GENETIC ADAPTIVE PARAMETER ESTIMATION *

JAMES R. GREMLING and KEVIN M. PASSINO †
Department of Electrical Engineering
The Ohio State University
2015 Neil Avenue
Columbus, Ohio 43210

Abstract

A genetic algorithm (GA) uses the principles of evolution, natural selection, and genetics to offer a method for parallel search of complex spaces. In this paper we develop a GA that can perform on-line adaptive parameter estimation for linear and nonlinear systems. First, we show how to construct a genetic adaptive parameter estimator where a GA evolves a best set of parameter estimates in real time for a model with a known structure. Next, we use several examples to illustrate the operation and performance of the genetic adaptive parameter estimator. We begin by showing how its performance compares to that of the conventional recursive least squares technique for two linear system examples. Next, we compare the GA to recursive least squares for a nonlinear ball on a beam system model. Finally, we examine the use of the GA for estimating vehicle parameters in an automated highway system (AHS) application.

Keywords: Adaptive estimation, genetic algorithms, failure detection and identification.

---

*This work was supported in part by National Science Foundation Grant IRI 9210332.
Bibliographic information for this paper: Gremling J.R., Passino K.M., "Genetic Adaptive Parameter Estimation," Int. Journal of Intelligent Control and Systems, Vol. 3, No. 4, pp. 465–503, 1999.

†Please address all correspondence to K. Passino; (614) 292-5716, email: passino@ee.eng.ohio-state.edu.

1 Introduction

The genetic algorithm (GA) has been used in a large number of applications, in areas ranging from economics and game theory to control system design. It is a stochastic process which attempts to find an optimal solution for a problem by using techniques that are based on Mendel's genetic inheritance concept and Darwin's theory of evolution and *survival of the fittest*. GAs have been used for parameter estimation and system identification [1, 2].

In this paper, we provide a new set of examples of parameter estimation and a comparative analysis with conventional methods. The goal of the first two examples is to show that the GA can perform adaptive parameter estimation in cases that are suitable for the recursive least squares method. This would support the use of the GA as an alternative method in cases where recursive least squares may be difficult to tune. In addition, we show that the GA can be used in cases where conventional methods may fail (nonlinear plants, for example). In particular, we use the GA to estimate parameters for a nonlinear ball on a beam system. In this example, the GA estimated model is compared to an approximate linear model that is estimated by recursive least squares. In a final example, we examine the ability of the GA to estimate specific parameters for a nonlinear automated highway system (AHS) application. Since the specific AHS parameters (rather than an overall estimated model) are of interest, recursive least squares is not used.

2 Relevant Background: A Base-10 Genetic Algorithm

In order for the GA to find the optimal solution to a particular problem, the parameters that comprise a solution must be encoded into a form upon which the GA can operate. To borrow a term from biology and genetics, any set of parameters which may be a solution to the given problem is called a *chromosome*, and the individual parameters in that possible solution are called *traits*. Since the GA will most likely be implemented on a digital computer, each trait must be encoded with a finite number of digits (called *genes*). The more genes in a given trait (or in a chromosome), the longer the GA will take for encoding and decoding purposes and in other operations, so a reasonable length should be chosen. The entire set of chromosomes (i.e. the entire set of candidate solutions to the given problem) upon which the GA will operate is called a *population*.

Here, it is important to discuss the assignment of the individual genes.

A particular gene can take any one of a given number of values (called *alleles*). The GA described in [3] is a base-2 GA, in which all traits are encoded as binary numbers and all genes may take a value of 0 or 1. For this study, however, a base-10 GA is used, in which each gene can take any value from 0 to 9 (an extra gene representing sign $\pm$ may be required for each trait). The base-10 approach was chosen for this study because it simplified the encoding/decoding procedure, and it provided for easy and intuitive monitoring of the dynamics of the operation of the GA. Simplification of the encode/decode procedure is especially important here since we are using the GA in a real-time system where encoding and decoding must occur within the sampling interval. Note that none of the techniques presented after this point require the use of a base-10 GA—a base-2 GA could just as easily be used. In the discussion th at follows, however, any mention of the GA implies a base-10 approach.

To evolve the best solution candidate (or chromosome), the GA employs the *genetic operators* of *selection, crossover,* and *mutation* for manipulating the chromosomes in a population. A brief description of these operators follows, and a more detailed description can be found in [3, 4, 5]. The GA uses these operators to combine the chromosomes of the population in different arrangements, seeking a chromosome that maximizes some user defined objective function (called the *fitness function*). This combination of the chromosomes results in a new population (i.e., the next *generation*). The GA operates repetitively, with the idea that, on average, the members of the population defining the current generation should be as good (or better) at maximizing the fitness function than those of the previous generation. The most fit member of the current generation (i.e., the member with the highest fitness function result, or "fitness value") at the time the GA terminates is often taken to be the solution of the GA optimization problem.

The first genetic operator used by the GA for creating a new generation is *selection*. To create two new chromosomes (or *children*) two chromosomes must be selected from the current generation as *parents*. As is seen in nature, those members of the population most likely to have a chance at *reproducing* are the members that are the most fit. The technique used in [3] for selection uses a "roulette wheel" approach. Consider a roulette wheel that is partitioned according to the fitness of each chromosome. The more fit chromosomes occupy a greater portion of the wheel and are more likely to be selected for reproduction. In [1], a selection method is chosen so that a given segment of the population corresponding to the most fit members (i.e. the D most fit members) are automatically selected for reproducing. Therefore, the least fit members have *no* chance of contributing any genetic

material to the next generation. Of the D most fit members of the curre nt population, parents are randomly chosen, with equal probability. The latter method of selection is used in this study.

