin of the coordinate system. This will improve the resolution of the membership function. Consequently the membership function can be assumed to have the following forms:

$$F_0 = k_0(x-a_0)(x-b_0).....a_0 = -b_0.....k_0 = 4/(a_0-b_0)^2 \qquad \text{a, b} \in [-1.2, +1.2] \qquad (5\text{-}3\text{-}2)$$

$$F_1 = k_1(x-0)(x-b_0e^{\alpha}).....k_1 = 4/(0.0-b_0e^{\alpha})^2 \qquad (5\text{-}3\text{-}3)$$

$$F_2 = k_2(x-b_0\sum_{i=1}^{1}2^{i-1}e^{i\alpha}/2^1)(x-b_0e^{2\alpha})...k_2 = 4/(b_0\sum_{i=1}^{1}2^{i-1}e^{i\alpha}/2^1-b_0e^{2\alpha})^2 \qquad (5\text{-}3\text{-}4)$$

$$F_3 = k_3(x-b_0\sum_{i=1}^{2}2^{i-1}e^{i\alpha}/2^2)(x-b_0e^{2\alpha})...k_3 = 4/(b_0\sum_{i=1}^{2}2^{i-1}e^{i\alpha}/2^2-b_0e^{3\alpha})^2 \qquad (5\text{-}3\text{-}5)$$

$$F_4 = k_4(x-b_0\sum_{i=1}^{3}2^{i-1}e^{i\alpha}/2^3)(x-b_0e^{4\alpha})...k_4 = 4/(b_0\sum_{i=1}^{3}2^{i-1}e^{i\alpha}/2^3-b_0e^{4\alpha})^2 \qquad (5\text{-}3\text{-}6)$$

$$F_5 = k_5(x-[b_0\sum_{i=1}^{4}2^{i-1}e^{i\alpha}]/2^4)(x-b_0e^{5\alpha})...k_5 = 4/([b_0\sum_{i=1}^{4}2^{i-1}e^{i\alpha}]/2^4-b_0e^{5\alpha})^2 \qquad (5\text{-}3\text{-}7)$$

:

:

$$F_j = k_j(x-[b_0\sum_{i=1}^{j-1}2^{i-1}e^{i\alpha}]/2^{j-1})(x-b_0e^{j\alpha}) \qquad (5\text{-}3\text{-}8)$$

$$\text{where } k_j = 4/([b_0\sum_{i=1}^{j-1}2^{i-1}e^{i\alpha}]/2^{j-1}-b_0e^{j\alpha})^2 \qquad (5\text{-}3\text{-}9)$$

The above membership functions from F_1 to F_j are illustrated on the right half plane of Figure (5-3-1)

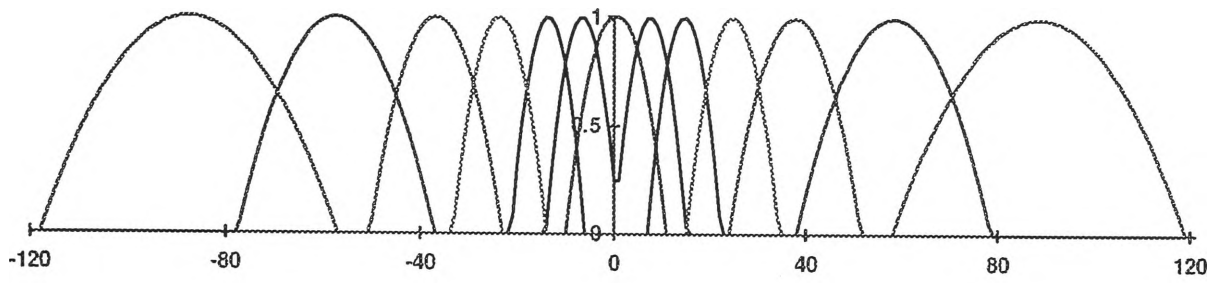


Figure (5-3-1) Membership Functions

On the right half plane ($x \geq 0$) $b_0 e^{6\alpha}$ should be equal to +1.2 . Thus we have the following equation (5-3-10):

$$b_0 e^{6\alpha} = 1.2 \qquad (5\text{-}3\text{-}10)$$

$$a_0 = -b_0 \qquad (5\text{-}3\text{-}11)$$

When b_0 is known, α will be obtained from

$$\alpha = [\ln(1.2/b_0)]/6 \qquad (5\text{-}3\text{-}12)$$

Assuming $b_0 = 0.1$, will give $\alpha = 0.41415$.

For the left half plane, the membership functions are defined by

$$F_0 = k_0(x-a_0)(x-b_0).....k_0 = 4/(a_0-b_0)^2 \qquad a, b \in [-1.2, +1.2] \qquad (5\text{-}3\text{-}13)$$

$$F_1 = k_1(x-0)(x-b_0(1+e^{\alpha})).....k_1 = 4/(0.0-b_0(1+e^{\alpha}))^2 \qquad (5\text{-}3\text{-}14)$$

$$F_2 = k_2(x - b_0\sum_{i=1}^{1} 2^{i-1}e^{i\alpha}/2^1)(x - b_0e^{2\alpha})\ldots k_2 = 4/(b_0\sum_{i=1}^{1} 2^{i-1}e^{i\alpha}/2^1 - b_0e^{2\alpha})^2 \quad (5\text{-}3\text{-}15)$$

$$F_3 = k_3(x - b_0\sum_{i=1}^{2} 2^{i-1}e^{i\alpha}/2^2)(x - b_0e^{2\alpha})\ldots k_3 = 4/(b_0\sum_{i=1}^{2} 2^{i-1}e^{i\alpha}/2^2 - b_0e^{3\alpha})^2 \quad (5\text{-}3\text{-}16)$$

:

:

$$F_j = k_j(x - [b_0\sum_{i=1}^{j-1} 2^{i-1}e^{i\alpha}]/2^{j-1})(x - b_0e^{j\alpha}) \quad (5\text{-}3\text{-}17)$$

$$k_j = 4/([b_0\sum_{i=1}^{j-1} 2^{i-1}e^{i\alpha}]/2^{j-1} - b_0e^{j\alpha})^2 \quad (5\text{-}3\text{-}18)$$

b) The membership function of $\Delta E(k)$

The membership functions of the ΔE(k) are assumed to be the same as the membership functions of E(k).

5.4 Fuzzy Rules and the Membership Functions of the Output

To derive the fuzzy rules and membership functions of the fuzzy controller output, two methods have been applied.

5.4.1 Method I

In the first method one of the membership functions of the output of the fuzzy controller are divided into two parts. In order to simplify the design of the fuzzy controller and reduce the number of the fuzzy rules, a PD (Proportional-plus-differential) controller's reference model is proposed. For an actuator with an accumulator the model [17] is

$$\begin{aligned} &U_{P+D}(k) = U_e(k) \oplus U_{\Delta e}(k), \\ &U(k) = U_{P+D}(k) \end{aligned} \quad (5\text{-}4\text{-}1)$$

where k is sampling time. When there is no accumulator then the model changes to

$$U_{P+D}(k) = U_e(k) \oplus U_{\Delta e}(k),$$
$$U(k) = U(k-1) + U_{P+D}(k) \qquad (5\text{-}4\text{-}2)$$

In these equations, The signal $\oplus$ is the fuzzy logic addition and $U_{P+D}(k)$ is the actual output of the fuzzy controller obtained from fuzzy inference according to the value of the membership functions of $E(k)$ and $\Delta E(k)$. $U_e(k)$ and $U_{\Delta e}(k)$ are the fuzzy output values caused by $E(k)$ and $\Delta E(k)$ respectively. $E(k)$ and $\Delta E(k)$ are used as the inputs to the fuzzy controller because they are the feedback variables of the control system. To simplify the inference process, if the $U_e(k)$ and $U_{\Delta e}(k)$ are assumed to have been generated by $E(k)$ and $\Delta E(k)$ respectively, the relationships between two outputs $U_e(k)$, $U_{\Delta e}(k)$ and two inputs $E(k)$, $\Delta E(k)$ are defined as follows:

a). The rules and the membership functions of the output related to E(k)

If $E(k) = e_0$ then $U_e(k) = U_{e0}(k)$

If $E(k) = e_1$ then $U_e(k) = U_{e1}(k)$

If $E(k) = e_2$ then $U_e(k) = U_{e2}(k)$

:

:

If $E(k) = e_n$ then $U_e(k) = U_{en}(k)$

The membership functions of $U_e(k)$ related to $E(k)$ are assumed as (5-4-3) and (5-3-4). In order to speed up the process of optimisation, the triangular membership functions are chosen because the number of multiplications during the optimisation will be reduced. The triangular membership functions are shown in figure (5-4-1)

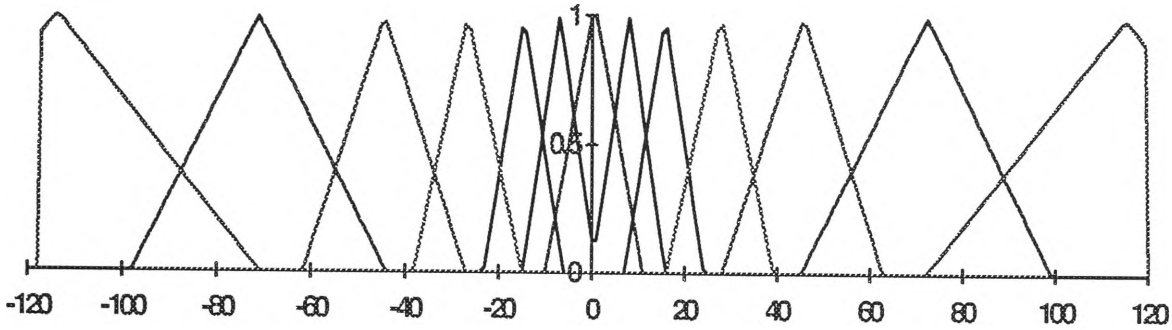


Figure (5-4-1) Membership Function of Output

$$F_i = k_i(x - a_i) \ldots . x \leq (a_i + b_i)/2 \ldots k_i = 2/(b_i - a_i) \qquad (5\text{-}4\text{-}3)$$

$$F_i = -k_i(x - b_i) \ldots . x \leq b_i \ldots . k_i = 2/(b_i - a_i),\ a,\ b \in [-1.0, +1.0].\ \text{and}\ F_i \geq 0.0 \qquad (5\text{-}4\text{-}4)$$

The parameters a_i and b_i (i=1, 11) will be obtained by optimising the control system using GAs.

b). The rules and the membership functions of the output related to ΔE(k)

If $\Delta E(k) = \Delta e_0$ then $U_{\Delta e}(k) = U_{\Delta e0}(k)$

If $\Delta E(k) = \Delta e_1$ then $U_{\Delta e}(k) = U_{\Delta e1}(k)$

If $\Delta E(k) = \Delta e_2$ then $U_{\Delta e}(k) = U_{\Delta e2}(k)$

:

:

If $\Delta E(k) = \Delta e_m$ then $U_{\Delta e}(k) = U_{\Delta em}(k)$

The membership functions of $U_{\Delta e}(k)$ related to $\Delta E(k)$ are given by (5-4-5) and (5-4-6):

$$F_i = k_i(x - a_i) \ldots . x \leq (a_i + b_i)/2 \ldots k_i = 2/(b_i - a_i) \qquad (5\text{-}4\text{-}5)$$

$$F_i = -k_i(x - b_i) \ldots . x \leq b_i \ldots k_i = 2/(b_i - a_i),\quad a,\ b \in [-1.0, +1.0].\ \text{and}\ F_i \geq 0.0 \qquad (5\text{-}4\text{-}6)$$

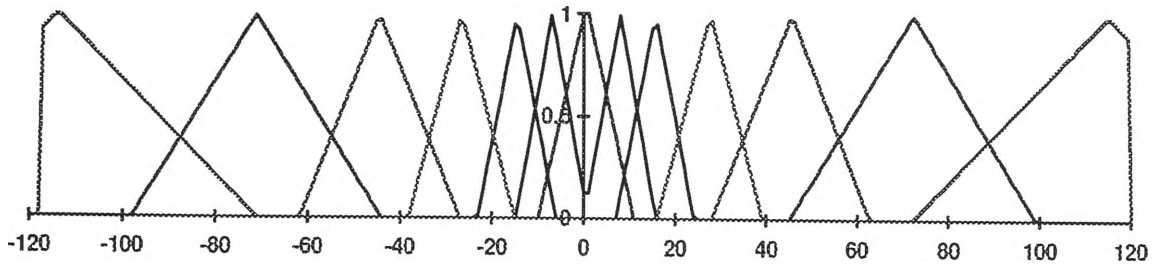


Figure (5-4-2) Membership Function of Output

The number of membership functions of outputs related to $E(k)$ and $\Delta E(k)$ is assumed to be 11. The number of membership functions determines the control performance. If a higher level of control performance is required more membership functions of outputs should be chosen. This will, however, increase the number of the membership functions to be optimised. Under these conditions there will be 44 parameters of a_i and b_i of the output membership functions to be optimised in the method. When all the parameters a_i and b_i are obtained, the crisp value of the output is calculated by the Central of area (COA) method as described in section 5.5 of this chapter.

5.4.2 Method II

In the second method the fuzzy rules for the outputs are directly optimised to tune the fuzzy controller. The fuzzy rules are:

If $E(k) = e_0$ and $\Delta E(k) = \Delta e_0$ then $U(k) = R_{00}$

If $E(k) = e_1$ and $\Delta E(k) = \Delta e_0$ then $U(k) = R_{10}$

If $E(k) = e_2$ and $\Delta E(k) = \Delta e_0$ then $U(k) = R_{20}$

:

If $E(k) = e_{n-1}$ and $\Delta E(k) = \Delta e_0$ then $U(k) = R_{n-1,0}$

:

If $E(k) = e_0$ and $\Delta E(k) = \Delta e_1$ then $U(k) = R_{01}$

If $E(k) = e_1$ and $\Delta E(k) = \Delta e_1$ then $U(k) = R_{11}$

If $E(k) = e_2$ and $\Delta E(k) = \Delta e_1$ then $U(k) = R_{21}$

:

If $E(k) = e_{n-1}$ and $\Delta E(k) = \Delta e_1$ then $U(k) = R_{n-1,1}$

:

:

If $E(k) = e_0$ and $\Delta E(k) = \Delta e_{n-1}$ then $U(k) = R_{0n-1}$

If $E(k) = e_1$ and $\Delta E(k) = \Delta e_{n-1}$ then $U(k) = R_{1n-1}$

If $E(k) = e_2$ and $\Delta E(k) = \Delta e_{n-1}$ then $U(k) = R_{2n-1}$

:

If $E(k) = e_{n-1}$ and $\Delta E(k) = \Delta e_{n-1}$ then $U(k) = R_{n-1,n-1}$

Where n is the number of membership functions of $E(k)$ and $\Delta E(k)$. Therefore, there are $(n-1)*(n-1)$ fuzzy rules in the fuzzy controller. The rule look-up table with nine membership functions is illustrated in Figure (5-4-3). Overall the N will be 81 fuzzy rules. If the number of membership functions is equal to nine, then there will be 81 fuzzy rules in the look-up table.

Δe \ e	nvb	nb	nm	ns	0	ps	pm	pb	pvb
nvp	R00	R01	R02	R03	R04	R05	R06	R07	R08
np	R10	R11	R12	R13	R14	R15	R16	R17	R18
nm	R20	R21	R22	R23	R24	R25	R26	R27	R28
ns	R30	R31	R32	R33	R34	R35	R36	R37	R38
0	R40	R41	R42	R43	R44	R45	R46	R47	R48
ps	R50	R51	R52	R53	R54	R55	R56	R57	R58
pm	R60	R61	R62	R63	R64	R65	R66	R67	R68
pb	R70	R71	R72	R73	R74	R75	R76	R77	R78
pvb	R80	R81	R82	R83	R84	R85	R86	R87	R88

Figure (5-4-3) Look-up table of a fuzzy controller

As described above, the $U(k)$ is only dependent on $R_{i,j}$. However every fuzzy rule $R_{i,j}$ will be determined by *E(k)* and $\Delta E(k)$. If the two membership functions are equal to 1 for a pair of *E(k)* and $\Delta E(k)$ then the control action of output from the fuzzy controller should be $R_{i,j}$. In general, the two membership functions are not equal to 1 and hence the control action is calculated by fuzzy logic operators as described in section 5.5 of this chapter.

In this approach all the fuzzy rules must be optimised. In order to avoid local minima points, genetic algorithms are used in this work. The encoding method used will be introduced later in this chapter.

5.5 Defuzzification of the Fuzzy Controller Output

In order to defuzzify the output of the system, two defuzzification methods proposed by Tsakamoto [57] have been applied in this work. The first method of COA is complex to calculate but produces more accurate results. The second method, (Tsakamoto's method) is computationally simple but its resolution is less than the COA method. In general it has been observed that the COA method is more appropriate when the membership functions are optimised and the Tsakamoto's method when fuzzy rules are optimised.

In the COA approach Z_e^* and $Z_{\Delta e}^*$ · which are the control outputs related to *E(k)* and $\Delta E(k)$ are obtained from (5-5-1)

$$Z_e^* = \frac{\sum_{j=1}^{q} z_j F_c(z_j)}{\sum_{j=1}^{q} F_c(z_j)}, \qquad Z_{\Delta e}^* = \frac{\sum_{j=1}^{q} z_j F_c(z_j)}{\sum_{j=1}^{q} F_c(z_j)} \qquad (5\text{-}5\text{-}1)$$

where q is the number of quantisation levels of the output Z, and F_c represents its membership function value based on the input value.

For the second method, the crisp control action is derived from (5-5-2)

$$U[k] = \lambda * \frac{\sum_{i=1}^{n} \omega_i x_i}{\sum_{i=1}^{n} \omega_i} \qquad (5\text{-}5\text{-}2)$$

Where n is the number of rules with firing strength (ω_i) greater than 0, x_i is the amount of control action recommended by the rule i and λ is a proportional parameter which will be optimised in order to make the fuzzy controller suitable for a system with a specific static gain.

Assume that $F_{m1}(E_1)$and $F_{m2}(E_1)$ denote the values of membership functions F_{m1} and F_{m2}, when E(k) is equal to E_1. $F_{m1}(\Delta E_1)$, and $F_{m2}(\Delta E_1)$ denote the values of membership function F_{m1} and F_{m2} when $\Delta E(k)$ is equal to ΔE_1, then

$\omega_1 = [\, F_{m1}(E_1) \Lambda\, F_{m1}(\Delta E_1)]$, and control action x_{r1} is equal to the value related to this rule.

$\omega_2 = [\, F_{m1}(E_1) \Lambda\, F_{m2}(\Delta E_1)]$, and control action x_{r2} is equal to the value related to this rule.

$\omega_3 = [\, F_{m2}(E_1) \Lambda\, F_{m1}(\Delta E_1)]$, and control action x_{r3} is equal to the value related to this rule.

