System Identification and Control Using Genetic Algorithms

Kristinn Kristinsson, *Member, IEEE*, and Guy A. Dumont, *Senior Member, IEEE*

Abstract—**It is shown how genetic algorithms can be applied for system identification of both continuous and discrete time systems. It is shown that they are effective in both domains and are able to directly identify physical parameters or poles and zeros. This can be useful because changing one physical parameter might effect every parameter of a system transfer function. The poles and zeros estimates are then used to design a discrete time pole placement adaptive controller. Simulations for minimum and nonminimum phase systems and a system with unmodeled dynamics are presented.**

I. Introduction

THE AREA of system identification has received a lot of attention over the last two decades. It is now a fairly mature field, and many powerful methods are at the disposal of control engineers. However, most of those techniques are for models that are linear in the parameters, and based upon the assumption of a smooth search space with ever present derivatives.

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 indeed can be described in a unified way [1]. Those recursive schemes are in essence local search techniques that search for the optimum by using a gradient-following technique. They often fail in the search for global optimum 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 feature of all these methods is that they go from one point in the search space to another at every sampling instant, as a new input–output pair becomes available. They do not iterate more than once on each datum received, as they need new data to direct the search.

A genetic algorithm (GA) is a parallel, global search technique that emulates natural genetic operators. Because it simultaneously evaluates many points in the parameter space, it is more likely to converge toward the global solution. It needs not assume that the search space is differentiable or continuous, and can also iterate several times on each datum received. A GA applies operators inspired by the mechanics of natural selection to a population of binary strings encoding the parameter space. At each generation, it explores different areas of the parameter space, and then directs the search to regions where there is a high probability of finding improved performance. By working with a population of solutions the algorithm can in effect search many local minima and thereby increases the likelihood of finding the global minima.

In this paper a GA is implemented as an estimator for discrete time and continuous time systems. The results obtained employing this algorithm are considered to be well suited to the adaptive control problem in combination with a pole placement scheme utilizing knowledge of poles and zeros. Although the use of GA's has gained some popularity in optimization [2], its use in adaptive control had not been investigated before. The algorithm is used on discrete time systems, both minimum and nonminimum phase to identify either parameters or poles-and-zeros. Although not intended as an alternative to least-squares based methods on linear-in-the-parameters systems, the genetic algorithm is compared to an instrumental variable method, using simulations.

The paper is organized as follows. In Section II, GA's are described and some of the simple genetic operators are explained. In Section III, a GA for system identification is implemented both in discrete and continuous time and simulation results are presented. The algorithm is also compared to a recursive instrumental variable algorithm. In Section IV a pole placement controller design based on knowledge of poles and zeros is outlined and simulation results are shown using the GA to identify plants with either minimum phase or nonminimum phase characteristic and unmodeled dynamics.

II. Genetic Algorithms

A. History of GA's

The underlying principles of GA's were first published by Holland in 1962 [3]. The mathematical framework was developed in the late 1960's, and is presented in Holland's pioneering book, *Adaptation in Natural and Artificial Systems* published in 1975 [4]. GA's have been used in many diverse areas such as function optimization [5], image processing [6], the traveling salesman problem [7], [8] and system identification [9]–[13]. In the last few years research devoted to GA's has significantly increased, as attested by the existence of several conferences on the topic, [14]–[17]. An excellent reference on GA's and their implementation is the recent book by Goldberg [18].

Manuscript received September 1, 1990; revised June 22, 1991, and January 18, 1992.

The authors are with the Department of Electrical Engineering, University of British Columbia, 2385 East Mall, Vancouver, BC, V6T 1Z4 Canada.

IEEE Log Number 9201408.

B. The Algorithm

A genetic algorithm differs from other search techniques by the use of concepts taken from natural genetics and evolution theory. First, the algorithm works with a population of strings, searching many peaks in parallel. By employing genetic operators it exchanges information between the peaks, hence reducing the possibility of ending at a local minimum and missing the global minimum. Secondly, it works with a coding of the parameters, not the parameters themselves. The coding that has been shown to be the optimal one [4] is binary coding. Intuitively, it is better to have few possible options for many bits than to have many options for few bits. Thirdly, the algorithm only needs to evaluate the objective function to guide its search. There is no requirement for derivatives or other auxiliary knowledge. The only available feedback from the system is the value of the performance measure (fitness) of the current population. Finally, the transition rules are probabilistic rather than deterministic. The randomized search is guided by the fitness value of each string and how it compares to others. By use of operators taken from population genetics the algorithm efficiently explores parts of the search space where the probability of finding improved performance is high.

GA's 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's fail are pathological, and generally fail to be optimized by any other known technique, except exhaustive search [19]. In a recent study by Goldberg [20], it has been shown that even though the algorithm is misled, it will converge for a wide range of starting conditions (initial populations), and under unfavorable conditions.

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

The basic element processed by a genetic algorithm is the string formed by concatenating substrings, each of which is a binary coding of a parameter of the search space. Thus, each string represents a possible solution to the problem. The genetic algorithm works with a set of strings, called the population. This population then evolves from generation to generation through the application of genetic operators. A genetic algorithm in its simplest form uses three operators: Reproduction, Crossover, and Mutation.

Reproduction: Reproduction is based on the principle of survival of the fittest. A fitness, $F(i)$, is assigned to each individual in the population where high numbers mean good fit. The fitness function can be any nonlinear, nondifferentiable, discontinuous, positive function, because the algorithm only needs a fitness assigned to each string. Based on a normalized fitness $F_n(i)$, the number of offsprings for each individual is calculated. First, the fitness is normalized with the average value of the fitness, $F_n(i) = F(i)/\bar{F}$. The strings with higher-than-average fitness will have more than one offspring, and those with below-average fitness will have less than one offspring on the average. To achieve this, the strings are selected according to what has become known as the stochastic remainder selection without replacement [18]. Accordingly, the strings receive a number of offsprings at least equal to the integer value of their normalized fitness. Then, the population is filled up by choosing another offspring for each of the strings with probability equal to the fractional part of the normalized fitness until the total number of offsprings equals the population size N.

The algorithm keeps track of the best string in the population and if it is not in the new population it is put into the new population by removing another string at random.

Ranking: It is important to regulate the number of offsprings an individual can have to maintain a diversity in the population. This is important for the first few generations, when a few "super" individuals can potentially take over a large part of the population, thereby reducing its diversity. The presence of super individuals can be sensed by monitoring the number of individuals receiving no offsprings. To control the reproduction, *ranking* can be introduced [21]. Whenever a certain ratio of the normalized fitness is receiving no offspring, the strings are sorted according to their fitness value. Then, instead of normalizing the fitness with the average, the normalized fitness is computed according to

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

where max 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].

Crossover: Reproduction directs the search toward the best existing individuals but does not create any *new* individuals. In nature, an offspring is rarely an exact clone of a parent, it usually has two parents and inherits genes from both. The main operator to work on the parents is crossover, which is applied with a certain probability, called crossover rate (p_c). 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 are cut randomly between any two bits on the strings. The new strings are then created by interchanging the tails. For example, the two strings:

00000000

11111111

with a crossover site at 3 become:

11100000

00011111.

Reproduction and crossover give GA's much of their power by directing the search toward the better areas using already existing knowledge.

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. The bits can only reproduce or die, so if at a 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 and is applied with a low probability p_m. In the case of binary coding, the mutation operator simply flips the state of a bit from 0 to 1 or vice versa. Mutation should be used sparingly because it is a random search operator, and with high mutation rates, the algorithm would become little more than a random search.

III. System Identification

A. Discrete Time Identification

Consider a system described by an ARMAX model

$$A(q^{-1})y(t) = B(q^{-1})u(t-d) + C(q^{-1})e(t) \tag{2}$$

where $A(q^{-1})$, $B(q^{-1})$ and $C(q^{-1})$ are polynomials in the backward shift operator, q^{-1}, i.e., $y(t-1) = q^{-1}y(t)$ and $y(t), u(t)$ and $e(t)$ 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 $A(q^{-1})$ and $C(q^{-1})$ are assumed to be monic. The objective is to estimate $A(q^{-1})$, $B(q^{-1})$ and the delay d, given the input $u(t)$ and the output $y(t)$. A sequence $\eta(t)$ can be defined, for calculating how well the estimates fit the system, as

$$\eta(t) = y(t) - \hat{y}(t) \tag{3}$$

with

$$\hat{A}(q^{-1})\hat{y}(t) = \hat{B}(q^{-1})u(t-\hat{d}) \tag{4}$$

where $\hat{y}(t)$ is the output of a deterministic system driven by the actual input $u(t)$ and $\hat{A}(q^{-1})$ and $\hat{B}(q^{-1})$ are the estimates of $A(q^{-1})$ and $B(q^{-1})$ respectively, found by minimizing $E[\eta^2(t)]$. This corresponds to an output-error identification method, akin to the instrumental variable (IV) case, and has a highly non linear search space (see Fig. 1). The fitness function needs to be maximized so it is chosen as

$$F(t) = \sum_{i=0}^{w} M - (\eta(t-i))^2 \tag{5}$$

where M is a bias term needed to ensure a positive fitness and w is the window size or the number of time steps the fitness is accumulated over, with an effect akin to that of the forgetting factor in RIV. It has been shown via simulation [22] that the variance of the parameter estimates reduces as the window size increases. However, execution slows down as the window size increases. Because the fitness function is not recursive, 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. There is however a practical upper limit because of time constraints. The input has to be persistently exciting in the window [22] for the algorithm to converge to the true value.