Once two parents are selected, the *crossover* operation is used. Crossover mimics natural genetics (i.e. "inheritance") in that it causes the exchange of genetic material between the parent chromosomes, resulting in two new chromosomes. Given the two parent chromosomes, crossover occurs with a user defined probability p_c. According to [3], if crossover occurs, a randomly chosen "cross site" is determined. All genes from the cross site to the end of the chromosome are switched between the parent chromosomes, and the children are created. Another approach to crossover, one that is used in [1] and in this study, is that crossover occurs exactly once (i.e. $p_c = 1$) for every *trait*, with the cross site within that trait chosen randomly. That is, all genes between the cross site and the end of the trait are exchanged between the parent chromosomes. Crossover helps to seek for other solutions near solutions that appear to be good.

After the children have been created, each child is subjected to the *mutation* operator. Mutation occurs on a gene by gene basis, each gene mutating with probability p_m. If mutation does occur, the gene that is to mutate will be replaced by a randomly chosen allele (in this case, a randomly chosen value between 0 and 9). The mutation operator helps the GA avoid a local solution to the optimization problem. If all of the members of a population should happen to converge to some local optimum, the mutation operator allows the possibility that a chromosome could be pulled away from that local optimum, improving the chances of finding the global optimum. However, since a high mutation rate results in a random walk through the GA search space, p_m should be chosen to be somewhat small. We have found, however, that in some instances in real-time systems, we need a slightly higher mutation rate. This is the case since the fitness function depends on the dynamically changing state of a system, so the locality of an optimum is time dependent and we must ensure that the GA is readily capable of exploring new opportunities for maximizing the time-varying fitness functions.

If a chromosome is generated by crossover and mutation, it is possible that one or more of its traits will lay outside of the allowable range(s). If this occurs, each trait that is out of range should be replaced with a randomly selected trait that does fall within the allowable range.

In addition to selection, crossover, and mutation, a fourth operator can be used by the GA. This operator, known as *elitism*, causes the single most fit chromosome of a population to survive, undisturbed, in the next generation. The motivation behind elitism is that after some sufficiently small

amount of time, a candidate solution may be found to be close to the optimal solution. To allow manipulation of this candidate solution would risk unsatisfactory performance by the GA. Therefore, with elitism, the fitness of a population (seen as the fitness of the best member of the population) *should* be a nondecreasing function from one generation to the next. If elitism is selected, the most fit member of the current generation is automatically chosen to be a member of the next generation. The remaining members are generated by selection, crossover, and mutation. Notice, also, that this frees us to raise the mutation probabilty since we know that we have a good solution available. In [1] , as well as in this study, elitism can involve more than just one member. That is, a certain number $\rho \cdot D$ (possibly more than one) of the most fit members will survive in the next generation without manipulation by crossover or mutation. If the most fit member would point to a local optimum in the GA search space, but a slightly less fit member points to the global optimum, they might both survive in the next generation with this new form of elitism.

To initialize the GA, a chromosome length must be chosen, along with the length and position of each trait on the chromosome. The allowable range for each trait must also be specified. The population size (denoted N) must be specified, along with the method of generating the first population. The individual members may be randomly generated, or they may be initialized to some set of "best guesses." In this study, a randomly generated initial population is always used. In addition, p_c and p_m must be specified. After this initialization, the GA can operate freely to solve its optimization problem.

3 Genetic Adaptive Parameter Estimation

Consider a general system

$$\dot{x} = G(x, u; \theta) \tag{1}$$

where x is a vector describing the state of the system, u is the system input, θ is a vector containing the parameters that describe the system, and G is a function that relates x, u, and θ and defines the operation of the system.

Since the GA is a technique often implemented on a digital computer, it generally operates in discrete time. A more appropriate definition of the general system is then written

$$x(k+1) = F\Big(x(k), u(k); \theta\Big) \tag{2}$$

if θ is a time-invariant vector, which will be the case in the applications studied in this paper. Note that the function F in Equation 2 is not the same as G in Equation 1. The GA-based parameter estimator assumes that the structure of the function F in Equation 2 is known and operates on the equation

$$\hat{x}(k+1) = F\Big(\hat{x}(k), u(k); \hat{\theta}(k)\Big) \tag{3}$$

where $\hat{\theta}(k)$ is the estimate of the true system parameter vector θ at time k, and $\hat{x}(k)$ is the state of this estimated system.

The way in which the GA operates on Equation 3 is as follows. Consider the block diagram shown in Figure 1. Each parameter estimate in the

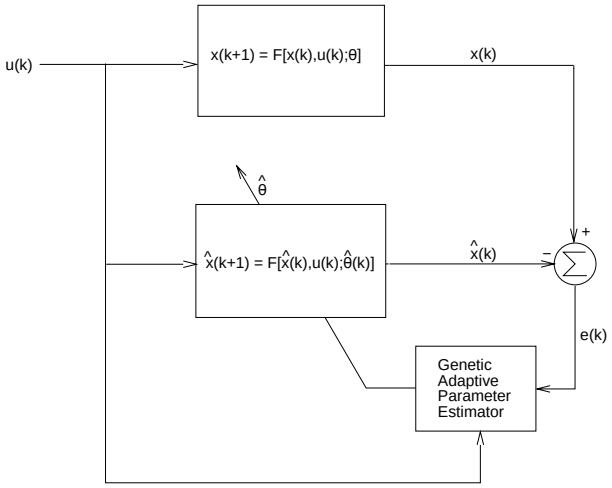


Figure 1: Block diagram for a general GA-based parameter estimator.

vector $\hat{\theta}(k)$ is encoded by the GA as a *trait* on a chromosome. Therefore, each chromosome completely represents a set of parameter estimates $\hat{\theta}(k)$ and hence it completely represents an estimated system (since the structure of F is assumed to be known). Since the GA works with a *population* of chromosomes, the block in Figure 1 representing the estimated system can be thought of as a population of candidate systems. The GA employs the selection, crossover, mutation, and elitism operators on this population of candidate systems to evolve a system that best represents the true system. At any time k, the current set of parameter estimates $\hat{\theta}(k)$ will be provided by the *most fit* chromosome in the population at time k (i.e. the chromosome with the highest *fitness value* at time k).

