GENETIC ALGORITHMS IN SYSTEM IDENTIFICATION AND CONTROL

By

Kristinn Kristinsson

B. Sc. Electrical Engineering University of Iceland, 1986

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1989

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Electrical Engineering
The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date:

Sept. 13 1989

Abstract

Current online identification techniques are recursive and involve local search techniques. In this thesis, we show how genetic algorithms, a parallel, global search technique emulating natural genetic operators can be used to estimate the poles and zeros of a dynamical system. We also design an adaptive controller based on the estimates. The algorithms are shown to be useful for continuous time parameter identifications and to be able to identify directly physical parameters of a system. Simulations and an experiment show the technique to be satisfactory and to provide unbiased estimates in presence of colored noise.

Table of Contents

List of Tables

List of Figures

Acknowledgement

I would like to use this opportunity for thanking all those that have made the completion of this thesis possible. I would especially like to thank Prof. Guy A. Dumont, my thesis advisor, for introducing the Genetic Algorithms to me and for his advice and guidance in my research. I would also like to thank Ye Fu for providing the RLS routines and Rob Ross for his assistance in operation the Pulp & Paper Centre's μVAX computer. At last special thanks go to Dr. K. Natarajan who read the final draft of this thesis.

Chapter 1

Introduction

1.1 General introduction

The area of system identification has been given a lot of attention over the years. Many methods have been used and many extended versions exist, but all of them are based upon eighteenth century mathematics which assumes smooth search space with ever present derivatives.

In the last few years Artificial Intelligence and learning have been gaining lot of popularity and have been entering many fields, but little has been done to apply them in the field of system identification and control.

1.2 "Standard" identification methods

On-line system identification methods used to date are based on recursive implementation of off-line methods such as least-squares, maximum-likelihood or instrumental variable. All those methods are based on the same principle and a unified description exists [23]. Those recursive schemes are in essence local search techniques that search for zero gradient by going in a direction suggested by the local gradient. They go to the nearest point that gives zero gradient and stay there. Nothing will get the methods to search further as long as the gradient stays zero. It is therefore very difficult for those methods to find a global maximum and they often fail in the search for global maximum if the search space is not differentiable or linear in the parameters. Because of

the linearity condition they have difficulty locating directly poles and zeros or physical parameters of a system.

Another aspect is that these methods are all serial. They go from one point in the search space to another at every sampling instant, as a new input-output pair becomes available. They are not capable of iterating more than once on each data they receive, they need new data to direct the search.

1.3 Motivation for this work

Genetic algorithms are a parallel, global search technique that emulates natural genetic operators. They search many points simultaneously and thus have the potential to converge more rapidly. In every generation new artificial chromosomes are created by taking parts of the fittest chromosomes of the previous generation and combining them to make a highly fit chromosome. They do not need to assume that the search space is differentiable or continuous because they go from one generation to another with transition rules that are probabilistic. This means the algorithms do not have to wait for new data, but can iterate a few times on each data they receive. They work with a population of binary coded strings so they can explore the search space in each generation and then direct the search to regions where there is a high probability of finding improved performance. Genetic algorithms have also been shown to excel in multimodal optimization [10], and thus have the potential to give unbiased estimates in presence of coloured noise.

In this thesis, a Genetic Algorithm (GA) is implemented as an estimator for discrete time systems. The results obtained employing this new identification method are particularly favourable and they are considered to be well suited to the adaptive

control problem. Although the use of GA has been gaining popularity, its use in adaptive control has not been investigated. The algorithm is used on few discrete time systems, both minimum and nonminimum phase and with or without colored noise. It is used to identify either parameters or poles-and-zeros. The encouraging results are then compared with Recursive Least Squares.

1.4 Outline of this thesis

This thesis is organised as follows:

Chapter 2. Genetic Algorithms are described and some of the simple genetic operators are explained. The algorithm is then used to find the maximum for a function with eleven local maxima.

Chapter 3. A GA for system identification is implemented both in discrete and continuous time and simulations results are shown. The algorithm is also compared to Recursive Least Squares algorithm.

Chapter 4. The pole placement controller design is outlined and simulations results are shown using the GA to identify plants with either minimum phase or nonminimum phase characteristic and unmodeled dynamics. One simulation is then done using Recursive Least Squares for the identification for comparison.

Chapter 5. An experiment with a water tank is explained and identification results are shown.

Chapter 6. Conclusions and suggestions for further work are given.

Chapter 2

Genetic Algorithms

2.1 History of Genetic Algorithms

The algorithms come out of work done by John H. Holland and his students at the University of Michigan. The underlying principles of genetic algorithms were first published by Holland in 1962 [18]. The mathematical framework was developed in the late 1960's and in 1975, Holland's pioneering book, Adaptation in Natural and Artificial Systems was published [19]. The same year it was shown by one of his student that Genetic Algorithms (GAs) are very useful in function optimization even on "difficult" domains that are multimodal, noisy and high-dimensional [10]. In 1983, Goldberg used GA to minimize power consumption in gas-pipelines and then combined a learning classifier system with a GA to detect leakage in the system [13]. In the last four years a lot of research has been devoted to GA, two conferences have been held [16,17] and two books have been written on the subject [15,9]. Genetic Algorithms have proven to be useful in many different applications [15], like function optimization, computer network design, travelling salesman problem, pattern recognition and many more.

2.2 The algorithm

Genetic algorithms differ from other search techniques by the use of concepts taken from natural genetics and evolution theory. They are different in four ways:

1. GAs work with coding of the parameters, not the parameters themselves.

2. GAs search from a population of points, not a single point.

3. GAs only need fitness values. There is no requirement for derivatives or other auxiliary knowledge.

4. GAs use probabilistic transition rules, not deterministic rules.

The parameters to be found by the GA, need to be coded as a finite length string over a finite search space. As an example, consider a stable real pole, (magnitude less than one) the search space would be on the interval [0,1] or if we want a resolution of 1/1000 the search space would be in the integer interval [0,1000] and with a binary coding this would be coded as a 10 bit string. The algorithm works with a population of strings, searching many peaks in parallel, hence reducing the possibility of ending at a local minimum and missing the global minimum. The only available feedback from the system is the value of the performance measure (fitness). Although transition rules are probabilistic, the algorithm is not simply a random search. It is a randomized search that is guided by the fitness values of each string. The algorithm uses information, already in the population, about things that have worked well in the past. By use of operators taken from population genetics the algorithm efficiently explores part of the search space where the probability of finding improved performance is high.

A genetic algorithm in its simplest form consists of 3 steps: (see Figure 2.1)

1. Reproduction

2. Crossover

3. Mutation

In the next three sections these will be described in details.

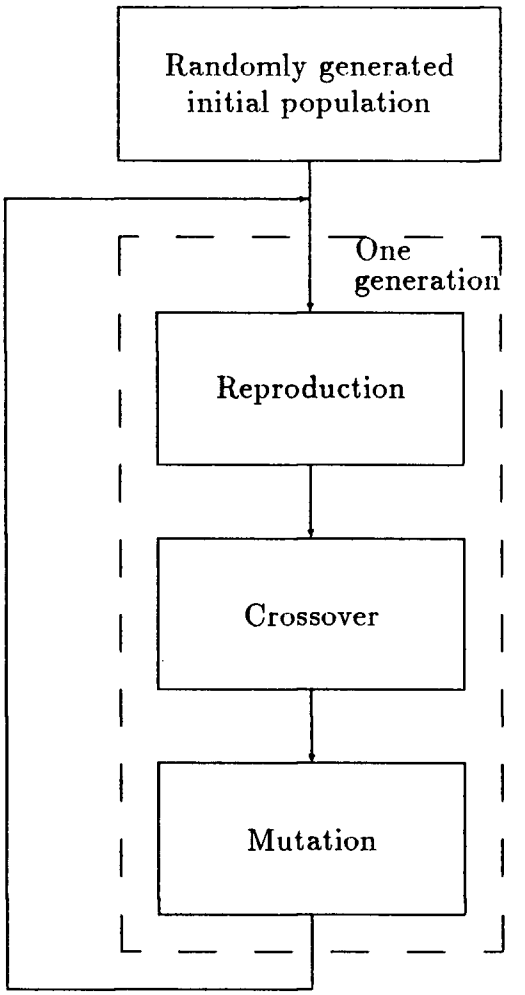

Figure 2.1: Genetic Algorithm

2.3 Coding

It has been shown that binary coding is in a certain sense the optimal coding [19]. Suppose we have the binary strings 1010010111 and 1110100110, by comparing them we can see some similarity, $1*10**011*$, where $*$ is a don't care. We call $1*10**011*$, a *schema* (plural, *schemata*). A string can be an instance of $2^{10} = 1024$ schemata which can be found by replacing any of the bits in the string by a don't care. The number of possible schemata on the alphabet $\{*, 0, 1\}$ for the binary coding is $3^{10} = 59094$ so by carefully selecting the string all schemata could be represented by 58 strings. If the coding is decimal we would need 3 decimal number to represent the same search space as a 10 bit binary string would. A three bit decimal string would be an instance of $2^3 = 8$ schemata but the number of possible schemata on the alphabet $\{*, 0, 1, \ldots, 9\}$ would be $11^3 = 1331$ so all schemata would have to be represented by 166 strings. So binary coding would need 3 times fewer strings to explore the search space. Therefore binary coding is chosen with each parameter corresponding to a fixed length binary substring of j bits $[0, \ldots, 2^j - 1]$. The value, (x), of the binary substring is mapped to an interval of the real numbers $[lb, ub]$ to give

$$y = \frac{x}{2^j - 1}(ub - lb) + lb \tag{2.1}$$

With n parameters, the final string consists of n concatenated substrings.

$$\begin{array}{|c|c|c|} \hline a_1 & \cdots & a_n \\ \hline 10\cdots01 & \cdots & 01\cdots11 \\ \hline \end{array} \tag{2.2}$$

2.4 Reproduction

In the reproduction part of the algorithm it is decided which strings are going to survive and which ones are going to disappear, based on what in biological terms, is known as

$F(i)$	$F_n(i)$	number of offsprings
100	0.50	1
10	0.05	0
200	1.00	1
300	1.50	2
210	1.05	1
290	1.45	1
310	1.55	1
280	1.40	2
120	0.60	1
180	0.90	0

Table 2.1: Fitness dependent reproduction.

the survival of the fittest principle. It is done by assigning a positive number, fitness $F(i)$, to each individual in the population. It must be positive because high fitness individuals should receive more offsprings than low fitness individuals. Based on the normalized fitness $F_n(i)$, the number of offsprings for each individual is calculated. The fitness function tells us how well the system, to optimize or control, is behaving under a certain string. The fitness function can be any nonlinear, nondifferentiable, discontinuous function, because the algorithm only needs a fitness value assigned to each string.

The number of offsprings is chosen according to the string normalized fitness. The fitness is normalized with the average value of the fitness,

$$F_n(i) = \frac{F(i)}{\frac{1}{N}\sum_{i=1}^{N} F(i)} \tag{2.3}$$

so the strings with above average fitness will have more than 1 offspring and those with below average fitness will have less than 1 offspring on the average (see Table 2.1). The strings are selected according to the expected value of the normalized fitness or what has become known as **Stochastic Remainder Selection without Replacement**

[15]. That means the strings will receive number of offsprings equal to the integer value of their normalized fitnesses and then the population is filled up by choosing another offspring for each of the strings with probability equal to their fractional part until the total number of offsprings are equal to the population size N.

The algorithm keeps track of the best string in the population and if it is not in the new population (because some other GA operators destroyed it) it randomly replaces another string in the new population.

2.4.1 Ranking

It is important to regulate the number of offsprings an individual can get, to maintain a diversity in the population. Especially for the first few generations, when a few "super" individuals can potentially take over a large part of the population, thereby reducing the diversity of the population. The presence of super individuals can be sensed by monitoring the number of individuals that are going to receive 0 offsprings. It is somewhat a better way than limiting the number of offsprings an individual can get, because it could be desirable to give a good individual many offsprings as long as the diversity is maintained.

To control the reproduction, *ranking* can be introduced [3]. Whenever a certain ratio of the normalized fitness is going to receive 0 offsprings, the strings are sorted according to their fitness values. Then, instead of calculating the normalized fitness as in Equation 2.3, the normalized fitness is given to each string according to

$$F_n(i) = \frac{2(max-1)}{N-1} rank(i) + 1 - (max-1)\frac{N+1}{N-1} \tag{2.4}$$

where max as shown in Figure 2.2 is a user defined value, $1 \leq max \leq 2$, and N is the population size. The range of the normalized fitness will then be [2-max,max]. This means that no matter how big the fitness is for the best string its normalized fitness

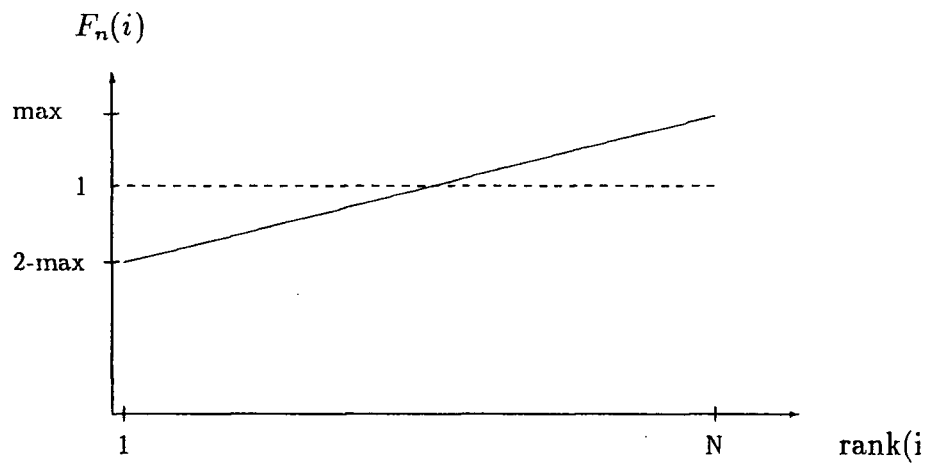

Figure 2.2: Ranking

will never be more than *max* when ranking is in effect. The lowest ranking string will similarly always be guaranteed $2 - max$ as its normalized fitness.

2.5 Crossover

Reproduction directs the search towards the best but does not create any *new* individuals. The offsprings are identical to their parent. In nature, the offsprings are not exact copy of the parents, they usually have two parents and then inherit their characteristic from both parents to make up a new individual. The main operator to work on the parents is crossover, the main searching operator. This operator takes valuable information from both parents and combines it to find a highly fit individual. To apply this operator, two strings from the reproduced population are mated at random and they are cut once randomly between two bits. The new strings are then created by interchanging the tails. It means that parent A will get the tail cut from parent B as it tail and vice versa. This can best be explained by an example. Suppose there are

two strings

00000000 and 11111111

and assume a random number generator comes up with a 3 as the cutting place or crossover site. Then the new strings will be

11100000 and 00011111

Reproduction and crossover give genetic algorithms much of their power. The search is emphasized towards the best and new regions are explored by using information about things that have worked well in the past.

2.6 Mutation

Even though reproduction and crossover come up with many new strings they do not introduce any new information into the population at the bit level. They work with the bits that are already in the population and do not get any new bits into the population. The bits can only reproduce or die, so if at certain position all the bits have the same value, there is no way that crossover and reproduction can get the lost bit back. To insure against such a loss and as a source of new bits, mutation is introduced. In the case of binary coding, the mutation operator simply flips the state of a bit from 0 to 1 or vice versa. But it should be used sparingly because it is a random search operator that searches the space randomly and the algorithm is intended to be a randomized searching algorithm, not a random search.

2.7 Mathematical Foundations

The theoretical properties of genetic algorithms can be studied using the theory of schemata[1] proposed by Holland [19]. The *defining length* of a schema, $\delta(h)$, is the

[1]see definition of schemata in Section 2.3

length between its outermost defining positions, for example $\delta(0 * 0) = 3 - 1 = 2$ and $\delta(* * 1) = 3 - 3 = 0$. The defining length is a measure of how often crossover may be destructive for a particular schema. For the schema $0 * 0$ there are two ways to destruct it by cutting it, but the schema $* * 0$ can not be destructed by crossover, providing both offsprings created by crossover are kept. The *order* of a schema, $o(h)$, on the other hand, is a measure of how often mutation will be destructive for a schema. The order of a schema is the number of defining positions for a string, for example $o(0 * 0) = 2$ and $o(* * 0) = 1$, or in other words, mutation can possibly destruct schema $0 * 0$ in two places but schema $* * 0$ in one place. In other words, schemata with short defining length and low order, stands the biggest chance of surviving into the next generation. This can be written as the Schemata Theorem [19]

Theorem 1 *Consider a GA using both crossover and mutation. The expected proportion of each schema represented in the population changes in one generation from* $m(h,t)$ *to*

$$m(h,t+1) \geq m(h,t)\frac{F(h,t)}{\bar{F}(t)}\left(1 - p_c\frac{\delta(h)}{l-1}(1 - m(h,t))\right)(1 - p_m)^{o(h)}$$

Where p_c *is the probability of a particular mating to undergo a crossover and* p_m *is the probability of a single bit to mutate during a generation. The average fitness of the strings at time* t *representing the schema* h *is denoted by* $F(h,t)$ *and* $\bar{F}(t)$ *is the average fitness of the population.*

What it means is that the number of schemata at time $t+1$ is greater or equal to the number at time t multiplied by the expected number of offsprings less those schemata that are destructed by crossover or mutation. In other words the schemata theorem states that the algorithm is going to converge towards the best, but there is no guarantee that it is converging to *the* optimum. As Goldberg [15] puts it (pp.74):

> "Convergent behavior without guarantee of optimality bothers many people who approach genetic algorithms from other, more traditional, optimization backgrounds. ...the fact of the matter is that genetic algorithms have no convergence guarantees in arbitrary problems. They do sort out interesting areas of a space quickly, but they are a weak method, without the guarantees of more convergent procedures. This does not reduce their utility. Quite the contrary, more convergent methods sacrifice globality and flexibility for their convergence."

GA have been shown to behave well on multimodal functions, although there is no known necessary and sufficient condition under which a function is genetically optimizable. However, numerous studies have shown that functions on which GA fail are pathological, and generally fail to be optimized by any other known technique except exhaustive search [4]. In a recent study by Goldberg [14] it has been shown that even though the algorithm is misled, it will converge for a wide range of starting conditions (initial population) and under unfavorable conditions.

2.8 Example

Suppose we have the function

$$(1 - cos2\pi t)\, sin^2 11\pi t \tag{2.5}$$

and wish to find the maximum on the time interval [0,1]. The function has 11 local maxima, the global one being in the middle as shown in Figure 2.3. By not knowing the underlying function itself, but only the values of it, it is very difficult to locate the maximum. If the direction of steepest gradient is followed the maximum will be the one closest to the starting point. The GA on the other hand should be able to find the maximum by climbing more than one peak at a time.

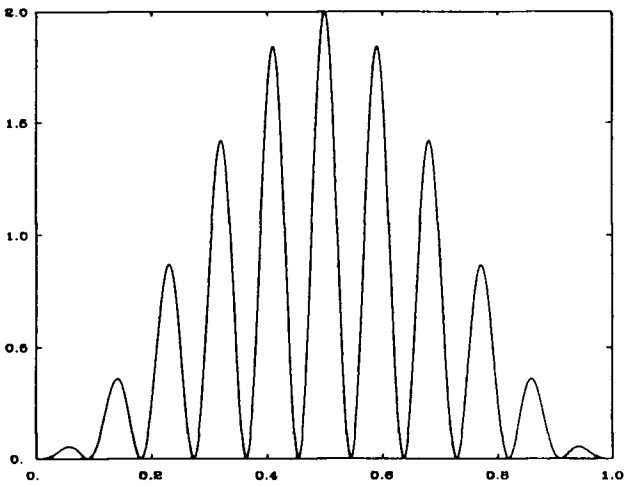

Figure 2.3: Function with 11 local maxima

Assume the interval [0,1] is coded as 10 bit binary string and the population size is 10. The initial population is chosen randomly and the binary string then mapped onto the time interval. The fitness is read from Figure 2.3. The average fitness is calculated and the number of offsprings for each individual found (see Table 2.2). Crossover and mutation are done by choosing mates and crossover site, both at random. Mutation is applied by mutating every bit of the new population with probability equal to p_m (approx. 1/1000), see Table 2.3. After three generations the algorithm is able to find a solution within 3.5% of the maximum, as can be seen in Figure 2.4. It is not until after 12 generations it finds *The Maximum*, but that is one of the underlying principles of the algorithm that it does its best while learning to do better.

2.9 Summary

Because the algorithm works with a population of strings, it is given more chance to locate the global maximum in a multimodal search space. It is in fact searching many points (peaks) in parallel and exchanging information between the peaks. The initial

	para-meters	fitness	normalized fitness	off-springs
1000001101	0.513	1.61	2.18	2
0011110101	0.239	0.78	1.05	2
0000001110	0.014	0.00	0.00	0
1100010111	0.773	0.85	1.16	1
1000110111	0.554	0.17	0.24	0
0101011011	0.339	0.86	1.16	1
1111010111	0.961	0.03	0.04	0
1001100101	0.599	1.67	2.26	2
0111010011	0.457	0.01	0.01	0
0101001011	0.324	1.40	1.89	2
		av. 0.74		

Table 2.2: Reproduction

reproduction	mate	x-site	new generation
10-00001101	3	2	1011110101
100-0001101	8	3	1001100101
00-11110101	1	2	0000001101
00111101-01	10	8	0011110111
110001-0111	7	6	1100110101
01010110-11	9	8	0101011011
100110-0101	5	6	1001100111
100-1100101	2	3	1000001101
01010010-11	6	8	0101001011
01010010-11	4	8	0101001001

Table 2.3: Crossover and mutation

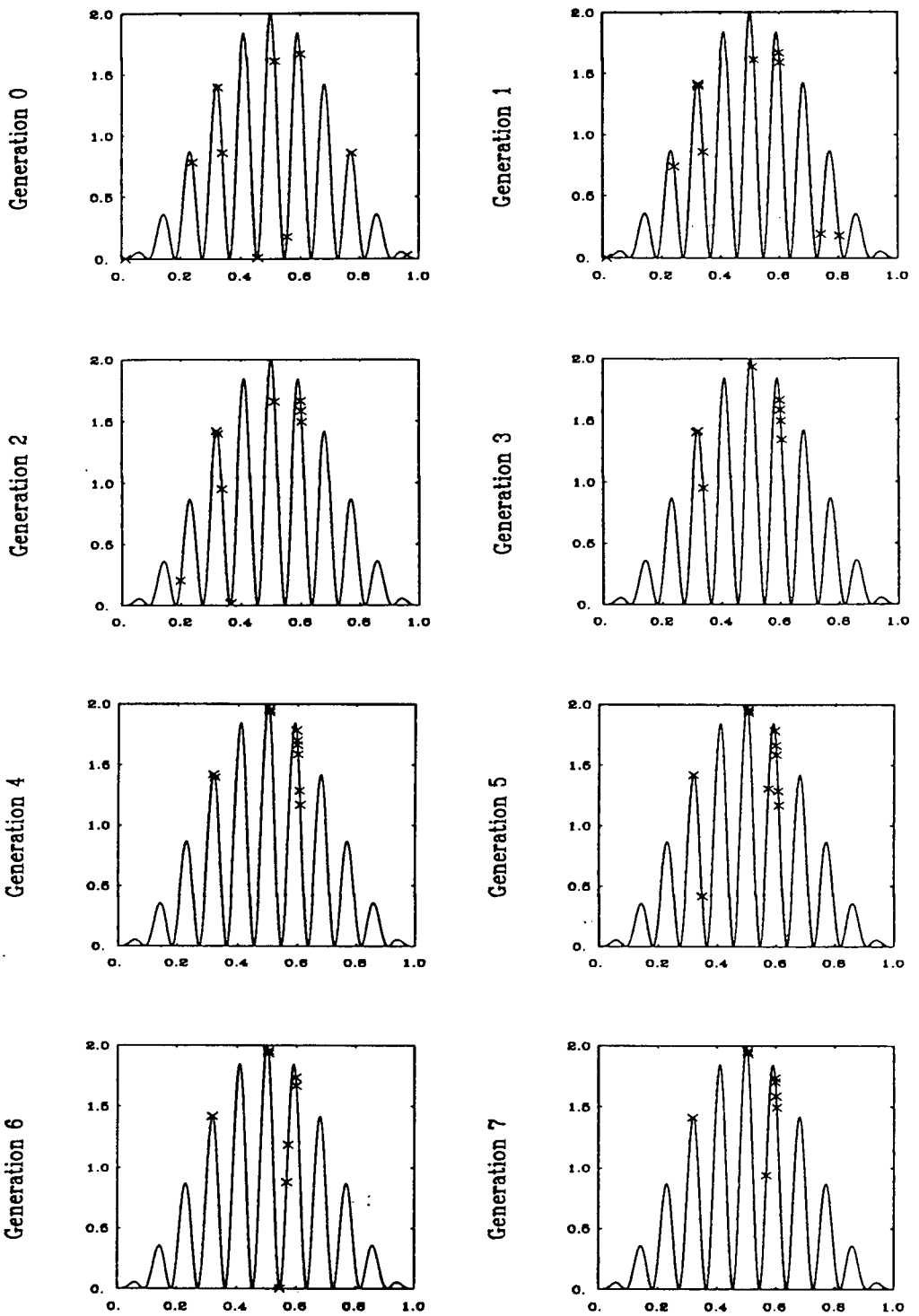

Figure 2.4: GA generations

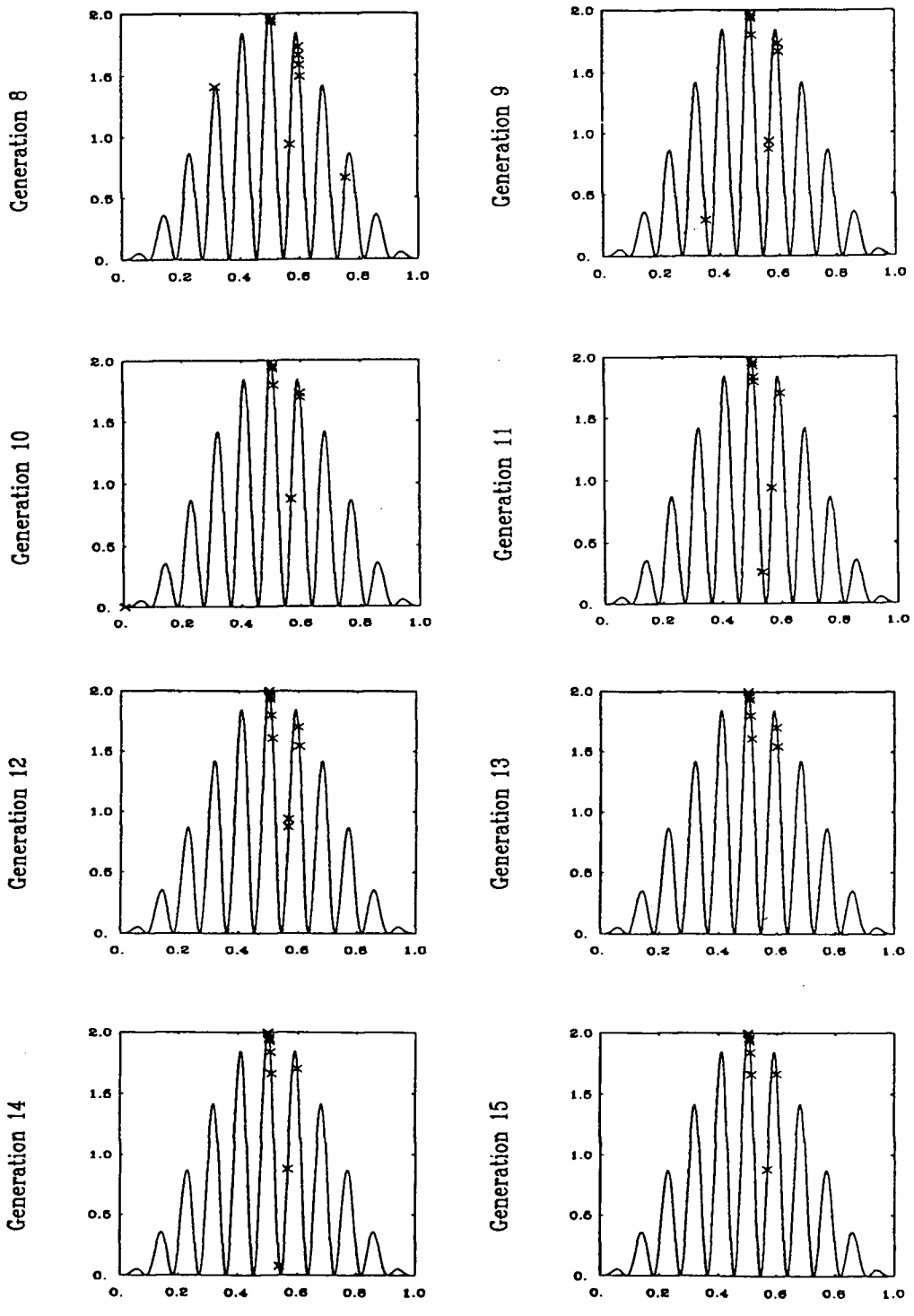

Figure 2.4: GA generations (continued)

population is generated randomly and the population size is kept constant throughout the process. The algorithm only requires payoff information (fitness) for each of the string, without the need for assumptions such as differentiability, thus making it very useful for discontinuous surfaces.