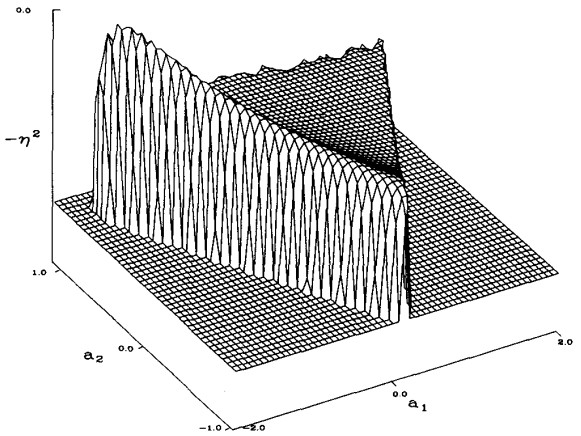


Fig. 1. Search space for $\eta^2(t)$ with PRBS input, for varying a_1 and a_2. The true parameters for a_1 and a_2 are -1.5 and 0.7 respectively

Pole-Zero Identification: Because they do not require linearity in the parameters, GA's can directly identify the poles and zeros of the system in (2). In pole-zero form, the plant can be written as

$$\begin{aligned} A(q^{-1}) &= (1 - p_1 q^{-1})(1 - \bar{p}_1 q^{-1}) \\ &\quad \cdots (1 - p_m q^{-1})(1 - \bar{p}_m q^{-1}) \\ B(q^{-1}) &= b_0(1 - z_1 q^{-1})(1 - \bar{z}_1 q^{-1}) \\ &\quad \cdots (1 - z_m q^{-1})(1 - \bar{z}_m q^{-1}) \end{aligned} \tag{6}$$

where $\bar{p}_i$ denotes the complex conjugate of p_i and $m = n/2$ 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. In the general case p_i is a complex number so there would be $4m$ parameters to be identified for A. But we can make use of the fact the roots always come in complex conjugate pairs or pairs of real roots. We can then denote the average of the pairs (α) and their deviation from the average (β), where positive deviation ($\beta > 0$) means pair of real roots ($\alpha \pm \beta$) and negative deviation ($\beta < 0$) means pair of complex roots ($\alpha \pm j\beta$). So we can write (6) as

$$\begin{aligned} A(q^{-1}) &= (1 - (\alpha_1 \pm \beta_1)q^{-1}) \\ &\quad \cdots (1 - (\alpha_m \pm \beta_m)q^{-1}) \\ B(q^{-1}) &= b_0(1 - (\gamma_1 \pm \delta_1)q^{-1}) \\ &\quad \cdots (1 - (\gamma_m \pm \delta_m)q^{-1}). \end{aligned} \tag{7}$$

We can represent our parameterization by a plane where the horizontal axis represents the average value and the vertical axis the deviation from the average. Every point in this new plane will represent two points, complex conjugate poles if it is in the lower half plane and two real poles if it is in the upper half plane. That gives search space for a stable system of the form seen in Fig. 4. A root-locus plot in the complex plane and the reparameterized plane are then shown in Fig. 5 to show the transform from the lower half plane to the upper half plane.

A "standard" system that has been used often in the literature [1], [23], [24] to test different identification methods is

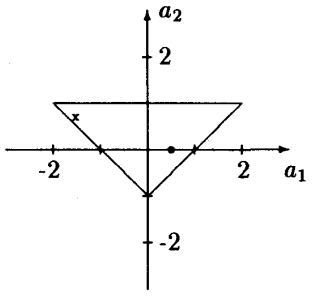


Fig. 2. Parameters.

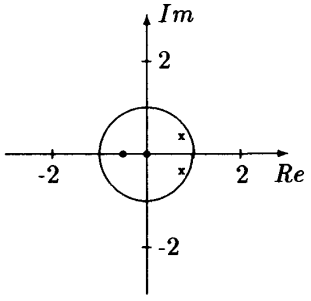


Fig. 3. Poles and zeros in the complex plane.

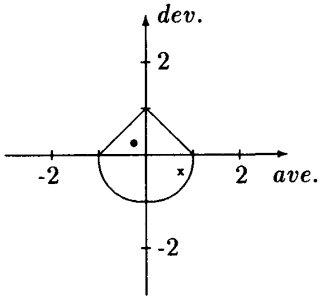


Fig. 4. Reparameterized complex plane.

(see Fig. 2):

$$\begin{aligned} A(q^{-1}) &= 1.0 - 1.5q^{-1} + 0.7q^{-2} \\ B(q^{-1}) &= 1.0(1.0 + 0.5q^{-1} + 0.0q^{-2}) \\ C(q^{-1}) &= 1.0 - 1.0q^{-1} + 0.2q^{-2} \end{aligned} \tag{8}$$

the poles and zeros are (see Fig. 3):

$$\begin{aligned} \text{poles}: &\quad (0.75 \pm j0.37) \\ \text{zeros}: &\quad [-0.5, 0] \end{aligned} \tag{9}$$

or in the reparameterized plane (see Fig. 4):

$$p_{1,2} = [0.75, -0.37] \quad z_{1,2} = [-0.25, +0.25]. \tag{10}$$

For a stable minimum phase plant, the poles and zeros are inside the unit circle (Fig. 3), therefore the search space can be limited to be the unit circle. For a nonminimum phase plant some zeros will be outside the unit circle, so one has to decide how big the search space is going to be depending on prior knowledge of the system.

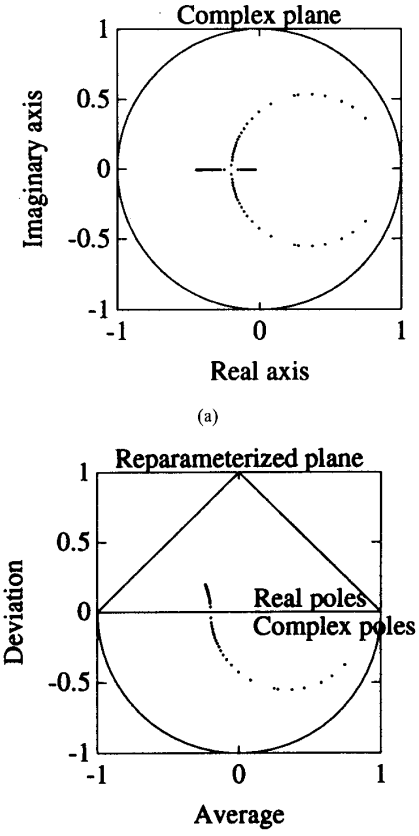


Fig. 5. Root-locus plots in the (a) complex plane and the (b) reparameterized plane

Results: The crossover rate, p_c, is chosen so as to give some of the population the opportunity to survive into the next generation without any changes. The mutation rate, p_m, 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 in Das and Goldberg [9]:

$$\begin{aligned} p_c &= 0.8 \\ p_m &= 0.01 \\ \text{population} &= 100 \end{aligned} \tag{11}$$

The previous second-order system was used, with six parameters to be identified, that is d and b_0 and poles-zeros α_1, β_1, γ_1, and δ_1. The delay is coded as a two bit string to give four choices for the delay, from 1 to 4. The gain, b_0, is assumed to be between 0 and 2. Because the system is assumed to be stable and minimum phase, a $[-1, 1]$ range for $\alpha_1, \beta_1, \gamma_1$ and δ_1 suffices. Choosing the resolution to be slightly finer than 0.02, requires seven bits for each parameter giving a 37 bit string, which leaves the search space with $2^{37} = 1.37\ 10^{11}$ alternatives.

A PRBS input with period of 127, and bit interval equal to four times the sampling interval is used as test signal. The GA is run for 600 generations, going through three generations for each input–output data, so each run is 200 samples. The

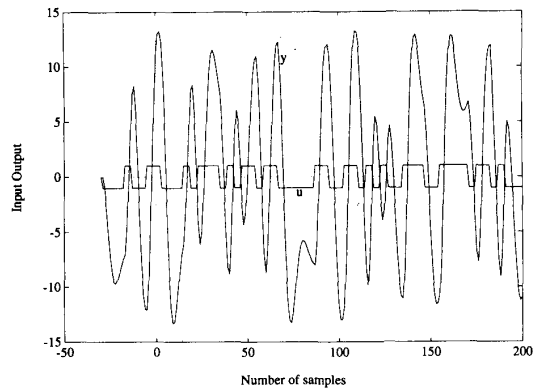


Fig. 6. PRBS input and output of a system without noise.