In this case, the fitness function for the GA is chosen to *minimize* the

squared error between the state $x(k)$ of the actual system and that of the estimated system $\hat{x}(k)$ over a window of the $W+1$ most recent data points. However, since the *maximally* fit member is sought, the fitness function takes the form

$$J = \alpha - \frac{1}{2}E^T E \tag{4}$$

where

$$E = \begin{bmatrix} e_{k-W} \\ e_{k-W+1} \\ \vdots \\ e_k \end{bmatrix}, \tag{5}$$

with $e_k = x(k) - \hat{x}(k)$ for $k > 0$ and $e_k = 0$ for $k \leq 0$. To guarantee that each member has a positive fitness value, α is selected to be the highest $\frac{1}{2}E^T E$ of any member of the population at time k (i.e., the $\frac{1}{2}E^T E$ of the worst member of the population).

The GA uses the current population of parameter estimates, along with the past and present values of the input $u(k)$, to generate past and present estimated system states $\hat{x}(k)$. Note that the value of $\hat{x}(k-W)$ (representing the beginning of the current window of data) is set equal to $x(k-W)$. A window of $\hat{x}$ values (i.e. $\hat{x}(k-W)$, $\hat{x}(k-W+1)$, $\ldots$, $\hat{x}(k)$) is generated for each candidate set of parameter estimates, and hence each candidate is assigned a fitness value according to Equations 4 and 5. Using these fitness values, the GA is able to select which candidates will be used as parents for generating the next population (or *generation*) of candidate sets of parameter estimates.

3.1 Computational Issues

In order to assess our ability to implement an algorithm (such as the genetic adaptive state estimator) in real time, computational complexity must be examined. For any given generation, let n_a represent the number of add operations that must take place. Also, let n_m, r_i, and r_f, represent the numbers of multiply operations, random integer generations, and random floating point number generations that must take place, respectively. The values for n_a and n_m are of primary interest in determining the complexity of the fitness calculation procedure, while r_i and r_f are of primary interest in terms of generating the next population of chromosomes.

First, we focus on fitness calculation. When looking at the $W+1$ most recent data points, each population member must undergo $W+1$ add operations, which will correspond to a calculation of the error between the actual system output and the estimated system output for each data point. Then, $W+1$ mulitply operations will be carried out, as the *squared* error is of sole interest here. All of the squared errors over the window will be added together, resulting in another W add operations. This leads to a total of $2W+1$ add operations and $W+1$ multiply operations for each member (in a population of size N) to generate E^TE. (Therefore, $W+2$ multiply operations are necessary for $\frac{1}{2}E^TE$ calculation for each member.)

Each member is examined, and after α is determined, N add operations are implemented to perform the remainder of the $J = \alpha - \frac{1}{2}E^TE$ calculations for the population. In all,

$$\begin{aligned} n_a &= 2N(W+1) \\ n_m &= N(W+2) \end{aligned} \tag{6}$$

are the numbers of add and multiply operations, respectively, that must take place for a population at any given generation. Depending on the encoding and decoding procedures used for chromosome manipulation, these numbers require some adjustment.

In order to determine how many random number generations must take place, we must look at the remainder of the operation of the GA (i.e., selection, crossover, and mutation). Since every two children are generated by two randomly chosen parents (from the pool of the best D population members), each child can be thought of as the result of one random selection from the pool of D possible parents, on average. Recalling that $\rho \cdot D$ members survive in the next population by elitism, $N - \rho \cdot D$ random integer generations then take place due to parent selection. For every two children generated, n_t crossover operations will occur, where n_t is the number of traits on a chromosome. This means that $\frac{n_t}{2}(N - \rho \cdot D)$ random integer generations will occur due to crossover operations. Finally, the number of mutation operations taking place in a given population can range from zero to $n_g(N - \rho \cdot D)$, with an average value of $p_m \cdot n_g(N - \rho \cdot D)$, where n_g is the number of genes per chromosome. Therefore the number of random integer generations r_i needed to produce a new population lay in a range

$$(\frac{n_t}{2} + 1)(N - \rho \cdot D) \leq r_i \leq (\frac{n_t}{2} + n_g + 1)(N - \rho \cdot D) \ , \tag{7}$$

with an average value

$$\bar{r}_i = (\frac{n_t}{2} + p_m \cdot n_g + 1)(N - \rho \cdot D) \ . \tag{8}$$

Since it is possible that mutation or crossover will lead to traits that lay outside the allowable ranges, the GA may be required to replace faulty traits with new, randomly generated traits. This will result in the generation of random floating point numbers. The number of random floating point number generations r_f will lay in a range

$$0 \leq r_f \leq n_t(N - \rho \cdot D) \ . \tag{9}$$

To examine the operation of a genetic adaptive parameter estimator consider the following examples.

4 A Linear Example

Consider a linear system with the following discrete time transfer function

$$H(z) = \frac{Y(z)}{U(z)} = \frac{K}{z^3 - az^2 - bz - c} \ , \tag{10}$$

which has the following difference equation

$$y(k) = Ku(k-3) + ay(k-1) + by(k-2) + cy(k-3) \ . \tag{11}$$

The estimation of K, a, b, and c will be studied in each of the following five cases.

Case 1: $K = 256.3$, $a = 0.55$, $b = 0.43$, and $c = -0.0475$. This plant has poles at -0.5 (somewhat fast), 0.1 (fast), and 0.95 (somewhat slow).

Case 2: $K = 374.8$, $a = 1.14$, $b = 0.4416$, and $c = -0.59248$. Two of the poles in this case are found at 0.92, and the other pole is at -0.7.

Case 3: $K = 875.1$, $a = 2.94$, $b = -2.8812$, and $c = 0.941192$. All three of this plant's poles are found at 0.98.

Case 4: $K = 12.6$, $a = -0.24$, $b = -0.0192$, and $c = -0.000512$. All three of this plant's poles are found at -0.08.