Genetic algorithms are inherently parallel. Indeed, all strings or individuals in a population evolve simultaneously without central coordination. To realize their full potential, they must be implemented on parallel computer architectures.

Chapter 3

System Identification

3.1 Background

Although a variety of techniques have been developed for system identification, none has proven to be effective in all domains. It would be nice to have a method that is sufficiently robust, that is, could be used on a broad class of problems. GAs have been used on a variety of problems as have been reported in [16,17,15,14]. In this chapter the algorithms are going to be applied to both discrete and continuous time systems. But first we look at some previous work in this area.

Etter et. al. [11] studied the system below and modelled it as having only two poles.

$$y(t) = \frac{1 + 10q^{-1}}{1 - 1.2q^{-1} + 0.6q^{-2}} u(t) \tag{3.6}$$

They identified a_1 and a_2 with the input as a white noise and used population size of 11. They showed that the GA did better in locating the true values than a random search did.

Das and Goldberg [8] worked with system of the form

$$y(t) = \frac{b_0 + b_1 q^{-1} + b_2 q^{-2}}{1 + a_1 q^{-1} + a_2 q^{-2}} \left(u(t) + e(t)\right) \tag{3.7}$$

with $b_0 = -0.2$, $b_1 = 0.1$, $b_2 = 0.4$, $a_1 = -1.6$ and $a_2 = 0.95$. The system they used was non-minimum phase and very oscillating ($\xi = 0.04$). They successfully identified the five parameters with the input as a PRBS signal and $e(t)$ as Gaussian noise with variance equal to 10% of the input.

Smith and DeJong [22] used GA to calibrate a nonlinear model of US. migration patterns.

There has been one application of GA for continuous time systems. Goldberg [12] identified mass-spring system with small damping ($\xi = 0.05$)

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = f(t) \tag{3.8}$$

where $m = 1.0$, $c = 0.1$ and $k = 1.0$. The force function, $f(t)$, was a two step staircase function and he identified directly the parameters of the continuous time system, m, c and k.

All the applications so far have been on open loop systems and for the discrete time systems have identified the parameters of the models which RLS can easily do. Nobody has seen the ability of the GA to identify directly the poles and zeros. When estimating poles and zeros with conventional estimation methods the problem is that the system is no longer linear in the parameters. Standard algorithms do not identify directly the poles and zeros. They change the system into a concatenation of second order systems and then calculate the poles and zeros for each 2^{nd} order block [24]. GAs on the other hand can directly identify the poles and zeros. There is really no difference from GA's point of view whether it is identifying the poles and zeros or the parameters. All it needs is a fitness value to assign to each string. The advantage of knowing the poles and zeros is simpler controller design as can be seen in Chapter 4.

GAs could also be used to identify physical parameters, like Goldberg did in [12] for a mass-spring system. For instance, using this method the friction coefficient in a motor drive could be identified directly. Traditionally discrete time estimation is used which results in coefficients that are nonlinearly dependent upon the sampling time. Because of GAs ability to deal with nonlinearity they can be used to identify continuous time systems.

The simulations were performed on Pulp and Paper Centre μVAX. Programs were written in PASCAL for the Genetic Algorithm and in FORTRAN for the RLS part.

3.2 Discrete time identification

Consider the system

$$A(q^{-1})y(t) = B(q^{-1})u(t-d) + C(q^{-1})e(t) \tag{3.9}$$

Where A,B and C are polynomials in the backward shift operator, q^{-1}, i. e. $y(t-1) = q^{-1}y(t)$ and y, u and e are the output, input and noise respectively. The noise $e(t)$ is a normally distributed random sequence with zero mean and a unit variance (σ_e^2). The polynomials A and C are assumed to be monic. The objective is to estimate $A(q^{-1})$, $B(q^{-1})$ and the delay d, when given the input $u(t)$ and the output $y(t)$. The estimates are denoted by '^'. Two sequences $\varepsilon(t)$ and $\eta(t)$, can be defined, for calculating how well the estimates fit the system, as:

$$\hat{A}(q^{-1})y(t) = \hat{B}(q^{-1})u(t-\hat{d}) + \varepsilon(t) \tag{3.10}$$

or

$$\eta(t) = y(t) - \hat{y}(t) \tag{3.11}$$

with

$$\hat{A}(q^{-1})\hat{y}(t) = \hat{B}(q^{-1})u(t-\hat{d})$$

Then we try to minimize $E[\varepsilon^2(t)]$ or $E[\eta^2(t)]$. The first case corresponds to the least-squares case and has a search space which is quadratic, the second is akin to the Instrumental Variable (IV) case and has a highly nonlinear search space.

Depending on the method used, the fitness function is chosen as

$$F(t) = \sum_{i=0}^{w} M - (\varepsilon(t-i))^2 \tag{3.12}$$

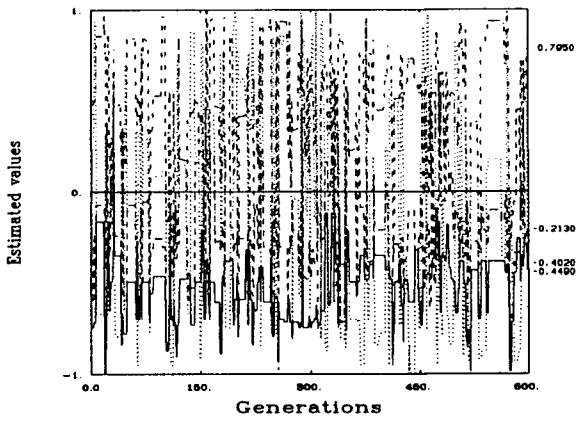

Figure 3.5: Window size = 5

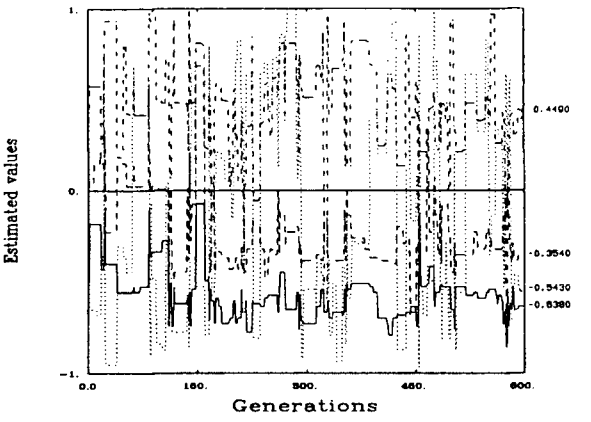

Figure 3.6: Window size = 10

or as

$$F(t) = \sum_{i=0}^{w} M - (\eta(t-i))^2 \tag{3.13}$$

where M is a bias term needed to ensure a positive fitness as explained in Chapter 2 and w is the window size or the number of time steps the fitness is accumulated over, with a effect akin to that of the forgetting factor in RLS. The effect of different window sizes can be seen in Figures 3.5 to 3.8 where the algorithm is run on a system with PRBS input and colored noise. For the moment just assume that the figures show some parameters of a second order system. From these figures it can be seen that the variance of the parameter estimates reduces as the window size increases. But there is a price to pay for increasing the window size. Implementing these fitness functions is expensive in terms of CPU time. Because of the nature of the algorithm, (i. e. coding, probabilistic transfer rules, etc.) no recursive version of the fitness function exists. So at every generation the algorithm has to calculate the estimated output for the whole window which makes the execution time proportional to the window size. There is also an advantage of not having a recursive fitness function. That means that

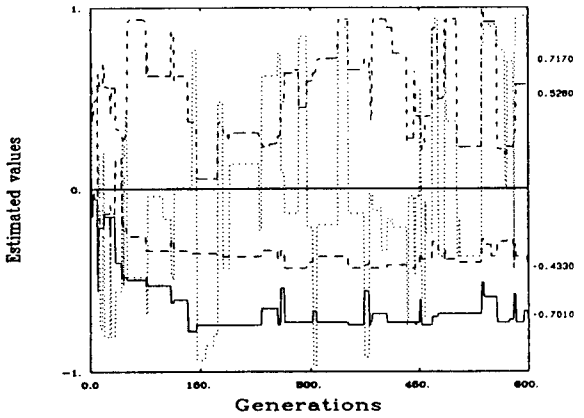

Figure 3.7: Window size = 20

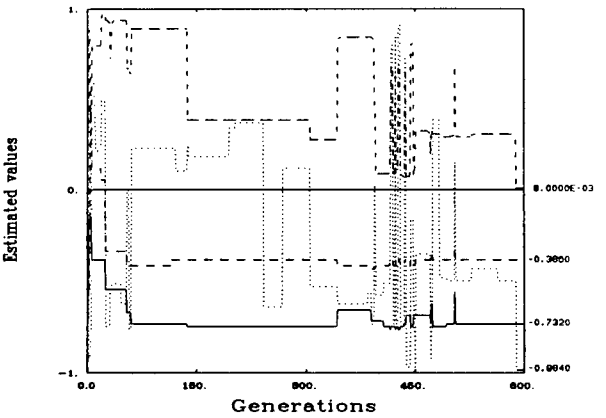

Figure 3.8: Window size = 30

the algorithm does not have to wait for new input-output data before coming up with new estimates. It can actually iterate as often as one likes for each sample but there is some upper limit because of time constraints and in the window the input has to be persistently excited (see Chapter 4, Section 4.4.4) for the algorithm to converge to a certain value. Figures 3.9 to 3.11 show parameter estimates for different number of trials (generations per sample) for a system with PRBS input and no noise. It is seen that the algorithm actually needs fewer generations (200,175,150) to converge as the number of trials increases from 1 to 3 and hence number of data points (200,87,50). But there should be some limit on number of trials as can be seen from Figure 3.12 where the algorithm uses 30 trials for each data it needs about 2250 generations to converge or 75 samples.

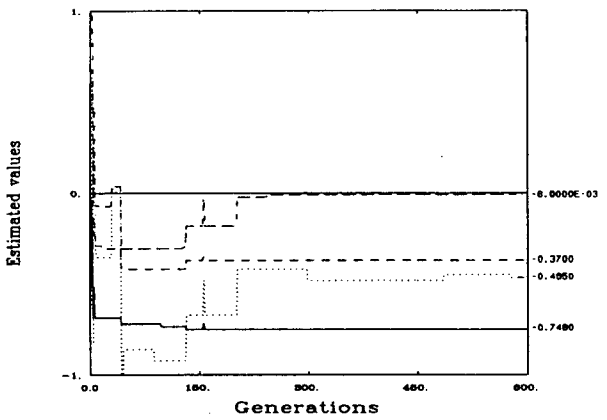

Figure 3.9: Number of trials = 1

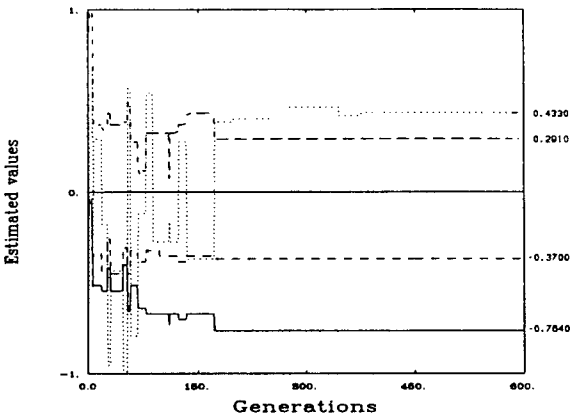

Figure 3.10: Number of trials = 2

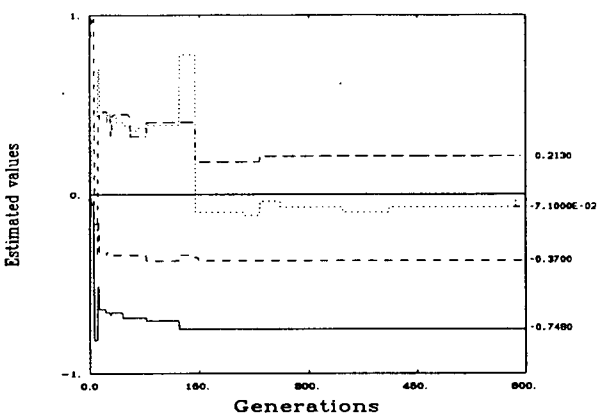

Figure 3.11: Number of trials = 3

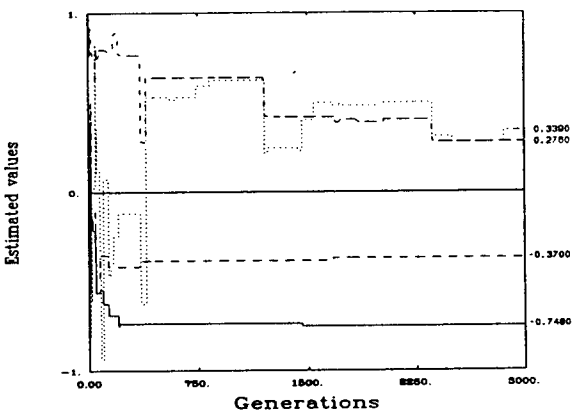

Figure 3.12: Number of trials = 30

3.2.1 Parameter identification

The system of Equation 3.9 can be described by the following polynomials

$$\begin{aligned} A(q^{-1}) &= 1 + a_1 q^{-1} + \cdots + a_n q^{-n} \\ B(q^{-1}) &= b_0(1 + b_1 q^{-1} + \cdots + b_n q^{-n}) \\ C(q^{-1}) &= 1 + c_1 q^{-1} + \cdots + c_n q^{-n} \end{aligned} \tag{3.14}$$

The GA can be used to identify the parameters in A and B and the delay, using either Equation 3.12 or 3.13 as a fitness function. For a second order stable system it gives a search space for a_1 and a_2 of the form seen in Figure 3.13. If the system is also inversely stable the search space for b_1 and b_2 is of the same form too.

3.2.2 Pole-zero identification

Because they do not require linearity in the parameters, genetic algorithms can directly identify the poles and zeros of the system. In pole-zero form, the plant can be written as:

$$\begin{aligned} A(q^{-1}) &= (1 - p_1 q^{-1})(1 - \bar{p}_1 q^{-1}) \cdots (1 - p_m q^{-1})(1 - \bar{p}_m q^{-1}) \\ B(q^{-1}) &= b_0(1 - z_1 q^{-1})(1 - \bar{z}_1 q^{-1}) \cdots (1 - z_m q^{-1})(1 - \bar{z}_m q^{-1}) \end{aligned} \tag{3.15}$$

Where $m = n/2$ if n is even and $m = (n+1)/2$ if n is odd. The parameters, $\bar{p}_m$ and $\bar{z}_m$ will be zero if n is odd. It can also be reparameterized so that a complex conjugate poles or two real poles will be represented by two parameters.

$$\begin{aligned} A(q^{-1}) &= (1 - (\alpha_1 \pm \beta_1)q^{-1}) \cdots (1 - (\alpha_m \pm \beta_m)q^{-1}) \\ B(q^{-1}) &= b_0(1 - (\gamma_1 \pm \delta_1)q^{-1}) \cdots (1 - (\gamma_m \pm \delta_m)q^{-1}) \end{aligned} \tag{3.16}$$

The parameters β_i and δ_i can be either imaginary (complex conjugate poles) or real (two real poles). Because the signs on β and δ are of no importance we can use the signs to decide if the numbers are imaginary or real, negative will mean complex number and

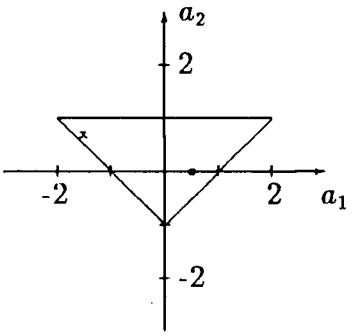

Figure 3.13: Parameters

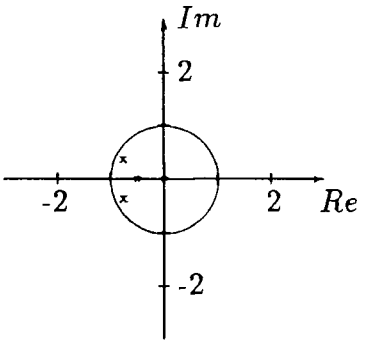

Figure 3.14: Poles and zeros in the complex plane

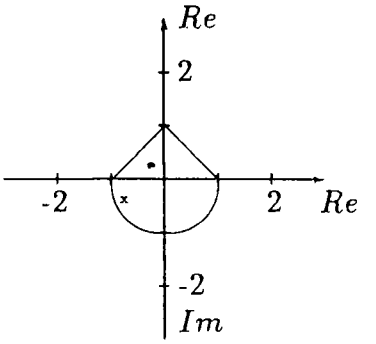

Figure 3.15: Reparameterized complex plane

positive will mean real numbers. As an example

$$\begin{aligned} 1+2q^{-1}+0q^{-2} &= (1-(-1+1)q^{-1})(1-(-1-1)q^{-1}) \equiv [-1,+1] \\ 1+2q^{-1}+2q^{-2} &= (1-(-1+j1)q^{-1})(1-(-1-j1)q^{-1}) \equiv [-1,-1] \end{aligned} \tag{3.17}$$

That gives search space for a stable system of the form seen in Figure 3.15. Where the lower half plane excluding the real axes, represents the complex conjugate poles and the upper half plane represents the real axes.

If the parameters for a second order system are given by [23] (see Figure 3.13) :

$$\begin{aligned} A(q^{-1}) &= 1.0-1.5q^{-1}+0.7q^{-2} \\ B(q^{-1}) &= 1.0(1.0+0.5q^{-1}+0.0q^{-2}) \\ C(q^{-1}) &= 1.0-1.0q^{-1}+0.2q^{-2} \end{aligned} \tag{3.18}$$

The poles and zeros are (see Figure 3.14) :

$$\begin{aligned} A(q^{-1}) &= 1.0-(0.75\pm j0.37)q^{-1} \\ B(q^{-1}) &= 1.0(1.0-(-0.25\pm 0.25)q^{-1}) \end{aligned} \tag{3.19}$$

or (see Figure 3.15) :

$$p_{1,2}=[0.75,-0.37] \equiv p_{1,2}=\begin{cases} 0.75+j0.37 \\ 0.75-j0.37 \end{cases} \tag{3.20}$$

and the zeros

$$z_{1,2} = [-0.25, +0.25] \equiv z_{1,2} = \begin{cases} -0.5 \\ 0.0 \end{cases} \tag{3.21}$$

For a stable minimum phase plant, the poles and zeros are inside the unit circle (Figure 3.14), therefore the search space can be limited to be the unit circle or a box enclosing the unit circle. For a nonminimum phase plant some of its zeros will be outside the unit circle, so one has to decide how big the search space is going to be depending on a priori knowledge of the system.

3.2.3 Results

Parameter settings

The crossover rate is chosen so as to give some of the population the opportunity to survive into the next generation without any changes. The mutation rate is chosen such that on the average one string in the population is mutated. Unless stated otherwise the genetic parameters have therefore been chosen as follows [8] :

$$\begin{aligned} p_c &= 0.8 \\ p_m &= 0.01 \\ population &= 100 \end{aligned} \tag{3.22}$$

Second order systems were used so six parameters needed to be identified, that is d and b_0 and then either parameters, b_1, b_2, a_1 and a_2 or poles-zeros, $\alpha_1, \beta_1, \gamma_1$ and δ_1. The delay is coded as two bit string to give 4 choices for the delay and the other parameters are coded as 7 bit strings, making totally a 37 bit string, which leaves the search space with $2^{37} = 1.37\ 10^{11}$ alternatives. The parameters have been concatenated as follows

$$|d|a_1|a_2|b_1|b_2|b_0| \tag{3.23}$$

or

$$|d|\alpha_1|\beta_1|\gamma_1|\delta_1|b_0| \tag{3.24}$$

depending on whether poles-zeros or parameters were identified. Upper and lower bound on the parameters are defined (see Figures 3.13 and 3.14) and the resolution of the coding is calculated using Equation 2.1.

	lower bound	upper bound	# of bits	resolution
d	1	4	2	1
b_0	0.0	2.0	7	0.016
a_1, b_1	-2.0	2.0	7	0.032
a_2, b_2	-1.0	1.0	7	0.016
$\alpha_1, \beta_1, \gamma_1, \delta_1$	-1.0	1.0	7	0.016

Table 3.4: Search space for a stable minimum phase system

Identification with PRBS

A PRBS signal is used as an input for the system of Equation 3.18. The PRBS input has a period of 127 with the bit interval equal to four times the sampling interval. The GA is run for 600 generations and 3 trials are used for each input-output data so 200 samples will be used. The window (forgetting factor) has been set as 30, i. e. the fitness function is calculated for the current input-output and the 30 previous samples.

Figure 3.16 shows the output for the PRBS signal when there is no noise in the system ($\sigma_e^2 = 0$). Figures 3.17[1] and 3.18 show the estimated parameters for each generation using instrumental variable criterion as given in Equation 3.13. The values of the estimates of the last generation are written at the right hand side of the graphs. After about 150 generations or 50 samples the algorithm comes up with unbiased estimates for all parameters except the zeros. Parameters $-\gamma_1$ and δ_1 should both be 0.250 so a bias of 0.321 for δ_1 seems rather big. But if b_1 and b_2 are calculated we

[1] Estimation of $-\alpha_1, \beta_1, -\gamma_1, \delta_1$

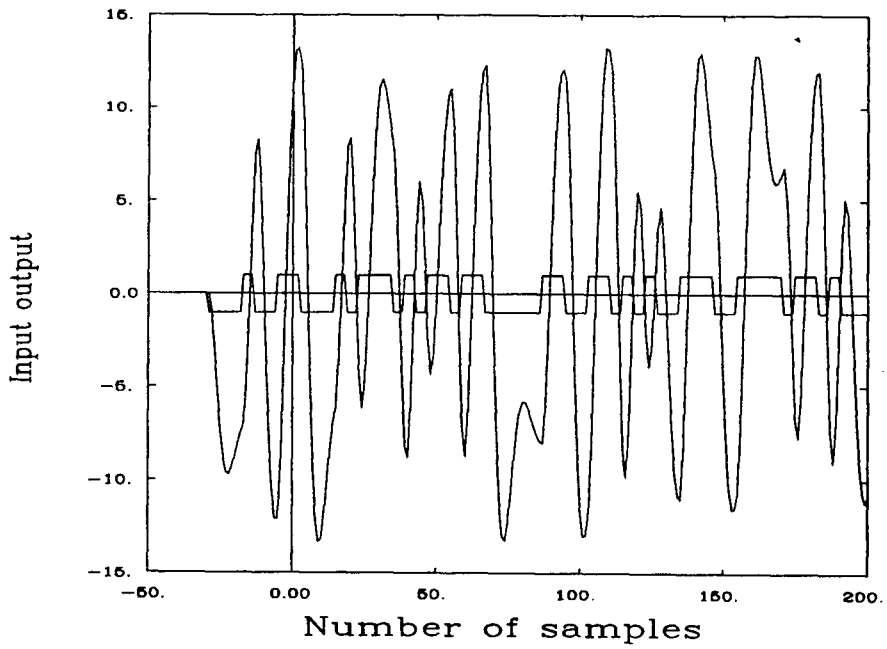

Figure 3.16: PRBS input and output of a system without noise

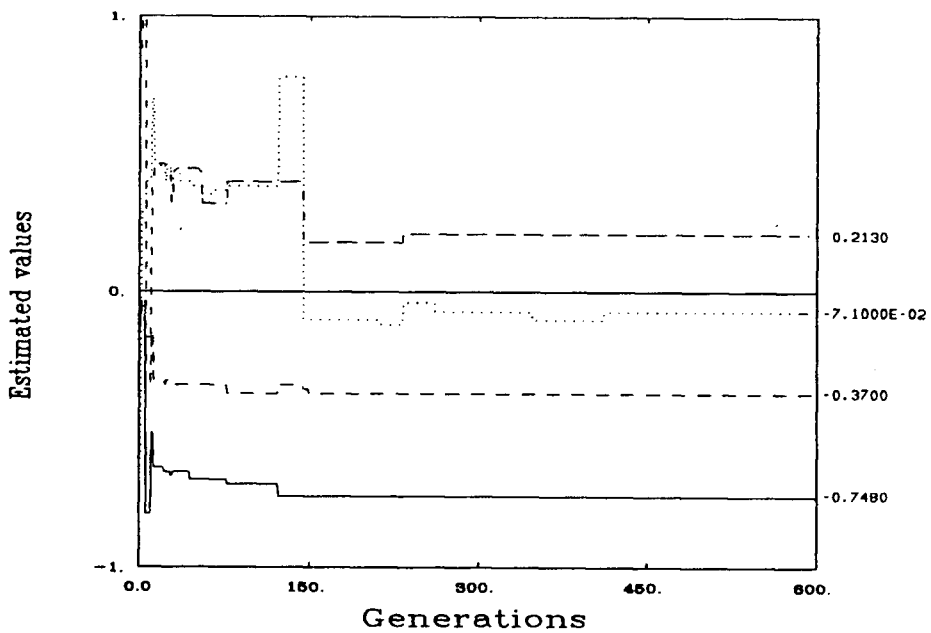

Figure 3.17: Pole-Zero estimate of a system without noise

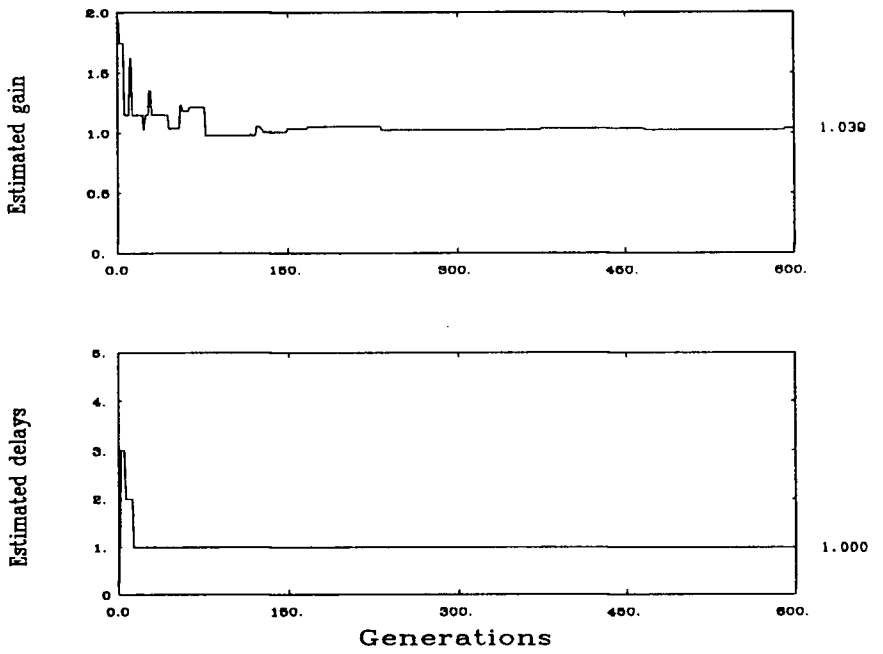

Figure 3.18: Estimation of gain and delay of a system without noise

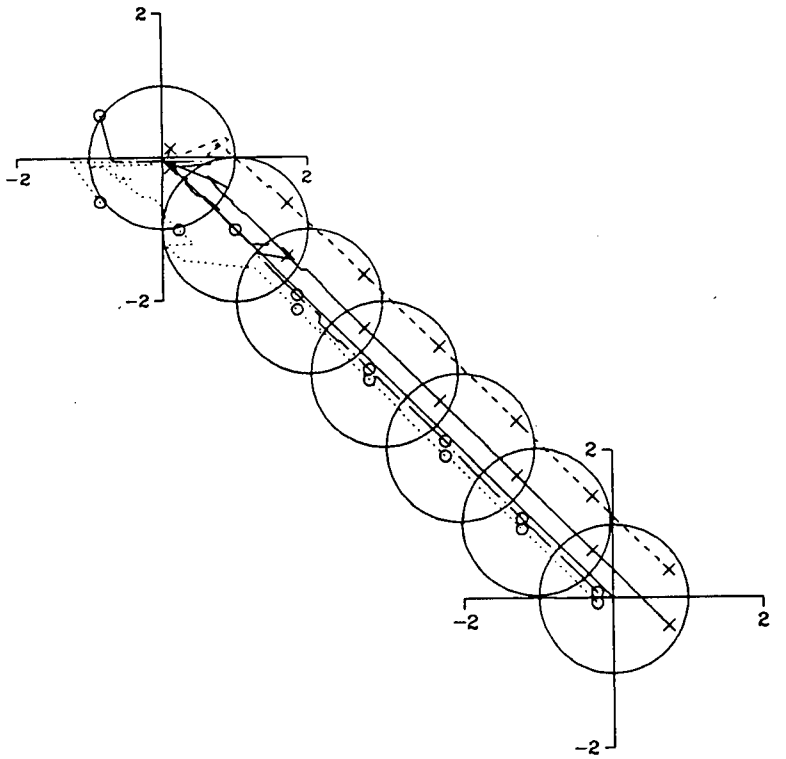

Figure 3.19: Pole zero locations

get 0.426 and 0.050 respectively (true values 0.50, 0.0). The steady state gain of the system is 7.5 so Equation 3.13 is less sensitive to changes in the zeros than the poles and it should also be emphasized that the algorithm does not necessarily converge to THE optimum. "It does its best while learning to do better." Figure 3.19 show how the poles and zeros move around in the complex plane for each generation where the initial generation is at the back and the final generation is in front. The unit circle is also plotted every 100 generations together with the estimates at that point.