$\omega_4 = [\, F_{m2}(E_1) \Lambda\, F_{m2}(\Delta E_1)]$, and control action x_{r4} is equal to the value related to this rule.

The symbol Λ is the 'minimum' operator.

Therefore:

$$U[k] = \lambda * \frac{\sum_{i=1}^{4} \omega_i x_i}{\sum_{i=1}^{4} \omega_i} \qquad (5\text{-}5\text{-}3)$$

The formula (5-5-3) is used in the second method for optimising the rules.

5.6 Design of the Fuzzy Controller Using Genetic Algorithms

The fuzzy control system can be shown by the block diagram of Figure (5-6-1). The fuzzification, defuzzification and fuzzy rules are included in the fuzzy controller.

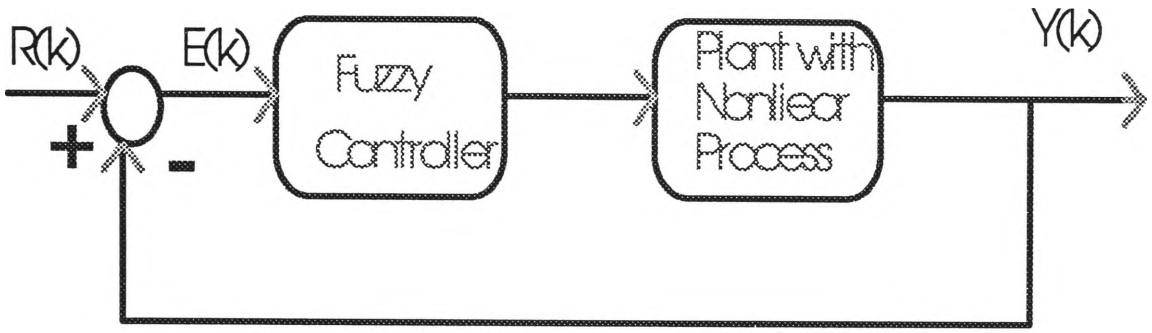


Figure (5-6-1) The configuration of the control system

In this section the design process developed using GAs algorithm is described by explaining the ecoding process, ojective fnction employed, and the search process used.

5.6.1 Encoding the Membership Functions for Method One

The membership functions of the output rules are defined by two points *a* and *b* across the *x* axis. After normalisation of *E(k)* and $\Delta E(k)$, the parameters, *a* and *b* should be within [-1, +1].

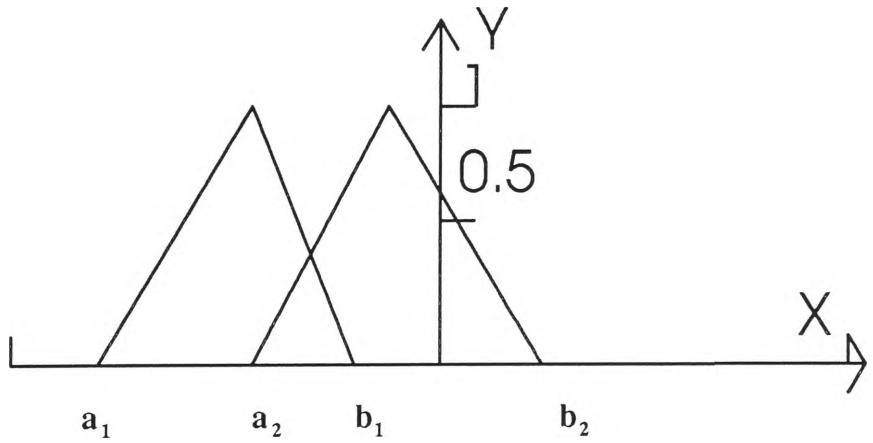


Figure (5-6-2) Two points a and b of a membership functions

In encoding the membership functions, the string consists of 11 bits, the most significant bit of which denotes the sign, and the remaining bits define the value of a or b. Since there are 11 rules used for $E(k)$ and $\Delta E(k)$, there will be a total of 44 parameters to be optimised. An increase in the membership functions will obviously increase the number of parameters to be optimised.

In order to realise the fuzzy logic addition mentioned in section 5.4 of this chapter, two fuzzy gains ω_1 and ω_2 are defined to perform fuzzy logic addition:

$$U_{P+D}(k) = \omega_1 * U_p(k) + \omega_2 * U_D(k) \qquad (5\text{-}6\text{-}1)$$

The parameters ω_1 and ω_2 should also be optimised.

5.6.2 Encoding the Fuzzy Rules for Method Two

If there are m membership functions for the fuzzy controller inputs $E(k)$ and $\Delta E(k)$ then m^2 fuzzy rules should be optimised. In this case every fuzzy rule Rij in figure (5-4-3) is encoded as an 8-bit binary string. The most significant bit of the binary string again denotes the positive or negative sign, and the remaining bits express the value of the fuzzy rule. After obtaining the values of the fuzzy rules, they are normalised by dividing them by 128. This ensures that the values of Rij in figure (5-4-3) are within the range of ± 1.0. For example, if an 8-bit binary string expressing R11 is equal to 10010000 then the fuzzy rule R11 is -16/128 or -0.125.

5.6.3 Objective Function

In order to reach a high level of control performance, the objective function is defined as formula (5-6-2). This provides different weights for various stages of the optimisation process.

$$Obj = \sum_{j=1}^{l} \sum_{k=k_j}^{k_{j+1}-1} w_j e^2(k) \qquad (5\text{-}6\text{-}2)$$

In this relationship l is the number of the separated phases, w_i is the weight for phase j, k_j is the number of sampling intervals for phase j, and $e(k) = R(k) - Y(k)$.

The value of $E(k)$ is significantly large during transient response compared to the steady state. A small value of w_i compensates for large $E(k)$ during the transient response. For example if the set point function is defined as :

$$\begin{aligned}
&R(k) = 0.5 && 0 < k < 20 \\
&R(k) = -0.5 && 20 < k < 40 \\
&R(k) = 0.8 && 40 < k < 60 \\
&R(k) = 0.0 && 60 < k < 80 \\
&R(k) = 1.0 && 80 < k < 100 \\
&R(k) = 0.6 && 100 < k < 120 \\
&R(k) = 1.0 && 120 < k < 140
\end{aligned} \qquad (5\text{-}6\text{-}3)$$

The objective function and typical values of w_i will be

$$Obj = \sum_{k=1}^{7} e^2(k) + \sum_{k=8}^{20} 100e^2(k) + \sum_{k=21}^{27} e^2(k) + \sum_{k=28}^{40} 100e^2(k) + \sum_{k=41}^{47} e^2(k)$$
$$\text{.....} + \sum_{k=48}^{60} 100e^2(k) + \sum_{k=61}^{67} e^2(k) + \sum_{k=68}^{80} 100e^2(k) + \sum_{k=81}^{87} e^2(k) + \sum_{k=88}^{100} 100e^2(k) \qquad (5\text{-}6\text{-}4)$$
$$\text{.....} + \sum_{k=101}^{107} e^2(k) + \sum_{k108}^{120} 100e^2(k) + \sum_{k=121}^{127} e^2(k) + \sum_{k=128}^{140} 100e^2(k)$$

As mentioned before in section 3.4 of chapter 3, evaluation function used in the GAs is the fitness function defined as:

$$\text{fitness function} = PM - Obj \qquad (5\text{-}6\text{-}5)$$

Where PM is an adaptive positive factor to guarantee a fitness function greater than zero.

The advantages of a set point function with varying amplitudes are to improve the control performance of the fuzzy controller and to make every fuzzy rule more effective for different levels of disturbance. It is well known that fuzzy control systems are nonlinear systems. Hence the change of the set point will cause the control performance to deteriorate if the optimised controller is globally optimal only for one set point. It is, therefore, necessary to modify the objective function used in the optimisation process for different levels of step responses.

In other words, if the controller designed by this method is a linear controller and the plant is also linear then the change of set point does not affect the control performance. On the other hand, for a nonlinear controller optimised for a given set point and controlling a nonlinear plant, the change of set point does surely affect the control performance. Hence a fuzzy controller optimised using an objective function can perform better than a linear controller under varying required conditions.

5.6.4 Procedure of Search Process

The search process used in the optimisation process progresses as follows:

(i) Select a small positive number ξ_{op} = 10.0. If a more accurate response is required the ξ_{op} should be chosen to be less than 10. The probability of mutation and crossover used in the genetic algorithm should be then provided.

(ii) Define the set point function and the weights w_i of the objective function.

(iii) Use genetic algorithms to minimise the objective function or maximise the fitness function. The procedures include crossover, mutation, calculation of fitness function of every individual and estimation of the adaptive positive factor.

(iv) Check the fitness of every individual and pick the best of the population. In order to detect the convergence of this search process a small positive number ε is defined to be compared with the minimum value of all the objective functions of the population. If this value is less than ε, then the search process is terminated. Otherwise, save the minimum objective function of this generation of search and continue the search process.

(v) Use method I and method II to establish a look-up table.

5.7 Summary

In this chapter the procedure developed in this work to design and tune a fuzzy controller has been discussed in detail. In order to achieve a better performance, a multi-level set point function with different amplitude has been chosen and an objective function with different weights has been developed. This is different from traditional optimal methods in which the controller is optimised for a constant set point. Using this strategy, a better performance under varying set points can be guaranteed for a nonlinear controller that controls a nonlinear system.

During the course of the chapter the procedures used for normalisation of input and output values and selection of objective function were also introduced. Based on the principles described in the chapter these procedures have been coded as a design tool using Borland C++.

Chapter 6 Validation of the Method

6.1 Introduction

In this chapter the developed methodology for tuning a fuzzy controller will be validated through computer simulation and experimental work. In the simulation work the fuzzy membership functions and fuzzy rules for a nonlinear system are optimised. In the experimental work the optimised controller tuned for a DC motor is applied in real time and the results are compared with a PD controller.

6.2 Validation Through Simulation

The configuration of the control system used in the computer simulation is illustrated in Figure (6-2-1). A nonlinear process is controlled by a fuzzy controller.

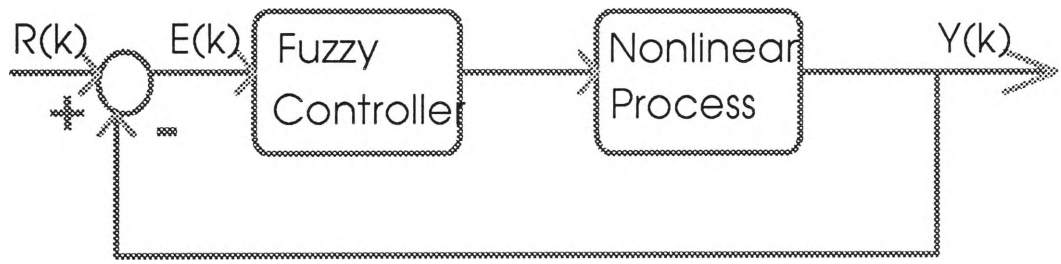


Figure (6-2-1) The control system with the non linear process

6.2.1 Optimisation of Membership Functions

For this simulation, a process with the following model is chosen:

$$A(q^{-1})Y(k) = q^{-d}B(q^{-1})F[U(k)] \qquad (6\text{-}2\text{-}1)$$

Where $A(q^{-1}) = (1-q^{-1})(1-1.78q^{-1}+0.784q^{-2})$,

$B(q^{-1}) = 1.36(1-0.7q^{-1})$ *and* $d = 2$

$F[U(k)] = 1.0\sin[U(k)\Pi/2]$

The model of the membership Functions of the error *E(k)* and change of error $\Delta E(k)$ have been defined as section 5.3. In this simulation there are 13 membership functions in the range of -1.0 to 1.0. It is also assumed that there are six membership functions on the right plane and six membership functions on the left plane, hence n is selected as six. There is also one membership function at the origin of the coordinate frame making a total of 13 membership functions. Thus:

$$b_0 e^{6\alpha} = 1.2 \qquad (6\text{-}2\text{-}2)$$

Assuming b_0 =0.1, then

$$a_0 = -b_0 \qquad (6\text{-}2\text{-}3)$$

$$\alpha = [\ln(1.2/b_0)]/6 \qquad (6\text{-}2\text{-}4)$$

$$\alpha = 0.41415.$$

The 13 membership functions are similar to the ones illustrated in figure (5-3-1) for a typical value b_0 and α.

Hence in order to define and optimise the membership functions of *E(k)* and $\Delta E(k)$, 52 variables are required to be tuned. In addition the weights ω_1 and ω_2 of the fuzzy logic addition defined in formula (5-6-1) of section 5.6 must also be optimised during the search process. This will increases the number of variables required to be optimised to 54.

In a fuzzy control system, a change in the reference point will cause a deterioration in the control performance if the controller is globally optimised only for one specific reference point. This can be a problem when a system is continuously disturbed by step functions of different amplitudes. For each step function, a new objective function optimising the fuzzy controller should be calculated.

In this work, such problem is avoided by employing an objective function as given by (6-2-5). As soon as the system arrives at a steady state condition, a weight of 100 is inserted into the objective function. This will increase the value of the objective function if the steady state error is too small. Hence small steady state error will result in a large change in objective function. The steady state error and rise time of the system response are directly proportional to the value of the chosen weight.

$$Obj = \sum_{k=1}^{7} e^2(k) + \sum_{k=8}^{20} 100e^2(k) + \sum_{k=21}^{27} e^2(k) + \sum_{k=28}^{40} 100e^2(k) + \sum_{k=41}^{47} e^2(k)$$
$$\ldots\ldots + \sum_{k=48}^{60} 100e^2(k) + \sum_{k=61}^{67} e^2(k) + \sum_{k=68}^{80} 100e^2(k) + \sum_{k=81}^{87} e^2(k) + \sum_{k=88}^{100} 100e^2(k)$$
$$\ldots\ldots + \sum_{k=101}^{107} e^2(k) + \sum_{k108}^{120} 100e^2(k) + \sum_{k=121}^{127} e^2(k) + \sum_{k=128}^{140} 100e^2(k) \qquad (6\text{-}2\text{-}5)$$

The reference input function R(k) used in this simulation is defined by

$$\begin{aligned} R(k) &= 0.5 && 0 < k < 20 \\ R(k) &= 0.0 && 20 < k < 40 \\ R(k) &= 0.8 && 40 < k < 60 \\ R(k) &= 0.0 && 60 < k < 80 \\ R(k) &= -0.5 && 80 < k < 100 \\ R(k) &= 0.0 && 100 < k < 120 \\ R(k) &= 1.0 && 120 < k < 140 \end{aligned} \qquad (6\text{-}2\text{-}6)$$

The optimised membership functions of the fuzzy outputs related to $E(k)$ and $\Delta E(k)$ are illustrated in Figures (6-2-2) and (6-2-3) respectively. The scale of the x-axis is 1/100 of the real value of the output. In these diagrams (6-2-2) M_i and N_i (i = 1, 13) are the membership functions of the fuzzy controller output related to the ith membership functions of $E(k)$ and $\Delta E(k)$ respectively.

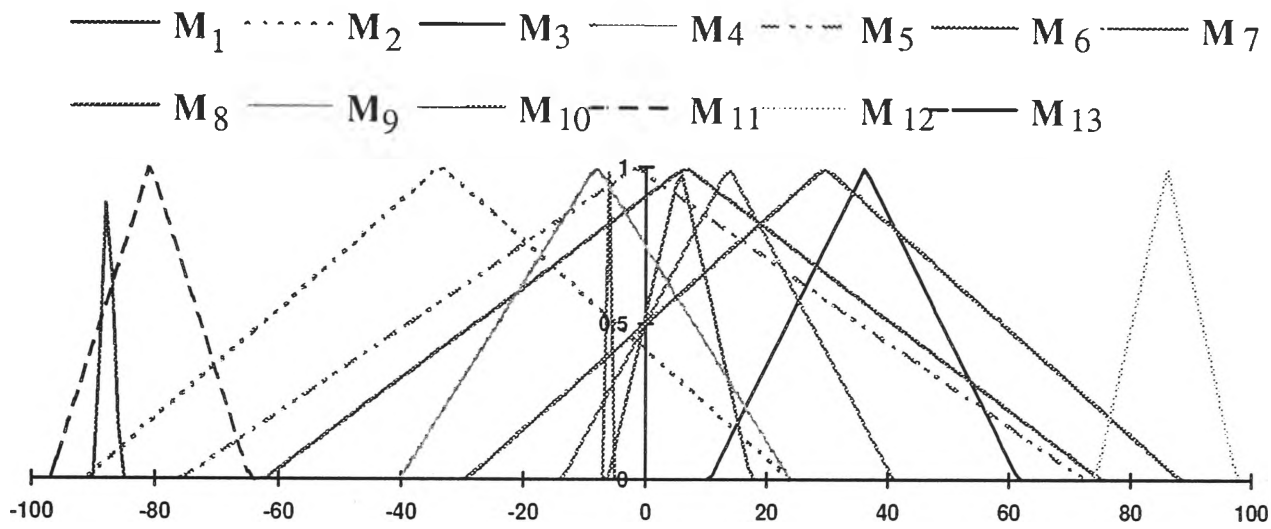


Figure (6-2-2) Optimised membership functions of fuzzy output related to membership functions of $E(k)$

The system response of the optimised controller to the variable reference input $R(k)$ is illustrated in Figure (6-2-4). The value of the objective function is 11.73 and the weights, ω_1 and ω_2, of the fuzzy logic addition, are 0.085 and 0.123 respectively. The search process lasted for 867 generations. The response has a maximum overshoot of 0.68%, a rise time of 7 sampling intervals and a steady state error of 0.03%.