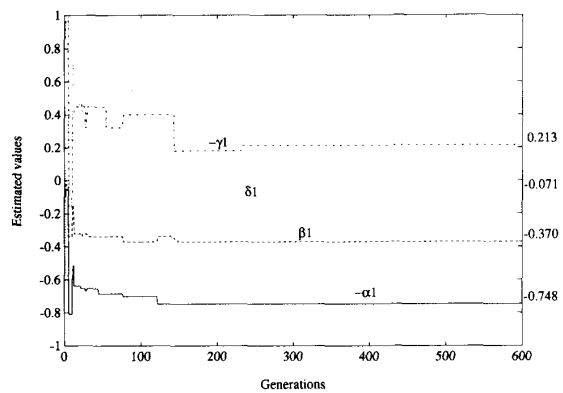


Fig. 7. GA, pole-zero estimate of a system without noise using PRBS input.

window size is 30, i.e., the fitness function is calculated for the current input–output and the 30 previous samples.

On a noise free system, the recursive instrumental variable method is known to converge in as many samples as there are parameters. It is however not able to estimate delays or poles and zeros. Fig. 6 shows the output of the noise-free process with the PRBS input. Figs. 7 and 8 show the estimated parameters for each generation. 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 converges to the true values for the poles but shows some bias for the zeros. Parameters $-\gamma_1$ and δ_1 (0.213 and -0.071) should both be 0.250 so a bias of 0.321 for δ_1 seems rather large. The reason for more bias in δ_1 than γ_1 is because for our particular problem the output of the system, and therefore our objective function, is 5 times more sensitive for deviations in γ_1 than δ_1. The reason the zeros are biased but the poles are not is because the steady-state gain of the system is 7.5 so (5) 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." Fig. 9 shows how the poles and zeros move around in the complex plane for each generation where the initial generation is at the back and the

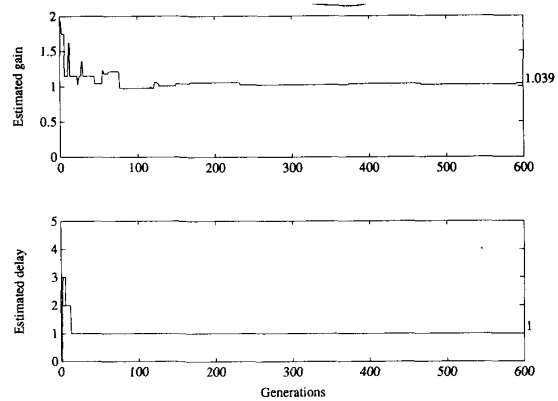


Fig. 8. GA, estimation of gain and delay of a system without noise using PRBS input.

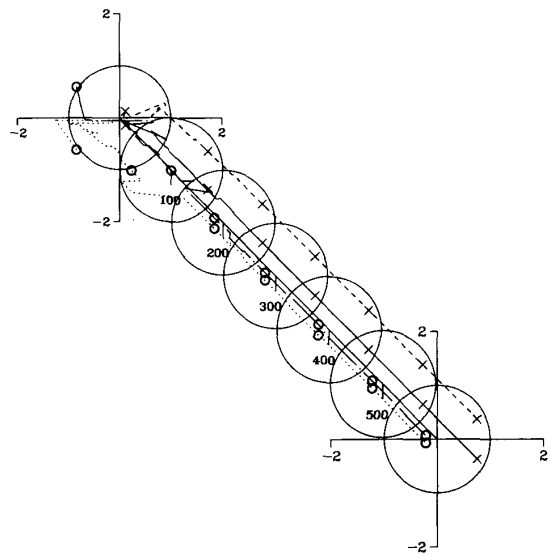


Fig. 9. GA, estimated pole zero locations of a system without noise using PRBS input. The unit circles with x and o marking locations of poles and zeros are drawn every 100 generations

final generation is in front. The unit circle is also plotted every 100 generations together with the estimates at that point.

B. Recursive Instrumental Variable Estimation

For the sole purpose of a comparison with a widely known identification method, we introduce a recursive instrumental variable (RIV) method [1]. The actual process is assumed to be described by

$$A(q^{-1})y(t) = B(q^{-1})u(t-1) + C(q^{-1})e(t) \qquad (12)$$

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

$$\begin{aligned} A(q^{-1}) &= 1.0 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a} \\ B(q^{-1}) &= 1.0 + b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b} \\ C(q^{-1}) &= 1.0 + c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c}. \end{aligned} \qquad (13)$$

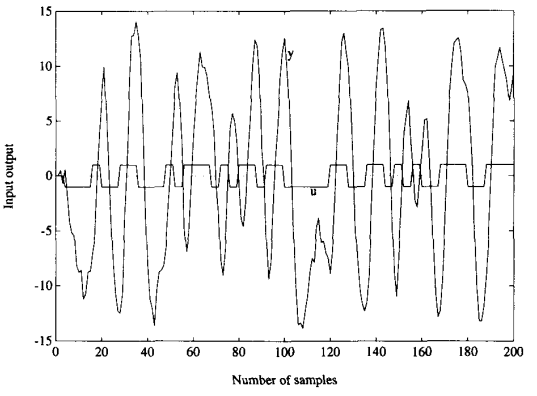


Fig. 10. PRBS input and output with noise

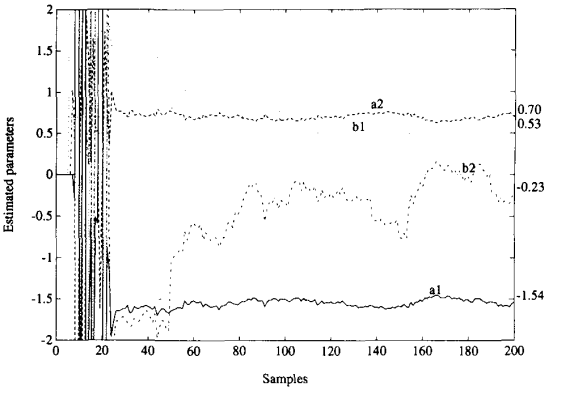


Fig. 11. RIV, parameter identification, $\lambda = 0.97$, using the PRBS input with noise. The solid lines are the true parameters.

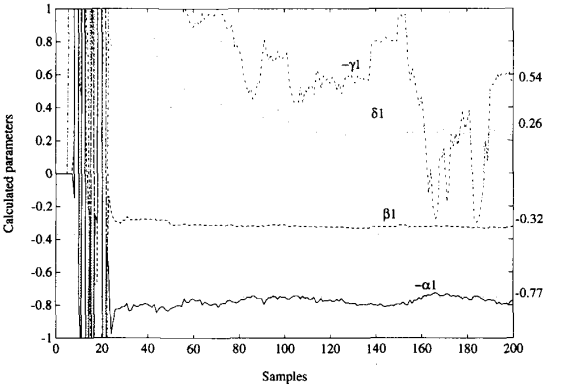


Fig. 12. RIV, pole zero identification calculated from the parameters. The solid lines are the true parameters.

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

$$\begin{aligned}\theta &= [a_1, \cdots, a_{n_a}, 1, b_1, \cdots, b_{n_b}]^T \\ \hat{\theta} &= \left[\hat{a}_1, \cdots, \hat{a}_{n_a}, 1, \hat{b}_1, \cdots, \hat{b}_{n_b}\right]^T \\ \varphi(t) &= [-y(t-1), \cdots, -y(t-n_a), u(t-1), \\ &\quad \cdots, u(t-n_b-1)]^T.\end{aligned} \tag{14}$$

The system output can be written as

$$y(t) = \varphi(t)^T \hat{\theta} + \epsilon(t). \tag{15}$$

Unbiased estimate of $\hat{\theta}$ can be obtained for the general case when $n_c \neq 0$, i.e., colored noise, using recursive instrumental variable (RIV) [23]. The estimates can be calculated by

$$\begin{aligned}\hat{\theta}_{t+1} &= \hat{\theta}_t + K_{t+1}\varepsilon_{t+1} \\ K_{t+1} &= \frac{P_t\zeta_{t+1}}{\lambda + \varphi_{t+1}^T P_t \zeta_{t+1}} \\ P_{t+1} &= \left(1 - K_{t+1}\varphi_{t+1}^T\right)\frac{P_t}{\lambda} \\ \varepsilon_{t+1} &= y_{t+1} - \varphi_{t+1}^T\hat{\theta}_t\end{aligned} \tag{16}$$

where λ is the forgetting factor. A common choice for the instruments is

$$\zeta(t) = [-\hat{y}(t-1), \cdots, -\hat{y}(t-n_a), u(t-1), \cdots, u(t-n_b-1)]^T \tag{17}$$

where $\hat{y}(t)$ is the output of a deterministic system, with the current estimates as its parameters, driven by the system input

$$\hat{y}(t) = \zeta(t)^T\hat{\theta}(t) \tag{18}$$

or

$$\hat{A}(q^{-1})\hat{y}(t) = \hat{B}(q^{-1})u(t-1) \tag{19}$$

which is the same equation as used for fitness calculation for the GA.