Case 5: $K = 172.3$, $a = 1.92$, $b = -1.2288$, and $c = 0.262144$. All three of this plant's poles are found at 0.64.

Parameter estimation is performed with noise as the training signal. In particular, at each sample, the training signal can take a random value between -1.0 and 1.0 (with a uniform distribution). Parameter estimation is performed using both recursive least squares and GA methods, and comparisons between the two are provided.

4.1 Recursive Least Squares

The recursive least squares approach begins with rewriting the difference equation in the form

$$y_k = \phi_k^T \theta \,, \tag{12}$$

where

$$\begin{aligned} y_k &= y(k) \,, \\ \phi_k &= \begin{bmatrix} u(k-3) \\ y(k-1) \\ y(k-2) \\ y(k-3) \end{bmatrix} \,, \end{aligned} \tag{13}$$

and

$$\theta = \begin{bmatrix} K \\ a \\ b \\ c \end{bmatrix} \,.$$

Since $\hat{\theta}$ is updated with the acquisition of each new training data point (i.e., at each time step), it actually should take the form $\hat{\theta}(k)$ where

$$\hat{\theta}(k) = \begin{bmatrix} \hat{K}(k) \\ \hat{a}(k) \\ \hat{b}(k) \\ \hat{c}(k) \end{bmatrix} \tag{14}$$

which is the current set of parameter estimates.

At each time step, $y(k)$ is used with ϕ_k to generate $\hat{\theta}(k)$ according to the update equation

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P(k)\phi_k \Big(y_k - \phi_k^T \hat{\theta}(k-1) \Big) \,, \tag{15}$$

where

$$P(k) = \frac{1}{\lambda_f} \left(P(k-1) - P(k-1)\phi_k \Big(\lambda_f I + \phi_k^T P(k-1) \phi_k \Big)^{-1} \phi_k^T P(k-1) \right) \,. \tag{16}$$

Here, λ_f is a "forgetting factor" that gives higher weight to more recent data points ($0 < \lambda_f \leq 1$). That is, the most recent data point will be

weighted with λ_f, while the $\bar{n}^{th}$ most recent data point will be weighted by $\lambda_f^{\bar{n}}$. Initially, $\lambda_f = 1$ is chosen to give all data equal weight.

To initialize the recursive least squares algorithm, $\hat{\theta}(0)$ and $P(0)$ are needed. Since a positive definite matrix must be chosen for $P(0)$, an acceptable choice is $\tilde{\alpha}I$ where $\tilde{\alpha} \gg 0$. For recursive least squares estimation of the five linear plants defined above, $\tilde{\alpha} = 25000.0$ has been chosen. All of the elements of $\hat{\theta}(0)$ were initialized to be zero. The results of the estimation of these plants can be seen in Figures 2 through 7, where the recursive least squares (RLS) algorithm is compared to the GA based approach which will be described below. In all cases except Case 4, the recursive least squares estimates were quite quick to converge. In theory, the estimates of Case 4 should converge, although this convergence was shown to be quite slow in simulation. By adjusting λ_f from 1 to 0.95, the parameter convergence for Case 4 was greatly assisted, as can be seen in Figure 6.

4.2 Genetic Algorithm

Recall the fitness function defined in Equations 4 and 5. Here, we have $e_k = y(k) - \hat{y}(k)$ for $k > 0$ and $e_k = 0$ for $k \leq 0$. The fitness function for the linear plant under discussion is then written

$$J = \alpha - \frac{1}{2} \sum_{j=k-W}^{k} \left(y(j) - \hat{K}(k)u(j-3) - \hat{a}(k)\hat{y}(j-1) - \hat{b}(k)\hat{y}(j-2) - \hat{c}(k)\hat{y}(j-3) \right)^2 , \tag{17}$$

where all $\hat{y}(j)$ in the window for time k are generated according to the equation

$$\hat{y}(j) = \hat{K}(k)u(j-3) + \hat{a}(k)\hat{y}(j-1) + \hat{b}(k)\hat{y}(j-2) + \hat{c}(k)\hat{y}(j-3) \ . \tag{18}$$

After some tuning, the following GA parameters were used for parameter estimation of the linear plant described above.
Case 1: $N = 30$, $p_m = 0.2$, $D = 20$, $\rho \cdot D = 2$, $W = 20$, and $g_t = 10$.
Case 2: $N = 50$, $p_m = 0.4$, $D = 30$, $\rho \cdot D = 4$, $W = 50$, and $g_t = 20$.
Case 3: $N = 100$, $p_m = 0.2$, $D = 60$, $\rho \cdot D = 10$, $W = 50$, and $g_t = 20$.
Case 4: $N = 20$, $p_m = 0.2$, $D = 10$, $\rho \cdot D = 2$, $W = 20$, and $g_t = 10$.
Case 5: $N = 50$, $p_m = 0.5$, $D = 30$, $\rho \cdot D = 4$, $W = 20$, and $g_t = 20$.
In all cases, the trait for $\hat{K}$ was allowed in the range $[-1000.0, 1000.0]$, and the traits for $\hat{a}$, $\hat{b}$, and $\hat{c}$ were allowed in the range $[-10.0, 10.0]$.

Originally, all of the GA parameters were chosen to be the same as those of Case 1, as a small population size, window length, and number of generations per time period would lead to a GA that is not very demanding in

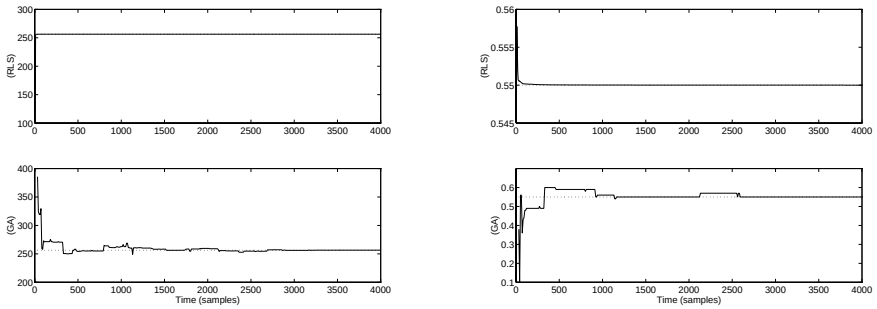
terms of computational resources. In cases where those parameter choices seemed insufficient for convergence of the GA estimates, the following procedure was used to tune the GA. First, the window length W was extended in order to allow the GA to look at more training data for evaluating the fitness of the population members. The goal of this step was a more accurate picture of the fitness of each member and an increased probability that the *best* members would be selected for mating. The window length would continue to be increased until it no longer seemed effective (i.e. until improvement in convergence for the GA estimates was no longer obvious).