The other parameters of the optimisation process are:

Population size = 280

Probability of mutation = 0.0062

Probability of crossover = 0.96

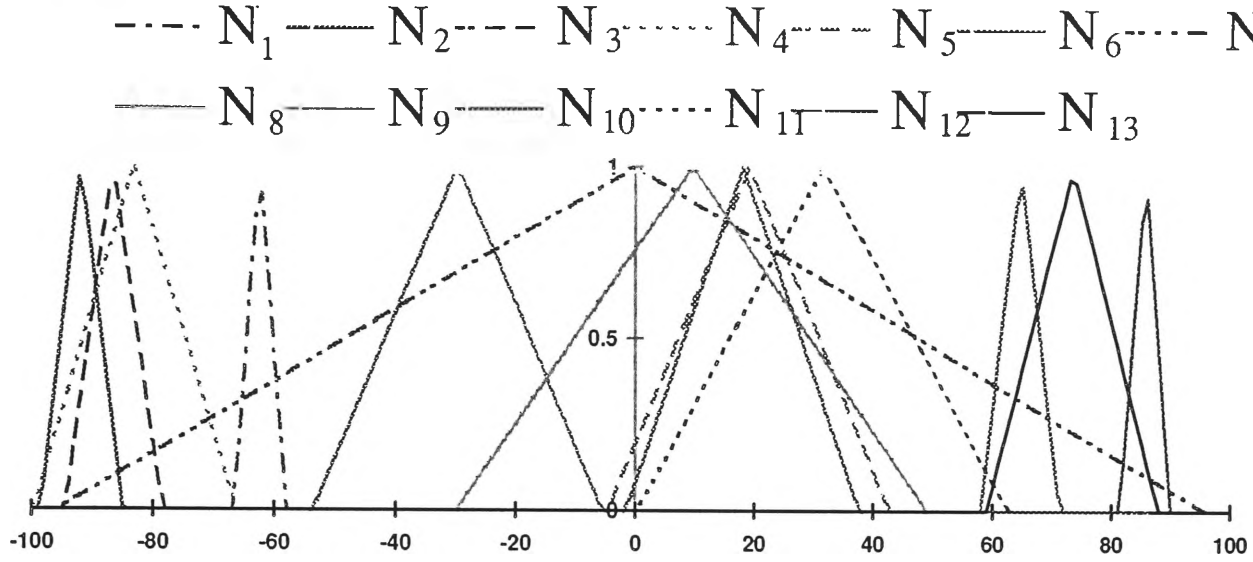


Figure (6-2-3) Optimised membership functions of fuzzy output related to membership functions of $\Delta E(k)$

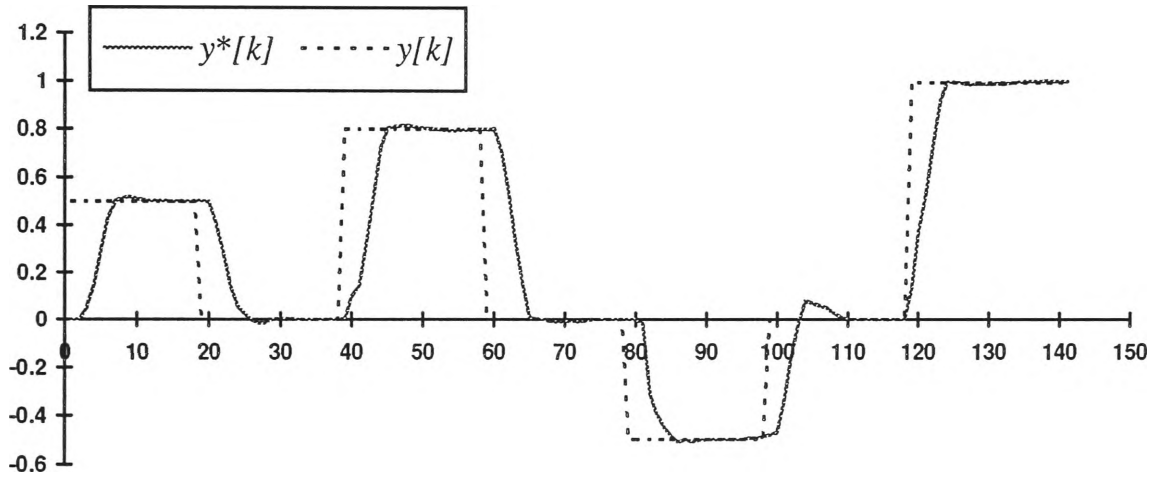


Figure (6-2-4) System responses optimised by the GAs for variable reference-point

The performance of the optimised fuzzy controller has been compared with a fuzzy controller intuitively tuned. The output membership functions of this controller related to the membership functions of $E(k)$ and $\Delta E(k)$ are respectively illustrated in Figures (6-2-6) and (6-2-7). The response of the system with this controller to the variable reference input is shown in Figure 6-2-5. This response has exhibited a maximum overshoot of 8.6%, a rise time of 7 sampling intervals and a steady state error of 3.4%.

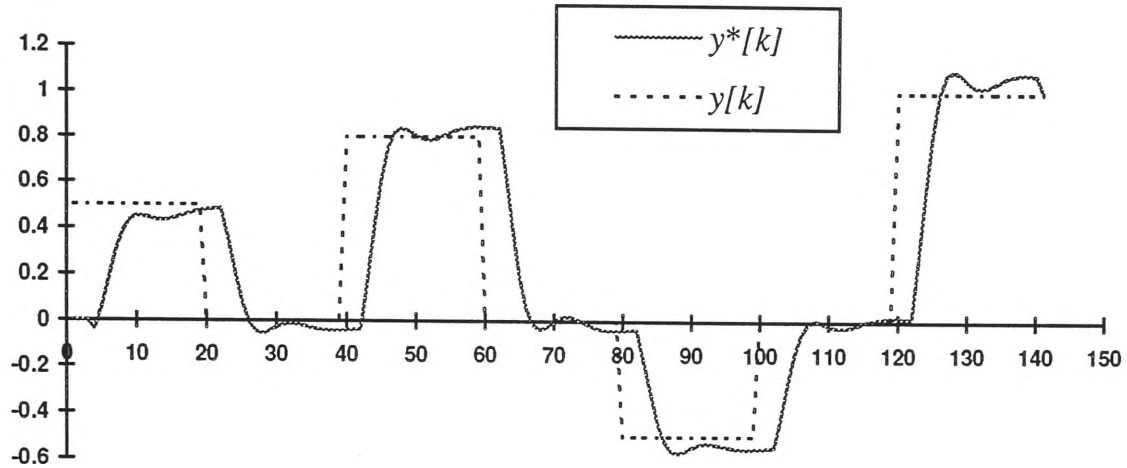


Figure (6-2-5) System response of the intuitively tuned controller for variable reference input

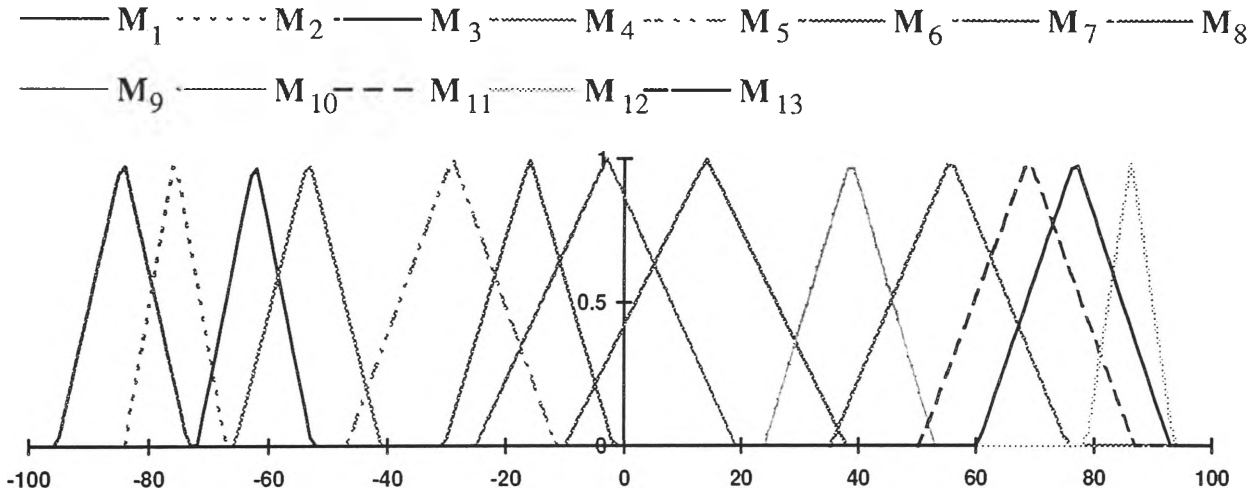


Figure (6-2-6) Membership functions of fuzzy output intuitively tuned related to membership functions of *E(k)*

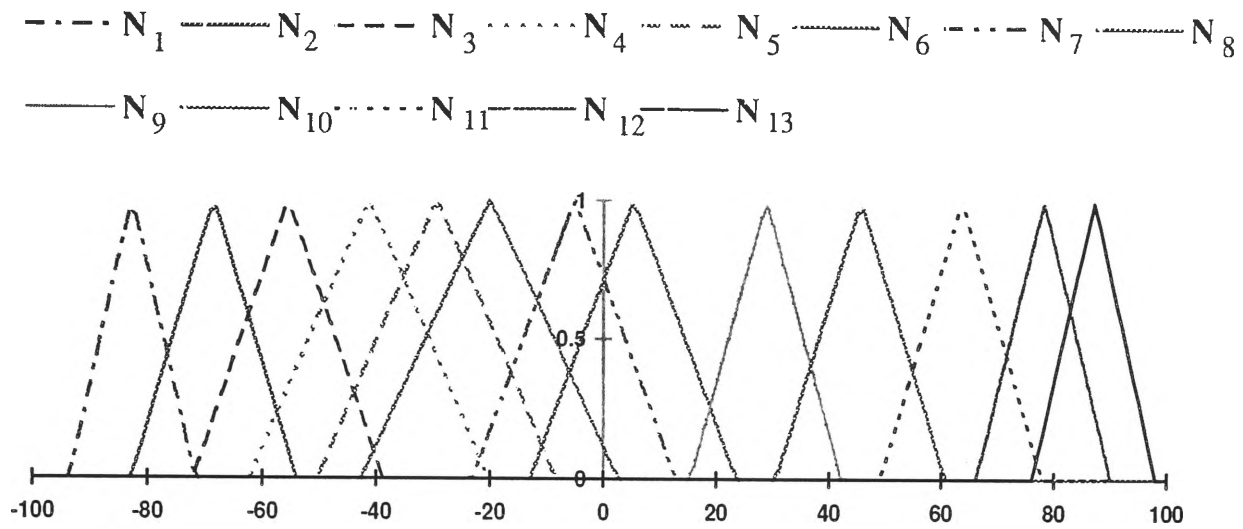


Figure (6-2-7) Membership functions of the fuzzy output tuned intuitively in relation to membership functions of $\Delta E(k)$

In the manual tuning, it is assumed that the membership functions N_1 to N_6 correspond to the error from maximum negative to minimum negative values, N_{13} to N_8 correspond to the error from maximum positive to minimum positive values and N_7 corresponds to the error in the vicinity of zero. A similar procedure is defined for change of error. The strategies used for manual tuning of the fuzzy controller are as follows:

1. Initialise fuzzy output membership functions related to error and change of error as
    a) The membership functions are isosceles triangular;
    b) Every triangular has equal span on x-axis;
    c) The membership functions are located on x-axis as shown in Fig (6-2-8).

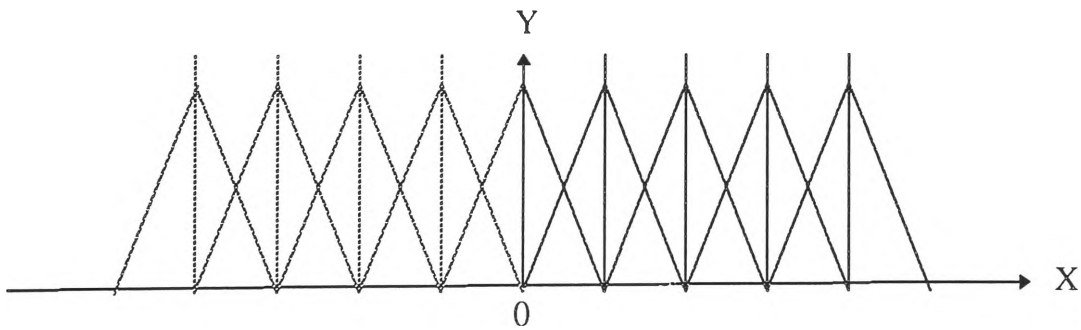


Figure (6-2-8) Initialised membership functions

2. Ignore the change of error and concentrate on using the error as the input to the fuzzy controller. Use the trial and error approach to tune the output membership functions of the fuzzy controller to get a step response with a short rise time. To ensure the stability of the system:

    a) If the rise time is too long, increase control action by shifting membership functions located on the right half plane to right when the set point is positive, or shifting membership functions located on the left half plane to left when the set point is negative.

    b) If the system is unstable, decrease the control action by shifting membership functions located on the right half plane to left when the set point is positive, or shifting membership functions located on the left half plane to right when the set point is negative

    c) Repeat steps a) and b) until the step response has a satisfactory rise time.

3. Include the membership functions of change of error into the tuning process and tune the output membership functions related to the change in error

    a) If the overshoot is too large then increase the control action by shifting membership functions located on the right-half plane to right and shifting membership functions located on the left-half plane to left.

    b) If the system response is too slow, decrease the control action by shifting membership functions located on the right-half plane to left and shifting membership functions located on the left-half plane to right.

    c) Repeat steps a) and b) until the step response has a shorter rise time.

4. After tuning the membership functions, use COA method as described in section 5.5 to defuzzify the control action.

Comparing Figure (6-2-4) and Figure (6-2-5) clearly illustrates that the performance of the fuzzy controller tuned by genetic algorithms is superior to the other. Moreover intuitive tuning of a fuzzy controller takes much longer than using the design tool.

In the experiment conducted, the method based on GAs converged after 867 generations which took 30 minutes. The intuitive method required 2400 minutes and 600 iterations.

An understanding about the "optimised" controller (MF) could greatly benifit the thesis and the reader.

6.2.2 Optimisation of Fuzzy Rules

The optimisation of the fuzzy rules is illustrated through two case studies and the results obtained from the optimised fuzzy controller will be compared with the results obtained from a fuzzy controller tuned intuitively. The intuitive method is performed as follows:

1. Initialise a fuzzy rule look-up with some values as shown in table (6-2-1).
2. Use the trial and error Method to tune the fuzzy rules to get a step response with short rise time while maintaining the stability of the system.
    a) If rise time is too long, increase λ or decrease control action obtained from the derivative of error by adding a small positive number such as 0.1 or 0.05 to the fuzzy rules at the left-top area of the table (6-2-1). Also increase control action obtained from error by adding a positive number to fuzzy rules at the right-top area of the table (6-2-1) when the set point is greater that initial value y(0).

    b) If the set point is greater that initial value *y(0)* and the rise time is too long then increase λ or decrease control action obtained from the derivative of error by

adding a small negative number such as -0.1 or -0.05 to the fuzzy rules at the left-bottom area of the table (6-2-1) and increase control action obtained from error by add a negative number to fuzzy rules at the right-bottom area of the table (6-2-1).

c) If the system is getting unstable or the response has big over-shoot, decrease the control action by adding a small positive number such as 0.1 or 0.05 to fuzzy rules at the left-top triangular area of the table (6-2-1) and by adding a negative number to fuzzy rules at the right- bottom triangular of the table (6-2-1). If necessary we can decrease λ ($0< \lambda <1.0$).

d) Repeat steps a), to c) until the step response has a shorter rise time and smaller over-shoot.

ė \ e	-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-1.0	-0.9	-0.8	-0.7	-0. 6	-0.5	-0.4	-0.3	-0.2	-0.1	0.0
-4	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0.0	0.1
-3	0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0.0	0.1	0.2
-2	-0.7	-0.6	-0.5	-0.4	-0. 3	-0.2	-0.1	0.0	0.1	0.2	0.3
-1	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0.0	0.1	0.2	0.30	0.4
0	-0.5	-0.4	-0.3	-0.2	-0.1	0.0	0.1	0.2	0.3	0.4	0.5
1	-0.4	-0.3	-0.2	-0.1	0.0	0.1	0.2	0.30	0.4	0.5	0.6
2	-0.3	-0.2	-0.1	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
3	-0. 2	-0.1	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
4	-0.1	0.0	0.1	0. 2	0.3	0.4	0.5	0.6	0.7	0.8	0.90
5	0.0	0.10	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Table (6-2-1) Fuzzy rules (Look-up table) for intuitively tuned controller

The system model assumed in the simulation is

$$A(q^{-1})Y(k) = q^{-d}B(q^{-1})F[U(k)] \qquad (6\text{-}2\text{-}7)$$

Where $A(q^{-1}) = (1-q^{-1})(1-1.78q^{-1}+0.784q^{-2})$,

$$B(q^{-1}) = 0.36(1-0.7q^{-1}) \; and \; d = 2$$

$$F[U(k)] = 1.0\sin[U(k)\Pi/2]$$

This model is the same as the model used in section 6.2.1 of this chapter. Hence the results obtained by optimising fuzzy rules and fuzzy membership functions can be compared. In this case the reference points are changed to demonstrate the robustness of the method to different set points. The objective function, however, has remained the same. The method was applied to tune the fuzzy controller using the following input function:

$$\begin{aligned} R(k) &= 0.5 && 0 < k < 20 \\ R(k) &= 0.0 && 20 < k < 40 \\ R(k) &= 1.0 && 40 < k < 60 \\ R(k) &= 0.0 && 60 < k < 80 \\ R(k) &= 0.8 && 80 < k < 100 \\ R(k) &= 1.0 && 100 < k < 120 \\ R(k) &= 0.6 && 120 < k < 140 \end{aligned} \qquad (6\text{-}2\text{-}8)$$

The membership functions related to $E(k)$ and $\Delta E(k)$ are shown in Figure(5-3-1) and the parameters used in the search process were:

No. of Generations = 1896 Population size = 380

Probability of mutation = 0.0028 Probability of crossover = 0.96

The maximum overshoot = 0.98% for the unit step target

The rise time = 6 sampling intervals for the unit step target

Steady State error = 0.06%

The response of the optimised fuzzy controlled system to the input function is shown in Figure (6-2-9). The value of the objective function is equal to 12.301 and λ = 0.867. According to (5-5-2) the control signal $U_{real}(k)$ of the fuzzy controller is produced by

$$U_{real}(k) = \lambda * U_{fuzzy}(k) \qquad 0 < \lambda < 1.0 \qquad (6\text{-}2\text{-}9)$$

Where $U_{real}(k)$ is the control signal, $U_{fuzzy}(k)$ is the crisp control value obtained from the look-up table and λ is a proportional factor converting the crisp control value to output control signal. In this process 122 variables have been optimised including the 121 fuzzy rules and λ. The optimised fuzzy rule table is shown in Table (6-2-2). The intuitively tuned fuzzy rule table is given in table (6-2-3).