3.3 Recursive Least Squares estimation

The actual process, or the system, is assumed to be described by an equation of the form

$$A(q^{-1})y(t) = B(q^{-1})u(t) + C(q^{-1})e(t) \tag{3.25}$$

where $e(t)$ is a white noise with zero mean and a variance σ_e^2 and the polynomials $A(q^{-1})$, $B(q^{-1})$ and $C(q^{-1})$ are given as

$$\begin{aligned} A(q^{-1}) &= 1.0 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a} \\ B(q^{-1}) &= q^{-1}(1.0 + b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}) \\ C(q^{-1}) &= 1.0 \end{aligned} \tag{3.26}$$

Now introduce the parameter vectors θ and $\hat{\theta}$ and a vector, $\varphi(t)$, with the previous inputs and outputs

$$\begin{aligned} \theta &= [a_1, \cdots, a_{n_a}, 1, b_1, \cdots, b_{n_b}]^T \\ \hat{\theta} &= \left[\hat{a}_1, \cdots, \hat{a}_{n_a}, 1, \hat{b}_1, \cdots, \hat{b}_{n_b}\right]^T \\ \varphi(t) &= [-y(t-1), \cdots, -y(t-n_a), u(t-1), \cdots, u(t-n_b-1)]^T \end{aligned} \tag{3.27}$$

The output of the model ($\hat{A}(q^{-1})$ and $\hat{B}(q^{-1})$) can then be written as

$$\hat{y}(t) = \varphi(t)^T \hat{\theta} \tag{3.28}$$

so the system output can be written as

$$y(t) = \hat{y}(t) + \epsilon(t) \tag{3.29}$$

Least squares is a prescription that one should take the value of $\hat{\theta}$ which makes the sum of the squares of the $\epsilon(t)$, $\sum_{t=1}^{N} \epsilon^2(t)$ as small as possible. It can be shown [23] that the LS estimate of θ is

$$\hat{\theta} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y \tag{3.30}$$

where

$$\Phi = \begin{bmatrix} \varphi(1)^T \\ \vdots \\ \varphi(N)^T \end{bmatrix} \qquad Y = \begin{bmatrix} y(1) \\ \vdots \\ y(N) \end{bmatrix} \tag{3.31}$$

The estimate of $\hat{\theta}$ can be made recursive by

$$\begin{aligned} \hat{\theta}_{t+1} &= \hat{\theta}_t + K_{t+1}\varepsilon_{t+1} \\ K_{t+1} &= \frac{P_t \varphi_{t+1}}{\lambda + \varphi_{t+1}^T P_t \varphi_{t+1}} \\ P_{t+1} &= \left(1 - K_{t+1}\varphi_{t+1}^T\right) \frac{P_t}{\lambda} \\ \varepsilon_{t+1} &= y_{t+1} - \varphi_{t+1}^T \hat{\theta}_t \end{aligned} \tag{3.32}$$

where λ is the forgetting factor. For numerical stability, the P matrix is factorized as

$$P_t = S_t S_t^T \tag{3.33}$$

where S is an upper-triangular matrix and S_t is then updated at each iteration [6].

3.3.1 Results

The Recursive Least Square (RLS) algorithm is run for the system of Equation 3.25 with the polynomials A, B and C described by Equation 3.18 with the noise variance, $\sigma_e^2 = 1.0$. The forgetting factor is set to 0.9 to resemble a window for the GA of 30

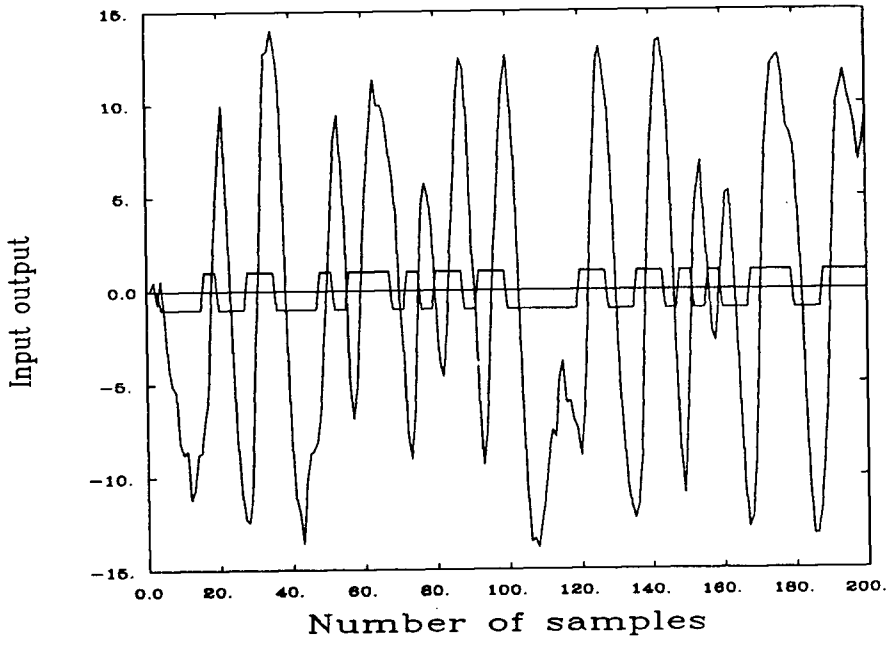

Figure 3.20: PRBS input and output with noise

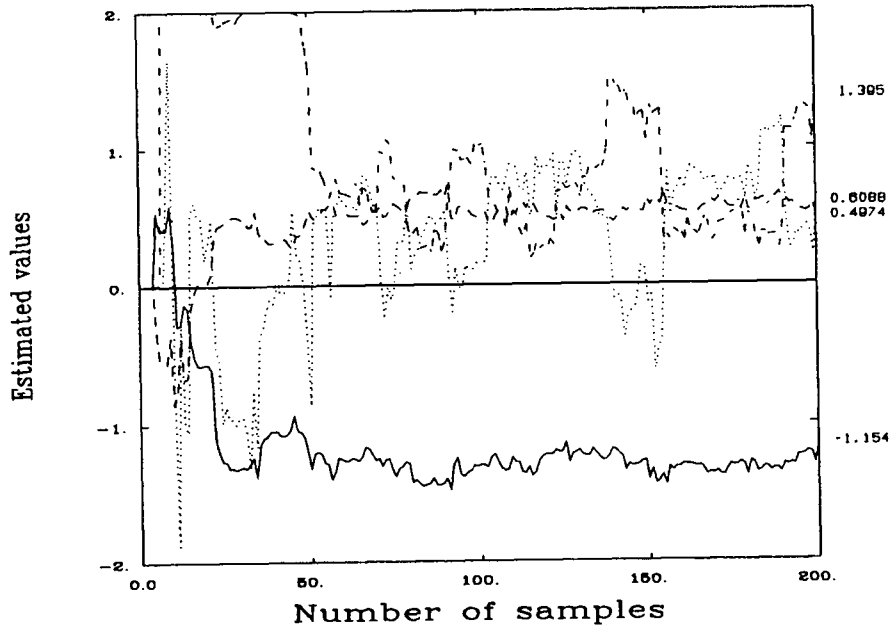

Figure 3.21: Parameter identification using RLS

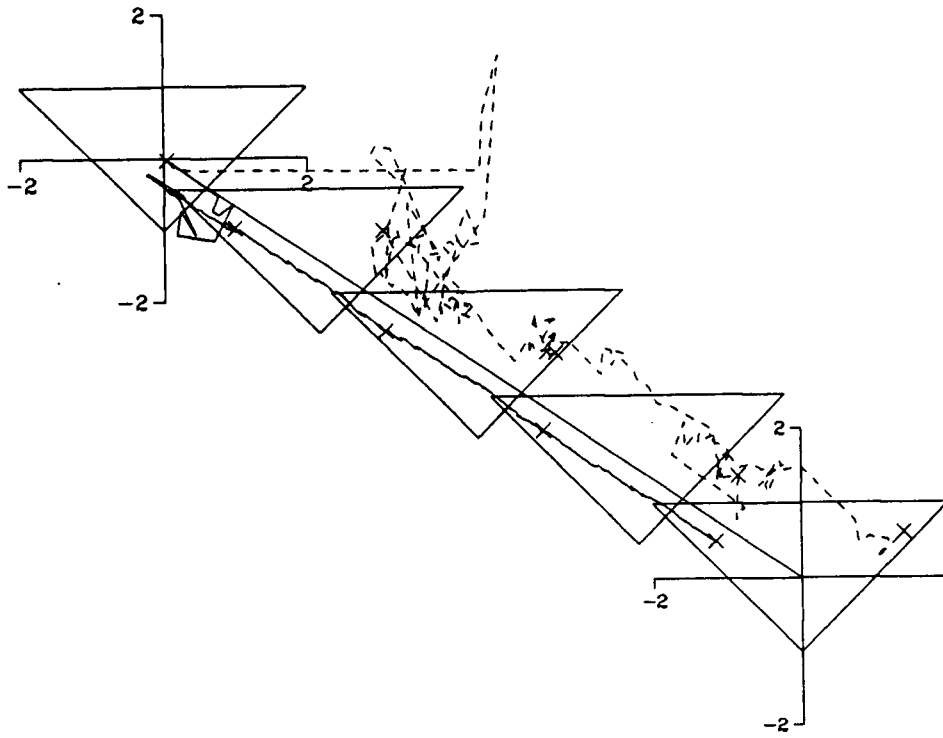

Figure 3.22: Parameters locations

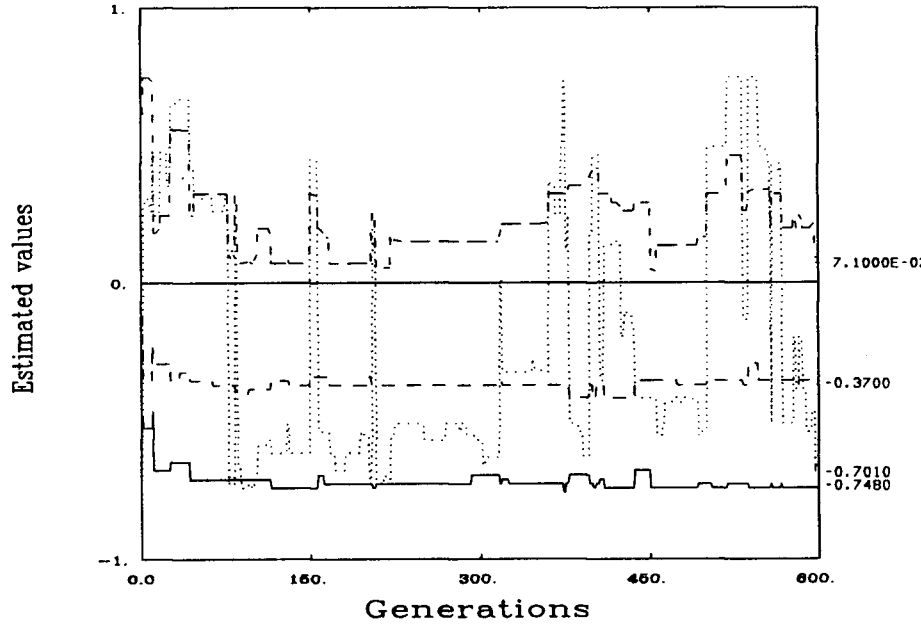

Figure 3.23: GA, pole zero identification

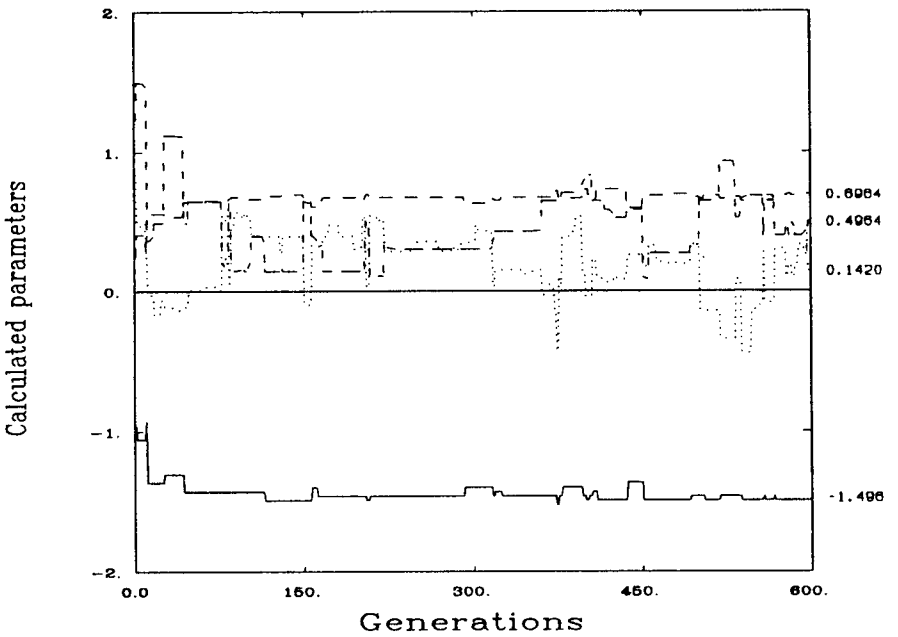

Figure 3.24: GA, parameters identification calculated from the pole zero identification

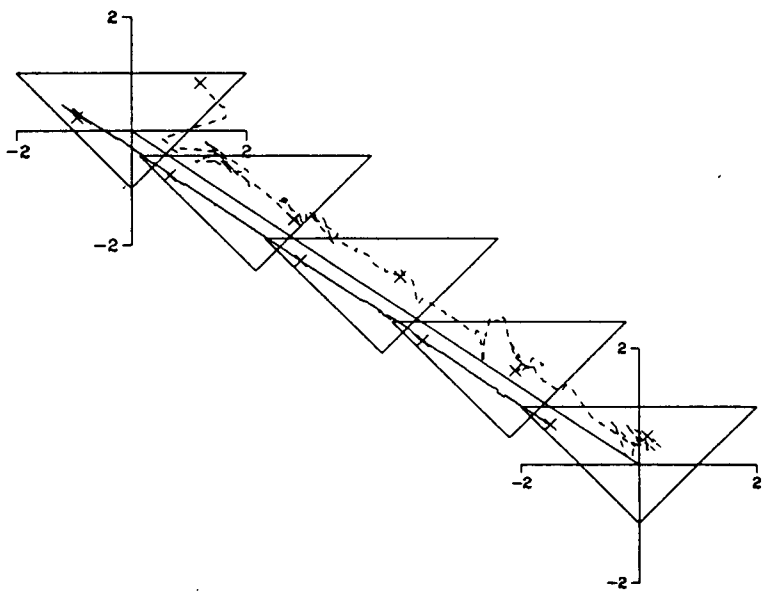

Figure 3.25: GA, parameters locations calculated from the pole zero identification

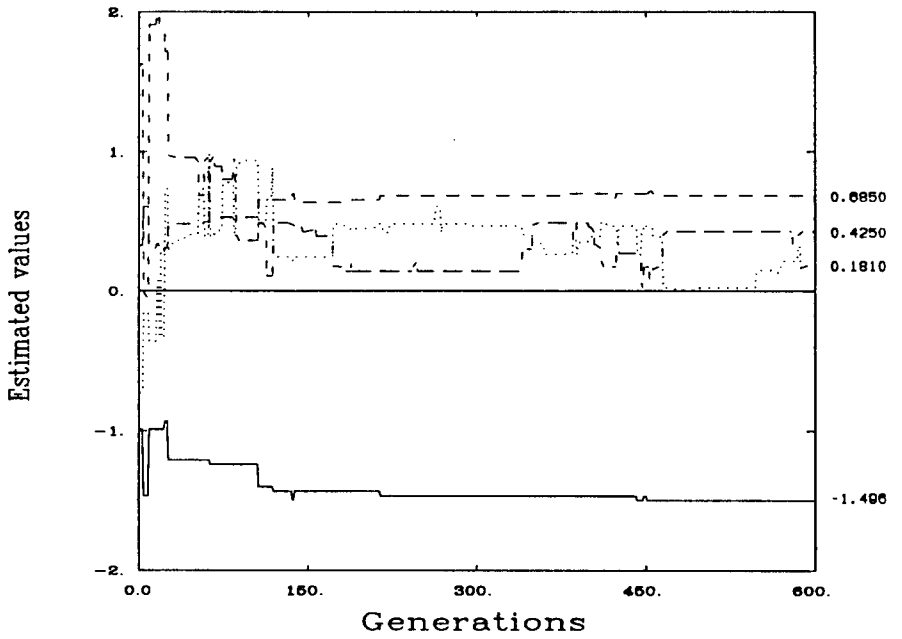

Figure 3.26: GA, parameters identification

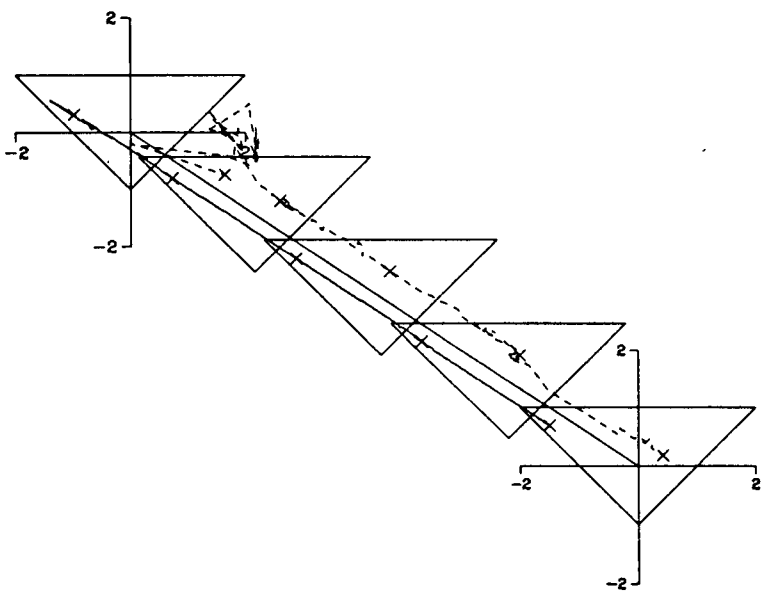

Figure 3.27: GA, parameters locations

steps ($0.9^{30} = 0.04$) and a_1, a_2, b_1 and b_2 are then identified.[2] Figure 3.21 shows the result using the input shown in Figure 3.20. It can be seen that the estimates have rather large variance especially b_1 and b_2 (the long dashed and dotted line respectively). Their value after 200 samples is 1.3957 and 0.6088 respectively but 10 samples before their values were 0.4214 and 1.2141 respectively so it is difficult to say what value they are converging to. The estimates for a_1 and a_2 have also a variance but much smaller and they converge to biased estimates with the final estimates as -1.1543 and 0.4974 respectively. Figure 3.22 shows plot of a_2 as a function of a_1 and b_2 as a function of b_1 with the time axis running out of the page and the triangle for a stable estimates plotted every 50 samples.

To compare those results with the GA, the GA is run identifying the same parameters as those identified by the RLS. That means that the gain, b_0 and the delay, d, are assumed to be known so there are only four parameters to be identified. Using same parameter settings as in Table 3.4 it gives a total string length of 28 bits, so the population size has been set to 50. The GA is run twice, first identifying the poles and zeros and secondly identifying the parameters. The results of the pole-zeros identification is shown in Figure 3.23, it is then converted into the parameters, Figure 3.24 and a 3-D figure is plotted, Figure 3.25. The parameters estimation is shown in Figure 3.26 and the corresponding 3-D figure is shown in Figure 3.27. It can be seen that in both cases the poles have almost zero bias, they are only limited by the resolution of the search space. They converge in about 50 generations for the pole-zero identification but in about 100 generations for the parameters identification or about twice as fast for the pole-zero identification than for the parameters. The zeros converge slowly for both cases but the final estimates are close to the true values (0.5 and 0.0) in both cases.

If GA is then compared to the RLS it can be seen that the RLS needs more than 50

[2]True values from Equation 3.18 are -1.5, 0.7, 0.5 and 0.0 respectively.

samples for the poles to converge but the zeros do not converge, whereas the GA needs between 50 and 100 generations for the poles to converge which means that with 3 trials per sample it needs between 17 and 33 samples and the zeros are slowly converging. So in terms of number of samples the GA converges faster. But as mentioned earlier the fitness function for the GA can not be calculated recursively so the algorithm has to calculate the outputs for all the window and to calculate every output it involves $(n_a + n_b + 1)$ multiplications and $(n_a + n_b)$ additions. The difference in bias of the estimates are mostly caused by different objective (cost) function, the RLS uses a simple least square whereas the GA uses IV alike objective function.

3.4 Continuous time identification

Consider n-th order system with a differential operator $s = \frac{d}{dt}$ and unknown coefficients a_i and b_i

$$y(t) = \frac{b_1 s^{n-1} + \cdots + b_n}{s^n + a_1 s^{n-1} + \cdots + a_n} u(t) \tag{3.34}$$

The goal is to estimate directly the unknown coefficients a_i's and b_i's using the knowledge of the continuous time input and output. Current techniques would find a model of the process with filtered input and output and then use any suitable method like RLS for identification of the parameters of the model and consequently the parameters of the continuous time system [20]. One way to do the estimation using the GA would be to find the parameters $\hat{\theta}$ ($\hat{\theta} = [\hat{a}_1, \cdots, \hat{a}_n, \hat{b}_1, \cdots, \hat{b}_n]^T$) such that the area of the difference between the actual output y and the output of the model $\hat{y}$ over a time window W is the smallest.

$$\min_{\hat{\theta}} \int_W (y(t) - \hat{y}(t, \hat{\theta}))^2 \tag{3.35}$$

At every sampling time T_s the area is calculated for previous W seconds, putting the initial conditions of $\hat{y}$ equal to the initial conditions of y at time $kT_s - W$. As in the

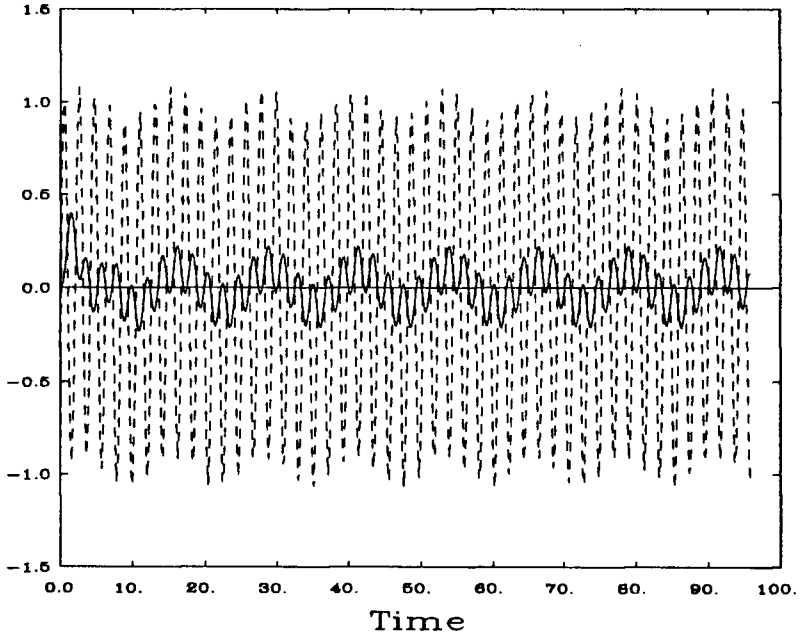

Figure 3.28: Continuous-time-system input and output

discrete time case, the algorithm can be run several times for each sampling time.