After the W tuning was completed, the population size N was increased, with the idea that a larger population would allow for a greater chance that some estimates would fall closer to the actual values at GA initialization. A secondary goal of increasing N was that the random search features inherent in the GA would have a better chance of finding a path to convergence if allowed to operate more frequently, which should be the case with a larger number of members upon which to operate. Naturally, the number of members D selected for mating was increased with N, as was the number of members $\rho \cdot D$ that would survive in the next generation. As with W, N was increased until it was no longer deemed effective.

At this point, p_m was increased. By increasing the randomness of the GA's search, an increased probability of finding a path to convergence seemed likely. As a last resort, the number of generations per time period g_t was increased, as this seemed like a "brute force" attempt at convergence. Essentially, a higher g_t might have facilitated convergence in a shorter amount of time, but it would not necessarily reduce the number of generations required. In addition, increasing g_t would lead to a heavy demand on computational resources. Once it was no longer effective to tune g_t, the entire GA tuning procedure could be repeated as long as improvements were exhibited.

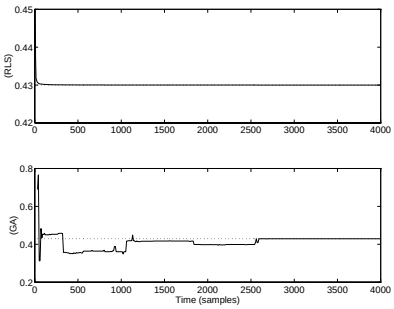
Since the genetic algorithm is a stochastic approach to optimization, there are no strict guidelines as to how to tune it. The procedure described above is an *ad hoc* approach—there may be other procedures that will work as well or better, depending on the situation in which the genetic algorithm is used. However, it seemed logical to use this procedure.

The results are shown in Figures 2 through 7 where the GA is compared to the recursive least squares algorithm. Although the performance of the GA would vary slightly from one execution to the next, the results shown represent a typical GA execution. In Case 1, the GA and RLS algorithm performed comparably. In Cases 2 and 5, the GA was successful, although outperformed by the RLS algorithm. In Case 3, the GA was significantly

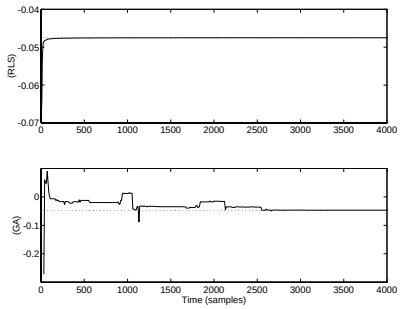


(a) Estimation of K.

(b) Estimation of a.



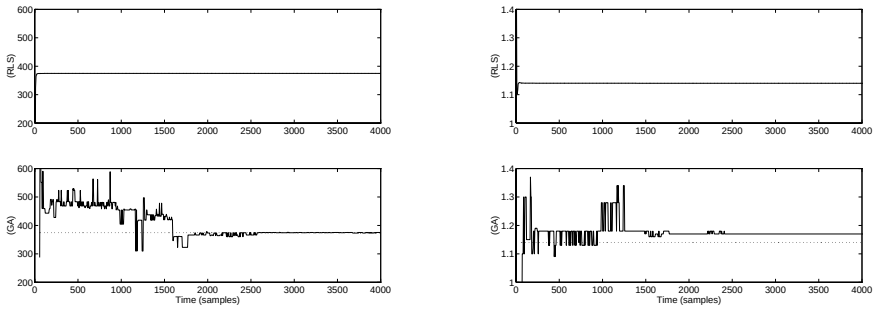
(c) Estimation of b.



(d) Estimation of c.

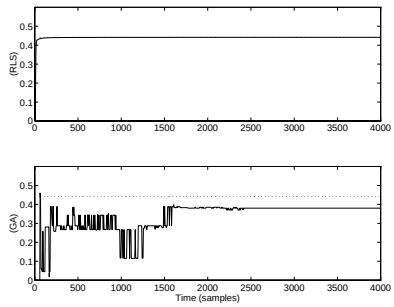
Figure 2: Comparison of the GA (bottom plot of each part) to the RLS algorithm (top plot of each part) for fixed parameter linear system, Case 1. Dotted lines indicate values of ideal parameters while solid lines represent parameter estimates.

outperformed by the RLS algorithm, as no combination of GA parameter values was found (even after some extensive tuning) that would yield acceptable results. It was expected that the RLS algorithm would perform well for parameter estimation of a linear plant. In three of the four aforementioned cases, the GA was somewhat competitive with the RLS algorithm. In Case 4, the GA converged very quickly, even after N was *reduced* from the initial value of 30. It is interesting to note that before λ_f was adjusted, the RLS algorithm required a very long time to converge for a few of the parameters. However, after λ_f was adjusted, the GA no longer outperformed the RLS algorithm. These results were not surprising. Theoretically, recursive least squares should always succeed for linear plants, but the GA seems to be able to provide a possible alternative in cases where recursive least squares is slow to converge.