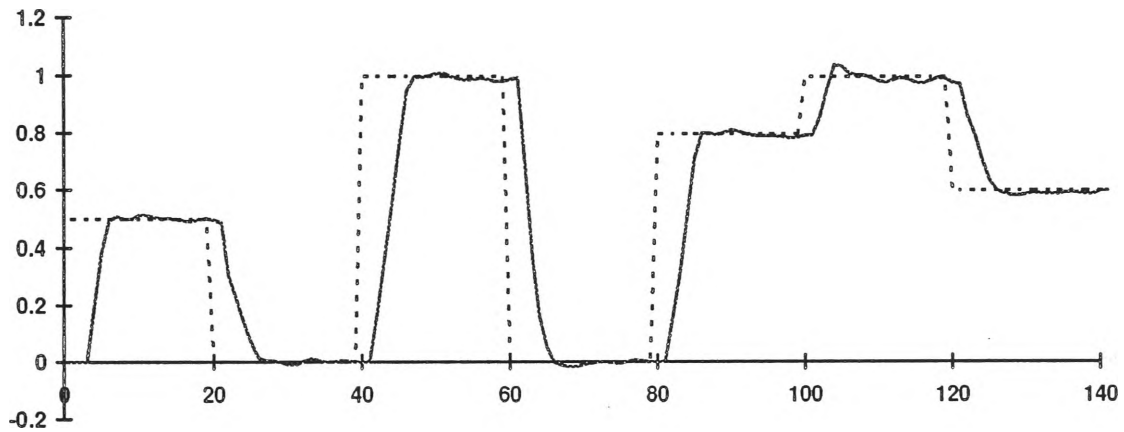


Figure (6-2-9) Response of the optimised fuzzy controller - Optimised fuzzy rules

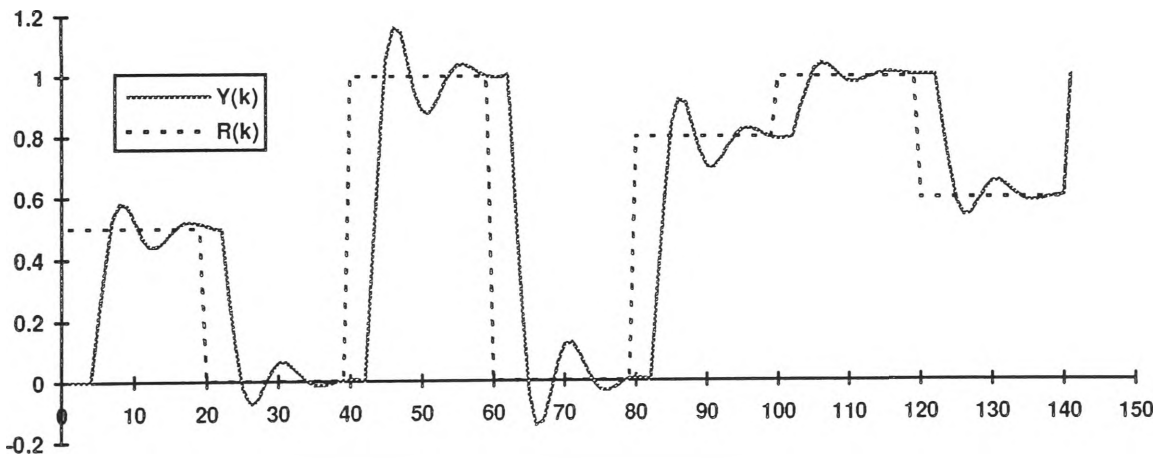


Figure (6-2-10) Response of fuzzy controller tuned intuitively

ė \ e	-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-0.999	-0.667	-0.534	-0.534	0.303	-.9992	-0.999	-0.667	0.303	-0.820	0.534
-4	-0.820	-0.534	0.6677	-0.534	-0.414	-0.9992	-0.3032	-0.667	-0.820	0.8201	-0.198
-3	0.667	-0.198	-0.820	-0.820	-0.303	-0.3032	-0.6677	-0.999	-0.414	-0.414	-0.098
-2	-0.303	-0.820	0.0000	-0.303	0.3032	-0.1988	-0.6677	0.3032	0.8201	-0.667	0.4140
-1	-0.414	0.098	-0.999	-0.999	-0.303	0.0000	0.0000	0.1988	0.8201	0.3032	-0.820
0	0.667	-0.198	-0.198	-0.098	-0.198	0.0000	0.3032	0.5342	0.3032	-0.098	0.4140
1	0.667	0.1988	0.667	-0.534	0.1988	0.5342	0.4140	0.3032	0.4140	-0.414	0.5342
2	0.667	0.0984	-0.820	-0.098	0.6677	0.6677	0.5342	0.5342	0.4140	0.5342	-0.667
3	0.303	0.5342	0.0000	0.8201	0.8201	0.8201	0.8201	-0.6677	0.8201	0.8201	-0.414
4	-0.99	0.4140	-0.534	0.8201	0.8201	0.5342	-0.0984	0.1988	0.0000	0.8201	0.4140
5	-0.09	0.0000	-0.198	0.6677	0.8201	0.8201	-0.4140	0.3032	0.1988	0.8201	0.8201

Table (6-2-2) Optimised fuzzy rules (Look-up table) $\lambda = 0.867$

ė \ e	-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-0.99	-0.86	-0.542	-0.32	-0.332	-.28	-0.24	-0.21	0.16	-0.08	0.01
-4	-0.801	-0.5342	-0.303	-0.26	-0.180	-0.2	-0.13	-0.17	-0.081	0.020	0.051
-3	0.6	-0.486	-0.28	-0.26	-0.23	-0.212	-0.16	-0.12	-0.010	0.14	0.098
-2	-0.52	-0.44	-0.41	-0.303	-0.103	-0.1	-0.06	0.02	0.23	0.22	0.24
-1	-0.40	-0.42	-0.36	-0.289	-0.225	0.00	0.00	0.108	0.22	0.30	0.20
0	-0.36	-0.31	-0.21	-0.098	-0.08	0.00	0.004	0.15	0.20	0.34	0.420
1	-0.33	-0.20	-0.17	-0.152	0.058	0.21	0.24	0.30	0.41	0.45	0.52
2	-0.27	-0.19	-0.122	-0.034	0.14	0.17	0.21	0.32	0.40	0.54	0.57
3	0.132	-0.18	0.00	0.201	0.28	0.296	0.32	0.28	0.38	0.42	0.60
4	-0.02	-0.14	0.009	0.022	0.31	0.32	0.436	0.52	0.6	0.82	0.840
5	0.001	0.00	0.168	0.377	0.41	0.61	0.64	0.68	0.74	0.83	0.92

Table (6-2-3) Fuzzy rules (Look-up table) for intuitively tuned controller, $\lambda = 0.75$

The performance of the intuitive controller to *R(k)* is provided in Figure (6-2-10). The performance has

Maximum overshoot = 16% for the unit step target

Rise time = 5 sampling intervals for the unit step target

and Steady State Error = 0.786%

It is obvious from the results obtained that optimised fuzzy controller has a better performance.

The fuzzy controller based on GA converged after 1896 generations and took 100 minutes. In comparison the intuitively tuned controller required 3600 minutes and completed after 600 iterations

6.3 Validation Through Experimental Work

In this part all stages of design and tuning of a fuzzy controller are applied to the position control of a DC servo motor. The overall block diagram of the system is shown in Figure (6-3-1).

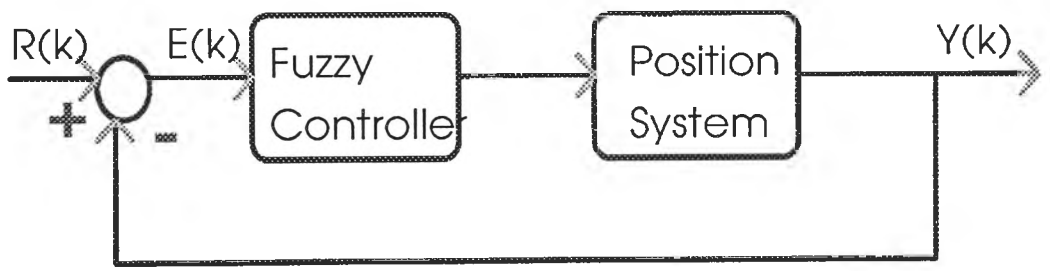


Figure (6-3-1) The control system with linearized position system

6.3.1 System Identification

The system to be controlled is a linearised position control system. The typical transfer function of such a system is given by

$$Y(s)=\frac{K_1 K_m e^{-dS}}{S(1+S\tau_m)}U(S) \qquad (6\text{-}3\text{-}1)$$

The discrete transfer function of this system with Zero-Order Hold is

$$Y(z^{-1}) = \frac{K_h(1-bz^{-1})z^{-d}}{(1-z^{-1})(1-az^{-1})}U(z^{-1}) \qquad (6\text{-}3\text{-}2)$$

For the identification of the parameters of the system an input/output data set of 80 samples was gathered from the system at a sampling interval of 20 ms with an 8 bit A/D converter. The identification package developed based on genetic algorithms and described in section 4.1 of chapter 4 produced the following transfer function for the system.

$$Y(z^{-1}) = \frac{0.26(1-0.36z^{-1})z^{-2}}{(1-z^{-1})(1-0.9268z^{-1})}U(z^{-1}) \qquad (6\text{-}3\text{-}3)$$

6.3.2 Design of the Fuzzy Controller Using the Design Tool

In this experiment the fuzzy rules are optimised to acquire the look-up table directly.
The parameters used in tuning the fuzzy controller are

Probability of crossover:	0.96
Probability of mutation:	0.009
Population:	196
Number of the variables	122

The objective function is assumed to be nearly the same as equation (6-2-5). However in order to decrease the number of multiplications in the search process the $e^2(k)$ in the equation (6-2-5) is changed to $|e(k)|$. In this case the weight is changed from 100 to 10 so that the objective function is changed to (6-2-4). The set points are given by (6-3-5).

$$Obj = \sum_{k=1}^{7}|e(k)| + \sum_{k=8}^{40}10|e(k)| + \sum_{k=41}^{47}|e(k)| + \sum_{k=48}^{80}10|e(k)| + \sum_{k=81}^{87}|e(k)| + \sum_{k=88}^{120}10|e(k)| + \sum_{k=121}^{127}|e(k)| + \sum_{k=128}^{140}10|e(k)| \qquad (6\text{-}3\text{-}4)$$

$$\begin{aligned} R(k) &= 0.6 && 0 < k < 40 \\ R(k) &= 0.0 && 40 < k < 80 \\ R(k) &= 1.0 && 80 < k < 120 \\ R(k) &= 0.0 && 120 < k < 140 \end{aligned} \qquad (6\text{-}3\text{-}5)$$

e / ė	-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-0.875	-1.000	-0.750	-1.000	0.000	-0.625	-0.125	-0.375	0.250	-0.250	-1.000
-4	-0.125	0.250	-0.625	0.750	0.250	-0.750	0.375	0.750	0.375	-1.000	-0.250
-3	-0.250	-0.375	-0.375	0.000	0.500	-0.625	0.125	0.500	0.250	-0.875	-0.500
-2	0.375	-1.000	0.500	0.125	0.875	-0.125	0.875	0.250	0.000	-0.125	0.875
-1	-0.125	0.125	-1.000	0.000	-0.250	-0.125	0.375	0.750	0.000	0.750	-0.625
0	-0.500	0.375	-0.250	-0.875	-0.375	0.000	0.375	0.125	-0.500	0.500	-0.375
1	-1.000	0.000	-0.375	0.125	-0.250	0.125	0.000	0.000	0.500	0.250	-0.125
2	-1.000	0.125	0.375	-0.250	-0.625	0.000	0.500	0.375	0.375	-0.500	-0.875
3	-0.625	-0.250	0.625	-1.000	-0.875	0.500	0.625	0.125	-1.000	-0.375	0.125
4	-0.875	-0.375	0.375	-0.500	0.250	0.500	0.875	-0.875	0.000	0.125	0.375
5	0.875	-0.875	0.875	0.875	-0.500	0.625	0.875	-0.125	0.000	-0.125	0.875

λ=0.626

Table (6-3-1) Optimised fuzzy rules (Look-up table)

The search process converged after 1246 generations. The derived optimised look-up table is shown in Table (6-3-1). At the same time a gain factor λ has been optimised and is equal to 0.626 and the optimised objective function is 32.92.

$$U_{real}(k) = \lambda * U_{fuzzy}(k) \qquad 0 < \lambda < 1.0 \qquad (6\text{-}3\text{-}6)$$

Where $U_{real}(k)$ is the real control action and $U_{fuzzy}(k)$ is the crisp control value of the fuzzy controller calculated based on the look-up table.

The computer simulation of the system response, the reference input and control action optimised are illustrated in figure (6-3-2). This result is obtained from the optimisation process based on the mathematical model of the real system identified in the section 6.3.1 of this chapter and is only a theoretical result.

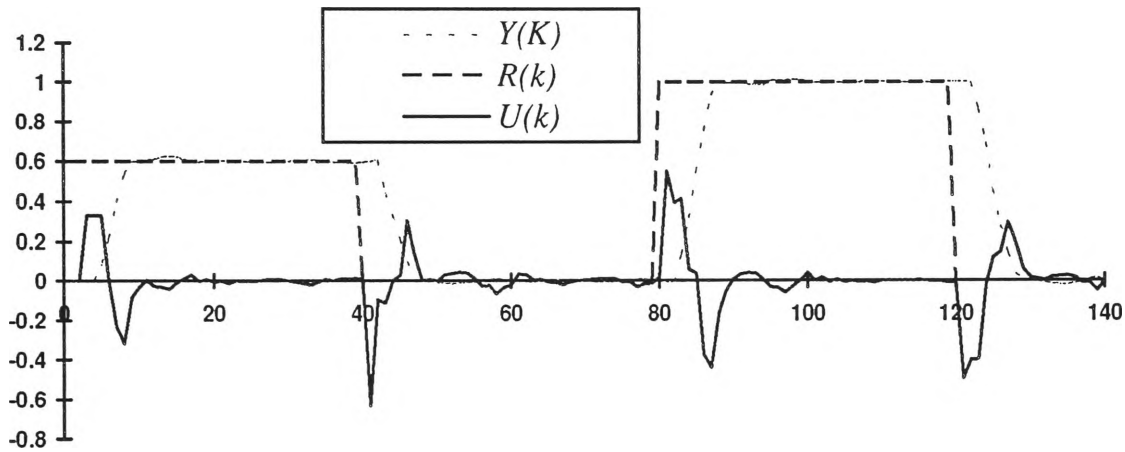


Figure (6-3-2) Computer simulation of system response and control action of the fuzzy controller

Where *U(k)* is the control action, *Y(k)* is the response of the fuzzy control system and *R(k)* is the reference input function. The value of λ and objective function used were 0.626 and 32.92 respectively. The system has produced a maximum overshoot of 1.2% and rise time of 6 sampling intervals.

The optimised fuzzy controller was then applied to the physical system using the optimised look up Table (6-3-1). The response of the system is shown in the figure (6-3-3). The response has a maximum overshoot of 4.26% and a rise time of 6 sampling intervals. The objective function is 35.62.

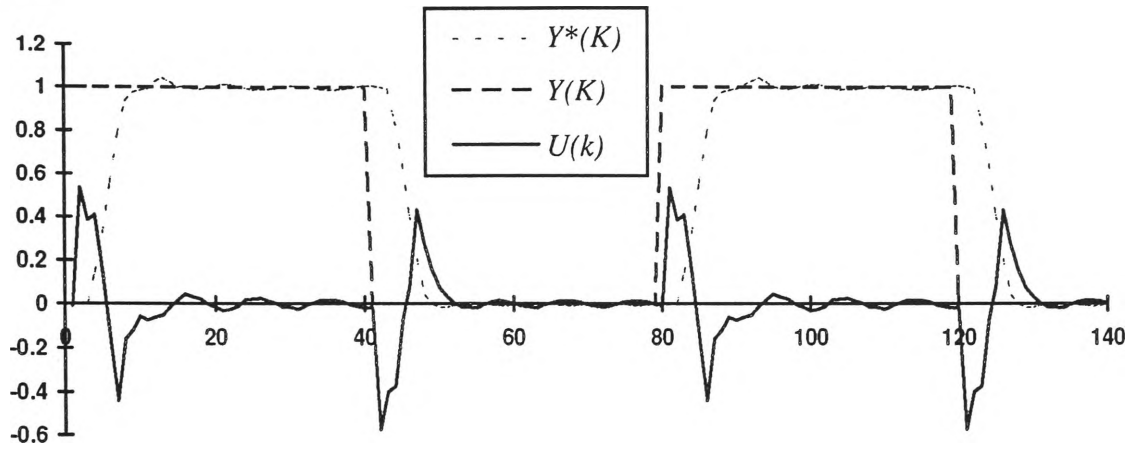


Figure (6-3-3) The system response and the control action of the fuzzy controller in real time.

6.4 Comparison of results with a Conventional PD Controller

A PD controller was applied to the DC motor to compare its performance with the fuzzy controller. The results including the control signal, system response and the set points for computer simulation and real-time control are shown in Figures (6-4-1) and (6-4-2) respectively. This PD controller was optimised by GAs based on the mathematical model identified in section 6.3.1. The parameters of the optimised PD controller had a proportional gain of 0.3672 and a derivative gain of 1.7344.

The following parameters were used in optimising the PD controller:

Number of variable = 2, Proportional gain and Derivative gain

Population size = 20; No. of Generations in the search process = 689;

Probability of mutation = 0.01; Probability of crossover = 0.96.