Results: The (RIV) algorithm is run for the system of (12) with the polynomials $A(q^{-1})$, $B(q^{-1})$, and $C(q^{-1})$ described by (8) with the noise variance, $\sigma_e^2 = 1.0$. The forgetting factor is set close to 1.0 because it is a time-invariant system, so a value of 0.97 is chosen. The parameters a_1, a_2, b_1 and b_2 are then identified, where the true values from (8) are -1.5, 0.7, 0.5 and 0.0 respectively. Fig. 11 shows the result using the input shown in Fig. 10. It can be seen that the estimates for a_1 and a_2 take about 25 samples to converge whereas the estimates for b_1 and b_2 take about 100 samples to converge and vary much more than the estimates for the a_i's. To compare those results with the GA we need to calculate the poles and zeros from the RIV estimates. Fig. 12 shows the pole-zeros calculated from the parameters. From the calculated poles and zeros locations, it can be seen that the poles converge in about 25 samples but the imaginary part (β_1) has a constant bias. The zeros have more problem converging but γ_1 converges in around 100 samples, and δ_1 does not converge within the 200 samples. Fig. 13 shows how the poles and zeros evolve with time, with the time axis running out of the page and the circles for 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 RIV.

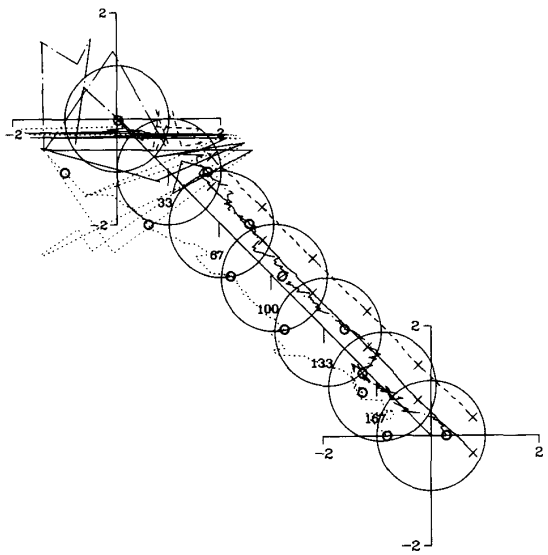


Fig. 13. RIV, calculated pole zero locations from the parameter estimates. The unit circles with x and o marking locations of poles and zeros are drawn every 50 samples

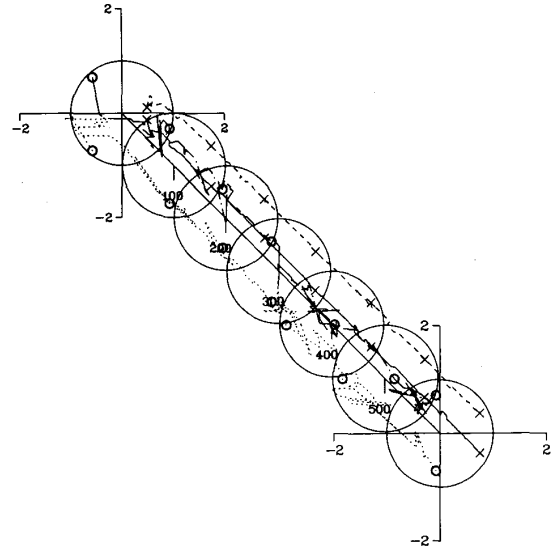


Fig. 15. GA, estimated pole zero locations. The unit circles with x and o marking locations of poles and zeros are drawn every 150 generations (i.e., 50 samples)

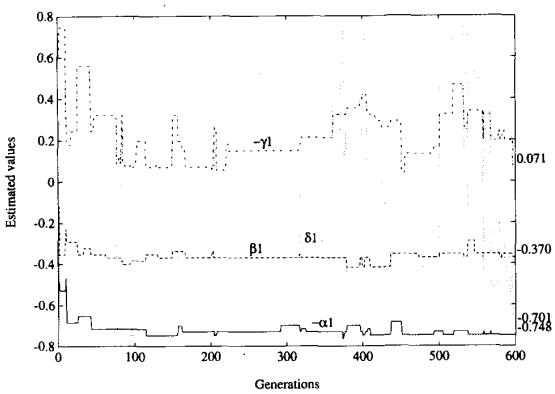


Fig. 14. GA, pole zero identification using PRBS input with noise

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 the same parameter settings as previously it gives a total string length of 28 bits, so the population size is reduced to 50. The results of the pole-zeros identification is shown in Fig. 14, and a 3-D Fig. that shows how the poles and zeros estimate evolve with time is shown, Fig. 15. It can be seen that the poles converge in about 50 generations and have almost zero bias, they are only limited by the resolution of the search space. The zeros on the other hand do not converge in the 600 generations (200 samples), $-\gamma_1$ is close to the true value of 0.25 but δ_1 is all over the search space.

If Fig. 12 and Fig. 14 are compared, it can be seen that RIV needs about 25 samples for the poles to converge but the zeros take about 100 samples. The GA needs around 50 generations for the poles to converge, which means that with 3 trials per sample it needs 17 samples but the zeros do not converge but are around the correct values. It can be seen that the imaginary part of the poles has a constant bias for the RIV estimate, whereas the GA gives zero bias for the poles. For the zeros, RIV seems to do a little better at least for $-\gamma_1$ but neither method converges really well for δ_1. The transients are much wilder for the RIV than for the GA where the estimates are forced to lie within certain bounds. The transients for the RIV can be reduced somewhat by reducing the initial value of the covariance matrix from the applied value of 10 000. In terms of number of samples the convergence rate is similar for both methods. But as mentioned earlier the fitness function for the GA cannot be calculated recursively so the algorithm has to calculate the output for the whole window. Every calculation involves $(n_a + n_b + 1)$ multiplications and $(n_a + n_b)$ additions and that has to be done for the whole window and all the population.

C. Continuous Time Identification

Consider a nth-order system, written in terms of its poles and zeros:

$$y(t) = \frac{z_0(p - z_1)\cdots(p - z_{n-1})}{(p - p_1)\cdots(p - p_n)}u(t) \tag{20}$$

where $p = d/dt$ and coefficients z_i and p_i are unknown. The goal is to estimate directly the unknown roots z_is and p_is 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 RIV for identification of the parameters of the model and consequently the parameters of the continuous time system [25]. The poles and zeros would then have to be calculated explicitly. One way to do the estimation using the GA is to find the parameters $\hat{\theta}$ ($\hat{\theta} = [\hat{z}_0, \cdots, \hat{z}_{n-1}, \hat{p}_1, \cdots, \hat{p}_n]^T$) such that the area of the difference between the actual output y

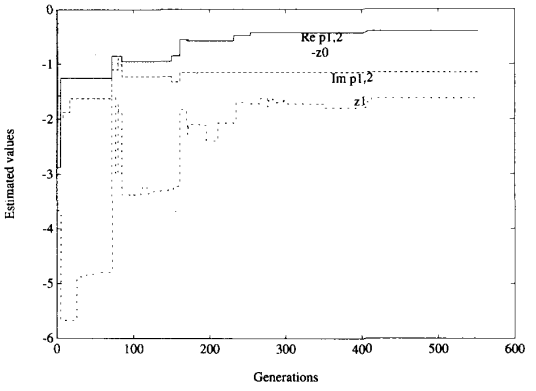


Fig. 16. GA, continuous-time estimates of poles and zero

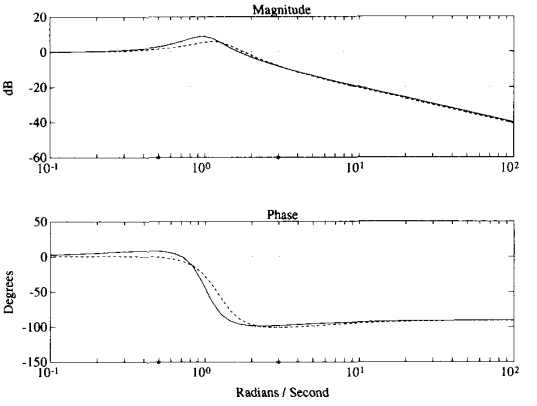


Fig. 17. Bode plot of the actual plant (solid lines) and the estimated plant after 450 generations (dashed line)

and the output of the model $\hat{y}$ over a time window W is the smallest:

$$\min_{\hat{\theta}} \int_W (y(t) - \hat{y}(t, \hat{\theta}))^2 \tag{21}$$

At every sampling time T_s the area is calculated for previous W s, setting $\hat{y}(kT_s - W) = y(kt_s - W)$. As in the discrete time case, the algorithm can be run several times for each sampling time.