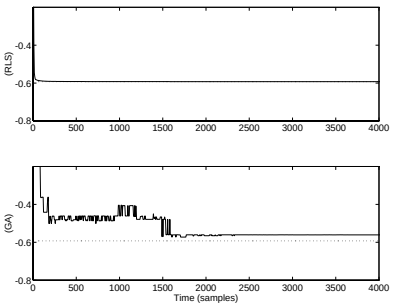


(a) Estimation of K.

(b) Estimation of a.



(c) Estimation of b.

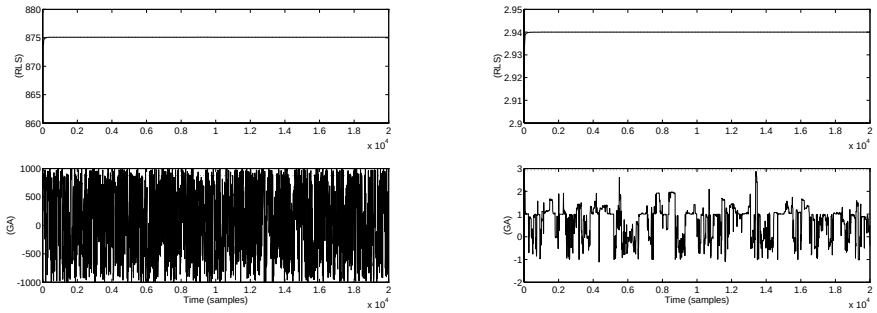


(d) Estimation of c.

Figure 3: Comparison of the GA (bottom plot of each part) to the RLS algorithm (top plot of each part) for fixed parameter linear system, Case 2. Dotted lines indicate values of ideal parameters while solid lines represent parameter estimates.

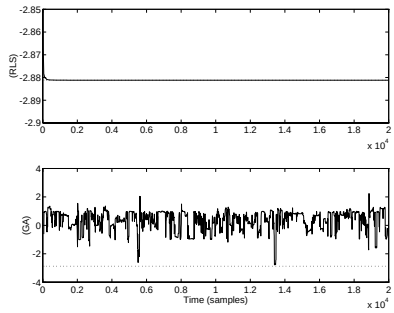
5 A Linear Example With a Varying Parameter

In Cases 1 and 2 of Section 4, both the GA and the RLS algorithm succeeded in estimating the plant parameters. It is interesting to see how each of the algorithms would perform if some parameter of the plant was not fixed at a constant value. Cases such as this do exist. Consider an automobile braking system, as in [6]. There is a noticeable difference in the response of "cold" brakes from that of "warm" or "hot" brakes. This would tend to indicate some parameter (or group of parameters) that changes as friction causes the brakes to heat up. The ability to detect changing parameters could lead to the design of controllers that would adapt to those changes. In the braking system example, the design of such an adaptive controller could lead to brake performance that feels the same to the driver, regardless of whether

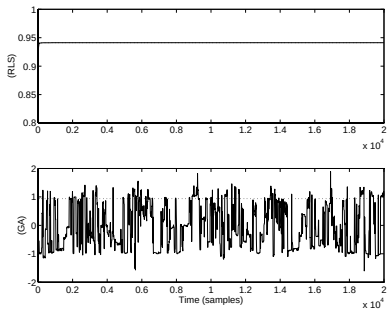


(a) Estimation of K.

(b) Estimation of a.



(c) Estimation of b.

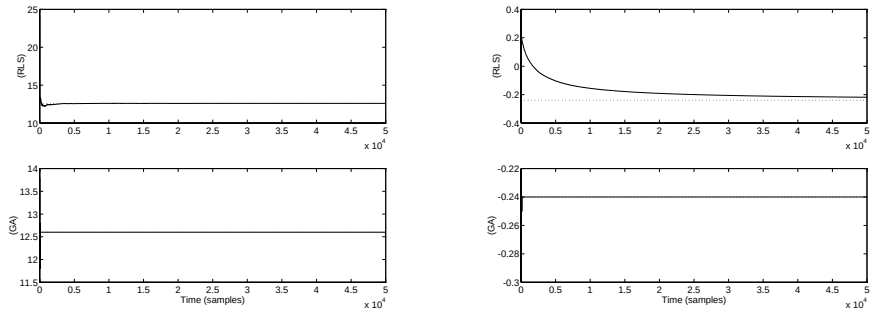


(d) Estimation of c.

Figure 4: Comparison of the GA (bottom plot of each part) to the RLS algorithm (top plot of each part) for fixed parameter linear system, Case 3. Dotted lines indicate values of ideal parameters while solid lines represent parameter estimates.

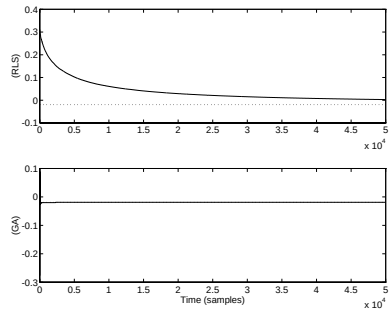
the brakes are cold or hot.

Therefore, the parameter K was chosen to vary in Cases 1 and 2 in the following manner. In Case 1, K varied from 200.0 to 300.0, and in Case 2, K varied from 300.0 to 450.0. Note that with both the RLS and the GA based estimator, there are no guarantees that the parameter estimates will converge to the ideal ones. The GA and the RLS algorithm used for estimating the parameters of these two plants were the same as those used for Cases 1 and 2 of Section 4. Since there were varying parameters to track, it made sense to use the "forgetting factor" λ_f and convert this into a "weighted" RLS approach. As a result, $\lambda_f = 0.9$ was used. (For the sake of comparison, when $\lambda_f = 1$ was used, the RLS had a great deal of difficulty tracking K. As a result, the estimates $\hat{a}$, $\hat{b}$, and $\hat{c}$ also suffered.) The results are given and compared in Figures 8 and 9. The results shown for the GA

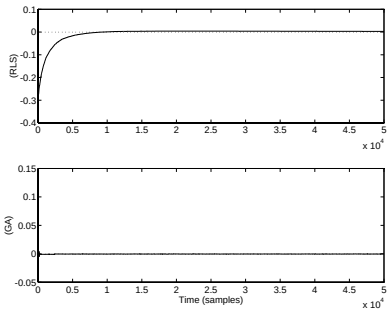


(a) Estimation of K.