Objective function:

$$Obj = \sum_{k=1}^{7}|e(k)| + \sum_{k=8}^{40}10|e(k)| + \sum_{k=41}^{47}|e(k)| + \sum_{k=48}^{80}10|e(k)| + \sum_{k=81}^{87}|e(k)| + \sum_{k=88}^{120}10|e(k)| + \sum_{k=121}^{127}|e(k)| + \sum_{k=128}^{140}10|e(k)|$$

(6-4-1)

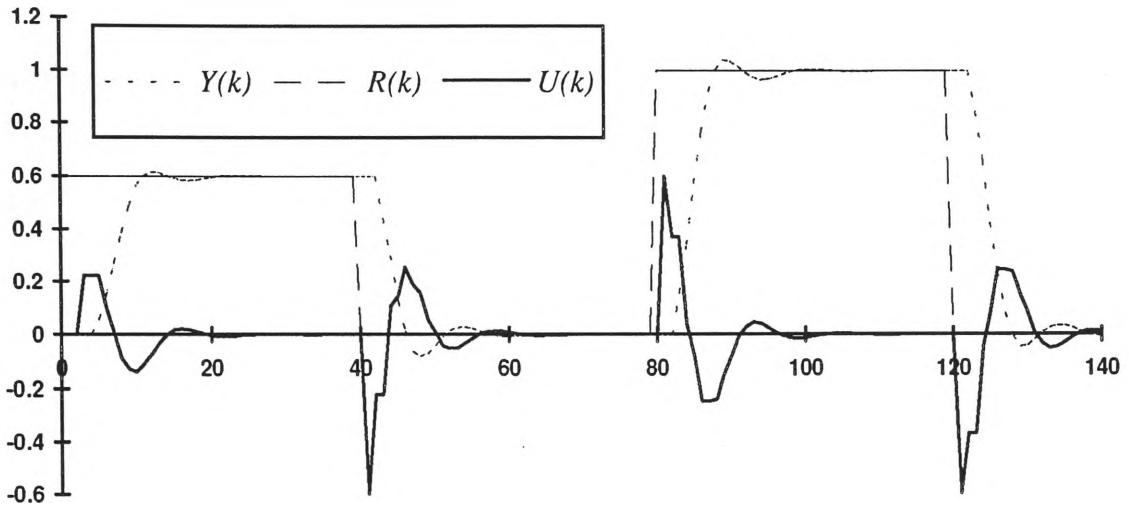


Figure (6-4-1) Computer simulation of system response and control action using conventional PD controller optimised by GAs

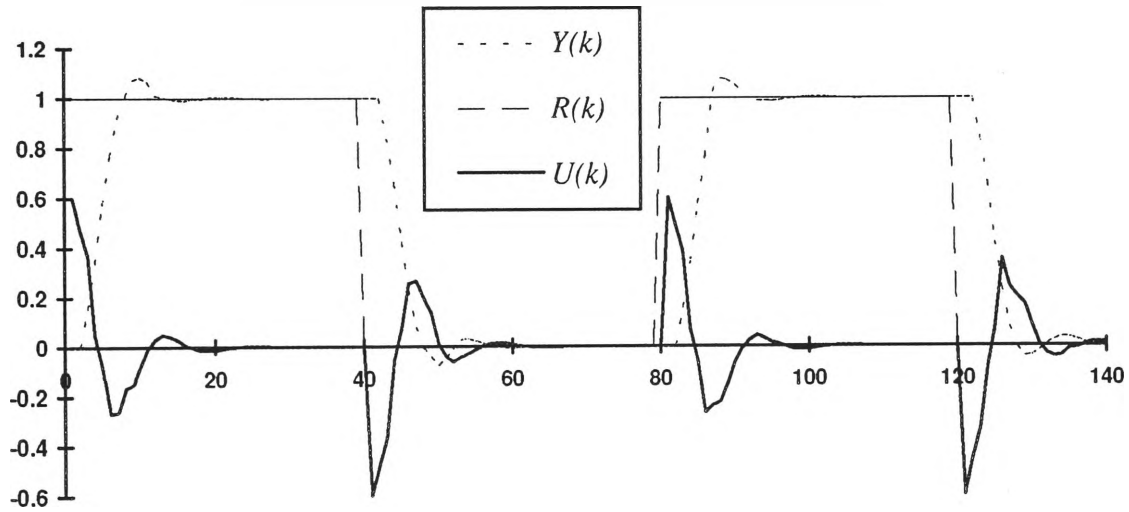


Figure (6-4-2) Real-time system response and control action of PD controller optimised by GAs

The PD control action was defined by

$$U(k) = Kp*error[k] + Kd*(error[k]-error[k-1]) \qquad (6\text{-}4\text{-}2)$$

A detailed comparison of the performance of the real-time fuzzy controller, PD regulator and computer simulations is provided in Table (6-4-1)

	Optimised Results			Real time Results		
	Objective Function	**Overshoot**	**Rise Time**	**Objective Function**	**Overshoot**	**Rise Time**
optimised Fuzzy Controller	32.92	1.2%	6.4*20 ms	35.62	4.26%	6.8*20 ms
PD Controller	47.46	4.2%	5.8*20 ms	62.46	8.1%	5.1*20 ms

Table (6-4-1) Comparison of experimental results

It is obvious from this table that the fuzzy controller designed by the design tool has a better control performance in terms of rise time, overshoot and objective function when compared with the PD controller optimised by the genetic algorithm.

6.5 Summary

In this chapter the developed methodologies were applied to a number of systems through computer simulation and real-time control. The results demonstrate that the design tool developed in this work based on GAs for tuning of fuzzy controllers can successfully produce a satisfactory controller. The methods developed in this technique has also proved to be quite robust in response to step inputs with variable gains. This is due to the usage of multi-level set points during the optimisation process. Finally, an optimised conventional PD controller was applied to the same real time system as the fuzzy controller. Results obtained indicate that the fuzzy controller can provide a better performance than PD controller.

Chapter 7 Conclusion

7.1 Introduction

The outcome of this work can be briefly described as a design tool which based on a small set of input/output data from a system produces an optimised fuzzy controller. In spite of a number of methodologies developed during the work, its major findings can be described as:

I. Development of a robust, efficient and flexible system identification methodology based on genetic algorithms
II. Development of a tuning tool for the design of fuzzy controllers based on genetic algorithms

In this chapter the results obtained in the work along the above two directions will be reviewed. The significance and drawbacks of the work will be also highlighted and some conclusions will be drawn. The possible future directions for the continuation of the work will be also proposed.

7.2 System Identification

The system identification method developed in this work has some unique features which can be summarised as:

(a) The conventional system identification methods will not produce an acceptable result unless a large set of input/output data is available. The system identification method

proposed in this work can still produce a satisfactory model for a system with a small set of input/output data.

(b) The method can be applied with equal success to all types of systems including SISO, MIMO and nonlinear systems of Hammerstine type.

(c) The new operations proposed in this work for GAs i.e., 'two-point crossover' and 'adaptive positive factor' for calculating fitness, ensure a fast convergence of the method.

(d) In contrast to conventional system identification methods, this approach can estimate the structure, and time delay of a system in addition to all its parameters.

(e) According to the comparison made with conventional IV (Instrumental Variable) algorithms, the method developed in this work estimates more accurate poles and zeros for the system.

7.3 Fuzzy Controller Tuning

The design methodology developed in this work operates on the system model identified using GA approach. The tuning can be carried out either on the membership functions or fuzzy rules.

In tuning the membership functions, 54 parameters derived from the membership functions are optimised. For fuzzy rules the number of parameters tuned is 122. In both methods GAs play a major role. Computer simulation and experimental work carried out in the work shows that both optimisation methods operate well in practice. The controllers tuned based on them outperforms other controllers.

Nonlinear controllers usually do not perform well if they operate at a reference point different from what they have been designed for. In this work an objective function with

multi-reference points was employed during design to overcome this problem. The computer simulation shows that such approach guarantees almost the same control performance for varying reference points.

It was also observed that the tuning process takes shorter than other design methods such as trial and error. The speed of convergence depends greatly on the values selected for probability of crossover and mutation. Inappropriate values can delay the convergence and hence increase the tuning time significantly. This is a major shortcoming of this approach. At present no systematic method for choosing these probabilities exists.

The developed method is not also suitable for on-line design of a controller due to the length of the tuning process.

7.4 Future Work

This work is perhaps the first step towards a versatile and user-friendly automatic design tool for fuzzy controllers. Further work along the following directions can improve the useability and performance of the developed methodology:

(a) More study of the developed tool by applying it to various real-time systems and observing the performance is necessary before the system can be recommended for industrial applications.

(b) The search time of the genetic algorithms is quite long for on-line applications at this stage. Further work is required to reduce this time by

- Implementing the search process in parallel using transputer or similar parallel computing architectures.
- Reducing the number of parameters and rules to be optimised.
- Increasing the speed of the search process by introducing novel short cuts.

(c) A windows based user-interface for the design tool will integrate all the routines developed in this work and will produce a more user friendly system. This can be done either in Visual Basic or Visual C++.

Bibliography

(1) E. H. Mamdani, "Applications of fuzzy algorithms for control of simple dynamic plant," *Proc.IEE* 121(12): 1585-1588(1974).

(2) . L. Ed. Davis, "Handbook of Genetic Algorithms", Van Nostrand, Reinhold, 1991.

(3) J. Kim, Y. Moon and B. P. Zeigler, "Designing fuzzy net controllers using GA optimisation", *Proceedings IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, P. 83-88, March 1994, AZ, USA.

(4) C. Karr, "Applying Genetics to Fuzzy Logic", *AI Expert*, Vol.6, No. 2, pp26-33, 1991.

(5) T. Takahama, S. Miyamoto, H. Ogura, M. Nakamura, "Acquisition of fuzzy control rules by Genetic Algorithms", *8th Fuzzy System Symposium*, pp241-244, 1992.

(6) Michael A. and Hideyuki. Takagi. "Integrating Design Stages of Fuzzy Systems using Genetic Algorithms", *Second IEEE Int. Conf. on Fuzzy Systems*, pp612-617, 1993, San Francisco. USA.

(7) John A. Bernard, "Use of a rule-based system for process control", *IEEE Control Systems Magazine*, pp. 3-33, October, 1988.

(8) X. Hu, B. Chung, "Heuristic leaning package for fuzzy control", *Proceedings ANZIIS-93*, Perth, Western Australia.

(9) Thomas Heckenthaler and Sebastian Engell, "Approximately Time-Optimal Fuzzy Control of a Two-Tank System". *IEEE Control System* 1994.

(10) M. Balazinski, E. Czogala and T. Sadowski, "Control of Mental-Cutting Process Using Neural Fuzzy Controller", *Second IEEE Int. Conf. on Fuzzy Systems*, pp161-166, 1993, San Francisco. USA.

(11) M. Sugeno (ED), "Industrial Applications of Fuzzy Control", Amsterdam, North Holland, PP 125-138, 1985.

(12) P. J. King and E. H. Mamdani, "The application of Fuzzy Control Systems to Industrial Processes", *Automatica*, 13(3): 235-242, 1977

(13) Van Der Rhee Floor, Hans R. Van Nauta Lemke, and Jaap G. Dijkman, "Knowledge Based Fuzzy Control of Systems". *IEEE Trans on Automatic Control*, 35(2): 148-155,1990.

(14) Labib Sultan and Talib Janabi, "An Advanced System for the Generation and Tuning of Fuzzy Controller Knowledge-Base", IAS ' 93, *Conf. Record of the 1993 IEEE Industrial Applications Conf. Twenty-Eighth IAS Annual Meeting*, PP. 2205-2209, Vol. 3, Oct. 1993.

(15) L. Zheng, "A practical Computer-aided tuning technique for fuzzy control", *Second IEEE Int Conf on Fuzzy Systems*, March, 1993, Vol 2.

(16) L. Zheng, "Human Operation Emulation Technique Using Fuzzy Control", *Proceedings of the instrument society of america, International technical conference*, (ISA/92). 1992

(17) L. Zheng, "A practical guide to tune of Proportional and Integral (PI) like fuzzy controllers", *Proceedings of IEEE Int. Conf on fuzzy systems*, pp. 633-640, 1992.

(18) A. Boscolo, F. Drius, "Computer aided tuning tool for fuzzy controllers", *Second IEEE Int Conf on Fuzzy Systems*, March, 1993, P. 291-296, Vol 1.

(19) Luis M. M. Custodio, Joao J. S. Sentieiro, and Carlos F. G. Bispo, "Production Planning and Scheduling Using a Fuzzy Decision System," *IEEE Trans. on Robotics and Automation*, vol. 10, NO. 2, April 1994.

(20) D. M. Etter, M. J. Hicks, and K. H. Cho, " Recursive adaptive filter design using an adaptive genetic algorithms," *Proc. IEEE int . Conf. Acoustics, Speech, Signal Processing*, vol. 2, pp. 635-638,1982.

(21) H. D. Joos, M. Schlothane, and G. Grœbel, "Multi-Objective Design of Controllers with Fuzzy Logic Using the Control Engineering Environment ANDECS", c1994 *IEEE*

(22) T. Hojo, T. Terano and S. Masui, "Design of Quasi-Optimal Fuzzy Controller by Fuzzy Dynamic Programming", *Second IEEE Int Conf on Fuzzy Systems*, March, 1993, Vol 2.

(23) J. Esogbue, R. E. Bellman, "Contribution to fuzzy dynamic programming", *2nd World Conf. Mathematics at Service of Man*, Las Palmas, 1982

(24) J. Baldwin, B. W. Pilsworth, "Dynamic programming for fuzzy system with fuzzy environment", *J. Math. Analysis and Applications*, , 85, 1-23, 1982

(25) K. Tanaka, M. Sano, "Design of fuzzy controllers based on frequency and Transient Characteristics", *Proc. IEEE Int. Conference on Fuzzy Systems*, pp111-116, 1993

(26) D. Sabharwal and K. S. Rattan, "A proportional-plus-derivative rule-based fuzzy controller", *IEEE Int Conf on Systems Engineering*, P. 229-233, Dayton USA, 1991.

(27) T. Brehm and K. S. Rattan, "Hybride fuzzy logic PID controller", *Proceedings of the IEEE 1993 National Aerospace and Electronics Conf.* P.807-813, 1993.

(28) Wen-Ruey. Hwang. Wiley. E. Thompson, "An improved method for designing fuzzy controllers for position control systems", *Second IEEE Int Conf on Fuzzy Systems*, March, 1993, P. 1361-1364, Vol 2.

(29) W. C. Daugherity "A Neural-Fuzzy Systems for the Protein Folding Problem" *IFIS' 93 . The third Int . Conf. on Industrial Fuzzy Control and Intelligent Systems*. PP. 47-49, 1993, Houston. USA

(30) E. Khan and P. Venkatapuram, "Neufuz: Neural network based fuzzy logic design algorithm", *Second IEEE Int Conf. on Fuzzy Systems*, March, 1993, P. 647-654, Vol 1.

(31) Yung-Yaw. Chen, "Rules Extraction for Fuzzy Control Systems", 1989 *IEEE Int. Conf. on System, Man and Cyb.*, pp. 526-527

(32) L. Fortuna, S. Graziani, S. Baglio, and G. Nunnari, "Improvements in fuzzy controller design", *IEEE Int Conf on Systems Engineering*, P. 237-240, Dayton USA, Aug. 1991.

(33) Cheng-Liang. Chen and Wen-Chih. Chen, "Fuzzy Controller Design by Using Neural Network Techniques", *IEEE Trans. on Fuzzy Systems*, pp. 235-244, Vol. 2. NO. 3. Aug. 1994.

(34) H. Takagi, I. Hayashi, "NN-driven Fuzzy Reasoning", *Int. J. Approximate Reasoning*, Vol 5, No. 3, pp116-132, 1991.

(35) H. Nomura, I. Hayashi, N. Wakami, " A self-tuning method of fuzzy control by decent method", *4th IFSA Congress*, Vol. Engineering, pp155-158, July 1991.

(36) Daihee Park, Abraham Kandel, and Gideon Langholz, "Genetic-based new fuzzy reasoning models with application to fuzzy control", *IEEE Trans on System, Man and Cybernetics*, vol. 24. NO. 1, January 1994.

(37) H. Ishibuchi, K. Nozaki and N. Yamamoto, "Selecting fuzzy rules by genetic algorithm for classification problems", *Second IEEE Int Conf. on Fuzzy Systems*, March, 1993, P. 1119-1124, Vol 2.

(38) S. Isaka and A. V. Sebald, "An optimisation approach for fuzzy controller design, controllers", *IEEE Trans on System, Man and Cybernetics*, vol. 22. NO. 6, pp. 1469-1473, Nov/Dec 1992.

(39) S. Chiu, S. Chand, D. Moore and A Chaudhary, "Fuzzy Logic for Control of Roll and Moment for a Flexible" *Wing Aircraft, IEEE Control Systems Magazine*, pp. 42-48, June 1992.

(40) L. A. Zadeh, "Fuzzy algorithms", *Information and Control* 12:94-102 (1968).

(41) L. Zadeh, "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes", *IEEE Trans. Sys., Man, Cybren.*, vol. smc-33, pp. 28-44, 1973.

(42) B. Kosko, "Neural Networks and Fuzzy System: A dynamically systems approach to machine intelligence", Prentice -Hall, 1992.

(43) T. Takagi and M. Sugeno. "Derivation of fuzzy control rules from human operator's control action". *In IFAC Syposium on Fuzzy Information, Knowledge representation and Decision A nalysis*, pp. 55-60, Marseille, France

(44) D. Goldberg., "Genetic Algorithms in Search , Optimization, and Machine Learning", Addison--Wesley, 1989.

(45) Z. Zhang and F. Naghdy, "Identification of Non-linear System Using Genetic Algorithms", CONTROL 95 Procedings, 1995, Australia.

(46) Ellot, H. and Wolovich, W. A, " Parameterization Issues in Multivariable Adaptive Control," *Automatica*, 20(1984), pp. 533-545.

(47) Z. Zhang and F. Naghdy, "System Identification Using Genetic Algorithms", EEC Procedings 1994 Sydney.

(48) Y. Davidor, H. Schwefel, "An introduction to adaptive optimisation algorithms based on principles of natural evolution", *Dynamic, Genetic and Chaotic programming*, John Wiley & Sons, INC.

(49) Bickles B.P., Fredrick E. Petry," Genetic Algorithms", *IEEE Computer Society* Press, 1992.

(50) K. Kristinsson and G. A. Dumont, "System Identification and Control Using Genetic Algorithms". *IEEE Trans Syst. Man and Cyber* ,VOL .22, NO.5, 1992.

(51) Zibo. Zhang and J. P. Rayner and A. D. Cheetham, "Non Linear System Identification with Fuzzy model Applied to Plasma Proces", SECOND ANNUAL JOINT CONFERENCE ON INFORMATION SCIENCES (JCIS'95), NC, USA.

(52) B. L. Andersen, W. C. Page, and J. R. Mcdonnell, "Multi-output system identification using evolutionary programming",

(53) _____"Reduced--Order Parameter Estimation for Continuous System from Sampled Data" *ASME Trans J Dynamic Syst*, 1990.

(54) M. Pawlak,. "On the series expansion approach to the identification of Hammerstein systems", *IEEE Transactions on Automatic Control.* Vol: 36 pp 763-7 , June 1991.

(55) S. Sagara, Z. J. Yank, K. Wada, "Recursive Identification Algorithms for Continuous System in Adaptive Procedure" , *Int. J. Contr*, 1991, vol. 53, n. 2.

(56) H. Akaike , "On Model Structure Testing in System Identification", *Int. J. Contr* . 1978 vol. 27.

(57) C. C. Lee. "Fuzzy logician control systems," parts I and II. *IEEE Trans. on Systems, Man and Cybernetics*, 20(2), March, April 1990.

(58) Tovonic, B., Muskinja, N.; Donlagic, D. "Parameter identification of non linaer systems", *6th Mediterranean Electrotechnical conference. Proceedings.* pp 848-51, Vol .2 1991.

(59) A. Pagni, R. Poluzzi, and G. Rizzotto, "Automatic synthesis and implementation of a fuzzy controller", *Second IEEE Int Conf on Fuzzy Systems*, March, 1993, P. 105-110, Vol 1.

(60) Back. Ad.; Tsoi, A.C. "Non linaer system identification using multilayer perceptrons with locally recurrent synaptic structure". *Neural Networks for Signal Processing, II Proceedings of the IEEE-SP Workshop* pp 444-53, 1992

(61) T. Takagi, M. Sugeno, "Fuzzy identification of system and its application to modeling and control", *IEEE Trans. SMC-15-1*, pp 116-132,1985.

(62) Chow, P.-C.; Chizeck, H. J. "Non linaer recursive identification of electrically stimulated muscle". *Images of the Twenty-First Century. Proceedings of the Annual Int Conf of IEEE Engineering in Medicine and Biology Society* pp 965-6 vol. 3 1989.

(63) H. Kang and G. J. Vachtsevanos, "A intelligent strategy to robot coordination and control", *in Proc. 28th IEEE Conf. Dec. and Contr.*, (Hawaii), pp. 2208-2213, Dec. 1990.

(64) Krzyzak, A. "On identification of non statioary Hammerstein systems by the Fourier series regression estimate", *Proceedings of the 28th IEEE conference on Decision and Control*, pp 626-9 vol. 1, 1989.