3.4.1 Results

This has been implemented in ACSL using the PASCAL subroutines from previous implementations of the algorithm. A second order system has been used with the transfer function:

$$\frac{y(t)}{u(t)} = \frac{0.0s + 1.0}{s^2 + 0.5s + 1.0} \tag{3.36}$$

The GA is used to find all four parameters of the plant. The search space has been defined as in Table 3.5. The total string length is 36 bits so the GA parameters have

	lower bound	upper bound	# of bits	precision
a_1, a_2, b_1, b_2	0.0	12.775	9	0.025

Table 3.5: Search space for continuous time parameters

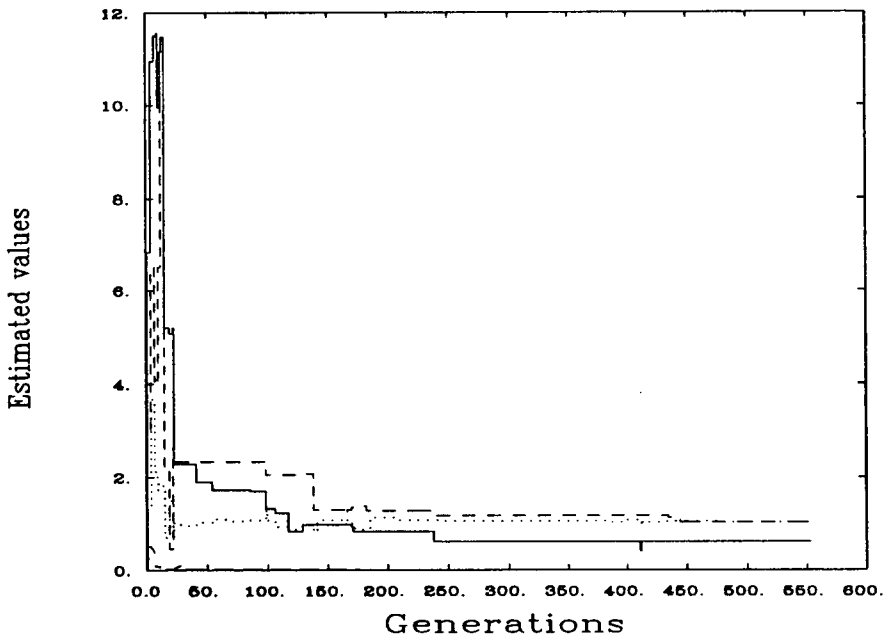

Figure 3.29: Continuous-time parameter estimates

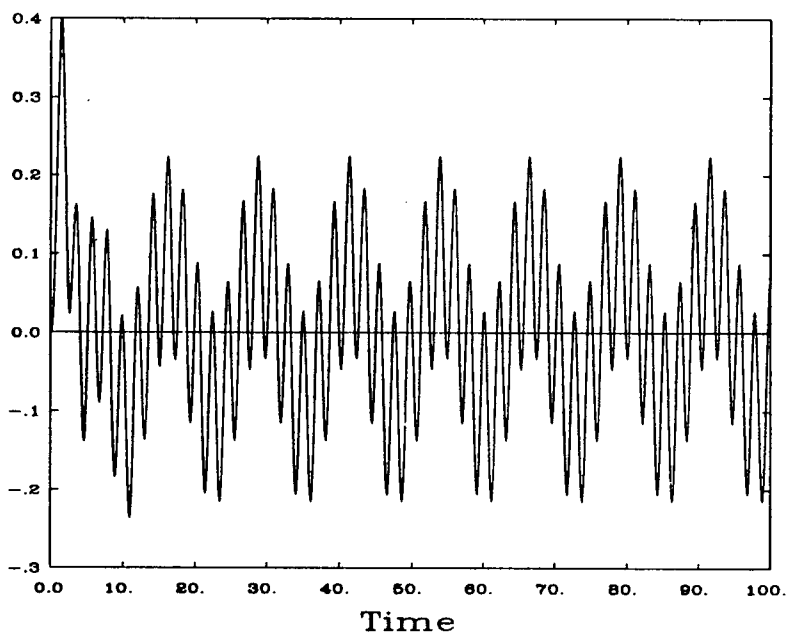

Figure 3.30: Actual output and the output using the final estimates

been set the same as in Equation 3.23. The input $u(t)$ has been chosen as (see the dashed line in Figure 3.28) :

$$u(t) = 0.08 \sin 0.5t + \sin 3.0t \tag{3.37}$$

The input is chosen so as to excite the system both above and below the natural frequency and they should have about the same amplitude in the output for the parameters to converge to the true value. The sampling time T_s is 0.5 seconds and the window is chosen as 8 seconds. The parameter estimates are shown in Figure 3.29. The convergence is slow but they approach the actual values and the estimate after 450 generations, i. e. 150 sampling intervals is

$$\frac{0.000s + 1.050}{s^2 + 0.600s + 1.025} \tag{3.38}$$

which has a natural frequency, $\omega_n = 1.01$ and a damping ratio, $\xi = 0.30$ instead of 1.0 and 0.25 respectively. If both the actual output and the output using the final estimates are plotted (Figure 3.30) one can see that the response is almost identical excluding the transient.

3.5 Friction compensation

Now look at the estimation of physical parameters, namely the friction of a motor. A motor with friction torque T_f and a load disturbance torque T_l can be described by the following model,

$$J\frac{d\omega}{dt} = KI(t) - T_f(t) + T_l(t) \tag{3.39}$$

where J is the total moment of inertia, K is the current constant and I is the motor current. Neglecting the load disturbance and introducing

$$I(t) = u(t) + \frac{\hat{T}_f(\omega)}{K} \tag{3.40}$$

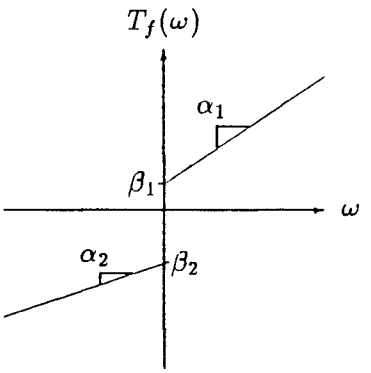

Figure 3.31: Friction model

the motor model can be written as

$$J\frac{d\omega}{dt} = Ku(t) + \left\{\hat{T}_f(\omega) - T_f(\omega)\right\} \tag{3.41}$$

If the estimates are good the term inside the bracket is going to vanish and the model looks like a frictionless motor.

Many models for the friction have been suggested but a model used in [5] has been adopted. The model is:

$$T_f(\omega) = \begin{cases} \alpha_1\omega + \beta_1 & \omega > 0 \\ \alpha_2\omega + \beta_2 & \omega < 0 \end{cases} \tag{3.42}$$

Therefore the estimation of $T_f(\omega)$ requires estimation of four parameters, α_1, β_1, α_2 and β_2. Only two of them can be estimated at a given time depending on whether the angular velocity ω is greater than zero or less than zero. GA can be applied to this problem ones the objective function has been defined. Assuming it is required to minimize the error between the actual angular velocity and the estimated one, the fitness function becomes

$$\min_{\theta} \int_W (\omega(t) - \hat{\omega}(t))^2 \, dt \tag{3.43}$$

where W is a time window over which the objective function is calculated.

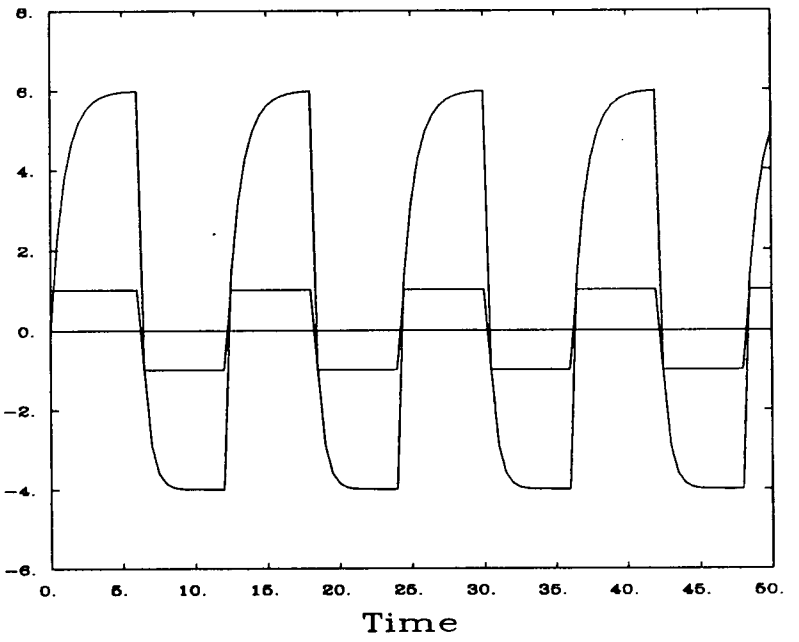

Figure 3.32: Motor input (I) and output (ω)

3.5.1 Results

The parameters of the friction model have been chosen as :

$$\begin{aligned} \alpha_1 &= 0.1 \quad \beta_1 = 0.4 \\ \alpha_2 &= 0.2 \quad \beta_2 = -0.2 \end{aligned} \tag{3.44}$$

and the motor is assumed to have $K = 1.0$ and $J = 0.1$. The parameters that are identified are α_1, β_1, α_2, $-\beta_2$ and they are assumed to lie between 0 and 1.2775 which with a string length of 9 gives precision of 0.0025 for each one of them. When the output is greater then zero α_1 and β_1 are identified and when the output is less than zero the other two, α_2 and β_2 are identified. Therefore two populations are maintained, one for α_1 and β_1 and the other for α_2 and β_2. The string length in each population is 18 so the population size is chosen as 50 for each one of them. The time window W for the integral of Equation 3.43 is chosen as 2 seconds and there are 3 trials for each

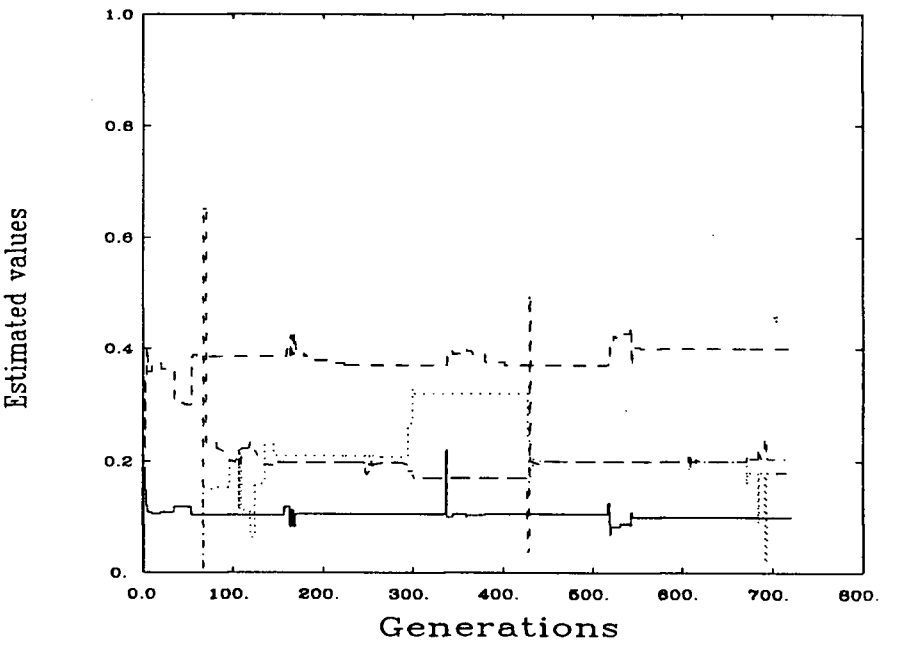

Figure 3.33: Friction parameters identification

sample which is sampled at the rate of 5 per second. The input is a square wave with period equal to 12 seconds, which can be seen together with the output in Figure 3.32. Figure 3.33 shows the estimated parameters using GA and the final value that the GA comes up with is:

$$\begin{aligned} \hat{\alpha}_1 &= 0.100 \quad \hat{\beta}_1 = 0.400 \\ \hat{\alpha}_2 &= 0.205 \quad -\hat{\beta}_2 = 0.180 \end{aligned} \tag{3.45}$$

which have almost zero bias. The algorithm takes less than 100 generations to find approximate estimates for each one of the populations and further refinements are then found along the way.

3.6 Summary

In this chapter it has been demonstrated how GAs can be used to estimate both continuous and discrete time systems and for identifying parameters, poles and zeros or

physical parameters of a system. The GA has proven to be a robust algorithm, whereas in all the applications the basic algorithm (procedures) stays the same. The only difference between these different applications is a different routine to calculate the fitness function.

In all the applications shown in this chapter, the GA has been able to converge towards the actual values of the parameters. In most cases it gives unbiased estimates but in some cases it takes time, like the identification of the zeros, primarily because the objective function is not as sensitive to changes in the zeros as changes in the poles and the GA is only looking for a good solution not necessarily the best.

In comparison to some widely known identification technique, RLS, the GA perform as well or even better in terms of number of samples required to converge. But GA can also easily be used on problem where RLS is difficult or even not possible to use because of the requirement of linearity in the parameters of the system.

Chapter 4

Controller design

4.1 Controller

There are a large variety of adaptive design techniques. In this thesis I have chosen to use an indirect scheme and I am not identifying the stochastic part, so an adaptive pole placement design has been chosen. It is a simple design method that makes use of the knowledge about the poles and zeros to obtain a desired transfer function or desired response.

A SISO plant is described by an ARMAX[1] model:

$$y(t) = \frac{B(q)}{q^d A(q)} u(t) + \frac{C(q)}{A(q)} e(t) \tag{4.46}$$

The control law for a two-degree of freedom pole-placement controller (Figure 4.34) can be written as

$$R(q)u(t) = -S(q)y(t) + T(q)y_r(t) \tag{4.47}$$

where y_r is the reference signal and R is assumed to be monic. To simplify the writing in the analysis that follows, the arguments of polynomials are suppressed. If Equation (4.47) is written in terms of the system input u(t) and then put into Equation (4.46) the closed loop system becomes

$$y(t) = \frac{TB}{q^d AR + BS} y_r(t) + \frac{q^d RC}{q^d AR + BS} e(t) \tag{4.48}$$

[1] Auto Regressive A(q), Moving Average C(q), eXternal signal B(q)

The desired closed loop transfer function is given by

$$H(q) = \frac{B_m(q)}{A_m(q)} \tag{4.49}$$

or

$$\frac{TB}{q^d AR + BS} = \frac{B_m}{A_m} \tag{4.50}$$

By choosing the desired closed loop transfer function and estimating the plant, the controller design has been reduced to finding the polynomials R, S and T that satisfy Equation 4.50. Some of the process zeros can be cancelled in the design.

If B is factorized as $B = B^+B^-$, where B^+ is cancelled stable process zeros and B^- is uncancelled process zeros, B_m must be written as, $B_m = B^-B'_m$ The parts of B that are not factors of B_m must be factors of $q^dAR + BS$ so B^+ must be a factor of R. Therefore R must be written as $R = B^+\bar{R}$. Generally the degree of $q^dAR + BS$ is higher than A_m so there must be some cancellation on the transfer function. So the Diophantine equations becomes

$$q^dA\bar{R} + B^-S = A_oA_m \tag{4.51}$$

and T can be found from

$$T = B'_mA_ot_0 \tag{4.52}$$

where t_0 is to ensure proper gain and A_o is the observer polynomial that is cancelled in the transfer function [1]. In order to design the controller we need to solve the Diophantine Equation 4.51 for $\bar{R}$ and S. In order to find a unique solution the degree of S has to be less than the sum of d and degree of A. We choose $degS = degA + d - 1$. Using the causality conditions ($degR \geq degS$) the degree of A_0 can be found from Equation 4.51 to be

$$degA_o \geq 2(degA + d) - degA_m - degB^+ - 1 \tag{4.53}$$

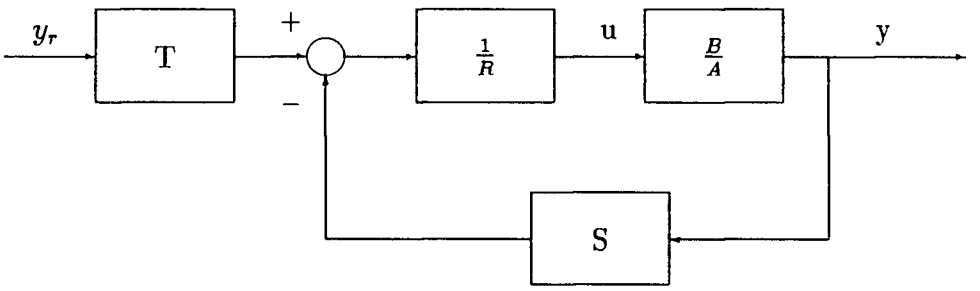

Figure 4.34: Two-degree of freedom controller

The degree of R can be found from Equation 4.51 as

$$deg\bar{R} = degA_o + degA_m - degA - d \tag{4.54}$$

This procedure assumes that the true plant A, B is known. When the true plant is not known, one can use the GA to do the estimation of the plant $\hat{A}$, $\hat{B}$ and then design an indirect adaptive control scheme as shown in Figure 4.35.

4.2 Parameter based design

By assuming that we have knowledge of A and B^- and that A_o is chosen we can solve Equation 4.51 for $\bar{R}$ and S. Define $n = degA$, $m = degB^-$, $k = degA_o$ and $j = degA_m$, then

$$\begin{aligned}
A(q) &= q^n + a_1 q^{n-1} + \cdots + a_n \\
B^-(q) &= b_0 (q^m + b_1 q^{m-1} + \cdots + b_m) \\
A_o(q) &= q^k + a_{1o} q^{k-1} + \cdots + a_{ko} \\
A_m(q) &= q^j + a_{1w} q^{j-1} + \cdots + a_{jw} \\
S(q) &= s_0 q^{n+d-1} + s_1 q^{n+d-2} + \cdots + s_{n+d-1} \\
\bar{R}(q) &= q^{k+j-n-d} + r_1 q^{k+j-n-d-1} + \cdots + r_{k+j-n-d}
\end{aligned} \tag{4.55}$$

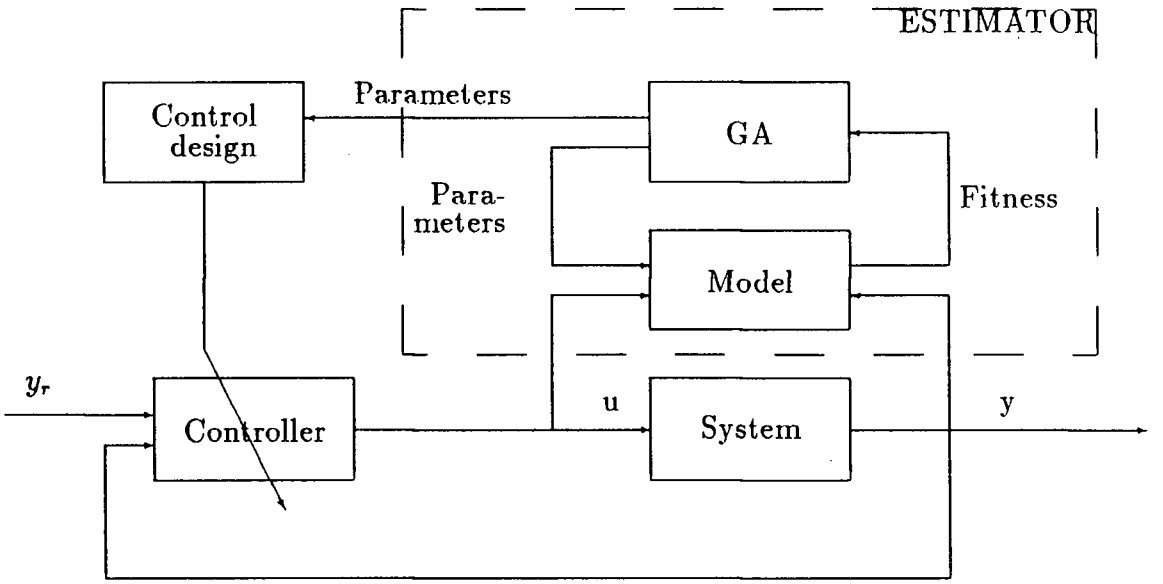

Figure 4.35: GA adaptive controller

In matrix form 4.51 can be written as

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ a_1 & \ddots & 0 & \vdots & & \vdots \\ \vdots & \ddots & 1 & 0 & \cdots & 0 \\ a_n & & a_1 & 1 & 0 & 0 \\ 0 & \ddots & \vdots & b_1 & \ddots & 0 \\ 0 & 0 & a_n & \vdots & \ddots & 1 \\ 0 & \cdots & 0 & b_m & & b_1 \\ \vdots & & \vdots & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & b_m \end{bmatrix} \begin{bmatrix} 1 \\ r_1 \\ \vdots \\ r_{k+j-n-d} \\ b_0 s_0 \\ \vdots \\ b_0 s_{n+d-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ a_{1o} & \ddots & 0 \\ \vdots & \ddots & 1 \\ a_{ko} & & a_{1o} \\ 0 & \ddots & \vdots \\ 0 & 0 & a_{ko} \end{bmatrix} \begin{bmatrix} 1 \\ a_{1w} \\ \vdots \\ a_{jw} \end{bmatrix} \tag{4.56}$$

where the upper right hand zero matrix has $degA_o - (degA + d - degA_m + degB^- - 1)$ number of rows and the lower left hand zero matrix has d number of rows. If A and B^- have no common factors, Equation 4.56 can be solved for r_i and s_i, to design the

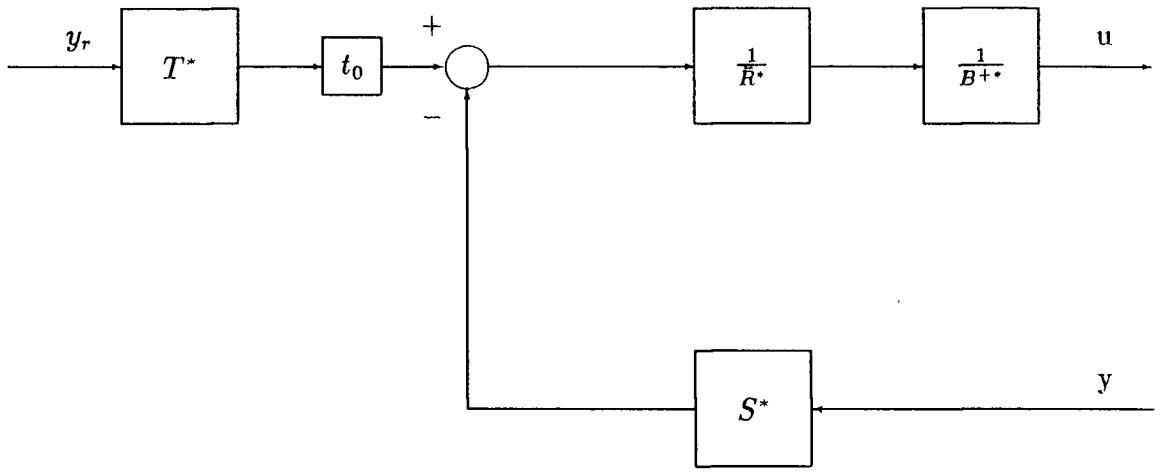

Figure 4.36: Parameter Controller

controller. Equation 4.50 in the backward shift operator (q^{-1}) becomes

$$\frac{q^{-(j-degB_m-(n+d-degB))}T^*q^{-(n+d-degB)}B^*}{A^*R^* + q^{-(n+d-degB)}B^*q^{-(k-(2(n+d)-j-degB^+-1))}S^*} \tag{4.57}$$

where * denotes that the polynomial is in the backward shift operator. If $degA_m - degB_m = degA + d - degB$ and the equality in Equation 4.53 is true, the transfer function becomes

$$\frac{T^*q^{-(n+d-degB)}B^*}{A^*R^* + q^{-(n+d-degB)}B^*S^*} \tag{4.58}$$

The controller can then be implemented as shown in Figure 4.36.

4.3 Pole-zero based design

Wittenmark and Evans [24] have come up with a pole-zero placement algorithm based on pole-zero parameterization. What they have done is to model a high order system as a concatenation of second order systems, identify the parameters of the second order subsystems and then calculate the poles and zeros. The solution of the Diophantine equation corresponds then to a solution of a linear triangular system of equations where the controller parameters can be calculated recursively.

Assume that the poles and zeros of A and B respectively, are known and the poles of A_m and A_o are chosen.

$$\begin{aligned} A(q) &= (q-p_1)\cdots(q-p_n) \\ B^-(q) &= b_0(q-z_1)\cdots(q-z_m) \\ A_m(q) &= (q-p_{1w})\cdots(q-p_{jw}) \\ A_o(q) &= (q-p_{1o})\cdots(q-p_{ko}) \end{aligned} \tag{4.59}$$

As before Equation 4.51 needs to be solved for $\bar{R}$ and S. The solution can be split up into three parts: One for finding coefficients of the terms with degree less than d, the second for finding coefficients of the terms with degree $m+n+d$ and higher and the third one for finding the rest of the coefficients. The polynomial $\bar{R}$ can be split up into two parts, one with terms of degree lower than m and the other with terms of degree m and higher.

$$\begin{aligned} \bar{R}(q) &= q^m R'' + R' \\ &= q^m(q^{k+j-m-n-d} + r''_1 q^{k+j-m-n-d-1} + \cdots + r''_{k+j-m-n-d}) \\ &\quad + \bar{r}_0 q^{m-1} + \bar{r}_1 q^{m-2} + \cdots + \bar{r}_{m-1} \end{aligned} \tag{4.60}$$

where R'' can be calculated recursively by long division of Equation 4.51. Similarly for the S polynomial.

$$\begin{aligned} S &= q^d S' + S'' \\ &= q^d(\bar{s}_0 q^{n-1} + \cdots + \bar{s}_{n-1}) + s''_0 q^{d-1} + \cdots + s''_{d-1} \end{aligned} \tag{4.61}$$

where S'' can be found by long division. By using the fact that $A(p_i) = 0$ and $B^-(z_i) = 0$ the coefficients in R' and S' can be calculated separately by evaluating Equation 4.51 for $q = p_i$ and $q = z_i$.

$$B^-(p_i)p_i^d S'(p_i) = A_o(p_i)A_m(p_i) - B^-(p_i)S''(p_i) \tag{4.62}$$

$$z_i^d A(z_i)R'(z_i) = A_o(z_i)A_m(z_i) - z_i^d A(z_i)z_i^m R''(z_i) \tag{4.63}$$

where z_i are uncancelled process zero ($z_i \in B^-$). By reparameterizing R' and S' into Newton coefficients, solution to Equations 4.62 and 4.63 can be found by solving a set of triangular system of equations.

$$R'(q) = \sum_{i=0}^{m-1} r'_i g_i(q) \tag{4.64}$$

$$S'(q) = \sum_{i=0}^{n-1} s'_i f_i(q) \tag{4.65}$$

where

$$g_i(q) = \begin{cases} 1 & i = 0 \\ \prod_{j=1}^{i}(q - z_j) & 1 \le i \le m-1 \end{cases} \tag{4.66}$$

$$f_i(q) = \begin{cases} 1 & i = 0 \\ \prod_{j=1}^{i}(q - p_j) & 1 \le i \le n-1 \end{cases} \tag{4.67}$$

and

$$g_i(z_k) = \begin{cases} 0 & 1 \le k \le i \\ \prod_{j=1}^{i}(z_k - z_j) & i+1 \le k \le m \end{cases} \tag{4.68}$$

$$f_i(p_k) = \begin{cases} 0 & 1 \le k \le i \\ \prod_{j=1}^{i}(p_k - p_j) & i+1 \le k \le n \end{cases} \tag{4.69}$$

Now, the parameters of R' and S' can be found by solving the following set of triangular equations

$$\begin{aligned} \frac{A_o(p_1)A_m(p_1)}{p_1^d B^-(p_1)} - \frac{S''(p_1)}{p_1^d} &= s'_0 \\ \frac{A_o(p_2)A_m(p_2)}{p_2^d B^-(p_2)} - \frac{S''(p_2)}{p_2^d} &= s'_0 + s'_1 f_1(p_2) \\ &\vdots \\ \frac{A_o(p_n)A_m(p_n)}{p_n^d B^-(p_n)} - \frac{S''(p_n)}{p_n^d} &= s'_0 + s'_1 f_1(p_n) + \cdots + s'_{n-1} f_{n-1}(p_n) \end{aligned} \tag{4.70}$$

and for R'

$$\begin{aligned}
\frac{A_o(z_1)A_m(z_1)}{z_1^d A(z_1)} - z_1^m R''(z_1) &= r'_0 \\
\frac{A_o(z_2)A_m(z_2)}{z_2^d A(z_2)} - z_2^m R''(z_2) &= r'_0 + r'_1 g_1(z_2) \\
&\vdots \\
\frac{A_o(z_m)A_m(z_m)}{z_m^d A(z_m)} - z_m^m R''(z_m) &= r'_0 + r'_1 g_1(z_m) + \cdots + r'_{m-1} g_{m-1}(z_m)
\end{aligned} \tag{4.71}$$

4.3.1 Multiple poles and zeros

Assume that the algorithm estimates that there is a pole p_i with multiplicity m_i. It implies that is has to find m_i coefficients from the pole. That can be done by replacing m_i equations in Equation 4.62 by the following

$$\begin{aligned}
B^-(p_i)p_i^d S'(p_i) &= A_o(p_i)A_m(p_i) - B^-(p_i)S''(p_i) \\
\frac{d}{dq}\left[B^-(q)q^d S'(q)\right]_{q=p_i} &= \frac{d}{dq}\left[A_o(q)A_m(q) - B^-(q)S''(q)\right]_{q=p_i} \\
&\vdots \\
\frac{d^{m_i-1}}{dq^{m_i-1}}\left[B^-(q)q^d S'(q)\right]_{q=p_i} &= \frac{d^{m_i-1}}{dq^{m_i-1}}\left[A_o(q)A_m(q) - B^-(q)S''(q)\right]_{q=p_i}
\end{aligned} \tag{4.72}$$

from the first equation we can get s'_{i-1} and the last one we get s'_{i-1+m_i}. In the case of multiple zeros we differentiate Equation 4.63 instead of Equation 4.62.