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:

$$\frac{y(t)}{u(t)} = \frac{1.0p + 1.0}{p^2 + 0.5p + 1.0} = \frac{1.0(p - (-1.0))}{p - (-0.25 \pm j0.968)}. \tag{22}$$

The GA is used to find all four parameters of the plant $(1.0, -1.0, -0.25, j0.968)$, with the same parameterization as before, where the complex axis, j, is represented by a minus sign. For a stable system the estimates should be in the left half-plane and the poles are assumed to be complex. This means that all the parameters except the gain, z_0, are negative. The string length is chosen as 9 bits and the maximum magnitude of all the parameters is chosen as 12.775 to give a resolution of 0.025. The total string length is then 36 bits so the GA parameters have been set as in (11). The input $u(t)$ has been chosen as

$$u(t) = 0.3 \sin 0.5t + \sin 3.0t. \tag{23}$$

The input is chosen so as to excite the system both above and below the natural frequency, with about the same amplitude in the output. The sampling time T_s is 0.5 s and the window is chosen as 8 s. The parameter estimates are shown in Fig. 16. The convergence toward the actual values is slow. After 450 generations, i.e., 150 sampling intervals the estimated plant is

$$\frac{0.925(p - (-1.625))}{p - (-0.400 \pm j1.150)} = \frac{0.925p + 1.531}{p^2 + 0.800p + 1.4825}. \tag{24}$$

It has a natural frequency $\omega_n = 1.22$ and a damping ratio $\xi = 0.329$ instead of 1.0 and 0.25 respectively. The Bode plot of the actual and the estimated plant after 450 generations are plotted in Fig. 17. It is seen that the estimated plant fits the Bode plot fairly well at the frequencies of the exciting signal.

D. Estimation of Friction

We now look at the estimation of a physical parameter, namely the friction of a motor. For our purposes a motor with friction torque T_f and a load disturbance torque T_l can be described by the following simple model [26]:

$$J\frac{d\omega}{dt} = KI(t) - T_f(t) + T_l(t) \tag{25}$$

where J is the total moment of inertia, K is the current constant and I is the motor current. It is suggested in [26] to neglect the load disturbance and introduce

$$I(t) = u(t) + \frac{\hat{T}_f(\omega)}{K} \tag{26}$$

where $\hat{T}_f$ is an estimate of the friction torque, the motor model can then be written as

$$J\frac{d\omega}{dt} = Ku(t) + \left\{\hat{T}_f(\omega) - T_f(\omega)\right\}. \tag{27}$$

With a good estimate, the term inside the bracket vanishes and the model looks like a frictionless motor.

Many models for the friction have been suggested but a model used in [26] has been adopted. The model is (see Fig. 18):

$$T_f(\omega) = \begin{cases} \alpha_1\omega + \beta_1 & \omega > 0 \\ \alpha_2\omega + \beta_2 & \omega < 0 \end{cases}. \tag{28}$$

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 any given time, depending on the sign of the angular velocity ω. GA can be applied to this problem once an 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

$$\min_{\theta} \int_W (\omega(t) - \hat{\omega}(t))^2 dt \tag{29}$$

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

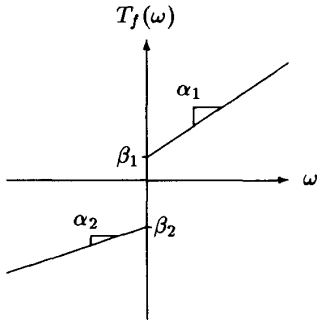


Fig. 18. Friction model.

Results: The parameters of the friction model have been chosen as:

$$\begin{aligned} \alpha_1 &= 0.1 \qquad \beta_1 = 0.4 \\ \alpha_2 &= 0.2 \qquad -\beta_2 = 0.2 \end{aligned} \tag{30}$$

and the motor is assumed to have $K = 1.0$ and $J = 0.1$. Here, as in [26], we assume perfect knowledge of K and J. 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 than zero α_1 and β_1 are identified and when the output is less than zero, 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 (29) is chosen as 2 s and there are 3 trials for each sample, with sampling time 0.2 s. The input is a square wave with period equal to 12 s, which can be seen together with the output in Fig. 19. Fig. 20 shows the estimated parameters using GA and the final value that the GA comes up with is

$$\begin{aligned} \hat{\alpha}_1 &= 0.100 \qquad \hat{\beta}_1 = 0.400 \\ \hat{\alpha}_2 &= 0.205 \qquad -\hat{\beta}_2 = 0.180 \end{aligned} \tag{31}$$

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. Here we are able to identify the parameters of the friction model directly, whereas using RIV as in [26] the estimates depend upon the sampling time and moment of inertia.

IV. Adaptive Control

A. Pole Placement Controller

We now look at how the GA's can be used in an indirect adaptive controller based on pole placement. 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. With the plant parameterized into poles and zeros it is possible to parameterize the controller such that the solution to the Diophantine equation corresponds to a solution of a linear triangular system of equations where the parameters can be calculated recursively [27] (see Appendix for details).

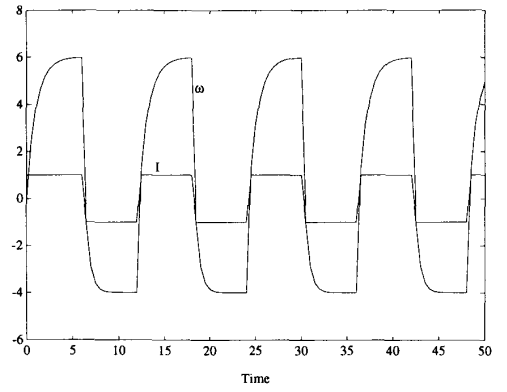


Fig. 19. Motor input (I) and output (ω)

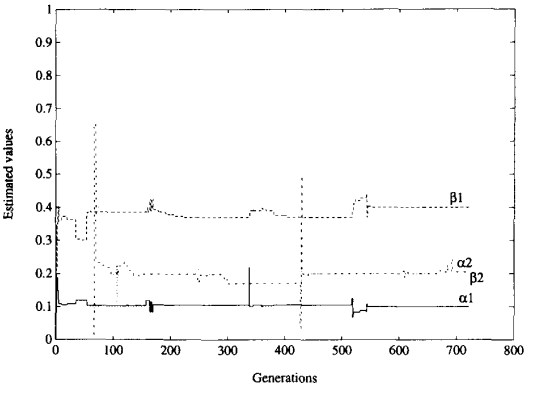


Fig. 20. GA, friction parameters identification

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 has deadbeat response and the observer, A_o, is chosen as $q^{n+d-m-1}$. A pulse is used to initialize the GA, i.e., to fill up the window.

Minimum Phase Plant: The plant to be controlled is the same one as used in Section III:

$$A(q^{-1})y(t) = B(q^{-1})u(t-1) + C(q^{-1})e(t) \tag{32}$$

with

$$\begin{aligned} A(q^{-1}) &= 1.0 - 1.5q^{-1} + 0.7q^{-2} \\ B(q^{-1}) &= 1.0(1.0 + 0.5q^{-1} + 0.0q^{-2}) \\ C(q^{-1}) &= 1.0 - 1.0q^{-1} + 0.2q^{-2}. \end{aligned} \tag{33}$$

The poles and zeros are identified using the criterion (5) with the same parameters settings as previously.

Fig. 21 shows the reference input and the output of the plant when the standard deviation of the white noise is $\sigma_e = 0.1$. By looking at Fig. 22 and Fig. 23 it can be seen that good estimates are found after 50 generations or 17 samples, but there is still a small bias in the estimates especially the zeros. It should be noted that the bias in the zeros corresponds to a small bias in the parameters. For example the final estimates for $-\gamma_1$ and δ_1 of 0.228 and 0.008 (true values 0.25 and 0.25) gives the parameters 0.456 and 0.052, which is not far from

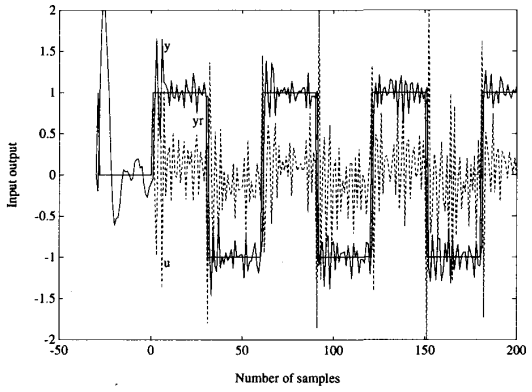


Fig. 21. Reference input and output of a minimum phase system with noise using pole-zeros estimates.

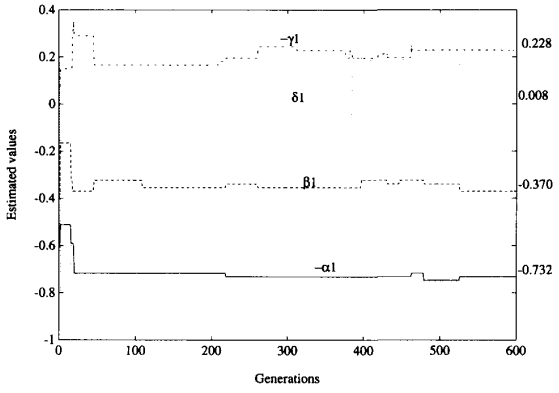


Fig. 22. Pole-zero estimates for a minimum phase system with noise.