(65) Hung-Yuan Chung., York-Yin Sun. "Analysis and Parameter Estimation of Non linear Systems with Hammerstein Model Using Taylor Series Approach", *IEEE Trans. on Circuits and Systems*, pp 1539-41. Vol. 35, No. 12, December 1988

(66) Lang Ziqiang, "A new method for the identification of Hammerstein model", *ACTA AUTOMATICA SINICA* Vol. 19, N0. 1, Jan., 1993.

(67) Narendra, K. S. and Gallman, P. G., "An Iterative Method for the Identification of Non linear System Using a Hammerstein Model", *IEEE Trans.*, AC-14, 1966, pp 546-550.

(68) W. Greblicki and M. Pawlak, "Identification of discrete Hammerstein systems using kernel regression estimates," *IEEE Trans. Automat. Control*, vol. AC-31, pp. 74-77, 1986.

(69) W. Greblicki and M. Pawlak, "Identification Hammerstein system Identification by non parametric regression estimations," *Int. J. Contr.*, vol. 45, pp. 343-354, 1987

(70) W. Greblicki and M. Pawlak, "Recursive non parametric identification of Hammerstein systems," J. Franklin Inst., vol. 326, pp. 461-481, 1989

(71) W. Greblicki and M. Pawlak, "Non parametric identification of Hammerstein systems ," *IEEE Trans. Inform. Thoery*, vol. IT-35, pp. 409-418, 1989

(72) K. S. Narendra and P. G. Gallman, "An iterative method for the identification of non linear systems using the Hammerstein model". *IEEE Trans. Automat. Contr.*, vol. AC-11, pp. 546-550, 1966

(73) S. A. Billings and S. Y. Fakhouri, "Non linear systems identification using the Hammerstein model", *Int. J. Syst. Sci.*, vol. 10, pp. 567-568, 1979.

(74) F. H. I. Chang and R. Luus, "A non iterative method for identification using the Hammerstein model". *IEEE Trans. Automat. Contr.*, vol. AC-16, pp.464-468, 1971

(75) J. C Stapleton and S. C. Baas, "Adaptive noise concellation for a class of non linear, dynamic reference channels". *IEEE Trans. Circuit and Systems*, vol. CAS-32, pp. 143-150, 1985

(76) M. Sugeno, T. Muroushi, T. Mori and Tatematsu. "Fuzzy Algorithmic Control of a Model Car by Oral Instructions". Fuzzy Sets and Systems, 32: 207-219, 1989.

(77) G. Lambert Torres, D. Mukhedkar, "Writing a fuzzy knowledge base", *Int. Conference on System, Man and Cybernestics*, 1990.

(78) F. H. I. Chang and R. Luus, "A noniterative method for identification using the Hammerstein model". *IEEE Trans. Automat. Contr.*, vol. AC-16, pp.464-468, 1971

(79) J. C Stapleton and S. C. Baas, "Adaptive noise concellation for a class of non linear, dynamic reference channels", *IEEE Trans. Circuit and Systems*, vol. CAS-32, pp. 143-150, 1985

(80) H. Kang and G. J. Vachtsevanos, "Model reference fuzzy control," *in Proc. 29th IEEEConf. Dec. and Contr.*, (Tampa, FL), pp. 751-756, Dec. 1989.

(81) Z. Zhang and F. Naghdy, "Application of Genetic Algorithms to System Identification", IEEE Conference, 1995, Western Australia.

Appendix 1 Source Code of Optimisation of Rules of the Fuzzy Controller

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <process.h>
#include <math.h>
#include <graphics.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
#include <complex.h>
        /* ************************************************************** */
        /*              Fuzzy controller optimization software            */
        /*              This program optimize fuzzy rules directly        */
        /*              and the number of membership functions is 11.     */
        /*      The file of Output (Y* and Y) is named as resp.dat        */
        /*      and the fuzzy rule table file is named as membe.dat        */
        /*              Initial data 268,122,8000,0.968,0.008.            */
        /*              Remainder Stochastic Sampling Without Replacement */
        /*      The system to be controlled is the following model        */
        /*      Y(K)=a1*Y(K-1)+a2*Y(K-2)+K*[U(K-d-1)+b1*U(K-d-2)]         */
        /**************************************************************** */
struct bitf0
        {
         unsigned chro1:1;
         unsigned chro2:1;
         unsigned chro3:1;
         unsigned chro4:1;
         unsigned chro5:1;
         unsigned chro6:1;
         unsigned chro7:1;
         unsigned chro8:1;
         unsigned chro9:1;
         unsigned chro10:1;
         unsigned chro11:1;
         unsigned chro12:1;
         unsigned chro13:1;
         unsigned chro14:1;
         unsigned chro15:1;
         unsigned chro16:1;
        };
```

```
struct bitf
    {
        struct  bitf0 chrom[44];
        float  fitness;
        unsigned  parent1,parent2;
        };
        struct bitf huge newpop[200];
        struct bitf huge oldpop[200];
        float  cmax=510000000000.0;int  xdt[16][16],xd[180],number;
  float far po[148];
  unsigned flag1,lchrom,gen1,gen,maxgen,nmutation=0,ncross=0,emate,popsize,maxj;
  float avg,max,min,pcross,pmutation,minobjfit,sumfitness,py[148];
  float ppy[148],pmutation0,resultc; float a[14],b[14],k[14];
  void   samplecopy();
  void initreport();
  void statistics(unsigned popsize);
  float objfunc(unsigned j);
  void decode(struct bitf pop,unsigned lchrom);
  void generation(void);//popsize,sumfitness,oldpop
  unsigned mutation(unsigned allele,float pmutation);
  unsigned flip(float tion);
  unsigned crossover(struct bitf huge *parent1,struct bitf huge *parent2,
  struct bitf huge *child1,struct bitf huge *child2,unsigned lchrom,float pcross,float pmutation);
  unsigned crossover1(struct bitf huge *parent1,struct bitf huge *parent2,
  struct bitf huge *child1,struct bitf huge *child2,unsigned lchrom,float pcross,float pmutation);
  unsigned select(unsigned popsize,float sumfitness);
  void   initdata();
  void   initialize();
  void   initpop();
  void   objp();
  void   chang();
  float  inputm(float at,float bt,float kt,float xt);
  void   memb(void);//popsize,sumfitness,oldpop

main(void) // simple genetic algorithms //
{
        flag1=0; resultc=288.8;number=140; gen1=0;
        unsigned k,k1;
a100:   for(k=0;k<=number;k++)
                {   ppy[k]=0.0;py[k]=0;}
    printf("\tYYYYYYYYYYYYYYYYYYYYYYYYY\n");
    memb();//popsize,sumfitness,oldpop
    objp();
a300: gen=0;
    initialize();
    flag1=0;chang();
    pmutation=pmutation0;
    for (gen=0;gen<=maxgen;gen++)
     {
        if(gen>260&&resultc>280.0)
         {   gen=1;
             pmutation=pmutation*.9;
```

```
                initpop();statistics(popsize);chang();
        }
        if(gen1==268)
                {
                        pmutation=pmutation*.98;
                        if(pmutation<=0.006)pmutation=0.006;gen1=1;
                }
        if(flag1==2)    goto a200;
        if(flag1==1)    goto a100;
        if(flag1==3)    goto a300;
        generation();
        statistics(popsize);
        printf("**sumfitness=%6.6f*******gen==%3i\n",sumfitness,gen);
        chang();gen1++;
    }
a200:   printf("*sumfitness=%2.9f******flag1=%i\n",sumfitness,flag1); return 0;
}
void chang()
{     int k,k1,k2;
        for(k=1;k<=popsize;k=k+1)
        {
         k2=1;k1=1;
         while(k2<=lchrom)
          {
           oldpop[k].chrom[k1].chro1=newpop[k].chrom[k1].chro1;
           oldpop[k].chrom[k1].chro2=newpop[k].chrom[k1].chro2;
           oldpop[k].chrom[k1].chro3=newpop[k].chrom[k1].chro3;
           oldpop[k].chrom[k1].chro4=newpop[k].chrom[k1].chro4;k2++;
           oldpop[k].chrom[k1].chro5=newpop[k].chrom[k1].chro5;
           oldpop[k].chrom[k1].chro6=newpop[k].chrom[k1].chro6;
           oldpop[k].chrom[k1].chro7=newpop[k].chrom[k1].chro7;
           oldpop[k].chrom[k1].chro8=newpop[k].chrom[k1].chro8;k2++;
           oldpop[k].chrom[k1].chro9=newpop[k].chrom[k1].chro9;
           oldpop[k].chrom[k1].chro10=newpop[k].chrom[k1].chro10;
           oldpop[k].chrom[k1].chro11=newpop[k].chrom[k1].chro11;
           oldpop[k].chrom[k1].chro12=newpop[k].chrom[k1].chro12;k2++;
           oldpop[k].chrom[k1].chro13=newpop[k].chrom[k1].chro13;
           oldpop[k].chrom[k1].chro14=newpop[k].chrom[k1].chro14;
           oldpop[k].chrom[k1].chro15=newpop[k].chrom[k1].chro15;
           oldpop[k].chrom[k1].chro16=newpop[k].chrom[k1].chro16;k2++;
           k1++;
          }
         oldpop[k].fitness=newpop[k].fitness;
         oldpop[k].parent1=newpop[k].parent1;
         oldpop[k].parent2=newpop[k].parent2;
//       for(k2=1;k2<=lchrom/10;k2++)oldpop[k].x[k2]=newpop[k].x[k2];
        }
}
void initreport(void)
{
        printf("\t*****SGA PARAMETERS*****\n");
        printf("\t population size          = %i\n",popsize);
```

```
        printf("\t chromosome length        = %i\n",lchrom);
        printf("\t max number of generation  = %i\n",maxgen);
        printf("\t crossover probability     = %f\n",pcross);
        printf("\t mutation probability      = %f\n",pmutation0);
        printf("*****initial generation statistics*****\n\n");
        printf("\t initial population max fitness = %f\n",max);
        printf("\t initial population avg fitness = %f\n",avg);
        printf("\t initial population min fitness = %f\n",min);
        printf("\t initial population sum fitness = %f\n",sumfitness);
}
void statistics(unsigned popsize)
{
        unsigned j;
        sumfitness=(float)newpop[1].fitness;
        min=(float)newpop[1].fitness;
        max=(float)newpop[1].fitness;
        for(j=2;j<=popsize;j=j+1)
        {
                sumfitness=sumfitness+newpop[j].fitness;
                //              printf("sumfitness==%9.1f,gen=%2i\n",sumfitness,gen);
                  //            printf("newpop[%2i]==%9.1f\n",j,newpop[j].fitness);
                if (newpop[j].fitness>=max)
                  {
                        max=newpop[j].fitness;
                   }
                if (newpop[j].fitness<=min)
                        min=newpop[j].fitness;
        }
//      printf("sumfitness==%9.1f\n",sumfitness);
        avg=sumfitness/(float)popsize;
}
float  objfunc(unsigned j)
{
 int ky,iy,xxa[8],xxb[8],kb;FILE *p1;
 float u1[148],xa[4],xb[4],error[148],muc,derror,memv[15],memc[15],
     obj,ue,ude,uinte,mucv[16];
 if(gen==0&&j==1)
 {
 for(iy=0;iy<=10;iy++)
 printf("a[%i]=%2.4f\tb[%i]=%2.4f\tk[%i]=%2.4f\n",iy,a[iy],iy,b[iy],iy,k[iy]);
 }
 decode(newpop[j],lchrom);
 uinte=(float)(xd[122]+16*xd[123])/256.0;//ude=(float)(xd[124]+16*xd[125])/256.0;
obj=0.0;ppy[1]=0.0;ppy[2]=0.0;error[1]=py[1]-ppy[1];error[2]=py[2]-ppy[2];
u1[0]=0.0;u1[1]=0.0;u1[2]=0.0;
for(kb=2;kb<=number-1;kb++)
{
 error[kb+1]=py[kb]-ppy[kb];derror=error[kb+1]-error[kb];
                    if(error[kb+1]>1.19)            {xxa[1]=11;xxa[2]=11;xa[1]=1.0;xa[2]=1.0;goto
oude;}//ude=deriviate;uinte=integer
 if(error[kb+1]<-1.19){xxa[1]=1; xxa[2]=1;xa[1]=1.0;xa[2]=1.0;goto oude;}
 for(iy=0;iy<=10;iy++)//error membershipe function computing
```

```
  {
  if(error[kb+1]>=a[iy]&&error[kb+1]<=b[iy])
   {
   memv[iy+1]=inputm(a[iy],b[iy],k[iy],error[kb+1]);
   if(memv[iy+1]<0.0)printf("memv[%i]=%f, a[%i]=%f, b[%i]=%f\n",iy,memv[iy],iy,a[iy],iy,b[iy]);
   }
   else memv[iy+1]=0.0;//error output value calculating
  }
  memv[12]=0.0;
/*begin to compute the ue */
 ky=1;
 while(memv[ky]<=0.0&&ky<=11)ky++;//to find the first point that is not equal to 0;
 if(ky>11) {printf("error!!! ky>11");return 0;}
 memc[1]=0.0;
 if(memv[ky]>0.0)//!!! calculate the area value !!!
 {
  xxa[1]=ky;xa[1]=memv[ky];
  if(ky+1<=11)
  {
   if(memv[ky+1]>0.0) {xa[2]=memv[ky+1];xxa[2]=ky+1;}
   else { xxa[2]=ky;xa[4]=memv[ky]; }
  } }
oude:                 if(derror>1.19)        {xxb[1]=11;        xb[1]=1.0;xxb[2]=11;xb[2]=1.0;goto
ende;}//ude=deriviate;uinte=integer
     if(derror<-1.19){xxb[1]=1; xb[1]=1.0;xxb[2]=1;xb[2]=1.0; goto ende;}
  for(iy=0;iy<=10;iy++)//error membershipe function computing
   {
   if(derror>=a[iy]&&derror<=b[iy])
    {
         memv[iy+1]=inputm(a[iy],b[iy],k[iy],derror);
         if(memv[iy+1]<0.0)printf("dmemv[%i]=%f,                                    a[%i]=%f,
b[%i]=%f\n",iy,memv[iy],iy,a[iy],iy,b[iy]);
    }
   else memv[iy+1]=0.0;//error output value calculating
  }
  memv[12]=0.0;
/*begin to compute the ue */
 ky=1;
 while(memv[ky]<=0.0&&ky<=11)ky++;//to find the first point that is not equal to 0;
          if(ky>11)          {printf("derror!!!          ky>10                    derror=%f,
memv[6]=%f\n",derror,memv[6]);for(iy=0;iy<=12;iy++)printf("memv[%i]=%f\t          a[%i]=%f\t
b[%i]=%f\n",iy,memv[iy],iy,a[iy],iy,b[iy]);return 0;}
 memc[2]=0.0;
 if(memv[ky]>0.0)//!!! calculate the area value !!!
 {
  xxb[1]=ky;xb[1]=memv[ky];
  if(ky+1<=11)
  {
   if(memv[ky+1]>0.0){ xb[2]=memv[ky+1]; xxb[2]=ky+1;  }
   else {xb[2]=memc[ky];xb[2]=ky;}
  }
 }
```

```
ende:    muc=0.0; memc[14]=0.0;
// if(fabs(error[kb+1])<0.02){u1[kb+1]=u1[kb];goto aoude;}
        //uinte=(xxc[1]+xxc[2]*fabs(error[kb+1]))*interror;
    if(xa[1]>xb[1]) memc[5]=xb[1];else memc[5]=xa[1];memc[9]=tan(((float)xdt[xxa[1]][xxb[1]]-
8)*0.18375)/10.0;
    if(xa[1]>xb[2]) memc[6]=xb[2];else memc[6]=xa[1];memc[10]=tan(((float)xdt[xxa[1]][xxb[2]]-
8)*0.18375)/10.0;
    if(xa[2]>xb[1]) memc[7]=xb[1];else memc[7]=xa[2];memc[11]=tan(((float)xdt[xxa[2]][xxb[1]]-
8)*0.18375)/10.0;
    if(xa[2]>xb[2]) memc[8]=xb[2];else memc[8]=xa[2];memc[12]=tan(((float)xdt[xxa[2]][xxb[2]]-
8)*0.18375)/10.0;
 for(iy=1;iy<=4;iy++)
 {
        muc=muc+memc[iy+4];
        memc[14]=memc[14]+memc[iy+8]*memc[iy+4];
 }
 if(muc==0.0) {printf("muc=0.0,\t%f\t",muc);muc=1.0;}
// u1[kb+1]=uinte*memc[14]/muc+u1[kb];
 u1[kb+1]=uinte*memc[14]/muc;
// if(fabs(error[kb+1])<0.2)
// u1[kb+1]=ude*u1[kb]+u1[kb+1];
 if(u1[kb+1]>1.0) u1[kb+1]=1.0;
 if(u1[kb+1]<-1.0) u1[kb+1]=-1.0;
aoude:ppy[kb+1]=1.9268*ppy[kb]-0.9268*ppy[kb-1]+0.660*(u1[kb-1]-0.360*u1[kb-2]);
    if((fabs(ppy[kb+1])-cmax)>=0) goto ag10;
// printf("u1[%2i]=%3.3f\tuinte=%f \n",kb,u1[kb],(xxc[1]+xxc[2]*error[kb+1])*interror/100.0);
}
  for(kb=1;kb<=number-1;kb++)
    {
        if(kb>=1&&kb<7){obj=obj+fabs(error[kb+1]);goto ag8;}
        if(kb>=7&&kb<20){obj=obj+10.0*fabs(error[kb+1]);goto ag8;}
        if(kb>=20&&kb<26){obj=obj+fabs(error[kb+1]);goto ag8;}
        if(kb>=26&&kb<40){obj=obj+10.0*fabs(error[kb+1]);goto ag8;}
        if(kb>40&&kb<46){obj=obj+fabs(error[kb+1]);goto ag8;}
        if(kb>=46&&kb<60){obj=obj+10.0*fabs(error[kb+1]);goto ag8;}
        if(kb>=60&&kb<66){obj=obj+fabs(error[kb+1]);goto ag8;}
        if(kb>=66&&kb<80){obj=obj+10.0*fabs(error[kb+1]);goto ag8;}
        if(kb>=80&&kb<86){obj=obj+fabs(error[kb+1]);goto ag8;}
        if(kb>=86&&kb<100){obj=obj+10*fabs(error[kb+1]);goto ag8;}
        if(kb>=100&&kb<106){obj=obj+fabs(error[kb+1]);goto ag8;}
        if(kb>=106&&kb<120){obj=obj+10*fabs(error[kb+1]);goto ag8;}
        if(kb>=120&&kb<126){obj=obj+fabs(error[kb+1]);goto ag8;}
        if(kb>=126&&kb<=number)obj=obj+10.0*fabs(error[kb+1]);
ag8:    if((obj-cmax)>=0.0)
          {
ag10:      printf("obj=%f\tj=%i\tppy[%i]=%f\tu1=%f\n",obj,j,kb,ppy[kb+1],u1[kb+1]);obj=cmax;
//
printf("ue=%f\tude=%f\tuinte=%f\n",ue,ude,(xxc[1]+xxc[2]*fabs(error[kb+1]))*interror);goto a10;
          }
    }
a10:  if(j==40||j==200||j==120||j==160||j==260)printf("obj=%f\tj=%i\n",obj,j);return obj;
}
```