4.3.2 Complex poles and zeros

Complex poles and zeros do not cause any problem because Equations 4.70 and 4.71 can be assumed to be defined on the complex plane. That means that some of the coefficients r_i and s_i will be complex so a filtering of complex signals has to be included (see Figure 4.38 Section 4.3.3). The total signal will however always be real as can be seen if z_1 and z_2 from Equation 4.71 are assumed to be complex conjugate zeros ($\bar{z}_1 = z_2 = a - jb$) and further more assume that $r'_0 = \alpha + j\beta$. Then the first two

equations in Equation 4.71 become

$$\begin{aligned} \alpha + j\beta &= r'_0 \\ \alpha - j\beta &= r'_0 + r'_1(\bar{z}_1 - z_1) \end{aligned} \tag{4.73}$$

so r'_1 is

$$r'_1 = \frac{\beta}{b} \tag{4.74}$$

Because r'_0 is used in combination with $-r'_1 z_1$ the total signal is always a real signal.

$$r'_0 - r'_1 z_1 = \alpha - j\beta - \frac{\beta}{b}(a + jb) = \alpha - \frac{a}{b}\beta \tag{4.75}$$

4.3.3 Implementation

The polynomials in the backward shift operator are

$$\begin{aligned} \bar{R}^* &= \sum_{i=0}^{k+j-m-n-d} r''_0 q^{-i} + q^{-(k+j-n-d-m+1)} \sum_{i=0}^{m-1} r'_i g_i(q^{-1}) q^{-m+1+i} \\ &= R''^* + q^{-(k+j-n-d-m+1)} R'^* \end{aligned} \tag{4.76}$$

$$\begin{aligned} S^* &= \sum_{i=0}^{n-1} s'_i f_i(q^{-1}) q^{-n+1+i} + q^{-n} \sum_{i=0}^{d-1} s''_i q^{-i} \\ &= S'^* + q^{-n} S''^* \end{aligned} \tag{4.77}$$

If the equality in Equation 4.53 holds ($k = 2(n+d) - j - degB^+ - 1$) Equation 4.77 becomes $\bar{R}^* = R''^* + q^{-(n+d-degB)} R'^*$ and the controller can be implemented as shown in Figure 4.37, where R'^* and S'^* are implemented using a ladder network as shown in Figure 4.38.

4.4 Results

In the simulation runs that follow, the input has been chosen to be a square wave with period of 60 steps, the desired closed loop system is supposed to have a deadbeat response and the observer is deadbeat as well. An impulse is used to initialize the GA, i. e. fill up the window.

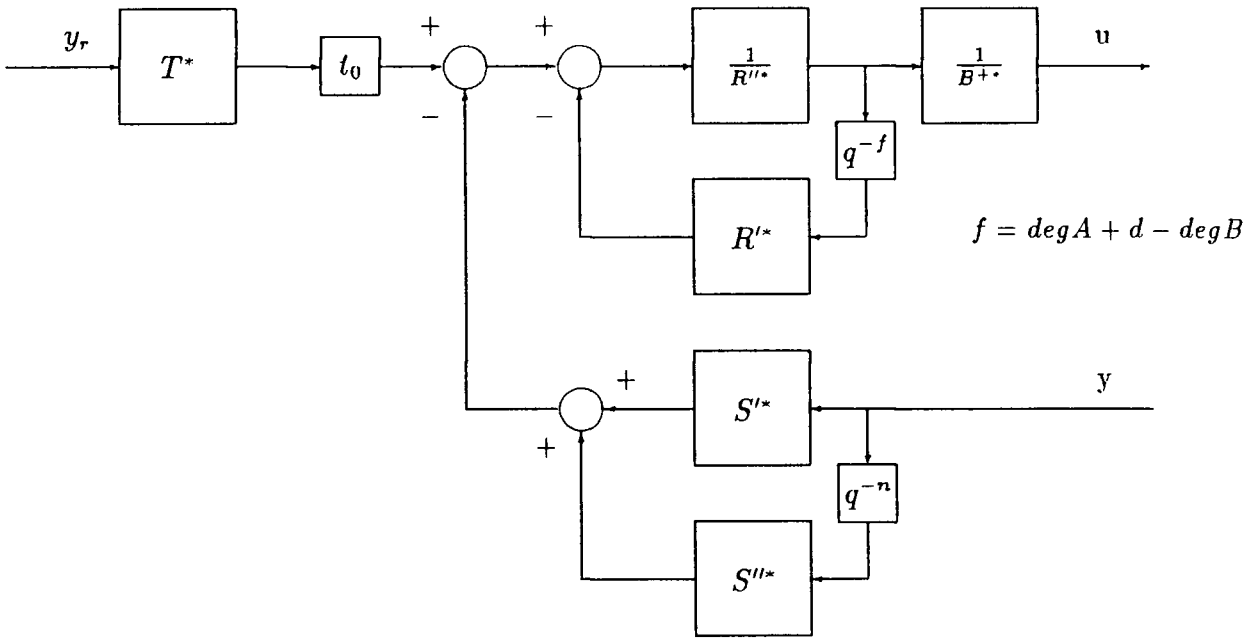

Figure 4.37: Factorized Controller

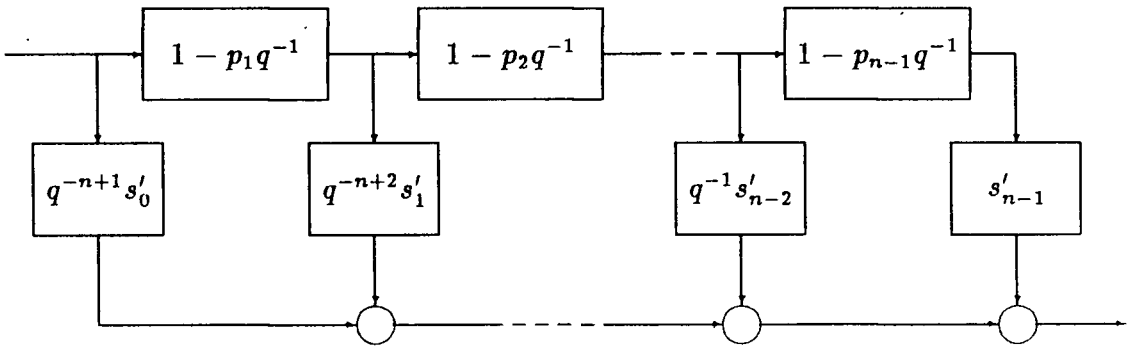

Figure 4.38: Ladder

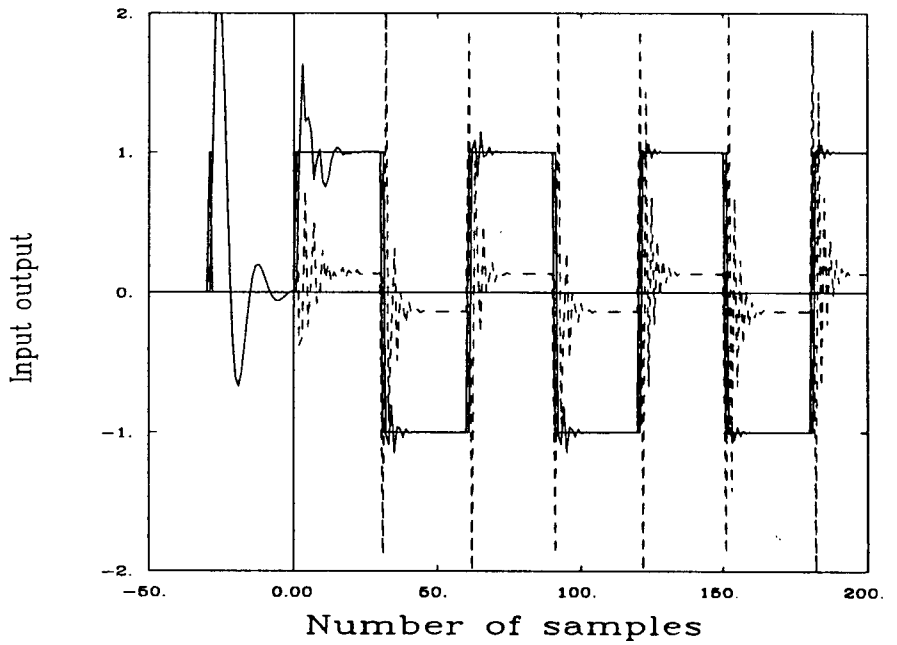

Figure 4.39: Reference input and output of a minimum phase system without noise

4.4.1 Minimum phase plant

The plant to be controlled is the same one as used in Chapter 3.

$$A(q^{-1})y(t) = B(q^{-1})u(t-1) + C(q^{-1})e(t) \tag{4.78}$$

with

$$\begin{aligned} A(q^{-1}) &= 1.0 - 1.5q^{-1} + 0.7q^{-2} \\ B(q^{-1}) &= 1.0(1.0 + 0.5q^{-1} + 0.0q^{-2}) \\ C(q^{-1}) &= 1.0 - 1.0q^{-1} + 0.2q^{-2} \end{aligned} \tag{4.79}$$

The poles and zeros of A and B are identified using the IV criterion from Equation 3.13 using the same parameters settings as in Table 3.4.

Without noise

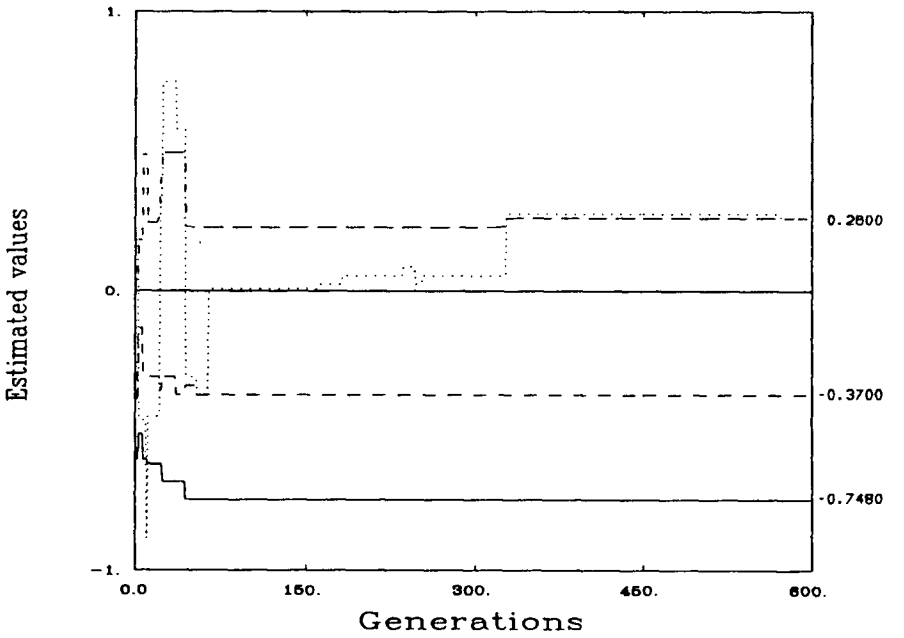

Figure 4.40: Pole-Zero estimates for a minimum phase system without noise

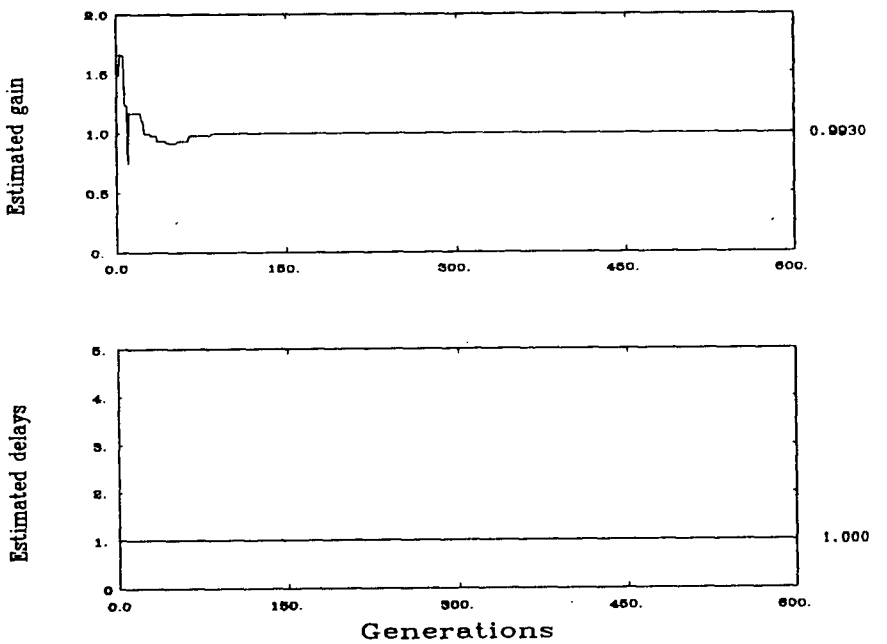

Figure 4.41: Estimates of gain and delay for a minimum phase system without noise

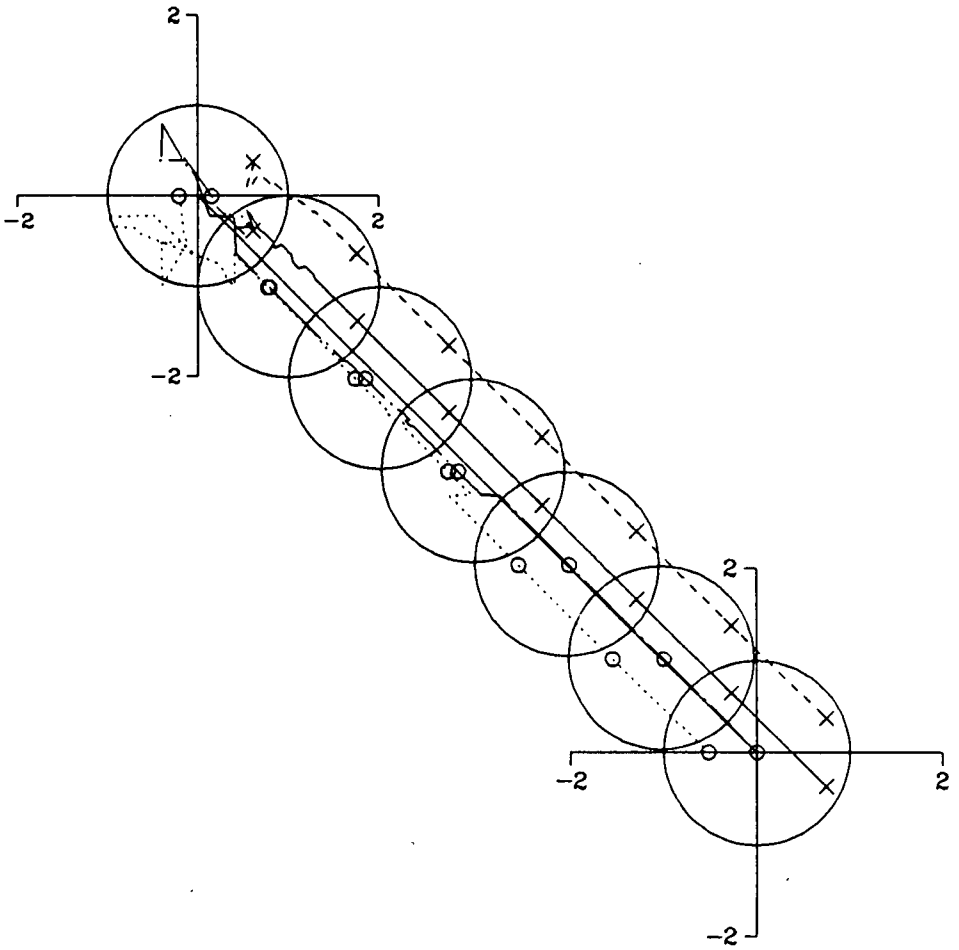

Figure 4.42: Pole zero locations

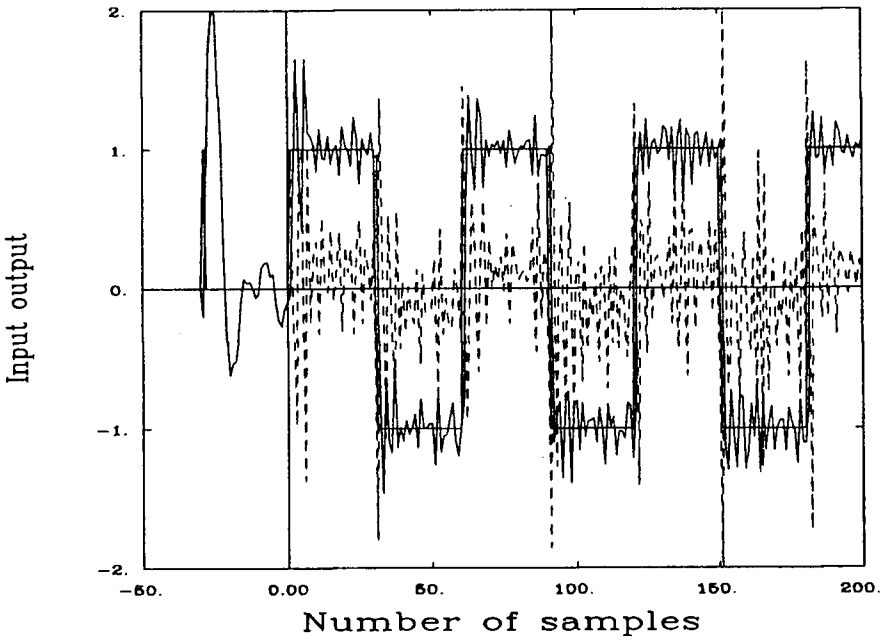

Figure 4.43: Reference input and output of a minimum phase system with noise using pole-zeros estimates

Figure 4.39 shows the reference input and output of the plant when there is no added noise in the system ($\sigma_e^2 = 0.0$). The output follows the reference input within the first 30 steps and there is a bit of oscillations in the control signal because of zeros cancellations. The estimates of the poles and zeros are plotted in Figure 4.40 and it can be seen that unbiased estimates are obtained after about 300 generations or 100 time steps. Prior to that there is a small bias in the estimates, particularly in the zeros, that contributes to the small ripples in the output whenever the reference input is changed. The gain converges to the true value after about 81 generations or 27 steps as can be seen in Figure 4.41. Figure 4.42 shows then the locations of the poles and zeros estimates in the complex plane.

With noise

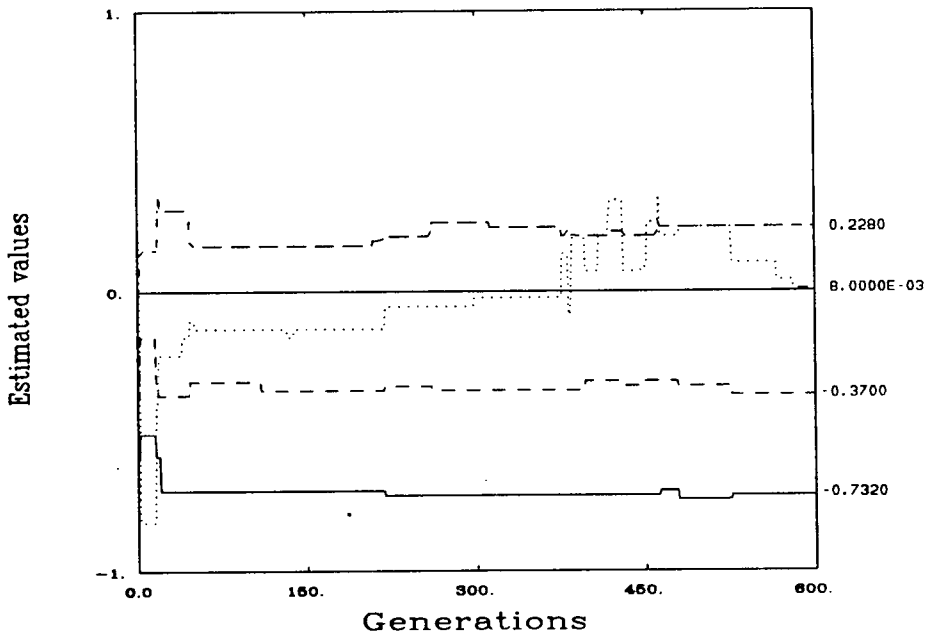

Figure 4.44: Pole-zero estimates for a minimum phase system with noise

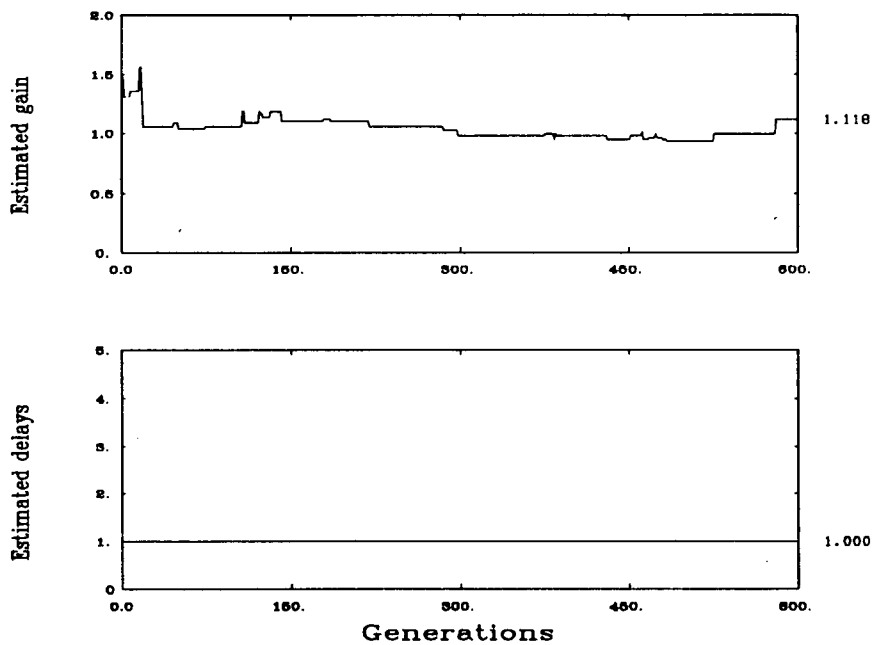

Figure 4.45: Estimates of gain and delay for a minimum phase system with noise

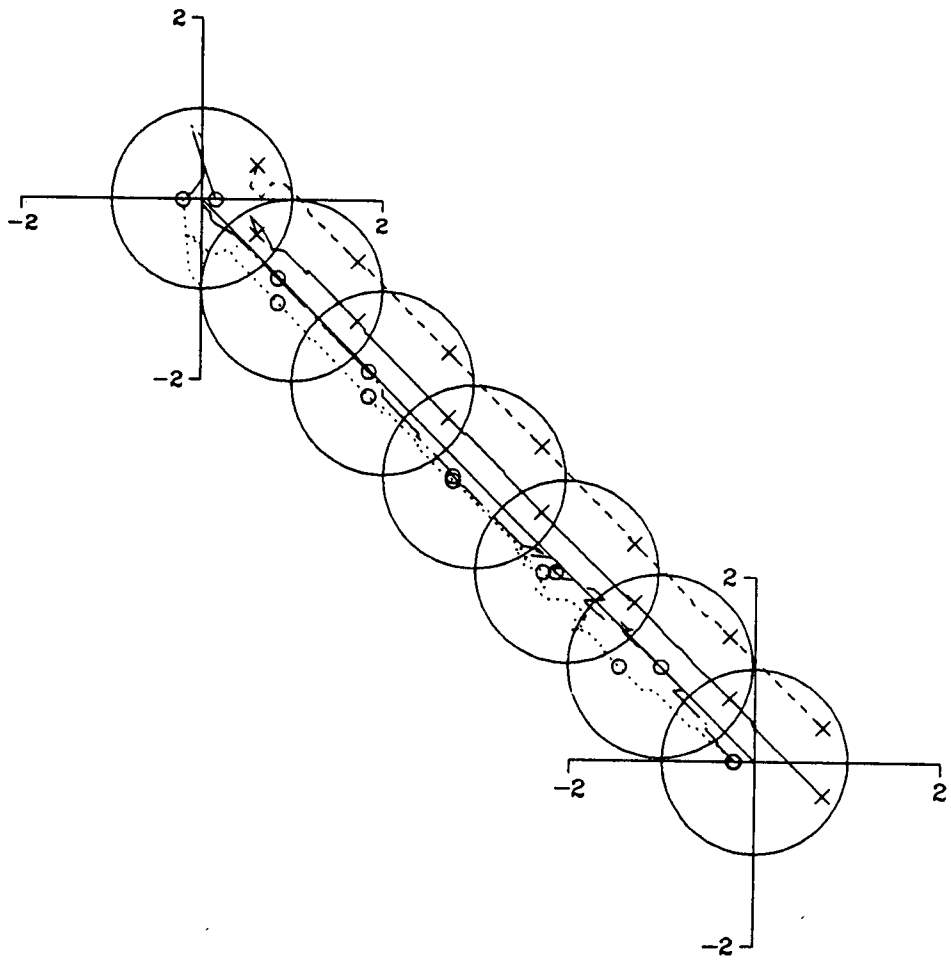

Figure 4.46: Pole zero locations

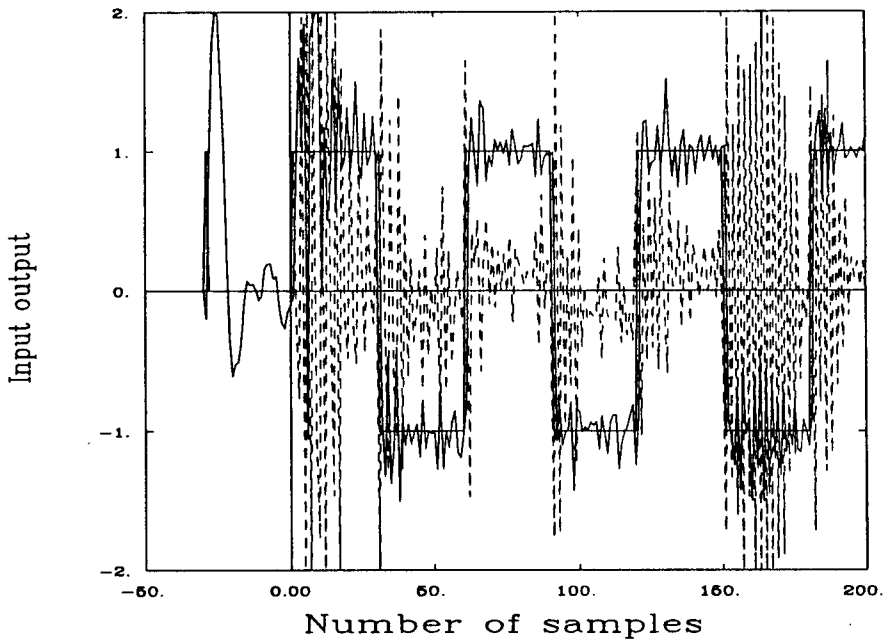

Figure 4.47: Reference input and output of a minimum phase system with noise using parameter estimates