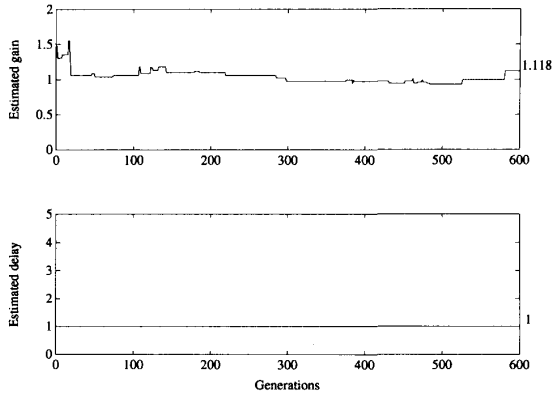


Fig. 23. Estimates of gain and delay for a minimum phase system with noise.

the true values of 0.5 and 0.0. The estimates in the complex plane for every generation are then shown in Fig. 24.

Nonminimum Phase Plant: The plant to be controlled is taken from Clarke [28] as

$$\frac{0.1q^{-1}(1+2q^{-1})(1+0q^{-1})}{(1-0.9q^{-1})(1-0.8q^{-1})}. \tag{34}$$

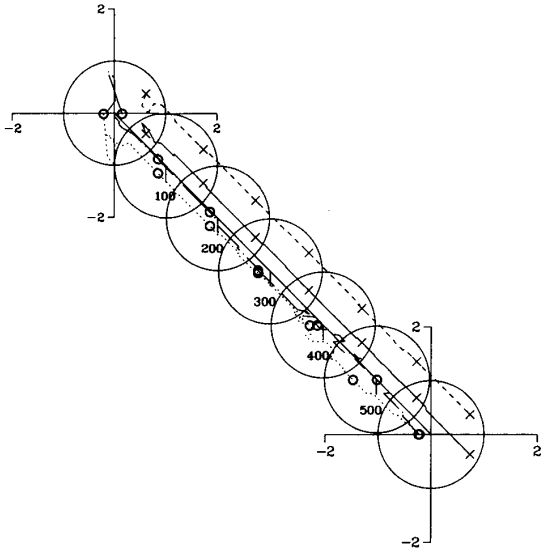


Fig. 24. Estimated pole zero locations for a minimum phase system with noise. The unit circles with x and o marking the locations of poles and zeros are drawn every 100 generations.

Using the parameterization given in Section III the plant parameters are as follows:

$$\begin{aligned} b_0 &= 0.1 \quad d = 1 \\ \alpha_1 &= 0.85 \quad \gamma_1 = 1.0 \\ \beta_1 &= 0.05 \quad \delta_1 = 1.0. \end{aligned} \tag{35}$$

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 to account for the nonminimum phase behavior. The observer is chosen to be deadbeat and the desired closed loop plant is assumed to have poles at 0.2 and 0.0 and because unstable process zeros $B^-(q^{-1})$ are not cancelled the desired transfer function becomes:

$$\frac{q^{-1}B^-(q^{-1})}{1-0.2q^{-1}}. \tag{36}$$

It is not until after 300 generations or 100 time steps that estimates (Fig. 26, Fig. 27 and Fig. 28) are found that give a good control and the output is following the reference input quite nicely as can be seen in Fig. 25. The estimates have a small bias but that does not seem to affect the output.

Unmodeled Dynamics: We use the same NMP plant as in the previous section (Section IV-A-2) but now the GA uses a first-order model

$$\frac{b_0q^{-d}(1+b_1q^{-1})}{1+a_1q^{-1}}. \tag{37}$$

As in Clarke [28], the system is initially run in open loop and then there are setpoint changes every 25th step. The window is chosen as 50 samples and the search space is defined as previously for a stable system, except for the zero, which is assumed to have 9 bits and be between -2 and 20 (population size = 50). The desired pole is set at 0.7 and the response obtained is as seen in Fig. 29. The estimated model becomes (see Fig. 30 and Fig. 31)

$$\frac{0.662q^{-3}(1+0.712q^{-1})}{1-0.937q^{-1}}. \tag{38}$$

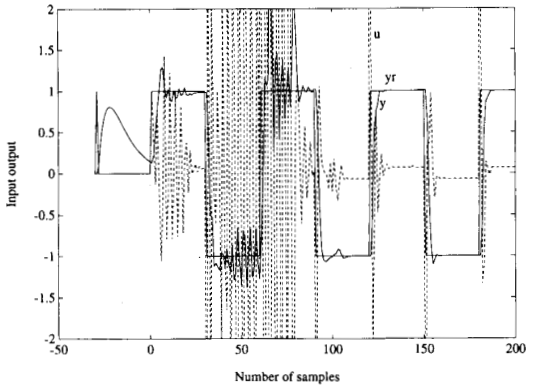


Fig. 25. Reference input and output for a nonminimum phase system

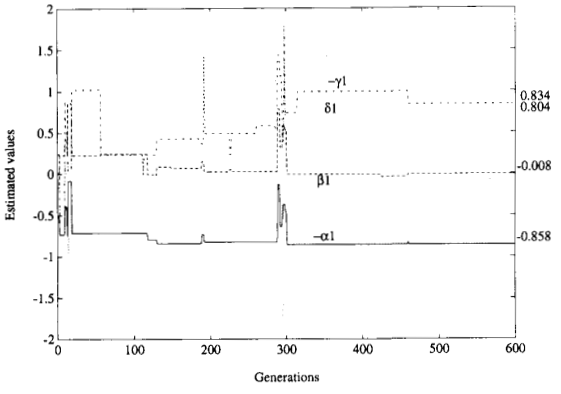


Fig. 26. Parameter estimate for a nonminimum phase system

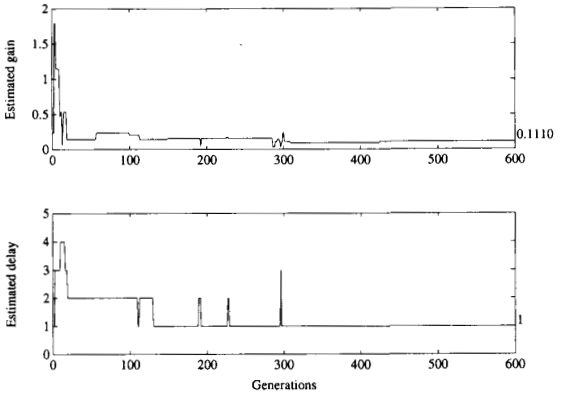


Fig. 27. Gain and delay estimate for a nonminimum phase system

Note that the estimated delay is now three sampling intervals, because it is trying to compensate for the nonminimum phase and the missing pole. Fixing the delay at one sampling interval as in [28] gives worse performance and can lead to a growing oscillatory response.

B. Multimodel Adaptive Control

An interesting feature of a genetic algorithm is that it simultaneously estimates several models of the process. It thus provides the opportunity for multimodel adaptive control. Multimodel adaptive control is an attractive way of dealing with abrupt system changes. For instance in the adaptive forgetting through multiple models (AFMM) method proposed by Anderson [29], standard Bayesian estimation theory is used to kill the least likely model and to create a new model from the most likely one. A genetic algorithm provides another way of regenerating the population of models while keeping its size constant. Multimodel adaptive control is also thought to have attractive robustness properties. Further, because GA's search for good regions (as opposed to points) of the parameter space, we believe they should even have enhanced robustness properties.

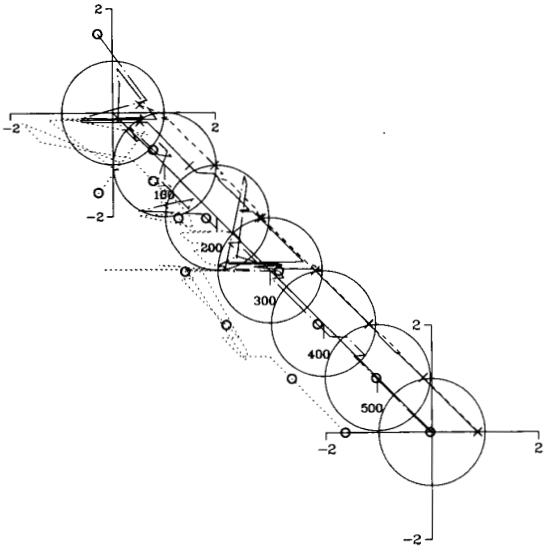


Fig. 28. Estimated pole zero locations for a nonminimum phase system. The unit circles with x and o marking the locations of poles and zeros are drawn every 100 generations

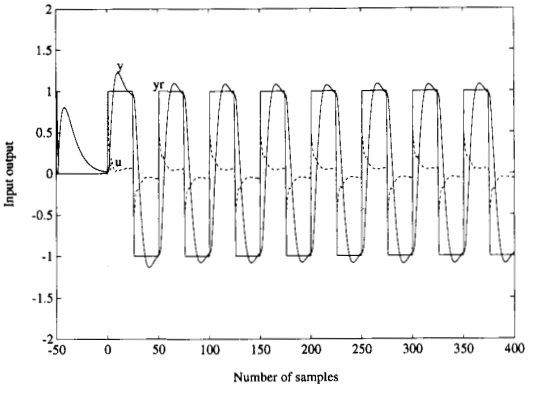


Fig. 29. Reference input and output for a system with unmodeled dynamics. The desired pole = 0.7.