```
float inputm(float at,float bt,float kt,float xt)
{
        float result;
        result=-kt*(xt-at)*(xt-bt);
        if(result<=0.0)result=0.0;
        return result;
}
void decode(struct bitf pop,unsigned lchrom)
{
 unsigned k,kx,kc;
 int accum,powerof2;accum=0;powerof2=1;
 kx=1;k=1; //11 bits coding
 while(k<=lchrom)
  {
   if(pop.chrom[kx].chro1==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro2==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro3==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro4==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   xd[k]=accum; k++;
   accum=0;powerof2=1;
   if(pop.chrom[kx].chro5==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro6==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro7==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro8==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   xd[k]=accum;
   accum=0;powerof2=1;k++;
   if(pop.chrom[kx].chro9==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro10==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro11==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro12==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   xd[k]=accum;
   accum=0;powerof2=1;k++;
   if(pop.chrom[kx].chro13==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro14==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro15==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
   if(pop.chrom[kx].chro16==1) {accum=accum+powerof2;powerof2=powerof2*2;}
   else  powerof2=powerof2*2;
```

```
   xd[k]=accum;
   accum=0;powerof2=1;k++;kx++;
  }
  kc=1;
  for(k=1;k<=11;k++)
  {
          for(kx=1;kx<=11;kx++)
          {
           xdt[k][kx]=xd[kc];kc++;
          }
   }
//                                        printf("lchrom==%3.i\n",lchrom);
}
void generation(void)//popsize,sumfitness,oldpop
{
  struct fit
   {
            float   rankfit[266];
            float       ranking[5];
            float       accum[266];
   }half;
  FILE *fp1,*fpu;
  unsigned jx,j,mate1,mate2,jcross,jcross1,halfcount1,halfcount,fitcount=0;
  unsigned jy=1,jrank; maxj=1;
  float objfit,objmin,objhalf,objmax;
  halfcount=popsize/2+gen/120;if(halfcount>280)halfcount=100;
  for(j=1;j<=popsize;j=j+2)
   {
    mate1=select(popsize,sumfitness);
    mate2=select(popsize,sumfitness);
    if(random(100)>=50)
          {
              jcross=crossover(&oldpop[mate1],&oldpop[mate2],
              &newpop[j],&newpop[j+1],lchrom,pcross,pmutation);
          }
    else          {
              jcross1=crossover1(&oldpop[mate1],&oldpop[mate2],
              &newpop[j],&newpop[j+1],lchrom,pcross,pmutation);
          }
    objfit=(float)objfunc(j);
    if(j==1)objhalf=objfit;
    else
       {
        if(objhalf>objfit)
         {
          objhalf=objfit;
          for(jcross=1;jcross<=number-1;jcross++) po[jcross]=ppy[jcross];
         }
       }
    half.accum[j]=objfit;
    if(cmax- objfit<=0)
              half.rankfit[j]=cmax;
```

```
  else {
                fitcount++;
                half.rankfit[j]=objfit;
           }
  newpop[j].parent1=mate1;
  newpop[j].parent2=mate2;
  objfit=(float)objfunc(j+1);
  if(objhalf>objfit)
       {
        objhalf=objfit;
        for(jcross=1;jcross<=number-1;jcross++) po[jcross]=ppy[jcross];
       }
  half.accum[j+1]=objfit;
  if(cmax- objfit<=0)
                half.rankfit[j+1]=cmax;
  else {
                fitcount++;
                half.rankfit[j+1]=objfit;
           }
  newpop[j+1].parent1=mate1;
  newpop[j+1].parent2=mate2;
}
jy=1; maxj=1;
half.ranking[2]=half.accum[1];
for(jrank=1;jrank<=popsize;jrank++)
  {
       if(half.ranking[2]>half.accum[jrank])//get min e*e subscript
         {
                half.ranking[2]=half.accum[jrank];
                maxj=jrank;
         }//find the order from small one to big one
  }
for(jrank=1;jrank<=popsize;jrank++)
  {
       for(jy=jrank+1;jy<=popsize;jy++)
         {
         if((half.accum[jrank]>half.accum[jy]))
         {
           half.ranking[1]=half.accum[jy];
           half.accum[jy]=half.accum[jrank];
           half.accum[jrank]=half.ranking[1];
         }
         }
  }
minobjfit=half.accum[1];
printf("\tresultc=%6.3f\n",resultc);
if(half.accum[1]<resultc)
  {
       objfit=(float)objfunc(maxj);
       resultc=half.accum[1];
       printf("\n\tobjfitmin==%6.3f\tpmutation=%f\n",objfit,pmutation);
       if((fp1=fopen("c:resp.dat","w"))!=NULL)
```

```
    for(j=1;j<=number;j++)
    fprintf(fp1,"\t%3i\t%6.3f\t%6.3f\n",j,po[j],py[j]);
    fprintf(fp1,"\tresultc=%6.3f\tpmutation=%2.6f\tgen=%i\n",resultc,pmutation,gen);
    fclose(fp1);
    printf("\tresultc=%6.3f\tpmutation=%f\tgen=%i\n",resultc,pmutation,gen);
    if((fpu=fopen("c:membe.dat","w"))!=NULL)
    {       decode(newpop[maxj],lchrom);
            for(j=1;j<=11;j++)
            {
                    for(jcross1=1;jcross1<=11;jcross1++)
                    fprintf(fpu,"%2.4f\t",tan(((float)xdt[j][jcross1]-8)*0.18375)/10.0);
                    fprintf(fpu,"\n");
            }
     }
              fprintf(fpu,"%f\n",(float)(xd[122]+16*xd[123])/256.0);
              fprintf(fpu,"%f\n",(float)(xd[124]+16*xd[125])/256.0);
    fclose(fpu);
 }
if(half.accum[1]<=2.18)
{
    objfit=(float)objfunc(maxj);
    flag1=2;
    printf("\n*flag1==%2i\tobjfitmin==%6.3f\tpmutation=%f\n",flag1,objfit,pmutation);
    printf("half.accum[1]==%6.2f****half.ranking[2]==%6.2f\n",half.accum[1],half.ranking[2])
;
    printf(" max newpop[%2i]\n",maxj);
    printf("*****gen=%4i ********accum of errors==%6.3f\n",gen,minobjfit);
    if((fp1=fopen("c:resp.dat","w"))!=NULL)
    for(j=1;j<=number;j++)
    fprintf(fp1,"\t%3i\t%6.3f\t%6.3f\n",j,po[j],py[j]);
    fclose(fp1);
    if((fpu=fopen("c:membe.dat","w"))!=NULL)
    {       decode(newpop[maxj],lchrom);
            for(j=1;j<=11;j++)
            {
                    for(jcross1=1;jcross1<=11;jcross1++)
                    fprintf(fpu,"%2.4f\t",tan(((float)xdt[j][jcross1]-8)*3.14/32.0));
                    fprintf(fpu,"\n");
            }
            fprintf(fpu,"%f\n",(float)(xd[122]+16*xd[123])/64.0);
            fprintf(fpu,"%f\n",(float)(xd[122]+16*xd[123])/256.0);
     }
    fclose(fpu);
}//objmax is called as adaptive fitness scaling factor
if(fitcount>=halfcount) objmax=half.accum[halfcount];
else objmax=half.accum[fitcount-1];
for(j=1;j<=popsize;j++)
{
    if(objmax-half.rankfit[j]<=0) newpop[j].fitness=0.005;
    else newpop[j].fitness=objmax-half.rankfit[j];
}
objfit=(float)objfunc(maxj);
```

```
   printf("**gen=%4i* maxj=%2i* errors==%6.3f\n",gen,maxj,minobjfit);
}
unsigned mutation(unsigned allele,float pmutation)
{
        unsigned muta;float mutate;
        mutate=(float)random(10000)/10000.0;
if(pmutation-mutate>=0.0)
                { nmutation++;      //perform mutation
                 if(allele==1)
                 muta=0;
                 else
                 muta=1;
                }
        else
        muta=allele;
        return muta;
}
unsigned flip(float tion)
{
        float truepm;
        truepm=(float)((float)random(100)/100.0)*(tion+.5);
        if((truepm-0.5)>0)
        return 1;
        else
        return 0;
}
unsigned crossover(struct bitf huge *parent1,struct bitf huge *parent2,struct bitf huge *child1,
struct bitf huge *child2,unsigned lchrom,float pcross,float pmutation)
{
        unsigned jc,jcross;int jx1,jx;
        float crop;
        crop=(float)random(100)/100.0;
        if(pcross-crop>=0.0)//biased wheight wheel
        {
                jcross=random(4*lchrom-1)+1;
                //printf("*********jcross=%4i*********\n",jcross);
                ncross++;
        }
        else jcross=lchrom*4;
   jc=1;jx=1;
   while(jx<=jcross)
    {
        child1->chrom[jc].chro1=mutation(parent1->chrom[jc].chro1,pmutation);
        child2->chrom[jc].chro1=mutation(parent2->chrom[jc].chro1,pmutation);
        jx++;
        child1->chrom[jc].chro2=mutation(parent1->chrom[jc].chro2,pmutation);
        child2->chrom[jc].chro2=mutation(parent2->chrom[jc].chro2,pmutation);
        jx++;
        child1->chrom[jc].chro3=mutation(parent1->chrom[jc].chro3,pmutation);
        child2->chrom[jc].chro3=mutation(parent2->chrom[jc].chro3,pmutation);
        jx++;
        child1->chrom[jc].chro4=mutation(parent1->chrom[jc].chro4,pmutation);
```

```
        child2->chrom[jc].chro4=mutation(parent2->chrom[jc].chro4,pmutation);
        jx++;
        child1->chrom[jc].chro5=mutation(parent1->chrom[jc].chro5,pmutation);
        child2->chrom[jc].chro5=mutation(parent2->chrom[jc].chro5,pmutation);
        jx++;
        child1->chrom[jc].chro6=mutation(parent1->chrom[jc].chro6,pmutation);
        child2->chrom[jc].chro6=mutation(parent2->chrom[jc].chro6,pmutation);
        jx++;
        child1->chrom[jc].chro7=mutation(parent1->chrom[jc].chro7,pmutation);
        child2->chrom[jc].chro7=mutation(parent2->chrom[jc].chro7,pmutation);
        jx++;
        child1->chrom[jc].chro8=mutation(parent1->chrom[jc].chro8,pmutation);
        child2->chrom[jc].chro8=mutation(parent2->chrom[jc].chro8,pmutation);
        jx++;
        child1->chrom[jc].chro9=mutation(parent1->chrom[jc].chro9,pmutation);
        child2->chrom[jc].chro9=mutation(parent2->chrom[jc].chro9,pmutation);
        jx++;
        child1->chrom[jc].chro10=mutation(parent1->chrom[jc].chro10,pmutation);
        child2->chrom[jc].chro10=mutation(parent2->chrom[jc].chro10,pmutation);
        jx++;
        child1->chrom[jc].chro11=mutation(parent1->chrom[jc].chro11,pmutation);
        child2->chrom[jc].chro11=mutation(parent2->chrom[jc].chro11,pmutation);
        jx++;
        child1->chrom[jc].chro12=mutation(parent1->chrom[jc].chro12,pmutation);
        child2->chrom[jc].chro12=mutation(parent2->chrom[jc].chro12,pmutation);
        jx++;
        child1->chrom[jc].chro13=mutation(parent1->chrom[jc].chro13,pmutation);
        child2->chrom[jc].chro13=mutation(parent2->chrom[jc].chro13,pmutation);
        jx++;
        child1->chrom[jc].chro14=mutation(parent1->chrom[jc].chro14,pmutation);
        child2->chrom[jc].chro14=mutation(parent2->chrom[jc].chro14,pmutation);
        jx++;
        child1->chrom[jc].chro15=mutation(parent1->chrom[jc].chro15,pmutation);
        child2->chrom[jc].chro15=mutation(parent2->chrom[jc].chro15,pmutation);
        jx++;
        child1->chrom[jc].chro16=mutation(parent1->chrom[jc].chro16,pmutation);
        child2->chrom[jc].chro16=mutation(parent2->chrom[jc].chro16,pmutation);
        jc++;jx++;
    }
    if(jcross!=lchrom*4)
     {jx1=jx;
     while(jx1>16)jx1=jx1-16;
     switch(jx1)
      {
case 1:{child1->chrom[jc].chro1=mutation(parent2->chrom[jc].chro1,pmutation);
        child2->chrom[jc].chro1=mutation(parent1->chrom[jc].chro1,pmutation);
        jx++;goto s1;}
case 2:{s1: child1->chrom[jc].chro2=mutation(parent2->chrom[jc].chro2,pmutation);
        child2->chrom[jc].chro2=mutation(parent1->chrom[jc].chro2,pmutation);
        jx++;goto s2;}
case 3:{s2: child1->chrom[jc].chro3=mutation(parent2->chrom[jc].chro3,pmutation);
        child2->chrom[jc].chro3=mutation(parent1->chrom[jc].chro3,pmutation);
```

```
        jx++;goto s3;}
case 4:{s3: child1->chrom[jc].chro4=mutation(parent2->chrom[jc].chro4,pmutation);
        child2->chrom[jc].chro4=mutation(parent1->chrom[jc].chro4,pmutation);
        jx++;goto s4;}
case 5:{s4: child1->chrom[jc].chro5=mutation(parent2->chrom[jc].chro5,pmutation);
        child2->chrom[jc].chro5=mutation(parent1->chrom[jc].chro5,pmutation);
        jx++;goto s5;}
case 6:{s5: child1->chrom[jc].chro6=mutation(parent2->chrom[jc].chro6,pmutation);
        child2->chrom[jc].chro6=mutation(parent1->chrom[jc].chro6,pmutation);
        jx++;goto s6;}
case 7:{s6: child1->chrom[jc].chro7=mutation(parent2->chrom[jc].chro7,pmutation);
        child2->chrom[jc].chro7=mutation(parent1->chrom[jc].chro7,pmutation);
        jx++;goto s7;}
case 8:{s7: child1->chrom[jc].chro8=mutation(parent2->chrom[jc].chro8,pmutation);
        child2->chrom[jc].chro8=mutation(parent1->chrom[jc].chro8,pmutation);
        jx++;goto s8;}
case 9:{s8: child1->chrom[jc].chro9=mutation(parent2->chrom[jc].chro9,pmutation);
        child2->chrom[jc].chro9=mutation(parent1->chrom[jc].chro9,pmutation);
        jx++;goto s9;}
case 10:{s9: child1->chrom[jc].chro10=mutation(parent2->chrom[jc].chro10,pmutation);
        child2->chrom[jc].chro10=mutation(parent1->chrom[jc].chro10,pmutation);
        jx++;goto s10;}
case 11:{s10: child1->chrom[jc].chro11=mutation(parent2->chrom[jc].chro11,pmutation);
        child2->chrom[jc].chro11=mutation(parent1->chrom[jc].chro11,pmutation);
        jx++;goto s11;}
case 12:{s11: child1->chrom[jc].chro12=mutation(parent2->chrom[jc].chro12,pmutation);
        child2->chrom[jc].chro12=mutation(parent1->chrom[jc].chro12,pmutation);
        jx++;goto s12;}
case 13:{s12: child1->chrom[jc].chro13=mutation(parent2->chrom[jc].chro13,pmutation);
        child2->chrom[jc].chro13=mutation(parent1->chrom[jc].chro13,pmutation);
        jx++;goto s13;}
case 14:{s13: child1->chrom[jc].chro14=mutation(parent2->chrom[jc].chro14,pmutation);
        child2->chrom[jc].chro14=mutation(parent1->chrom[jc].chro14,pmutation);
        jx++;goto s14;}
case 15:{s14: child1->chrom[jc].chro15=mutation(parent2->chrom[jc].chro15,pmutation);
        child2->chrom[jc].chro15=mutation(parent1->chrom[jc].chro15,pmutation);
        jx++;goto s15;}
case 16:{s15: child1->chrom[jc].chro16=mutation(parent2->chrom[jc].chro16,pmutation);
        child2->chrom[jc].chro16=mutation(parent1->chrom[jc].chro16,pmutation);
        jx++;jc++;break;}
default:break;
        }
    while(jx<=4*lchrom)
    {
        child1->chrom[jc].chro1=mutation(parent2->chrom[jc].chro1,pmutation);
        child2->chrom[jc].chro1=mutation(parent1->chrom[jc].chro1,pmutation);
        jx++;
        child1->chrom[jc].chro2=mutation(parent2->chrom[jc].chro2,pmutation);
        child2->chrom[jc].chro2=mutation(parent1->chrom[jc].chro2,pmutation);
        jx++;
        child1->chrom[jc].chro3=mutation(parent2->chrom[jc].chro3,pmutation);
        child2->chrom[jc].chro3=mutation(parent1->chrom[jc].chro3,pmutation);
```

```
        jx++;
        child1->chrom[jc].chro4=mutation(parent2->chrom[jc].chro4,pmutation);
        child2->chrom[jc].chro4=mutation(parent1->chrom[jc].chro4,pmutation);
        jx++;
        child1->chrom[jc].chro5=mutation(parent2->chrom[jc].chro5,pmutation);
        child2->chrom[jc].chro5=mutation(parent1->chrom[jc].chro5,pmutation);
        jx++;
        child1->chrom[jc].chro6=mutation(parent2->chrom[jc].chro6,pmutation);
        child2->chrom[jc].chro6=mutation(parent1->chrom[jc].chro6,pmutation);
        jx++;
        child1->chrom[jc].chro7=mutation(parent2->chrom[jc].chro7,pmutation);
        child2->chrom[jc].chro7=mutation(parent1->chrom[jc].chro7,pmutation);
        jx++;
        child1->chrom[jc].chro8=mutation(parent2->chrom[jc].chro8,pmutation);
        child2->chrom[jc].chro8=mutation(parent1->chrom[jc].chro8,pmutation);
        jx++;
        child1->chrom[jc].chro9=mutation(parent2->chrom[jc].chro9,pmutation);
        child2->chrom[jc].chro9=mutation(parent1->chrom[jc].chro9,pmutation);
        jx++;
        child1->chrom[jc].chro10=mutation(parent2->chrom[jc].chro10,pmutation);
        child2->chrom[jc].chro10=mutation(parent1->chrom[jc].chro10,pmutation);
        jx++;
        child1->chrom[jc].chro11=mutation(parent2->chrom[jc].chro11,pmutation);
        child2->chrom[jc].chro11=mutation(parent1->chrom[jc].chro11,pmutation);
        jx++;
        child1->chrom[jc].chro12=mutation(parent2->chrom[jc].chro12,pmutation);
        child2->chrom[jc].chro12=mutation(parent1->chrom[jc].chro12,pmutation);
        jx++;
        child1->chrom[jc].chro13=mutation(parent2->chrom[jc].chro13,pmutation);
        child2->chrom[jc].chro13=mutation(parent1->chrom[jc].chro13,pmutation);
        jx++;
        child1->chrom[jc].chro14=mutation(parent2->chrom[jc].chro14,pmutation);
        child2->chrom[jc].chro14=mutation(parent1->chrom[jc].chro14,pmutation);
        jx++;
        child1->chrom[jc].chro15=mutation(parent2->chrom[jc].chro15,pmutation);
        child2->chrom[jc].chro15=mutation(parent1->chrom[jc].chro15,pmutation);
        jx++;
        child1->chrom[jc].chro16=mutation(parent2->chrom[jc].chro16,pmutation);
        child2->chrom[jc].chro16=mutation(parent1->chrom[jc].chro16,pmutation);
        jx++;jc++;
        }
  }
 return jcross;
}
unsigned crossover1(struct bitf huge *parent1,struct bitf huge *parent2,struct bitf huge *child1,
struct bitf huge *child2,unsigned lchrom,float pcross,float pmutation)
{
        int jc,jx,jx1,jcross1;
        float crop;
        crop=(float)random(100)/100.0;
        if(pcross-crop>=0.0)//biased wheight wheel
        {
```