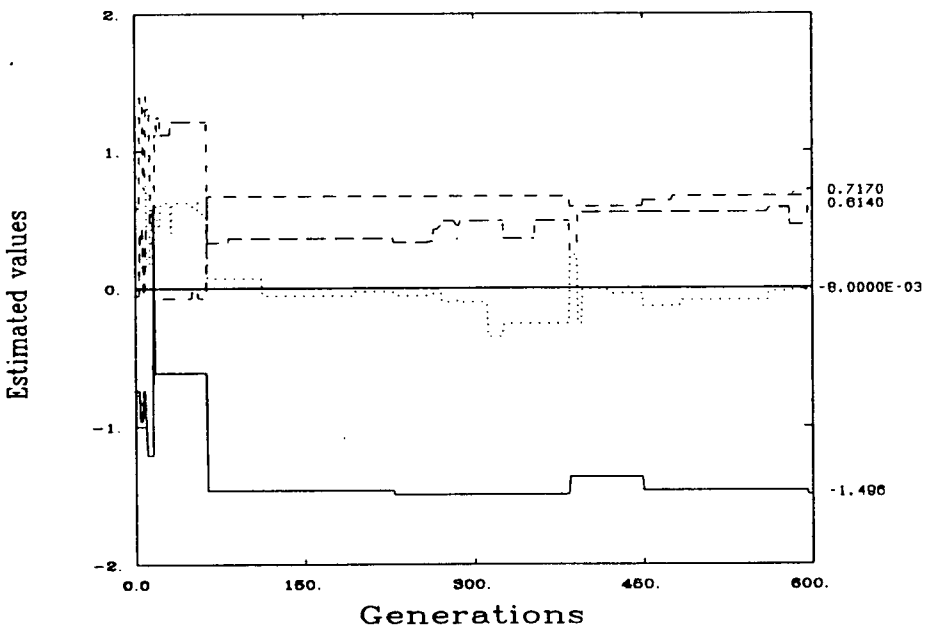

Figure 4.48: Parameter estimates for a minimum phase system with noise

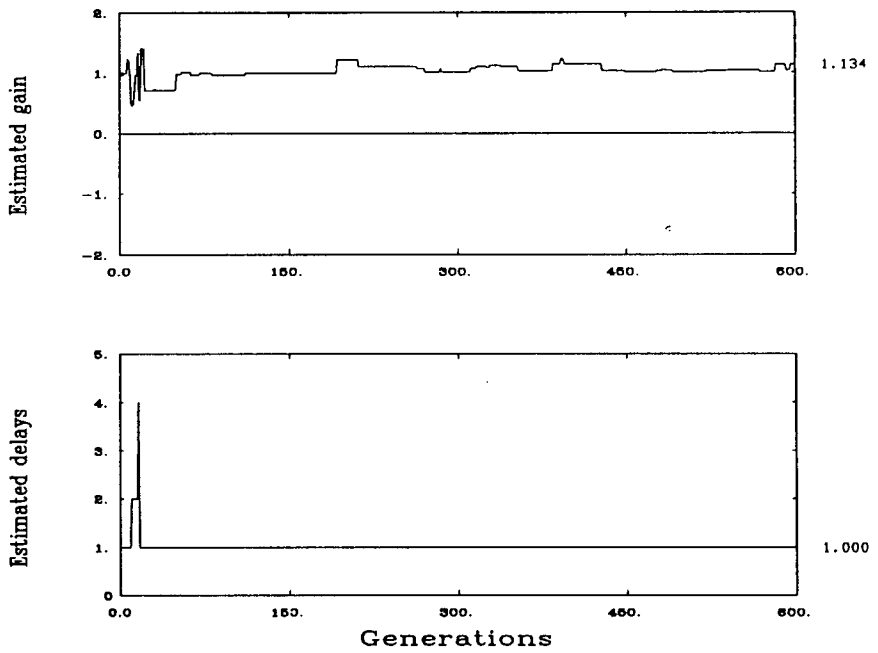

Figure 4.49: Estimates of gain and delay for a minimum phase system with noise

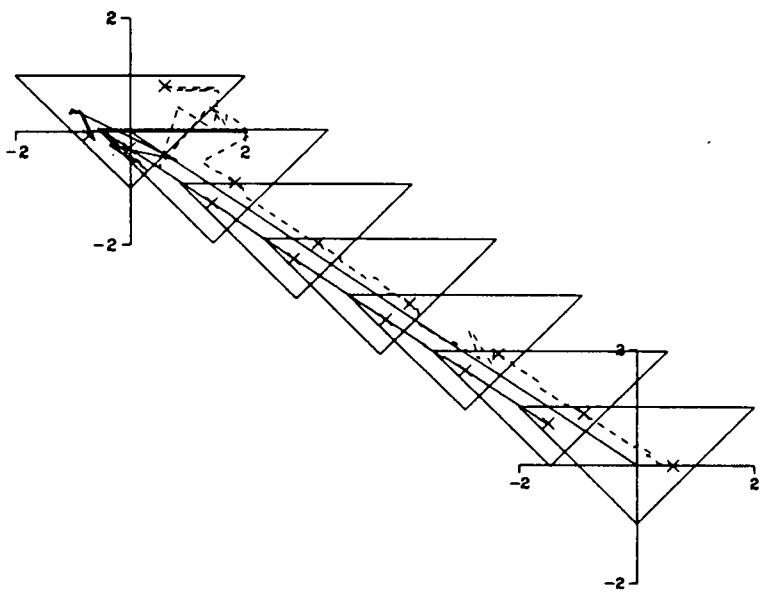

Figure 4.50: Parameters locations

Figure 4.43 shows the reference input and the output of the plant when a colored noise is added to the output. The white noise, $e(t)$, has standard deviation $\sigma_e^2 = 0.1$. By looking at Figures 4.44 and 4.45 it can be seen that good estimates are found after 50 generations or 17 steps, but there is still a small bias in the estimates especially the zeros. It should though be remembered that even though the zeros have a big bias, if the parameters are calculated they do not have a big bias. For example the final estimates for the zeros of 0.228 and 0.008 (true value 0.25 and 0.25) gives the parameters 0.456 and 0.052 which is not far from the true values of 0.5 and 0.0. The estimates in the complex plane for every generation are then shown in Figure 4.46.

To see how the controller based on its parameters behaved compared to the controller based on the poles and zeros, the GA was run again but now estimating the parameters not the poles and zeros. It was shown (Figure 4.47) that the parameters controller does as good job as the pole-zero controller. The estimates take a little bit longer time to converge to acceptable level (Figures 4.48 and 4.50) than they did in Figures 4.44 and 4.45 and the zeros take long time to find refined values because the objective function is more sensitive to changes in the poles than zeros because of steady state gain of 7.5. Figure 4.50 shows then the $a_1 - a_2$ and $b_1 - b_2$ plane for each generation and how the estimates evolve.

4.4.2 Nonminimum phase plant

The plant to be controlled is taken from Clarke [7].

$$\frac{0.1q^{-1}(1+2q^{-1})(1+0q^{-1})}{(1-0.9q^{-1})(1-0.8q^{-1})} \tag{4.80}$$

Using the parameterization given in Chapter 3 the plant parameters are as follows:

$$\begin{aligned} b_0 &= 0.1 \quad d = 1 \\ \alpha_1 &= 0.85 \quad \gamma_1 = 1.0 \\ \beta_1 &= 0.05 \quad \delta_1 = 1.0 \end{aligned} \tag{4.81}$$

The search space for the poles and the gain is the same as previously but the search space for the zeros has been doubled in size (see Table 4.6) to account for the nonminimum phase behaviour. The observer is chosen to be deadbeat and the desired closed

	lower bound	upper bound	# of bits	precision
γ_1, δ_1	-2.0	2.0	7	0.032

Table 4.6: Search space for nonminimum phase

loop plant is assumed to have poles at 0.2 and 0.0 and because unstable process zeros B^- are not cancelled the desired transfer function becomes:

$$\frac{q^{-1}B^-}{1 - 0.2q^{-1}} \tag{4.82}$$

It is not until after 300 generation or 100 time steps that estimates (Figures 4.52 and 4.53) are found that give a good control and the output is following the reference input quite nicely as can be seen in Figure 4.51. The estimates have a small bias but that does not seem to affect the output.

4.4.3 Unmodeled dynamics

We use the same NMP plant as in the previous section (Section 4.4.2) but now the GA uses a first order model

$$\frac{b_0 q^{-d}(1 + b_1 q^{-1})}{1 + a_1 q^{-1}} \tag{4.83}$$

Similar test sequence as used by Clarke [7], i. e. the system is initially run in open loop and then there are setpoint changes every 25^{th} step. The window is chosen as 50

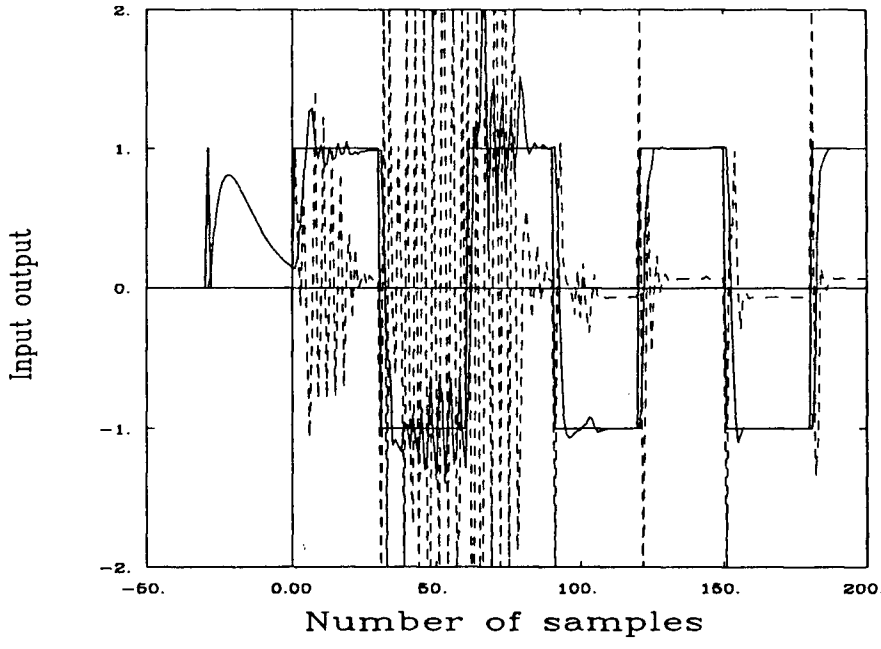

Figure 4.51: Reference input and output for a nonminimum phase system

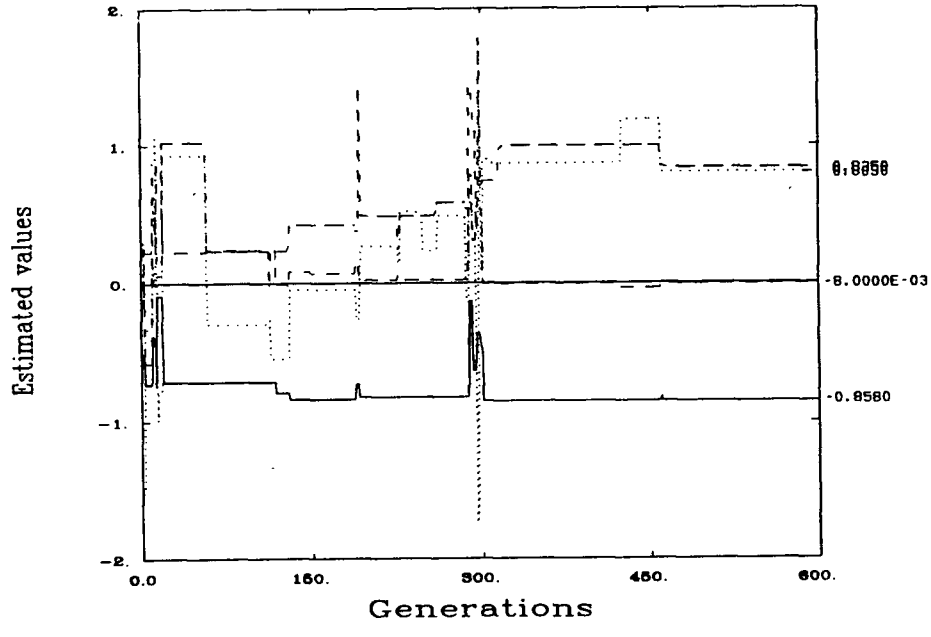

Figure 4.52: Parameter estimate for a nonminimum phase system

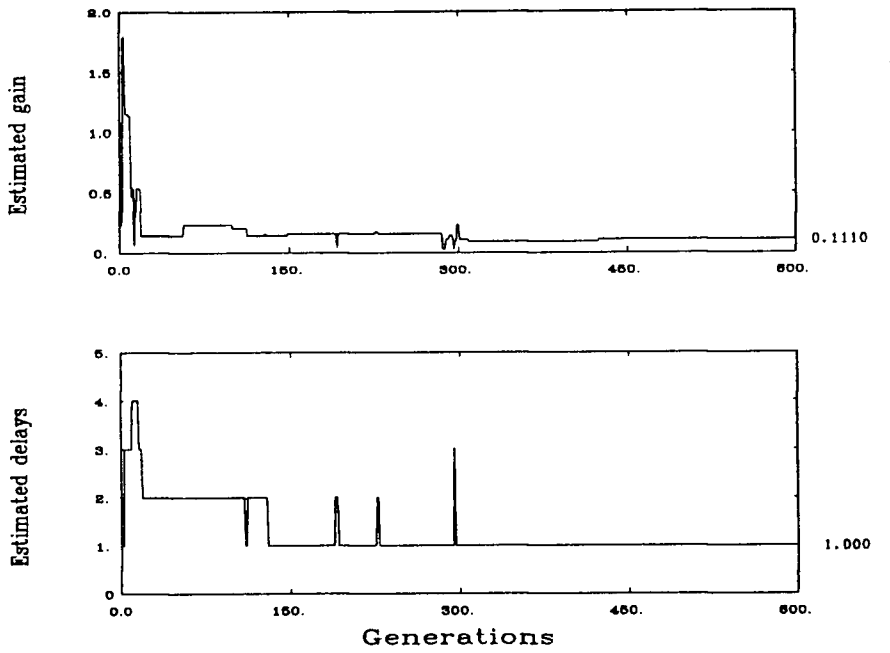

Figure 4.53: Gain and delay estimate for a nonminimum phase system

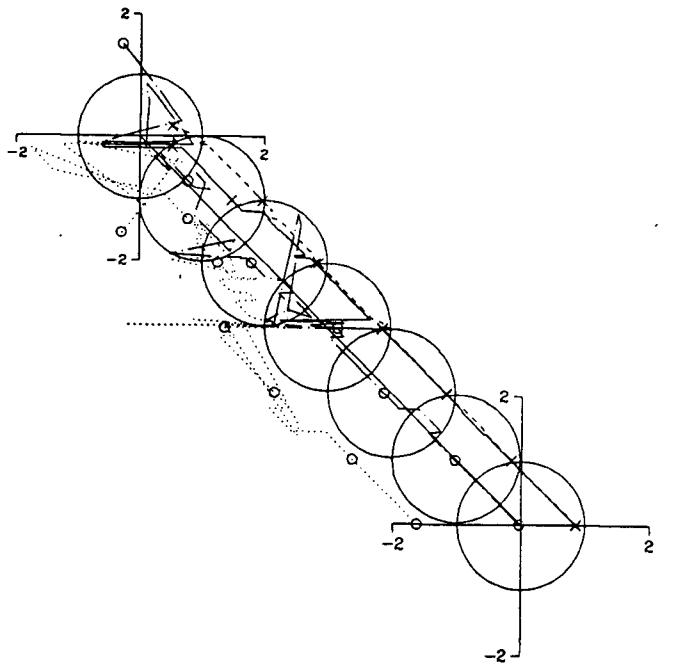

Figure 4.54: Pole zero locations for a nonminimum phase system

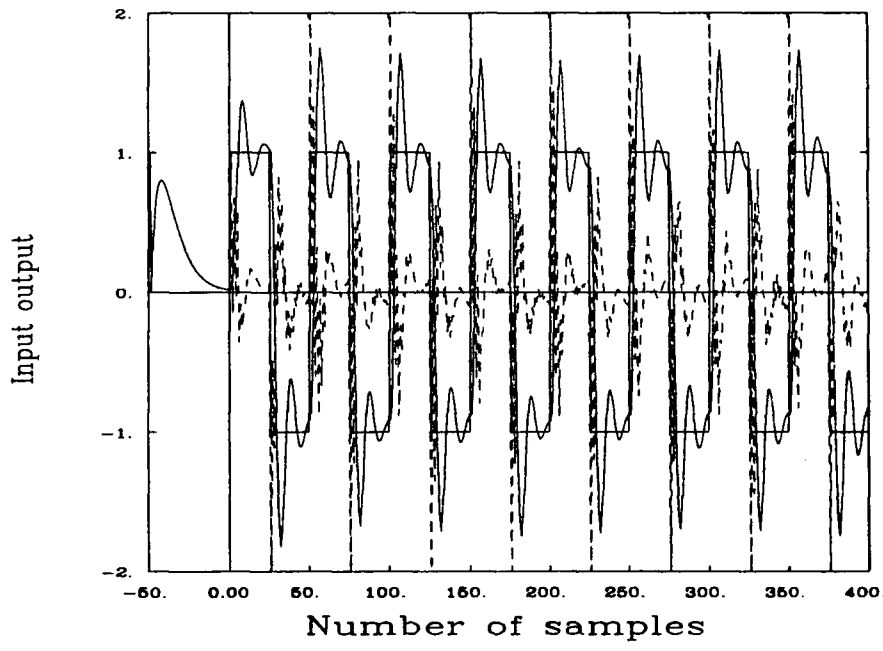

Figure 4.55: Input-Output for 3 parameters estimate with unmodeled dynamics

samples and the search space is defined as in Table 4.7. The desired pole was set

	lower bound	upper bound	# of bits	precision
d	1	4	2	1
b_0	0.0	2.0	7	0.016
a_1	-1.0	1.0	7	0.016
b_1	-2.0	20.0	9	0.043

Table 4.7: Search space for unmodeled dynamics

at 0 (dead-beat). For the first simulation the delay d is assumed to be known so only the gain, the pole and the zero (b_0, a_1 and b_1) are identified. That gives a response that has about 70% overshoot and has a low damping as shown in Figure 4.55 and the parameter estimates are as shown in Figures 4.56 and 4.57 with the estimates for the zero not converging to a certain value. But when all the four parameters (b_0, b_1, d and a_1) are identified (population size = 50, total string length = 23) the overshoot is reduced to about 30% and the damping is higher, Figure 4.58, and the system is

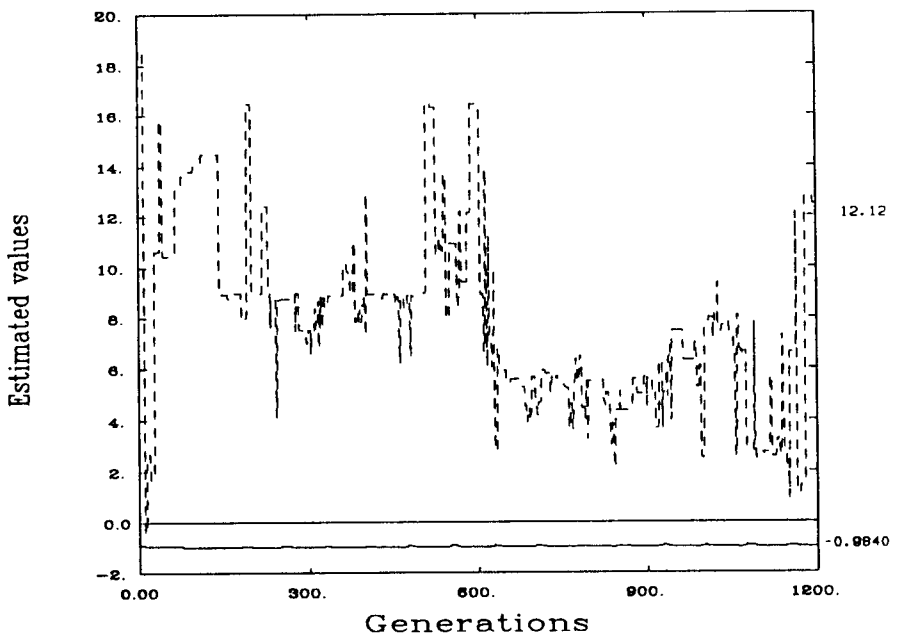

Figure 4.56: Parameter estimate for unmodeled dynamics

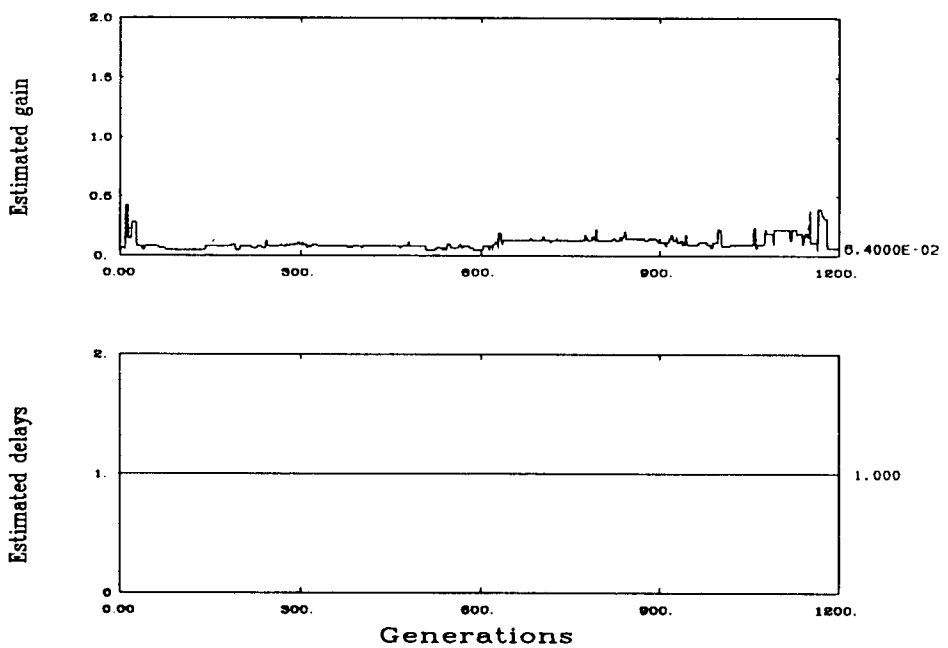

Figure 4.57: Gain and delay estimate for unmodeled dynamics

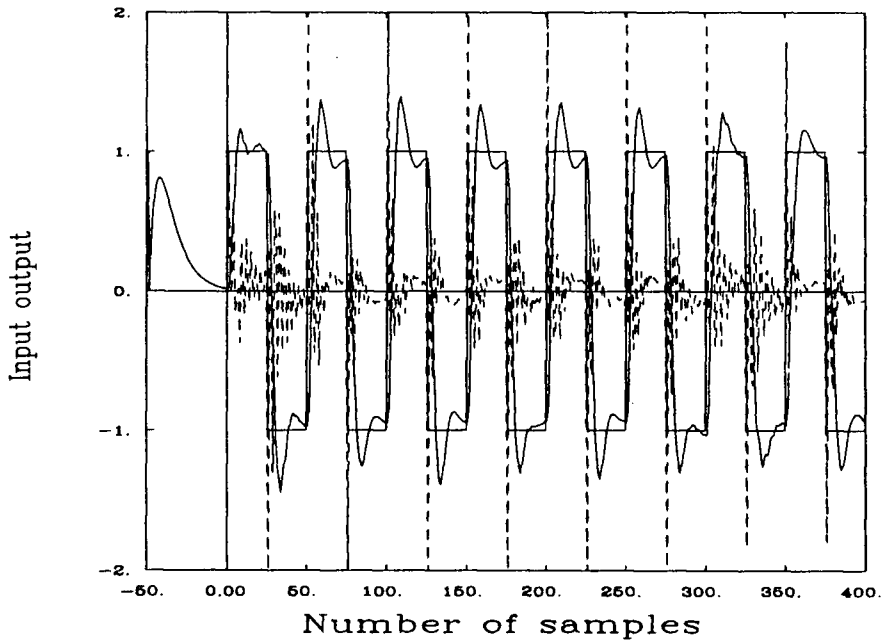

Figure 4.58: Input-Output with dead beat control

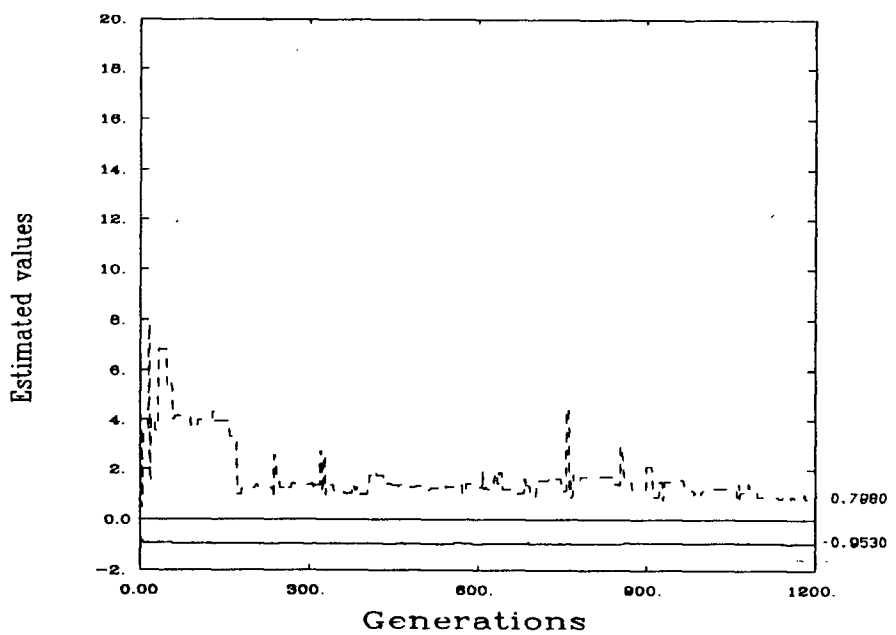

Figure 4.59: Parameter estimate for unmodeled dynamics with dead beat control

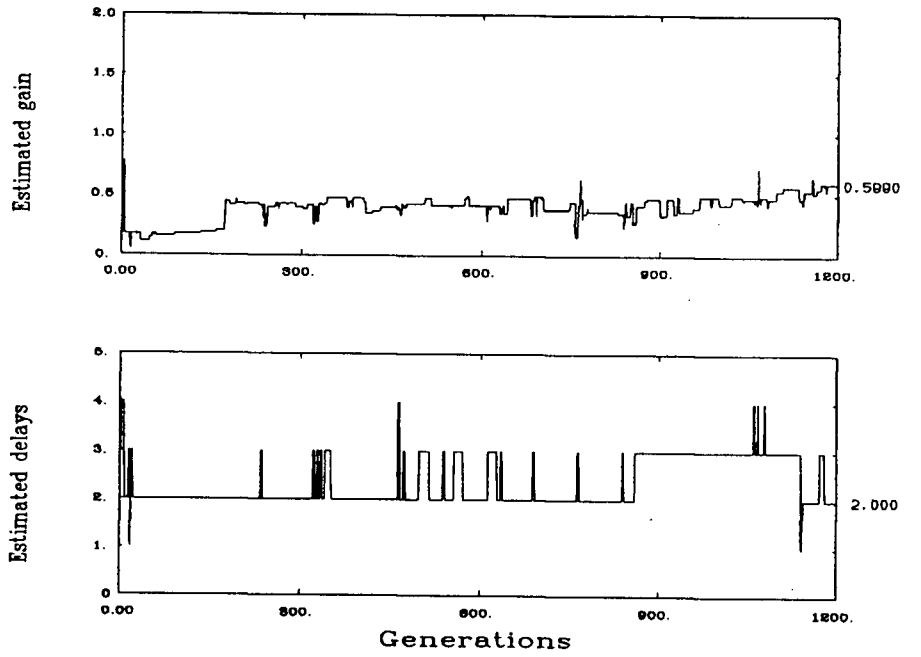

Figure 4.60: Gain and delay estimate for unmodeled dynamics with dead beat control