(b) Estimation of a.



(c) Estimation of b.



(d) Estimation of c.

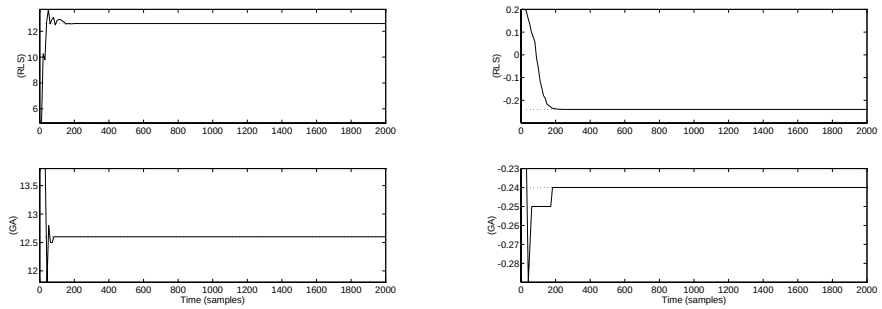
Figure 5: Comparison of the GA (bottom plot of each part) to the RLS algorithm (top plot of each part) for fixed parameter linear system, Case 4. Dotted lines indicate values of ideal parameters while solid lines represent parameter estimates.

represent a typical GA execution.

Note that the GA was able to respond to the variation of the parameter K in both cases, although the estimate $\hat{K}$ was noisy compared to that of the weighted RLS algorithm. There is still no guarantee that either algorithm will yield parameter estimates that converge to the actual parameters, but there seems to be some evidence here that the GA may be a valuable alternative method for on-line estimation when there is the possibility of parameter variations.

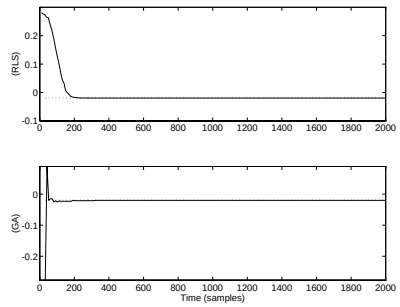
6 The Ball on a Beam System

A third example in which the GA can be compared to the recursive least squares approach is the ball-beam system. This system consists of a beam

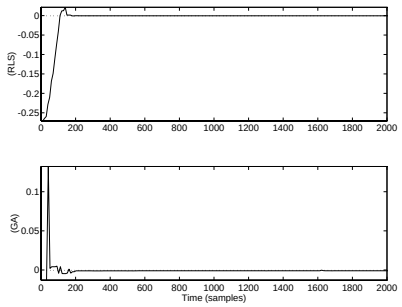


(a) Estimation of K.

(b) Estimation of a.



(c) Estimation of b.



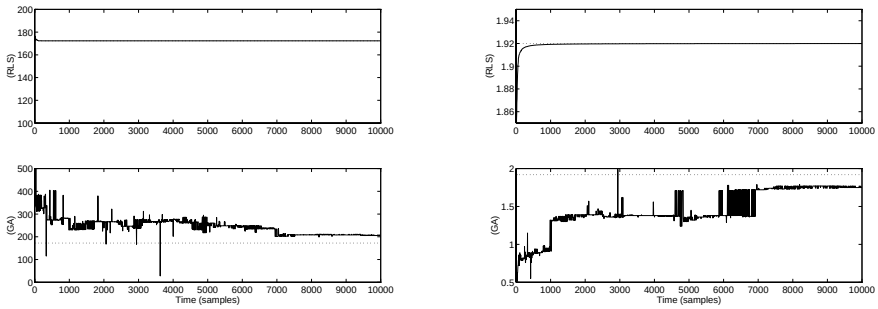
(d) Estimation of c.

Figure 6: Comparison of the GA (bottom plot of each part) to the RLS algorithm with $\lambda_f = 0.95$ (top plot) for fixed parameter linear system, Case 4. Dotted lines indicate values of ideal parameters while solid lines represent parameter estimates.

with a groove in the center along the length of the beam. A ball is placed in this groove and a control system must vary the tilt of the beam so that the position of the ball follows a reference position. The beam is 31 units long (a unit being the distance—approximately 0.75 cm—between the photo-diodes used to determine ball position). Position 0.0 refers to the left end of the beam, while position 31.0 refers to the right end of the beam. The angle of the beam is measured from the horizontal, with counter-clockwise being the positive direction. The mathematical model of the system is

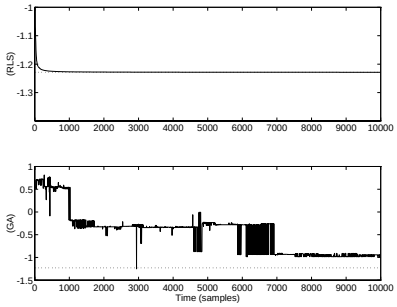
$$\dot{x}_1 \;=\; x_2 \;, \tag{19}$$

$$\dot{x}_2 \;=\; a_1 u + a_2 tan^{-1}(a_3 x_2)(e^{-a_4 x_2^2} - 1) \;.$$

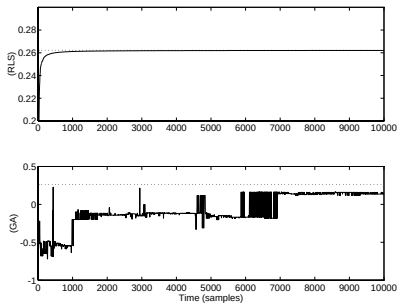


(a) Estimation of K.

(b) Estimation of a.



(c) Estimation of b.