```
            jcross1=random(4*lchrom-1)+1;
            //printf("*********jcross=%4i*********\n",jcross);
            ncross++;
        }
        else jcross1=1;
        jc=lchrom;jx=4*lchrom;
while(jx>=jcross1)
{
        child1->chrom[jc].chro16=mutation(parent1->chrom[jc].chro16,pmutation);
        child2->chrom[jc].chro16=mutation(parent2->chrom[jc].chro16,pmutation);
        jx--;
        child1->chrom[jc].chro15=mutation(parent1->chrom[jc].chro15,pmutation);
        child2->chrom[jc].chro15=mutation(parent2->chrom[jc].chro15,pmutation);
        jx--;
        child1->chrom[jc].chro14=mutation(parent1->chrom[jc].chro14,pmutation);
        child2->chrom[jc].chro14=mutation(parent2->chrom[jc].chro14,pmutation);
        jx--;
        child1->chrom[jc].chro13=mutation(parent1->chrom[jc].chro13,pmutation);
        child2->chrom[jc].chro13=mutation(parent2->chrom[jc].chro13,pmutation);
        jx--;
        child1->chrom[jc].chro12=mutation(parent1->chrom[jc].chro12,pmutation);
        child2->chrom[jc].chro12=mutation(parent2->chrom[jc].chro12,pmutation);
        jx--;
        child1->chrom[jc].chro11=mutation(parent1->chrom[jc].chro11,pmutation);
        child2->chrom[jc].chro11=mutation(parent2->chrom[jc].chro11,pmutation);
        jx--;
        child1->chrom[jc].chro10=mutation(parent1->chrom[jc].chro10,pmutation);
        child2->chrom[jc].chro10=mutation(parent2->chrom[jc].chro10,pmutation);
        jx--;
        child1->chrom[jc].chro9=mutation(parent1->chrom[jc].chro9,pmutation);
        child2->chrom[jc].chro9=mutation(parent2->chrom[jc].chro9,pmutation);
        jx--;
        child1->chrom[jc].chro8=mutation(parent1->chrom[jc].chro8,pmutation);
        child2->chrom[jc].chro8=mutation(parent2->chrom[jc].chro8,pmutation);
        jx--;
        child1->chrom[jc].chro7=mutation(parent1->chrom[jc].chro7,pmutation);
        child2->chrom[jc].chro7=mutation(parent2->chrom[jc].chro7,pmutation);
        jx--;
        child1->chrom[jc].chro6=mutation(parent1->chrom[jc].chro6,pmutation);
        child2->chrom[jc].chro6=mutation(parent2->chrom[jc].chro6,pmutation);
        jx--;
        child1->chrom[jc].chro5=mutation(parent1->chrom[jc].chro5,pmutation);
        child2->chrom[jc].chro5=mutation(parent2->chrom[jc].chro5,pmutation);
        jx--;
        child1->chrom[jc].chro4=mutation(parent1->chrom[jc].chro4,pmutation);
        child2->chrom[jc].chro4=mutation(parent2->chrom[jc].chro4,pmutation);
        jx--;
        child1->chrom[jc].chro3=mutation(parent1->chrom[jc].chro3,pmutation);
        child2->chrom[jc].chro3=mutation(parent2->chrom[jc].chro3,pmutation);
        jx--;
        child1->chrom[jc].chro2=mutation(parent1->chrom[jc].chro2,pmutation);
        child2->chrom[jc].chro2=mutation(parent2->chrom[jc].chro2,pmutation);
```

```
        jx--;
        child1->chrom[jc].chro1=mutation(parent1->chrom[jc].chro1,pmutation);
        child2->chrom[jc].chro1=mutation(parent2->chrom[jc].chro1,pmutation);
        jx--;jc--;
    }
    if(jcross1!=1)
    {
    jx1=jx;jx1=4*lchrom-jx1;
    while(jx1>16)jx1=jx1-16;
    jx1=16-jx1;
    switch(jx1)
        {
case 16:{ child1->chrom[jc].chro16=mutation(parent2->chrom[jc].chro16,pmutation);
        child2->chrom[jc].chro16=mutation(parent1->chrom[jc].chro16,pmutation);
        jx--;goto ca1;}
case 15:{ca1: child1->chrom[jc].chro15=mutation(parent2->chrom[jc].chro15,pmutation);
        child2->chrom[jc].chro15=mutation(parent1->chrom[jc].chro15,pmutation);
        jx--;goto ca2;}
case 14:{ ca2:child1->chrom[jc].chro14=mutation(parent2->chrom[jc].chro14,pmutation);
        child2->chrom[jc].chro14=mutation(parent1->chrom[jc].chro14,pmutation);
        jx--;goto ca3;}
case 13:{ca3: child1->chrom[jc].chro13=mutation(parent2->chrom[jc].chro13,pmutation);
        child2->chrom[jc].chro13=mutation(parent1->chrom[jc].chro13,pmutation);
        jx--;goto ca4;}
case 12:{ ca4: child1->chrom[jc].chro12=mutation(parent2->chrom[jc].chro12,pmutation);
        child2->chrom[jc].chro12=mutation(parent1->chrom[jc].chro12,pmutation);
        jx--;goto ca5;}
case 11:{ ca5: child1->chrom[jc].chro11=mutation(parent2->chrom[jc].chro11,pmutation);
        child2->chrom[jc].chro11=mutation(parent1->chrom[jc].chro11,pmutation);
        jx--;goto ca6;}
case 10:{ca6: child1->chrom[jc].chro10=mutation(parent2->chrom[jc].chro10,pmutation);
        child2->chrom[jc].chro10=mutation(parent1->chrom[jc].chro10,pmutation);
        jx--;goto ca7;}
case 9:{ ca7:child1->chrom[jc].chro9=mutation(parent2->chrom[jc].chro9,pmutation);
        child2->chrom[jc].chro9=mutation(parent1->chrom[jc].chro9,pmutation);
        jx--;goto ca8;}
case 8:{ca8: child1->chrom[jc].chro8=mutation(parent2->chrom[jc].chro8,pmutation);
        child2->chrom[jc].chro8=mutation(parent1->chrom[jc].chro8,pmutation);
        jx--;goto ca9;}
case 7:{ ca9: child1->chrom[jc].chro7=mutation(parent2->chrom[jc].chro7,pmutation);
        child2->chrom[jc].chro7=mutation(parent1->chrom[jc].chro7,pmutation);
        jx--;goto ca10;}
case 6:{ ca10: child1->chrom[jc].chro6=mutation(parent2->chrom[jc].chro6,pmutation);
        child2->chrom[jc].chro6=mutation(parent1->chrom[jc].chro6,pmutation);
        jx--;goto ca11;}
case 5:{ ca11: child1->chrom[jc].chro5=mutation(parent2->chrom[jc].chro5,pmutation);
        child2->chrom[jc].chro5=mutation(parent1->chrom[jc].chro5,pmutation);
        jx--;goto ca12;}
case 4:{ ca12: child1->chrom[jc].chro4=mutation(parent2->chrom[jc].chro4,pmutation);
        child2->chrom[jc].chro4=mutation(parent1->chrom[jc].chro4,pmutation);
        jx--;goto ca13;}
case 3:{ ca13: child1->chrom[jc].chro3=mutation(parent2->chrom[jc].chro3,pmutation);
```

```
        child2->chrom[jc].chro3=mutation(parent1->chrom[jc].chro3,pmutation);
        jx--;goto ca14;}
case 2:{ ca14: child1->chrom[jc].chro2=mutation(parent2->chrom[jc].chro2,pmutation);
        child2->chrom[jc].chro2=mutation(parent1->chrom[jc].chro2,pmutation);
        jx--;goto ca15;}
case 1:{ ca15: child1->chrom[jc].chro1=mutation(parent2->chrom[jc].chro1,pmutation);
        child2->chrom[jc].chro1=mutation(parent1->chrom[jc].chro1,pmutation);
        jx--;jc--;break;}
default:break;
   }
   while(jx>=1)
        {
        child1->chrom[jc].chro16=mutation(parent2->chrom[jc].chro16,pmutation);
        child2->chrom[jc].chro16=mutation(parent1->chrom[jc].chro16,pmutation);
        jx--;
        child1->chrom[jc].chro15=mutation(parent2->chrom[jc].chro15,pmutation);
        child2->chrom[jc].chro15=mutation(parent1->chrom[jc].chro15,pmutation);
        jx--;
        child1->chrom[jc].chro14=mutation(parent2->chrom[jc].chro14,pmutation);
        child2->chrom[jc].chro14=mutation(parent1->chrom[jc].chro14,pmutation);
        jx--;
        child1->chrom[jc].chro13=mutation(parent2->chrom[jc].chro13,pmutation);
        child2->chrom[jc].chro13=mutation(parent1->chrom[jc].chro13,pmutation);
        jx--;
        child1->chrom[jc].chro12=mutation(parent2->chrom[jc].chro12,pmutation);
        child2->chrom[jc].chro12=mutation(parent1->chrom[jc].chro12,pmutation);
        jx--;
        child1->chrom[jc].chro11=mutation(parent2->chrom[jc].chro11,pmutation);
        child2->chrom[jc].chro11=mutation(parent1->chrom[jc].chro11,pmutation);
        jx--;
        child1->chrom[jc].chro10=mutation(parent2->chrom[jc].chro10,pmutation);
        child2->chrom[jc].chro10=mutation(parent1->chrom[jc].chro10,pmutation);
        jx--;
        child1->chrom[jc].chro9=mutation(parent2->chrom[jc].chro9,pmutation);
        child2->chrom[jc].chro9=mutation(parent1->chrom[jc].chro9,pmutation);
        jx--;
        child1->chrom[jc].chro8=mutation(parent2->chrom[jc].chro8,pmutation);
        child2->chrom[jc].chro8=mutation(parent1->chrom[jc].chro8,pmutation);
        jx--;
        child1->chrom[jc].chro7=mutation(parent2->chrom[jc].chro7,pmutation);
        child2->chrom[jc].chro7=mutation(parent1->chrom[jc].chro7,pmutation);
        jx--;
        child1->chrom[jc].chro6=mutation(parent2->chrom[jc].chro6,pmutation);
        child2->chrom[jc].chro6=mutation(parent1->chrom[jc].chro6,pmutation);
        jx--;
        child1->chrom[jc].chro5=mutation(parent2->chrom[jc].chro5,pmutation);
        child2->chrom[jc].chro5=mutation(parent1->chrom[jc].chro5,pmutation);
        jx--;
        child1->chrom[jc].chro4=mutation(parent2->chrom[jc].chro4,pmutation);
        child2->chrom[jc].chro4=mutation(parent1->chrom[jc].chro4,pmutation);
        jx--;
        child1->chrom[jc].chro3=mutation(parent2->chrom[jc].chro3,pmutation);
```

```
        child2->chrom[jc].chro3=mutation(parent1->chrom[jc].chro3,pmutation);
        jx--;
        child1->chrom[jc].chro2=mutation(parent2->chrom[jc].chro2,pmutation);
        child2->chrom[jc].chro2=mutation(parent1->chrom[jc].chro2,pmutation);
        jx--;
        child1->chrom[jc].chro1=mutation(parent2->chrom[jc].chro1,pmutation);
        child2->chrom[jc].chro1=mutation(parent1->chrom[jc].chro1,pmutation);
        jx--;jc--;
        }
 }
        return jcross1;
}
void    initdata(void)
{
        char cha,*p,p1[100],c;   int j;
//      void cleardevice();
        printf("**** A basic Genetic Algorithms for Design of Fuzzy controller****\n\n");
        printf("********************* by using Borland C++ ***************\n\n");
        p=" ZIBO ZHANG March, 1994  Version 1.0 All Rights Reserved";
        printf("******** %s ******** \n",p);
        p=" SGA Data Enter and Initialization";
        printf("************* %s *************\n",p);
        p=" please input population size number, then press 'return'";
        printf("**** %s **** \n",p);
        fflush(stdin);
        scanf("%i",&popsize);
        p=" please input chromosome length, then press 'return'";
        printf("**** %s **** \n",p);
        scanf("%i",&lchrom);
        p=" please input max generations, then press 'return'";
        printf("**** %s **** \n",p);
        scanf("%i",&maxgen);
        p=" please input crossover probability, then press 'return'";
        printf("**** %s **** \n",p);
        scanf("%f",&pcross);
        p=" please input mutation probability, then press 'return'";
        printf("**** %s **** \n",p);
        scanf("%f",&pmutation0);
        randomize();
        nmutation=0,ncross=0;
}
void initialize(void)
{
        if(flag1==0)
        initdata();
        initpop();
        statistics(popsize);
        initreport();
}
void initpop()
{
        float objfit;
```

```
        unsigned j,j2,j1,jxz;
        for(j=1;j<=popsize;j=j+1)
        {     j2=1;j1=1;
                while(j2<=lchrom*4)
                {
                newpop[j].chrom[j1].chro1=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro2=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro3=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro4=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro5=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro6=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro7=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro8=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro9=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro10=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro11=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro12=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro13=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro14=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro15=(unsigned)flip(0.5);j2++;
                newpop[j].chrom[j1].chro16=(unsigned)flip(0.5);j2++;j1++;
                }
         objfit=(float)objfunc(j);
         if((cmax-objfit)<0.)
            newpop[j].fitness=0.;
         else
         {
          newpop[j].fitness=cmax-objfit;
         }
         newpop[j].parent1=0;newpop[j].parent2=0;
         printf("newpop[%2i].fitness=%6.6f\t",j,newpop[j].fitness);
        }
 }
void objp(void)
{
   int kb,mod,kby,zhang;
   float xxm[5],uy[5],sh[7];
/*   for(kb=1;kb<=number;kb++)
   {     py1[kb]=0.0;u1[kb]=0.0;}
         uy[1]=1.0;u1[0]=-1.0;    //number of data
         for(kb=1;kb<=5;kb++)sh[kb]=uy[1];
         for(kb=1;kb<=(number-1)/6;kb++)
         {
                 xxm[1]=sh[4]+sh[5];//(+)=NOR
         if(xxm[1]==0.0)xxm[1]=1.0;else xxm[1]=-1.0;
         for(kby=2;kby<=5;kby++)
          {zhang=6-kby;sh[zhang+1]=sh[zhang];}//shift register
                 sh[1]=xxm[1];
         for(kby=(kb-1)*6;kby<=kb*6;kby++)py[kby]=xxm[1];
                 printf("u1[%2i]=%2.2f\t",kb,py[kb]);
         }*/
         for(kb=1;kb<=number;kb++)
```

```
        {
        if(kb>=1&&kb<20){py[kb]=0.5;goto oag8;}
        if(kb>=20&&kb<40){py[kb]=0.0;goto oag8;}
        if(kb>=40&&kb<60){py[kb]=1.0;goto oag8;}
        if(kb>=60&&kb<80){py[kb]=0.0;goto oag8;}
        if(kb>=80&&kb<100){py[kb]=0.8;goto oag8;}
        if(kb>=100&&kb<120){py[kb]=1.0;goto oag8;}
        if(kb>=120&&kb<=number)py[kb]=0.6;
oag8:   printf("py[%2i]=%2.2f\t",kb,py[kb]);
        }
}
unsigned select(unsigned popsize,float sumfitness)
{
        float rand,partsum=0.0;
        unsigned j=0;
        rand=((float)random(1000)/1000.0)*sumfitness;
        while(((partsum-rand)<0) && (j<popsize))
        {
                j=j+1;
                partsum+=oldpop[j].fitness;
        }
        return j;
}
void memb(void)//popsize,sumfitness,oldpop
{
        FILE *p1;
        int jx,j,m[10];
        float x,e,t,aa[18],bb[18],kk[18];
        bb[0]=0.1;t=0.49698;kk[0]=100;aa[0]=-0.1;
        aa[1]=0.0;bb[1]=.3;aa[2]=0.1;bb[2]=0.5;aa[3]=0.3;bb[3]=0.7;
        aa[4]=0.5;bb[4]=1.0;aa[5]=0.7;bb[5]=1.2;
        kk[1]=4.0/((bb[1]-aa[1])*(bb[1]-aa[1]));
 printf("k[0]=%3.5f\tk[1]=%3.6f\tb[1]=%3.5f\n",kk[0],kk[1],bb[1]);
        for(jx=2;jx<=6;jx++)
        {
//       aa[jx]=0.0;
//       for(j=1;j<=jx-1;j++)aa[jx]=aa[jx]+pow(2.0,j-1)*pow(2.7182,j*t);
//       aa[jx]=aa[jx]*bb[0]/pow(2.0,jx-1);
//       bb[jx]=bb[0]*pow(2.71828,jx*t);
         kk[jx]=4.0/((aa[jx]-bb[jx])*(aa[jx]-bb[jx]));
 printf("k[%i]=%3.5f\ta[jx]=%2.4f\tb[jx]=%3.5f\n",jx,kk[jx],aa[jx],bb[jx]);
        }
        x=0.0;jx=1+120;
        x=aa[1];jx=100*x+120;
        if((p1=fopen("c:abk.dat","w"))!=NULL)
        for(j=1;j<=5;j++)  //the left side of y axis
        {
         fprintf(p1,"%1.6f\t%1.6f\t%1.6f\n",-bb[6-j],-aa[6-j],kk[6-j]);
         a[j-1]=-bb[6-j];b[j-1]=-aa[6-j];k[j-1]=kk[6-j];
        }
        for(j=6;j<=11;j++)//the right side of y axis
        {
```

```
        fprintf(p1,"%1.6f\t%1.6f\t%1.6f\n",aa[j-6],bb[j-6],kk[j-6]);
        a[j-1]=aa[j-6];b[j-1]=bb[j-6];k[j-1]=kk[j-6];
        }
        fclose(p1);
}
```