V. Conclusion

In this paper it has been demonstrated how a genetic algorithm can be used to estimate both continuous and discrete

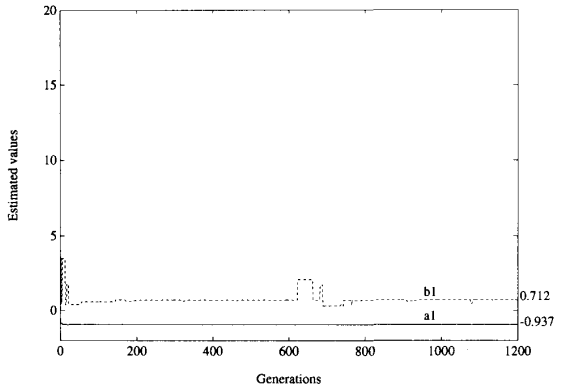


Fig. 30. Parameter estimates for unmodeled dynamics with the desired pole = 0.7.

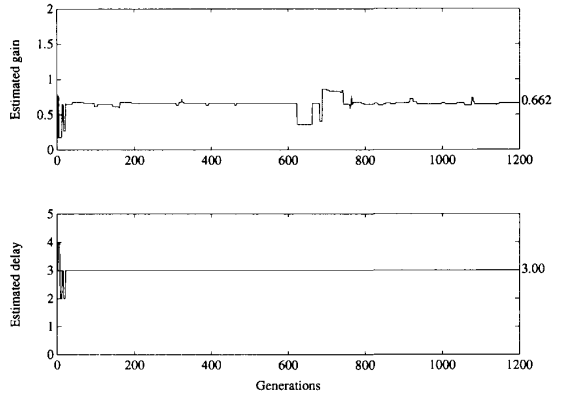


Fig. 31. Gain and delay estimate for unmodeled dynamics with the desired pole = 0.7.

time systems and for identifying poles and zero or physical parameters of a system. The algorithm has proven to be robust, as in all the applications the basic algorithm stays the same, the only differences being in the encoding of the string, and a different routine to calculate the fitness function.

In all the applications shown in this paper, the GA has been able to converge toward the actual values of the parameters, although there are no known necessary and sufficient conditions for its convergence. In most cases unbiased estimates are obtained. In some cases, like the identification of the zeros, convergence is slow, primarily because the objective function is not as sensitive to changes in the zeros as it is to 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 techniques, RIV, it performs as well or even better in terms of number of samples required to converge. We see this methodology of interest for problems where the application of conventional techniques is difficult, due to severe nonlinearities, discontinuities, etc... For instance, this methodology has recently been used to successfully identify the compliances of an industrial-size hydraulic manipulator, using normal operating data [30].

It has been shown how knowledge of the poles-and-zeros of a plant can be used to design a pole-placement controller. Simulation results show that the algorithm is as well fit for doing the identification as the RIV. It is able to handle both minimum and nonminimum phase systems and has also shown its ability to function in presence of unmodeled dynamics. We are also actively pursuing the use of GA's in multimodel adaptive control.

But as mentioned previously, the GA's are fairly computer intensive and the CPU time ratio for the comparison done between RIV and GA is about 1:50 on a μ-VAX. They are inherently parallel so they could make use of a parallel architecture. As a result of this study, we have been working on a parallel implementation of GA's, and obtained a system capable of real time performance.

APPENDIX
POLE PLACEMENT CONTROLLER DESIGN BASED ON POLE-ZERO PARAMETERIZATION

Wittenmark and Evans [27] have come up with a pole-zero placement algorithm based on pole-zero parameterization. By assuming knowledge of the plants poles and zeros the solution of the Diophantine equation corresponds to a solution of a linear triangular system of equations where the controller parameters can be calculated recursively.

Here we show modifications to Wittenmark and Evans algorithm where we cancel some process zeros. Furthermore, the $S(q)$ polynomial does not necessarily contain the factor q^d and some minor typographical mistakes in Wittenmark and Evans are corrected.

A SISO plant is described by an ARMAX model:

$$y(t) = \frac{B(q)}{q^d A(q)} u(t) + \frac{C(q)}{A(q)} e(t). \tag{39}$$

The control law for a two-degree of freedom pole-placement controller (Fig. 32) can be written as

$$R(q)u(t) = -S(q)y(t) + T(q)y_r(t) \tag{40}$$

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. The closed loop system is then

$$y(t) = \frac{BT}{q^d AR + BS} y_r(t) + \frac{q^d CR}{q^d AR + BS} e(t). \tag{41}$$

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

$$q^d A\bar{R} + B^- S = A_o A_m \tag{42}$$

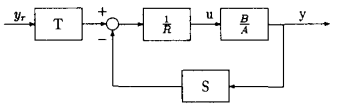


Fig. 32. Two-degree of freedom controller

and T can be found from

$$T = B'_m A_o t_0 \tag{43}$$

where t_0 is to ensure proper gain and A_o is the observer polynomial that is cancelled in the transfer function [24]. In order to design the controller we need to solve the Diophantine equation (42) for $\bar{R}$ and S. In order to find a unique solution, $\deg S < \deg A + d$. We choose $\deg S = \deg A + d - 1$. Using the causality conditions ($\deg R \geq \deg S$) the degree of A_0 can be found from (42) to be

$$\deg A_o \geq 2(\deg A + d) - \deg A_m - \deg B^+ - 1 \tag{44}$$

The degree of R can also be found from (42) as

$$\deg \bar{R} = \deg A_o + \deg A_m - \deg A - d \tag{45}$$

Now if we assume that the poles and zeros, A and B respectively, are known and the roots of A_m and A_o are chosen, we get following polynomials:

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

where $j = \deg A_m$ and $k = \deg A_o$. Considering the degrees of the polynomials in (42) it is easily seen that it can be written as

$$q^d A(q^m R'' + R') + B^-(q^d S' + S'') = A_o A_m \tag{46}$$

if the equality in (44) holds the degrees of the polynomials are:

$$\deg R' = m-1 \qquad \deg R'' = n + d - \deg B - 1$$
$$\deg S' = n-1 \qquad \deg S'' = d-1$$

R'' and S'' can be calculated recursively by long division of (42). 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 (42) for $q = p_i$ and $q = z_i$.

$$B^-(p_i)p_i^d S'(p_i) = A_o(p_i)A_m(p_i) - B^-(p_i)S''(p_i) \tag{47}$$
$$z_i^d A(z_i)R'(z_i) = A_o(z_i)A_m(z_i) - z_i^d A(z_i) z_i^m R''(z_i) \tag{48}$$

where z_i are uncancelled process zero ($z_i \in B^-$). Solution to (47) and (48) can easily be obtained by reparameterizing R' and S' into Newton coefficients as proposed by Wittenmark and Evans [27]:

$$R'(q) = r'_0 + \sum_{i=1}^{m-1} r'_i g_i(q) \tag{49}$$

$$S'(q) = s'_0 + \sum_{i=1}^{n-1} s'_i f_i(q) \tag{50}$$

where

$$g_i(q) = \prod_{j=1}^{i}(q - z_j) \qquad f_i(q) = \prod_{j=1}^{i}(q - p_j).$$

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

$$\begin{aligned} \frac{A_o(p_1)A_m(p_1)}{p_1^d B^-(p_1)} - \frac{S''(p_1)}{p_1^d} &= s'_0 \\ \frac{A_o(p_2)A_m(p_2)}{p_2^d B^-(p_2)} - \frac{S''(p_2)}{p_2^d} &= s'_0 + s'_1 f_1(p_2) \\ &\vdots \\ \frac{A_o(p_n)A_m(p_n)}{p_n^d B^-(p_n)} - \frac{S''(p_n)}{p_n^d} &= s'_0 + s'_1 f_1(p_n) + \cdots \\ &\quad + s'_{n-1} f_{n-1}(p_n) \end{aligned} \tag{51}$$

and for R'

$$\begin{aligned} \frac{A_o(z_1)A_m(z_1)}{z_1^d A(z_1)} - z_1^m R''(z_1) &= r'_0 \\ \frac{A_o(z_2)A_m(z_2)}{z_2^d A(z_2)} - z_2^m R''(z_2) &= r'_0 + r'_1 g_1(z_2) \\ &\vdots \\ \frac{A_o(z_m)A_m(z_m)}{z_m^d A(z_m)} - z_m^m R''(z_m) &= r'_0 + r'_1 g_1(z_m) + \cdots \\ &\quad + r'_{m-1} g_{m-1}(z_m). \end{aligned} \tag{52}$$

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 (47) by the following

$$\begin{aligned} B^-(p_i)p_i^d S'(p_i) &= A_o(p_i)A_m(p_i) - B^-(p_i)S''(p_i) \\ \frac{d}{dq}\Big[B^-(q)q^d S'(q)\Big]_{q=p_i} &= \frac{d}{dq}[A_o(q)A_m(q) \\ &\quad - B^-(q)S''(q)\Big]_{q=p_i} \\ &\vdots \\ \frac{d^{m_i-1}}{dq^{m_i-1}}\Big[B^-(q)q^d S'(q)\Big]_{q=p_i} &= \\ \frac{d^{m_i-1}}{dq^{m_i-1}}\Big[A_o(q)A_m(q) &- B^-(q)S''(q)\Big]_{q=p_i} \end{aligned} \tag{53}$$

from the first equation we can get s'_{i-1} and the last one we get s'_{i-1+m_i-1}. In the case of multiple zeros we differentiate (48) instead of (47).