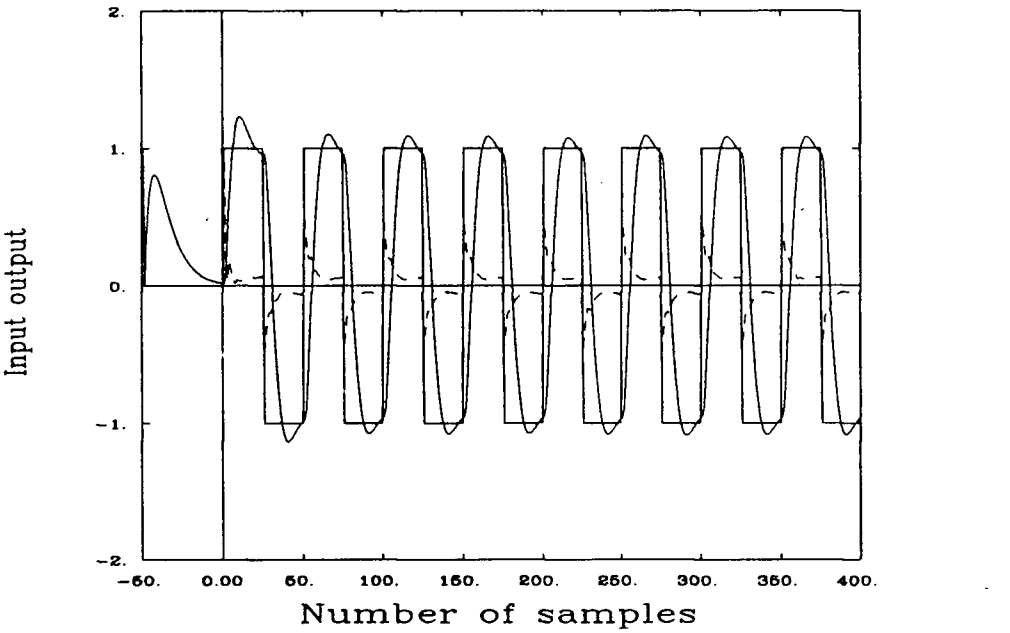

Figure 4.61: Input-Output with desired pole = 0.7

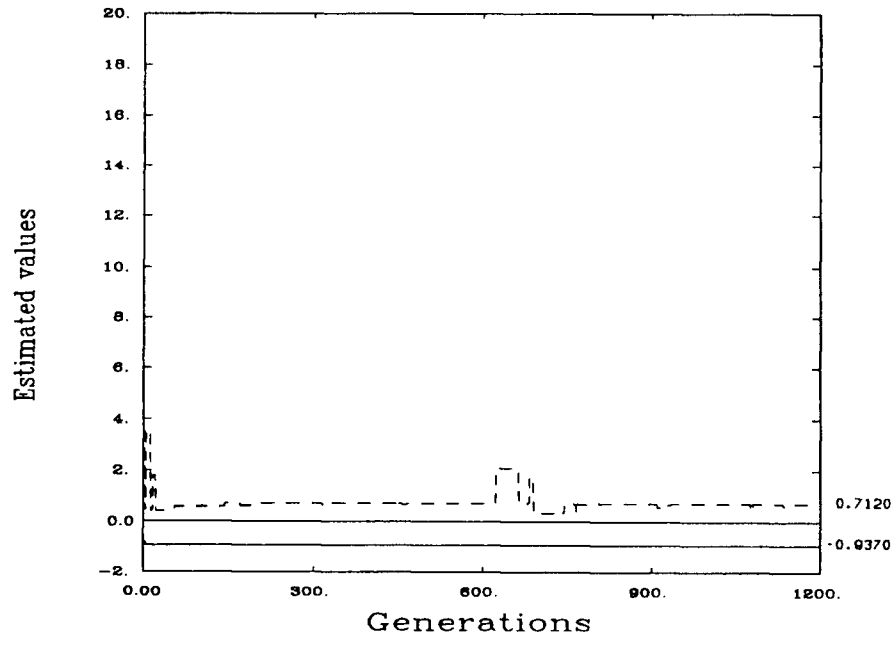

Figure 4.62: Parameter estimates for unmodeled dynamics with desired pole = 0.7

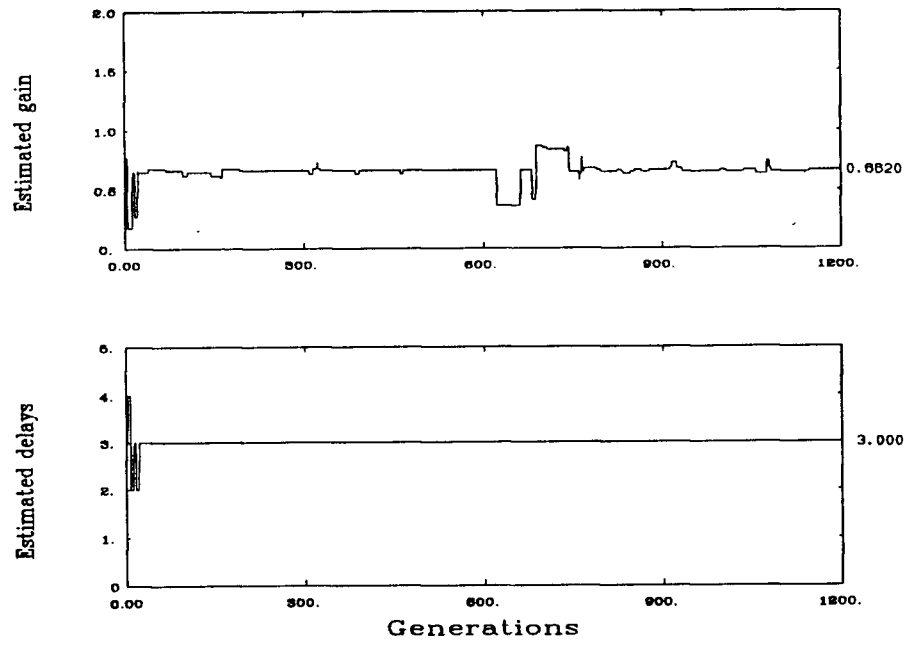

Figure 4.63: Gain and delay estimate for unmodeled dynamics with desired pole = 0.7

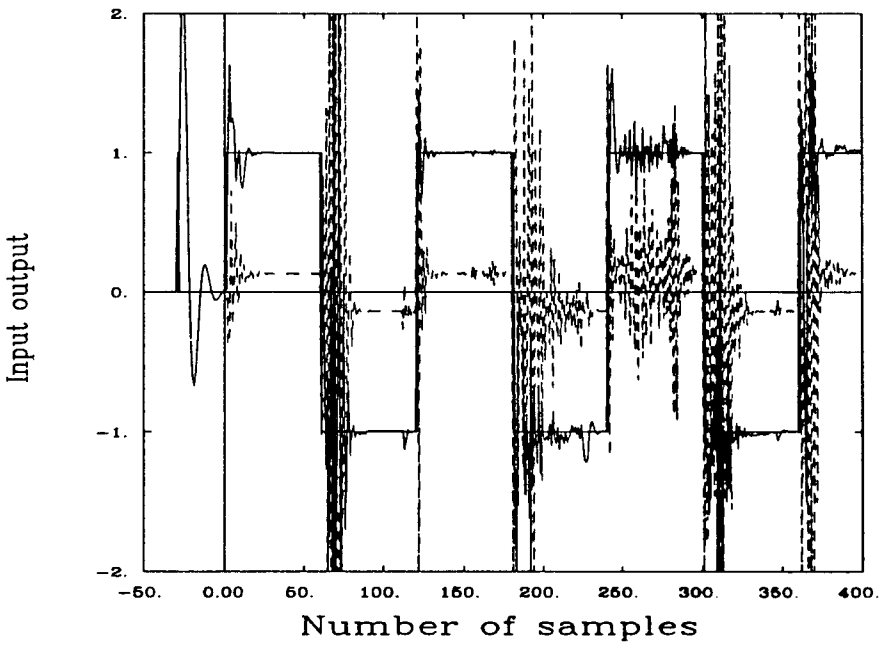

Figure 4.64: Reference input and output of a system with window size = 30

estimated to be (see Figures 4.59 and 4.60)

$$\frac{0.599q^{-2}(1+0.798q^{-1})}{1-0.953q^{-1}} \tag{4.84}$$

Further improvement is obtained if a slower response is chosen (desired pole at 0.7), Figure 4.61. The estimated model becomes (see Figures 4.62 and 4.63)

$$\frac{0.662q^{-3}(1+0.712q^{-1})}{1-0.937q^{-1}} \tag{4.85}$$

4.4.4 Persistently exciting signal

To show the effect of a persistent excitation, the minimum phase system of Equation 4.79 without noise ($\sigma_e^2 = 0.0$) is run for a step change every 60 sample and a window size of 30 and 60. For the smaller window the algorithm comes up with good estimates after about 150 generations (see Figures 4.65 and 4.66), but is not able to keep them because the input and the output does not change over the window. The estimates

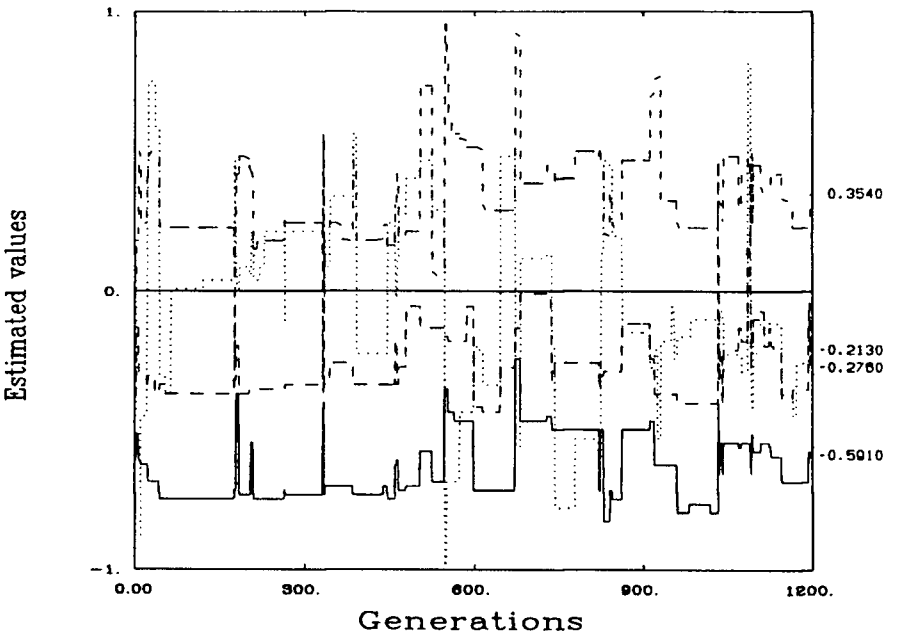

Figure 4.65: Parameters for a window size = 30

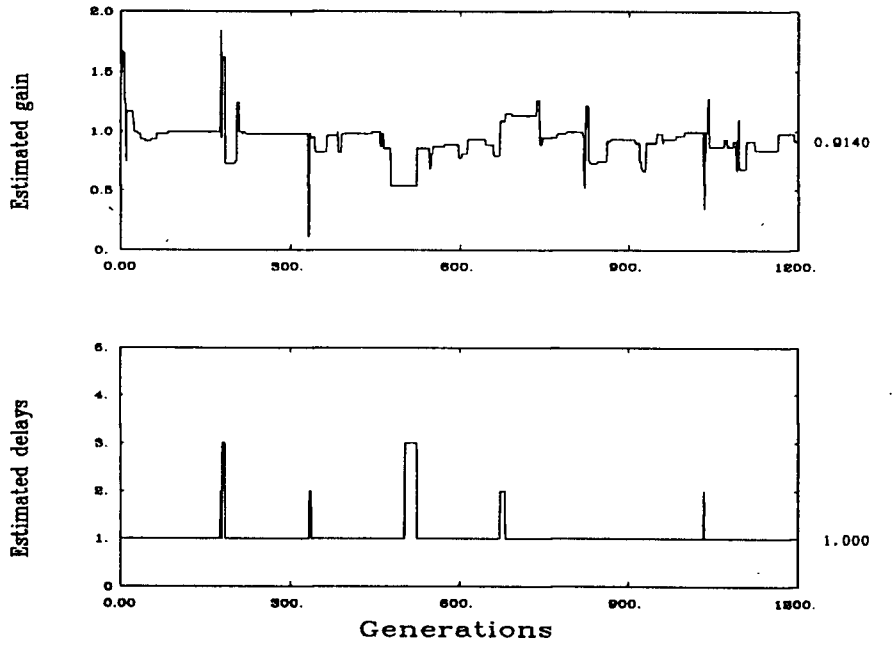

Figure 4.66: Gain and delay for a window size = 30

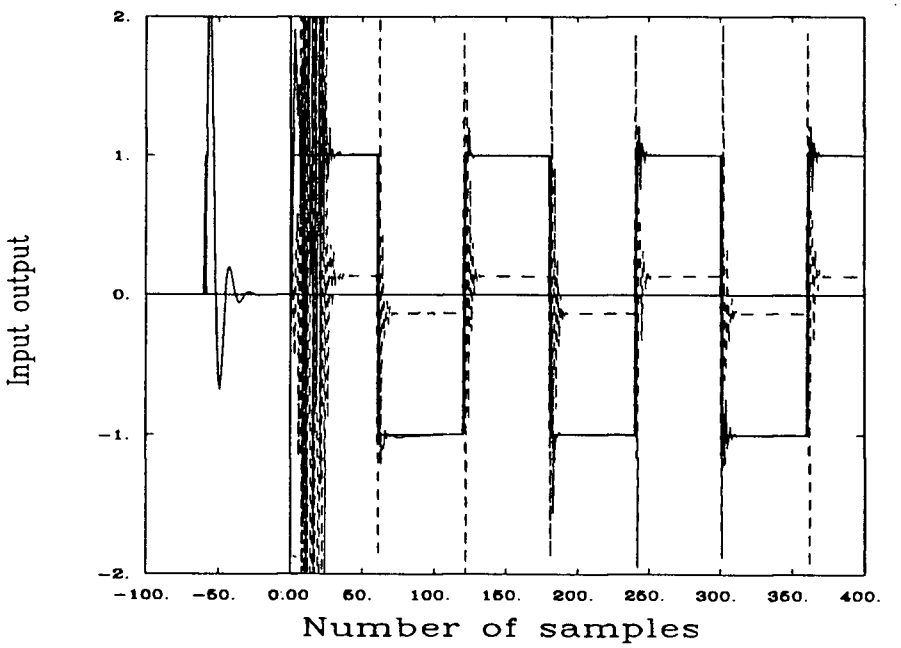

Figure 4.67: Reference input and output of a system with window size = 60

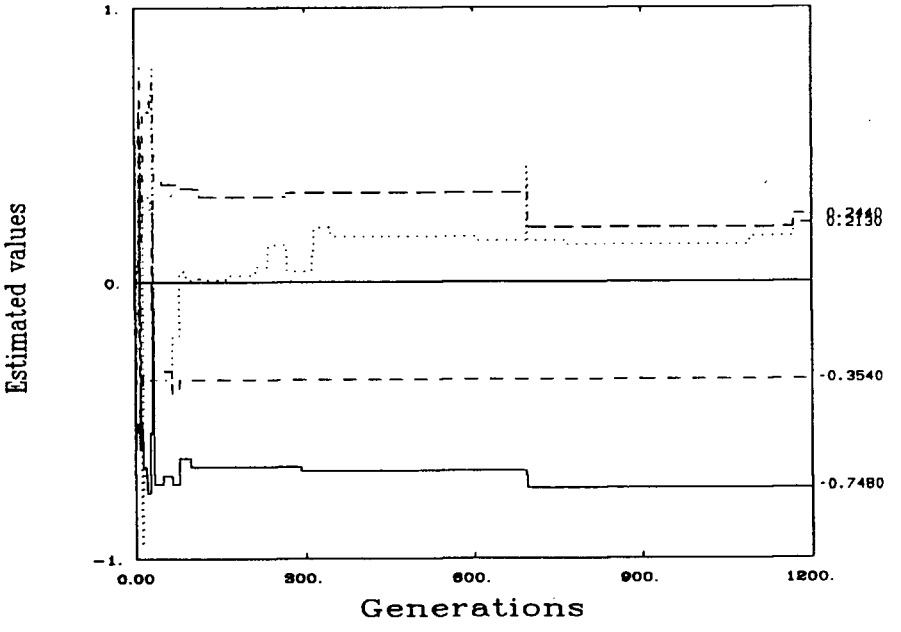

Figure 4.68: Parameters for a window size = 60

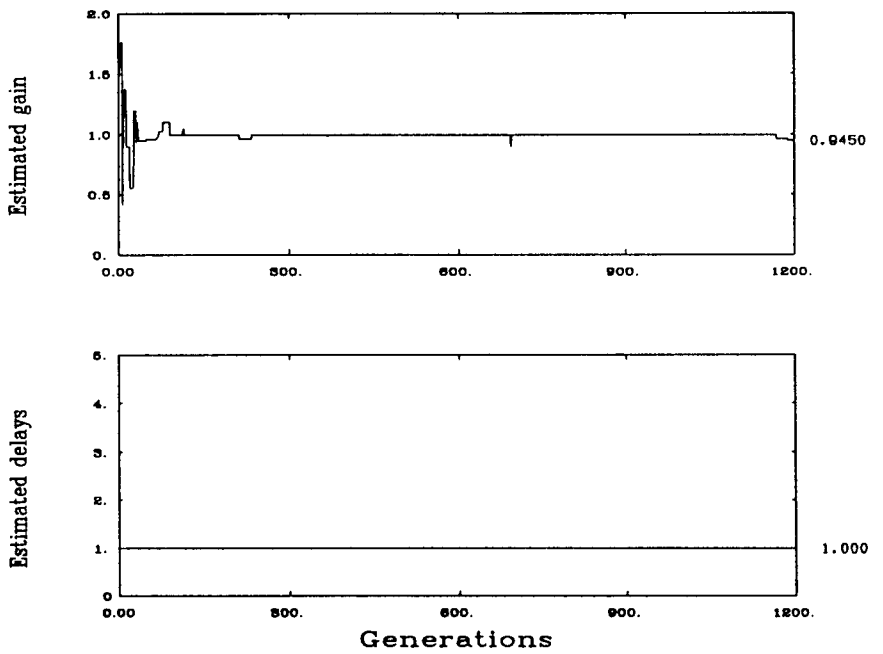

Figure 4.69: Gain and delay for a window size = 60

starts to deteriorate and the algorithm is not able to bring them back consequently the output does not follow the input very well (Figure 4.64). When the window is increased to 60, to be able to include a step change in the window at all time, the estimates are shown to converge to the true value (Figures 4.68 and 4.69). Because of a small bias in the estimates the output has a small overshoot at every setpoint change (Figure 4.67).

4.4.5 Recursive Least Squares

To compare the GA to some method that is widely known, a standard RLS algorithm is used on the same system as before with noise variance $\sigma_e^2 = 0.1$. The same pole placement controller design is used. A deadbeat observer is chosen and the closed loop poles and zeros are set at zero (deadbeat). The forgetting factor is set to 0.9 to resemble a window for the GA of 30 steps ($0.9^{30} = 0.04$) and a_1, a_2, b_1 and b_2 are then identified. Figure 4.70 shows the reference input and the output of the system. It can be seen that

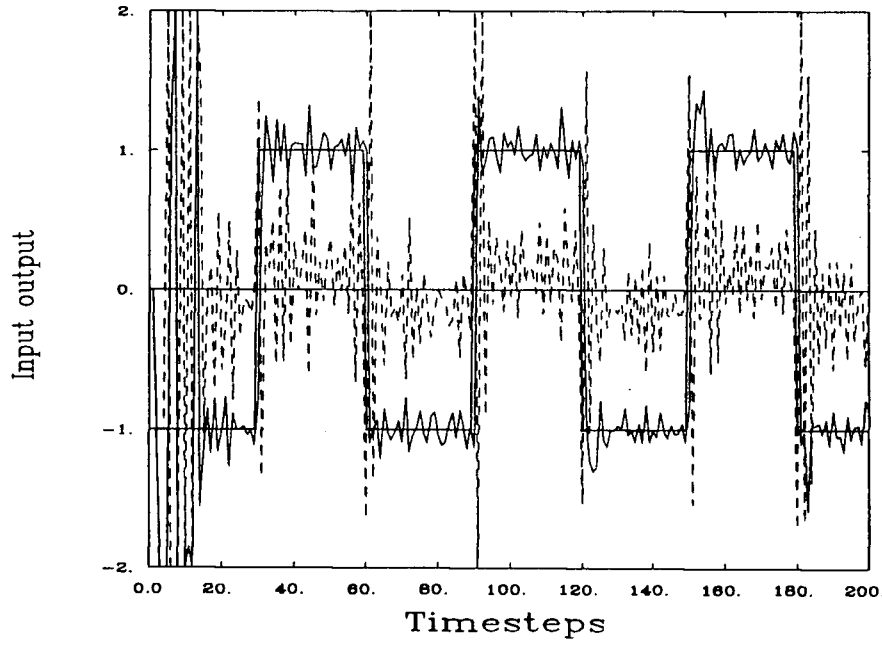

Figure 4.70: Reference input and output of a system using RLS

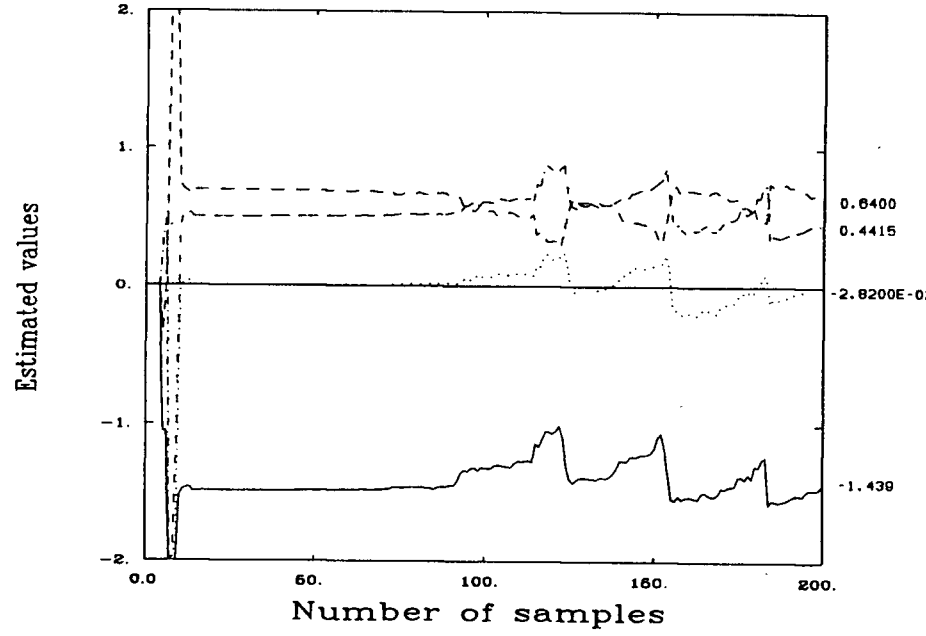

Figure 4.71: Parameter estimates for RLS

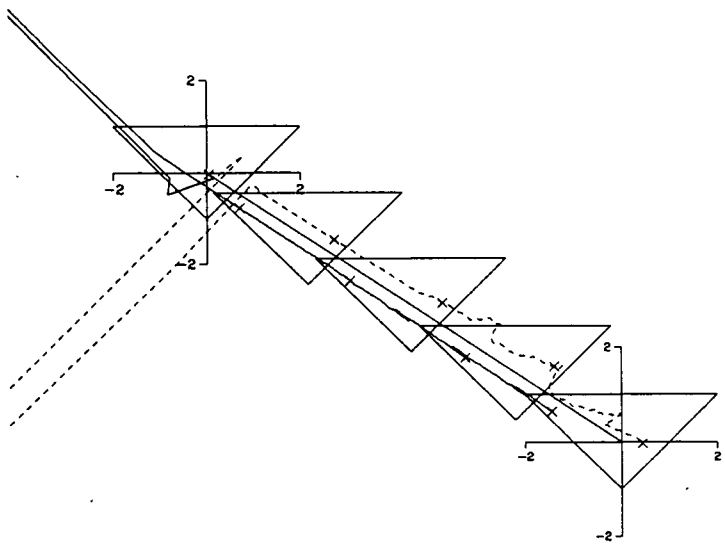

Figure 4.72: Parameter locations for RLS

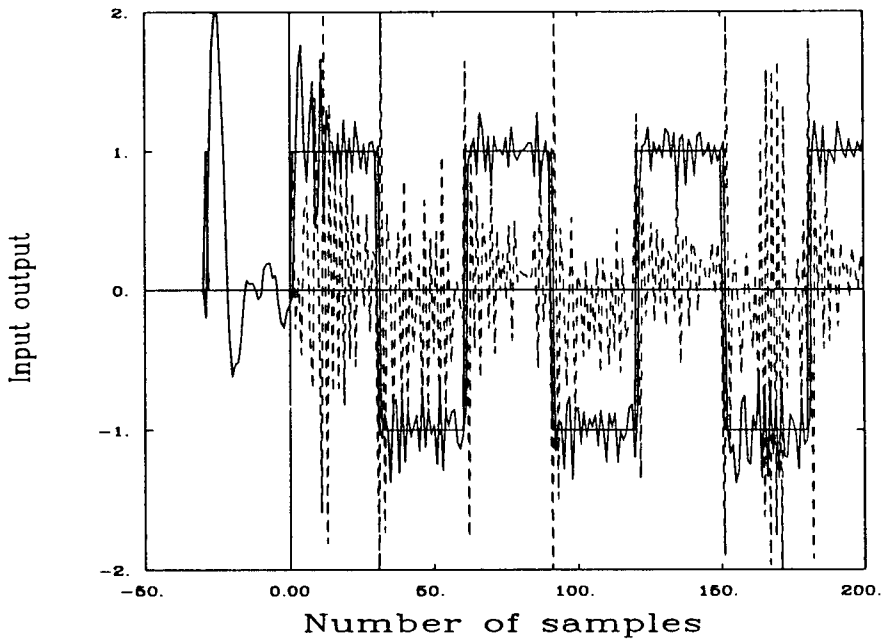

Figure 4.73: Reference input and output of a system using GA to compare to RLS

the overshoots becomes larger as the parameter estimates deteriorates (Figure 4.71). Figure 4.72 shows plot of a_2 as a function of a_1 and b_2 as a function of b_1. To compare those results with GA, the GA has been run for same parameter estimates (only pole and zeros, not gain and delay) using IV criterion and the results are shown in Figures 4.73, 4.74 and 4.75. Comparing Figure 4.70 and Figure 4.73 one can see that there is not much of a difference in their response. Both have transients while searching for good parameters and after few steps output follows the input. The RLS is quicker to converge to a value whereas the GA is satisfied with suboptimal estimates.