Complex Poles and Zeros: Complex poles and zeros do not cause any problem because (51) and (52) can be assumed to be defined on the complex plane [27].

Implementation: The polynomials in the backward shift operator are

$$\begin{aligned} \bar{R}^* &= \sum_{i=0}^{n+d-\deg B-1} r''_i q^{-i} + q^{-(n+d-\deg B)} \cdot \sum_{i=0}^{m-1} r'_i g_i(q^{-1}) q^{-m+1+i} \\ &= R''^* + q^{-(n+d-\deg B)} R'^* \end{aligned} \tag{54}$$

$$\begin{aligned} S^* &= \sum_{i=0}^{n-1} s'_i f_i(q^{-1}) q^{-n+1+i} + q^{-n} \sum_{i=0}^{d-1} s''_i q^{-i} \\ &= S'^* + q^{-n} S''^*. \end{aligned} \tag{55}$$

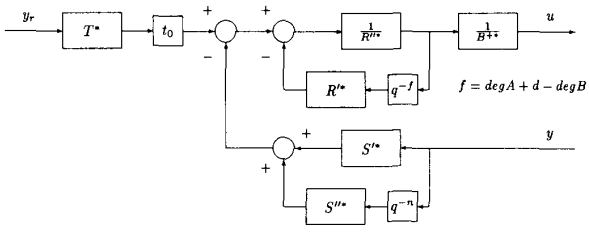


Fig. 33. Factorized controller.

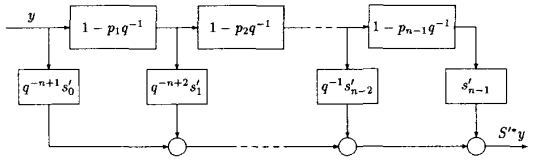


Fig. 34. Ladder.

The controller can be implemented as shown in Fig. 33, where R'^* and S'^* are implemented using a ladder network as shown in Fig. 34.

REFERENCES

[1] L. Ljung and T. Söderström, *Theory and Practice of Recursive Identification*. Cambridge MA: MIT Press, 1983.

[2] I. P. Androulakis and V. Venkatasubramanian, "A genetic algorithm framework for process design and optimization," *Computers Chem. Eng.*, vol. 15, no. 4, pp. 217-228, 1991.

[3] J. H. Holland, "Outline for a logical theory of adaptive systems," *J. ACM*, vol. 3, pp. 297-314, July 1962; also in A. W. Burks, Ed., *Essays on Cellular Automata*, Univ. Illinois Press, 1970, pp. 297-319.

[4] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Mich. Press, 1975.

[5] K. A. DeJong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation (CCS), Univ. Mich., Ann Arbor, MI, 1975.

[6] J. M. Fitzpatrick, J. J. Grefenstette, and D. Van Gucht, "Image registration by genetic search," in *Proc. IEEE Southeastcon '84*, 1984, pp. 460-464.

[7] D. E. Goldberg and R. Lingle, Jr., "Alleles, loci and the traveling salesman problem," in *Proc. Int. Conf. Genetic Algorithms and Their Appl.*, 1985, pp. 154-159.

[8] J. J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," in *Proc. Int. Conf. Genetic Algorithms and Their Applications*, 1985, pp. 160-165.

[9] R. Das and D. E. Goldberg, "Discrete-time parameter estimation with genetic algorithms," preprints from the *Proc. 19th annual Pittsburgh Conf. Modeling and Simulation*, 1988.

[10] D. M. Etter, M. J. Hicks, and K. H. Cho, "Recursive adaptive filter design using an adaptive genetic algorithm," *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, vol. 2, pp. 635-638, 1982.

[11] D. E. Goldberg, *System identification via genetic algorithm*, unpublished manuscript, Univ. Mich., Ann Arbor, MI, 1981.

[12] K. Kristinsson and G. A. Dumont "Genetic algorithms in system identification," *Third IEEE Int. Symp. Intelligent Contr.*, Arlington, VA, 1988, pp. 597–602.

[13] T. Smith and K. A. DeJong, "Genetic Algorithms applied to the calibration of information driven models of US migration patterns," in *Proc. 12th Annu. Pittsburgh Conf. Modeling and Simulation*, 1981, pp. 955-959.

[14] J. J. Grefenstette, Ed., *Proc. Int. Conf. Genetic Algorithms and their Applications*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1985.

[15] ——, *Genetic Algorithms and Their Applications: Proc. Second Int. Conf. Genetic Algorithms*. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[16] D. J. Scheffer, Ed., *Proc. 3rd Int. Conf. Genetic Algorithms*. Palo Alto, CA: Morgan Kaufmann, 1989.

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

[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[19] A. D. Bethke, "Genetic algorithms as function optimizers," Ph.D. dissertation (CCS), Univ. Mich., Ann Arbor, MI, 1980.

[20] D. E. Goldberg, "Simple genetic algorithms and the minimal deceptive problem," *Genetic algorithms and simulated annealing*. London: Pitman, 1987, pp. 74-88.

[21] J. E. Baker, "Adaptive selection methods for genetic algorithms," *Proc. Int. Conf. Genetic Algorithms and Their Applications*, 1985, pp. 101-111.

[22] K. Kristinsson, "Genetic algorithms in system identification and control," M.A.Sc. thesis in electrical engineering, Univ. Brit. Columbia, Aug. 1989.

[23] T. Söderström, L. Ljung, and I. Gustavsson, "A comparative study of recursive identification methods," Rep. 7427, Lund Inst. Technol., Lund, Sweden, 1974.

[24] K. J. Åström and B. Wittenmark, *Computer Controlled Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[25] R. Johansson, *Identification of Continuous Time Dynamic Systems*, Tech. Rep., Dept. Automat. Contr., Lund Inst. Technol., Lund, Sweden, 1986.

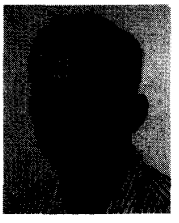
[26] C. Canudas, K. J. Åström, and K. Braun, "Adaptive compensation in DC-motor drives," *IEEE J. Robotics Automat.*, vol. RA-3, no. 6, pp. 680-685, Dec. 1987.

[27] B. Wittenmark and B. J. Evans, "An adaptive pole placement controller based on pole-zero parameterization," preprint from the *8th IFAC/IFORS Symp. Identification and System Parameter Estimation*, Beijing, China, 1988, pp. 98-103.

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

[29] P. Anderson, "Adaptive forgetting in recursive identification through multiple models," *Int. J. Contr.*, vol. 42, no. 5, pp. 1175-1193, 1985.

[30] F. Wan, N. Sepehri, K. Kristinsson, G. A. Dumont, and P. D. Lawrence, "On identification of hydraulic compliance using a genetic algorithm," in *Proc. 13th Canadian Congress Appl. Mecahnics CANCAM'91*, Winnipeg, MN, Canada, June 2–6, 1991.



Kristinn Kristinsson (S'90–M'92) was born in Hafnarfjordur, Iceland, in 1964. He received the B.Sc. degree in electrical engineering (Verkfraedingaprof) in 1986 from University of Iceland, Reykjavik, Iceland, and the M.A.Sc. degree in electrical engineering in 1989 from University of British Columbia, Vancouver, BC, Canada, where he is currently working towards the Ph.D. degree at the Department of Electrical Engineering in the Field of process control.

His research interests include adaptive process control, computer simulation and modelling and applications of modern control theory to industrial processes, particularly pulp and paper processes. During the summer of 1988 he held a Summer University Graduate Fellowship at the University of British Columbia and currently holds the Paprican F. L. Mitchell Memorial Fellowship.

Guy A. Dumont (M'78–SM'84) was born in Calais, France in 1951. He received a diplome d'Ingenieur from the Ecole Nationale Superieure d'Arts et Metiers, Paris, in 1973 and the Ph.D. degree in electrical engineering in 1977 from McGill University, Montreal, PQ, Canada.

He worked for Tioxide S.A., Calais, France, from 1977 to 1979, where he implemented a self-tuning regulator on a TiO_2 rotary kiln. From July 1979 to July 1989, he was with PAPRICAN, where he headed the Control Engineering Section, first in Montreal, then in Vancouver. In 1989 he joined the Department of Electrical Engineering at UBC to occupy the PAPRICAN/NSERC Chair of Industrial Process Control. He is involved in applications of advanced control theory to pulp and paper process and his current interests are in stochastic and adaptive control, process modelling and identification and intelligent control. In 1979, he was corecipient (with Dr. P.R. Belanger) of the IEEE T-AC Honourable Paper Award.