4.5 Summary

It has been shown how knowledge of the plant (A and B), either its parameters or poles-and-zeros, can be used to design a pole-placement controller. Simulations results

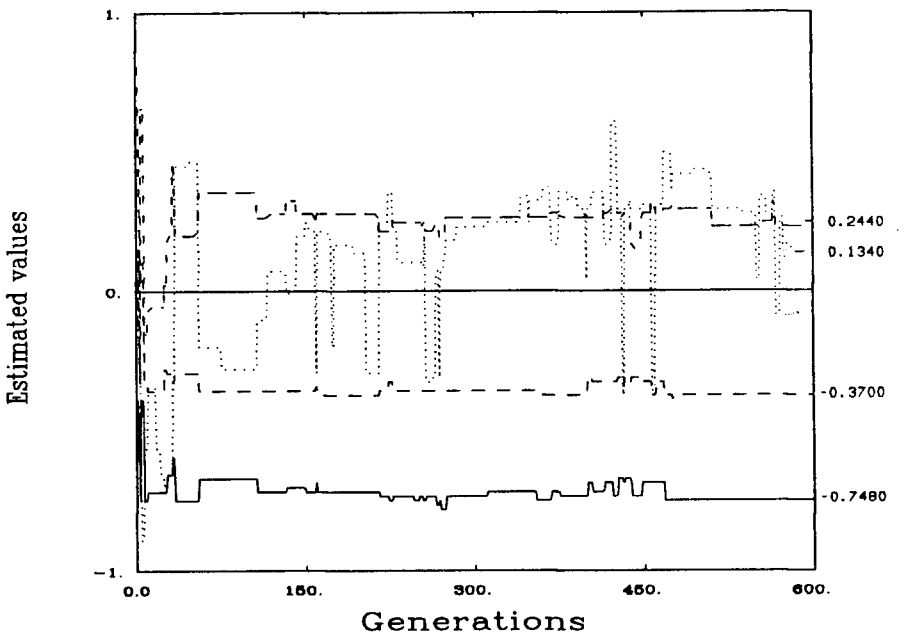

Figure 4.74: Parameter estimates for GA to compare with RLS

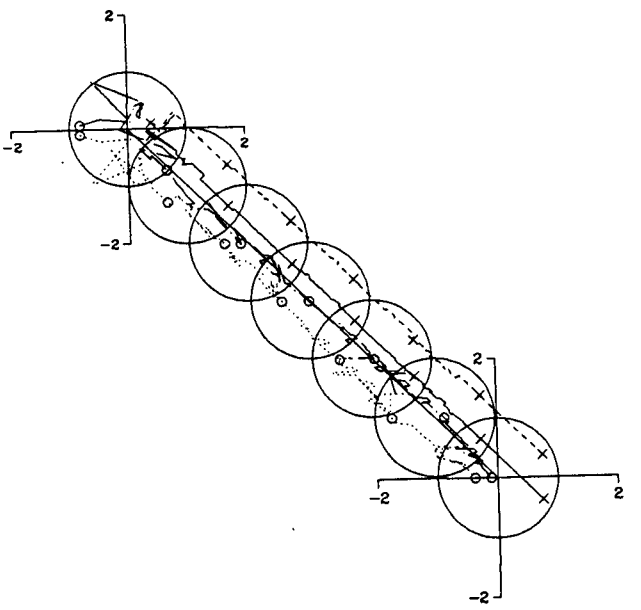

Figure 4.75: Parameter locations for GA

show that the GA is as well fit for doing the identification as the RLS. The GA has proven to be able to handle both minimum and nonminimum phase systems and has also shown its ability to control when there is unmodeled dynamics.

Chapter 5

Experiment

5.1 Water level in a tank

An experiment was carried out on a tank system at the Pulp and Paper Centre. The tank has a sensor to measure the height h of the water and a pump to pump the water, given a drive voltage u, into the tank. The outflow of the tank is a function of the tank level so the dynamics will be nonlinear. Therefore the tank can be described by a nonlinear differential equation of the form [2]:

$$\frac{dh}{dt} = -A\sqrt{h} + Bf(u(t)) \tag{5.86}$$

Where A depends on gravity and the ratio between the effective outlet area and the cross section of the tank and B depends on the cross section of the tank and also relates the pump flow to the drive voltage u of the pump motor electronics. A linear model of the tank is given in Åström and Östberg [2] as:

$$H(s) = \frac{KT}{Ts + 1} \tag{5.87}$$

where T depends on A and the initial height and K depends on the sensor and the constant B. A ZOH is used together with the A/D converter to read the water height so the Z-transform would be:

$$H(z) = \frac{KT(1 - e^{-\frac{T_s}{T}})z^{-1}}{(1 - e^{-\frac{T_s}{T}}z^{-1})} \tag{5.88}$$

where T_s is the sampling time.

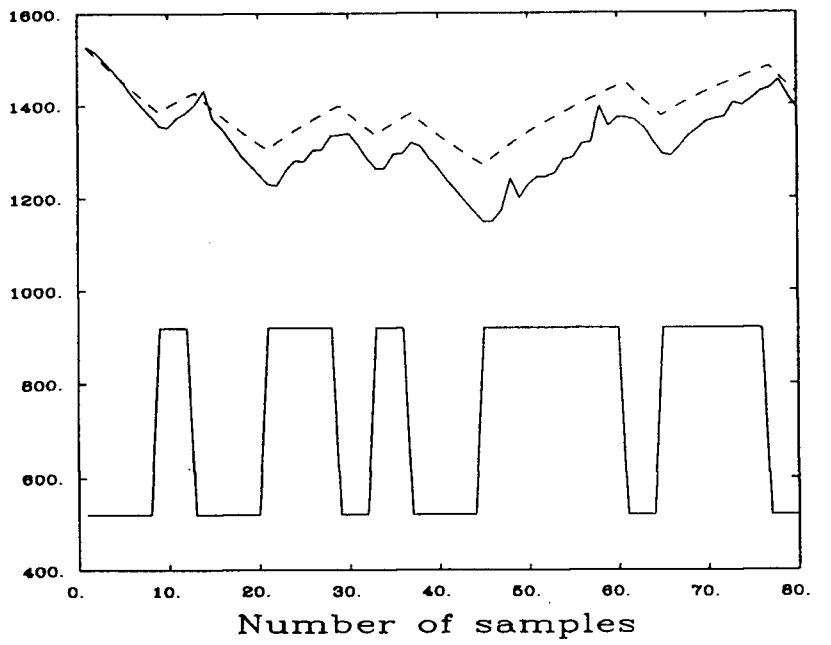

Figure 5.76: Tank Input-Output

5.2 Simulation results

To collect the data the sampling time is chosen as 0.55 sec. and 80 samples are obtained using PRBS as the input (see figure 5.76). Because of the prohibitive time it takes to run the GA on an IBM AT we were not able to do any online control on the tank. The GA is therefore run offline using the IV fitness function (see equation 3.13) to identify directly the poles and zeros. The algorithm assumes that there are two poles, one zero and it also identifies the gain b_0.

$$\frac{b_0 q^{-1}(1 + b_1 q^{-1})}{(1 + p_1 q^{-1})(1 + \bar{p}_1 q^{-1})} \tag{5.89}$$

The poles are decoded as in chapter 3. Both the poles and the zero are assumed to be stable so they are assumed to lie between -1 and +1. The gain is assumed to be in the range $[0, 10]$ The length of each string is chosen as 11 which give resolution of about 1/1000 for the poles and the zero and about 5/1000 for the gain. With four

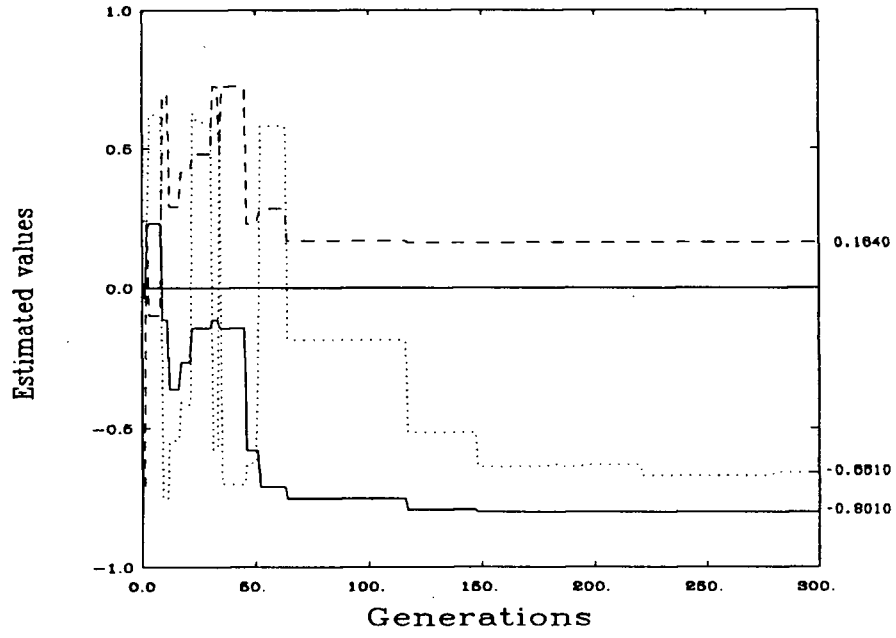

Figure 5.77: Parameter estimate for a tank

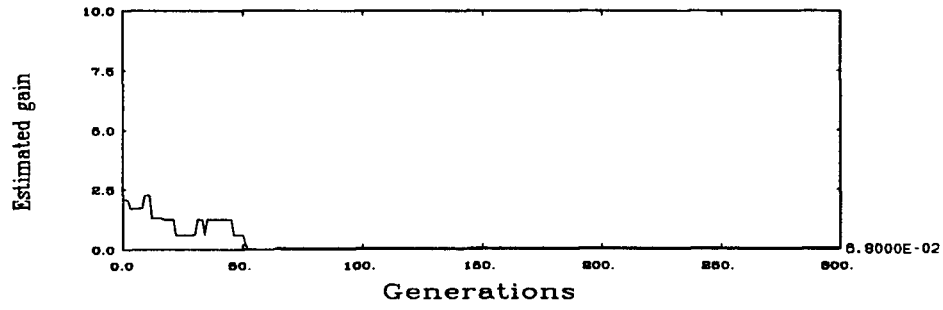

Figure 5.78: Estimated gain for a tank

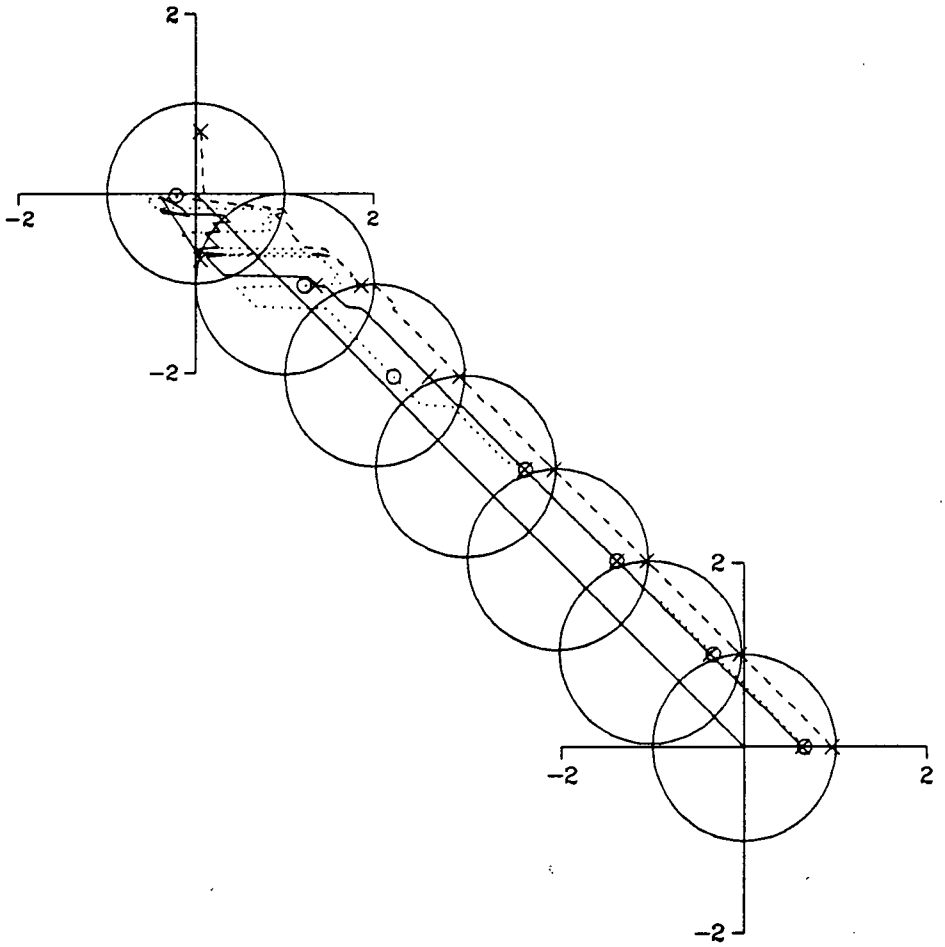

Figure 5.79: Pole zero locations for a tank

parameters to identify, the total string length is 44 bits so the population size is set to 100. The probability of crossover and mutation is chosen as before to be 0.80 and 0.01 respectively and 6 generations were generated each sampling interval.

Figure 5.77 and 5.78 show the estimated parameters for each generation using the input output data of figure 5.76. The estimated system after 300 generations is:

$$\frac{0.068q^{-1}(1-0.661q^{-1})}{(1-0.637q^{-1})(1-0.965q^{-1})} \simeq \frac{0.068q^{-1}}{(1-0.965q^{-1})} \tag{5.90}$$

So the zero cancels one of the poles and the system is a first order system with a delay and a pole close to the unit circle. The time it takes to fill up the tank is about 10 seconds so with a sampling time close to half a second the pole should be according to equation 5.88 about -0.95 so the estimates seems good. In figure 5.76 the output of the tank is shown if the final estimated parameters were used (dashed line). It can be seen that the estimated output is not far from the actual one and it should be pointed out that the estimated output is put equal to the actual output at the beginning of every window, which means that the two output are much closer than suggested in figure 5.76. In figure 5.79 the locations of the poles and the zero are shown in the complex plane for every generation where the cancellation of one of the poles with the zero can be clearly seen.

Chapter 6

Conclusions

In this thesis a new approach was taken to the identification problem. The usual hill climbing algorithms that follow the steepest gradient were abandoned for a method that uses concepts from evolutionary theory called Genetic Algorithms. They proved to be able to identify both discrete time and continuous time systems and could give unbiased estimates in the presence of colored noise. They showed some advantage over RLS and could be used in cases were the system is not linear in the parameters were RLS can not be used, for example identify physical parameters, delays and pole-zeros. They were used to design an adaptive pole-placement controller and gave good control for a variety of problems as demonstrated by simulations.

An experiment was presented. The algorithm was tested on a real data from a tank system. The algorithm behaved well but because of the prohibitive time it takes for the algorithm to run on an IBM-AT, no real time online control was attempted.

Genetic Algorithms have proven to be useful on wide range of applications without any changes in the basic algorithm. The only interface with the system the algorithm is working on is through the objective or fitness function. That function is the only thing that needs to be changed from one application to another. Because the GAs search within a population not from a single value they are insensitive to noise. As with the RLS, there must be some priori knowledge about the system to identify, the search space that the parameters are likely to lie within must be specified and also the resolution. For a proper choice of the resolution the algorithm will prevent the

estimates from jumping around and hence could be used to filter some noise. It should also be remembered that the algorithm is a randomized search technique, so there is no guarantee of optimality, the algorithm does its best while learning to do better.

An area for further research is the exploitation of more than the best string in the population for the design of a robust controller. For example the average of the ten best strings in the population could be used for preventing abrupt changes in the estimates. Also dominance could be used for a changing plant and for a multimodal search space, like the example from chapter 2, some sort of distribution among the peaks could be maintained by introducing sharing, that is the individuals are prevented from crowding around one particular peak by punishing them for being too close together. That could be particularly useful in changing environment where by maintaining diversity in the population the algorithm does not put all of its effort into searching around a particular peak.

Finally, GAs are parallel algorithm, so every attempt to run the algorithm on non-parallel computer is bound to be slow. For our case the algorithm uses little bit less than 1 second of CPU time for each generation, on a μ-VAX (1 MIPS) for population size of 100, string length of 37 and window size of 30 steps. Once parallel computer architectures become readily available, GA will become more attractive.

Bibliography

[1] Åström, K. J. and B. Wittenmark, (1984). *Computer controlled systems*, Prentice Hall Inc., Englewood Cliffs N.J.

[2] Åström, K. J. and A. B. Östberg, (1985). "A teaching laboratory for process control," *Proceedings of the American Control Conference*, pp. 1380-1385, vol. 3.

[3] Baker, J. E., (1985). "Adaptive Selection Methods for Genetic Algorithms," *Proceedings of an International conference on Genetic Algorithms and Their Applications*, pp. 101-111.

[4] Bethke, A. D., (1980). *Genetic algorithms as function optimizers.* Doctoral dissertation (CCS), University of Michigan, Ann Arbor, MI.

[5] Canudas, C., K. J. Åström and K. Braun, (1987). "Adaptive compensation in DC-motor drives," *IEEE Journal of robotics and automation*, pp. 680-685, vol. RA-3, No. 6, December.

[6] Clarke, D. W., (1981). "Implementation of self tuning controllers," *Self-tuning and adaptive control: Theory and applications*, (eds. C.J. Harris and S.A. Billings), pp. 144-165.

[7] Clarke, D. W., (1984). "Self-tuning control of nonminimum-phase systems," *Automatica*, vol. 20, pp. 501-517.

[8] Das, R. and D. E. Goldberg, (1988). "Discrete-Time parameter estimation with

Genetic algorithms," *Proceedings of the* 19^{th} *annual Pittsburgh conference on modelling and simulation.*

[9] Davis, L. (Ed.) (1987). *Genetic Algorithms and Simulated Annealing*, London: Pitman.

[10] DeJong, K. A., (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation (CCS), University of Michigan, Ann Arbor.

[11] Etter, D. M., M. J. Hicks, and K. H. Cho, (1982). "Recursive adaptive filter design using an adaptive genetic algorithm." *Proceedings of the IEEE International conference on Acoustics, Speech and Signal Processing*, pp. 635-638, vol. 2.

[12] Goldberg, D. E., (1981). *System identification via genetic algorithm*, Unpublished manuscript, University of Michigan, Ann Arbor, MI.

[13] Goldberg, D. E., (1983). *Computer-aided gas pipeline operation using genetic algorithms and rule learning.* Doctoral dissertation (civil engineering), University of Michigan, Ann Arbor.

[14] Goldberg, D. E., (1987). "Simple genetic algorithms and the minimal deceptive problem," *Genetic algorithms and simulated annealing*, Lawrence, D. (Ed.), Pitman Publishing, pp. 74-88.

[15] Goldberg, D. E., (1989). *Genetic algorithms in Search, optimization and Machine Learning*, Addison-Wesley.

[16] Grefenstette, J. J. (Ed.), (1985). *Proceedings of an International conference on genetic algorithms and their applications*, Hillsdale, NJ: Lawrence Erlbaum Associates.

[17] Grefenstette, J. J. (Ed.), (1987). *Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms*, Hillsdale, NJ: Lawrence Erlbaum Associates.

[18] Holland, J. H., (1970). "Outline for a logical theory of adaptive systems," A. W. Burks (Ed.), *Essays on cellular automata*, pp. 297-319, University of Illinois Press.

[19] Holland, J. H., (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

[20] Johansson, R., (1986). *Identification of continuous time dynamic systems*, Technical Report, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[21] Kristinsson K. and G. A. Dumont (1988). "Genetic algorithms in system identification", Third IEEE international symposium on intelligent control, Arlington, VA, USA.

[22] Smith, T. and K. A. DeJong, (1981). "Genetic Algorithms applied to the calibration of information driven models of US migration patterns," *Proceedings of the* 12^{th} *Annual Pittsburgh conference on Modelling and Simulation*, pp. 955-959.

[23] Söderström, T., L. Ljung, and I. Gustavsson, (1974). *A comparative study of recursive identification methods*, Report 7427, Lund Institute of Technology, Lund, Sweden.

[24] Wittenmark, B. and B. J. Evans, (1988). "An adaptive pole placement controller based on pole-zero parameterization," *Preprint from the* 8^{th} *IFAC/IFORS symposium on identification and system parameter estimation*, pp.98-103, Beijing, China.

Appendix A

Genetic algorithms procedures

```
Program PPCGA

PROCEDURE Select a population ;
  BEGIN
    FOR all the population size DO
      FOR all the bits DO
        IF random > 0.5 THEN bit := 1
          ELSE bit := 0 ;
  END ;


PROCEDURE Schemata ;
  BEGIN
    { Count how many 1 there are in each bit position using one counter for
      each bit position. Then count lost bits by counting number of counters
      that have value equal to 0 or population size. Then count converged
      bits by counting number of counters that have value less than converged
      % or those with value greater than (1 - converged) %. }
  END ;


PROCEDURE System ;
```

BEGIN

{ Choose either PRBS input (no control) or setpoint changes }

IF setpoint changes THEN

Every bitinterval multiply the input y_{ref}(t) with -1.0 ;

IF PRBS input THEN

Every bitinterval make the PRBS sequence using shift register as given in Eykhoff [1974] ;

Find e(t), the normally distributed random sequence ;

y(t) := $\frac{B(q^{-1})}{A(q^{-1})}u(t) + \frac{C(q^{-1})}{A(q^{-1})}e(t)$;

Call the controller design with the best estimate to find the next controller output, u(t) ;

END ;

PROCEDURE Convert the strings into the parameters ;

BEGIN

FOR all the population DO

FOR all the substrings DO

BEGIN

x := decimal value of the binary substring ;

resolution := $\frac{\text{maximum value - minimum value}}{2^{\text{length of substring}} - 1}$;

y := x * resolution + minimum value ;

END ;

END ;

PROCEDURE Fitness evaluation ;

```
BEGIN
  FOR all the population DO
    BEGIN
      FOR t-window size TO t-0 DO
        BEGIN
          IF RLS THEN
            BEGIN
              ε := y − (B̂/Â) u ;
              fitness := bias - ε² + fitness ;
            END
          ELSE
            BEGIN
              ŷ := (B̂/Â) u ;
              η := y − ŷ ;
              fitness := bias - η² + fitness ;
            END
        END ;
    END ;
END ;
```

PROCEDURE Quicksort;

Use algorithm given in the book DATA STRUCTURE TECHNIQUES

by Thomas A. Standish page 25-27

or any other sorting algorithm

PROCEDURE Rank;

BEGIN

a := $2\frac{max-1}{popsize-1}$;

b := 1 - $\frac{max-1}{popsize-1}(popsize+1)$;

FOR all the popsize DO

fitness := a * rank + b ;

END ;

PROCEDURE Offspring ;

BEGIN

FOR all the population DO

BEGIN

Normalized fitness := fitness / meanfit ;

Intfit := integer value of normalized fitness ;

Offspring count := Intfit ;

Throw a dice to decide if the string gets one offspring for the fractional part or not ;

END ;

IF there are too few or too many in the population THEN choose the one that were most likely to get additional offspring and did not or the one that were most unlikely to get additional offspring and did, depending on whether the population size is too small or too large respectively. ;

END ;

```
PROCEDURE Copies ;
  BEGIN
    WHILE population size is not full DO
      BEGIN
        Next string ;
        WHILE offspring count ≤ 0 DO
          BEGIN
            Copy string ;
            Offspring count := offspring count - 1 ;
          END ;
      END ;
  END ;


PROCEDURE Crossover ;
  BEGIN
    FOR half the population size DO
      BEGIN
        Find the first string that has not been used ;
        Choose another string randomly ;
        Apply crossover with p_c probability ;
        IF crossover THEN
          BEGIN
            Choose crossover point randomly ;
```

```
            Copy first half of first string up to crossover point and
              second half of second string from crossover point to
              the end into the first offspring ;
            Copy first half of second string up to crossover point and
              second half of first string from crossover point to the
              end into the second offspring ;
          END
        ELSE
          BEGIN
            Copy first string into first offspring ;
            Copy second string into second offspring ;
          END ;
      END ;
  END ;


PROCEDURE Mutation ;
  BEGIN
    FOR all the population size DO
      FOR all the bits in each string DO
        Mutate each bit with p_m probability ;
  END ;


BEGIN
  Get all parameters ;
  Initialize random generators ;
```

```
Initialize system input and output ;
Print initial values ;
Select a population ;
Start with the initial estimate as 1, 0, ..., 0 ;
FOR kids := 1 TO number of kids DO
   BEGIN
      Schemata ;
      IF (kids-1) / trial = integer THEN system ;
      Convert the strings into the parameters ;
      Fitness evaluation ;
      Calculate the average fitness ;
      Offspring ;
      Count how many receive 0 offsprings ;
      IF (receive 0 offsprings) > (fit0pct * popsize) THEN
         BEGIN
                Quicksort ;
                Rank ;
                Offspring ;
         END ;
      Copies ;
      Crossover ;
      Mutation ;
      Find the best string ;
      IF it is not in the new population THEN
         Replace a string randomly chosen with the best one ;
```

```
        Print report ;
        Make the new generation the current one ;
    END ;
END ;
```