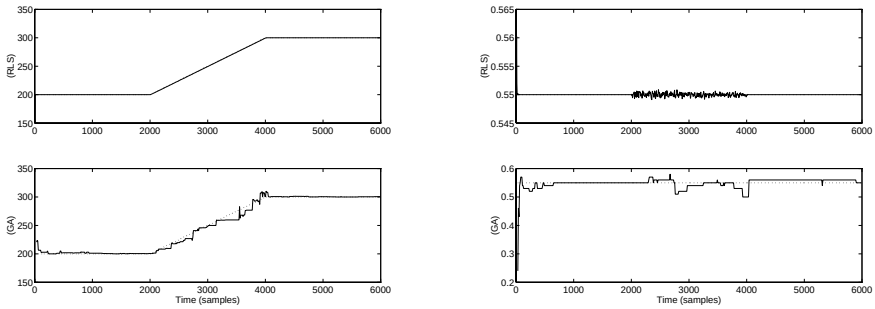
(d) Estimation of c.

Figure 7: Comparison of the GA (bottom plot of each part) to the RLS algorithm (top plot of each part) for fixed parameter linear system, Case 5. Dotted lines indicate values of ideal parameters while solid lines represent parameter estimates.

Here, the state x_1 refers to the ball position, and the input u refers to the angle of the beam. The parameters a_1, a_2, a_3, and a_4 are to be estimated in this example. The actual values are $a_1 = -514.96$, $a_2 = 9.84$, $a_3 = 100$ and $a_4 = 10^4$. These values were determined experimentally using a ball-beam set up in our laboratory.

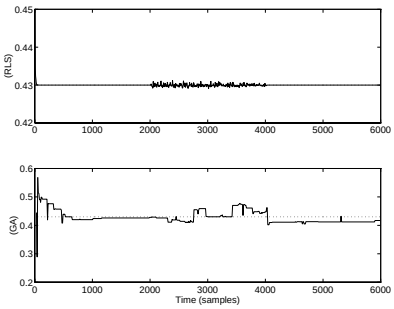
Since the least squares technique works best for linear systems, a linear system that might approximate the ball-beam model must be proposed for recursive least squares estimation. Since the RLS algorithm must estimate the parameters of the linear approximation, the actual parameters of the nonlinear ball-beam model will still be unknown (to the RLS algorithm). However, the parameters of the linear approximation may be useful enough for on-line controller design (i.e. adaptive control [7]).

As mentioned before, the GA has the advantage of not being restricted to

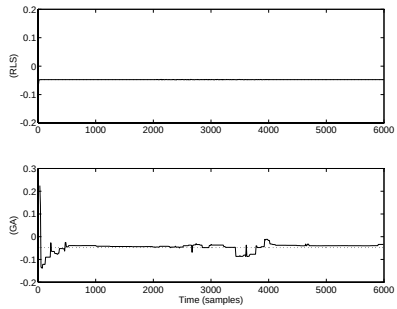


(a) Estimation of K.

(b) Estimation of a.



(c) Estimation of b.



(d) Estimation of c.

Figure 8: Comparison of the GA (bottom plot of each part) to the RLS algorithm with $\lambda_f = 0.9$ (top plot) for varying parameter linear system, Case 1. Dotted lines indicate values of ideal parameters while solid lines represent parameter estimates.

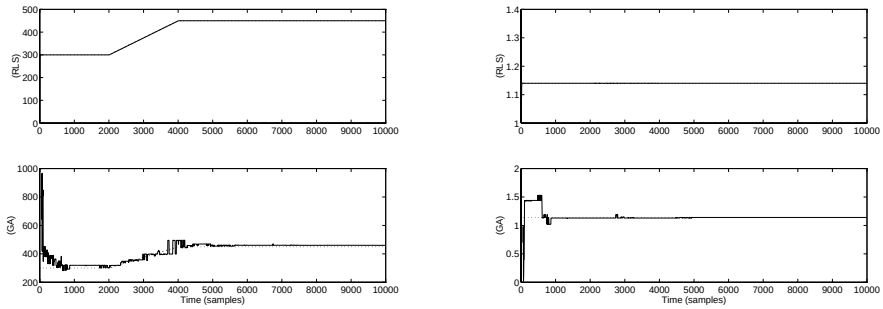
parameter estimation for linear systems. Therefore the actual parameters of the ball-beam model, a_1, a_2, a_3, and a_4, can be estimated (recall Equations 19 and 20).

Using a forward looking difference, a discrete time approximation to the ball-beam model is

$$x_1(k+1) \quad = \quad x_1(k) + Tx_2(k) \; , \tag{20}$$

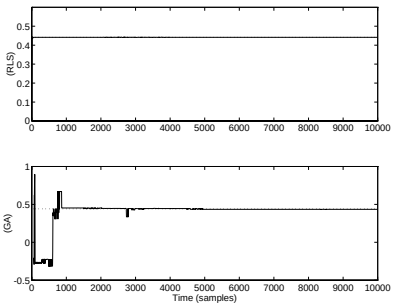
$$x_2(k+1) \quad = \quad x_2(k) + Ta_1u(k) + Ta_2tan^{-1}(a_3x_2(k))(e^{-a_4x_2^2(k)} - 1) \; ,$$

where T is the sampling period. Here, $T = 0.01$ is chosen for all identification investigations.

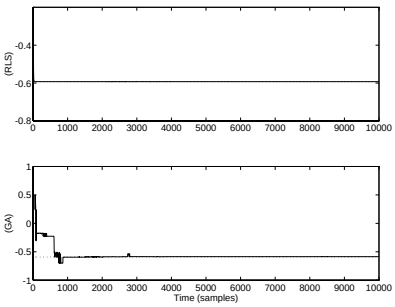


(a) Estimation of K.

(b) Estimation of a.



(c) Estimation of b.



(d) Estimation of c.

Figure 9: Comparison of the GA (bottom plot of each part) to the RLS algorithm with $\lambda_f = 0.9$ (top plot) for varying parameter linear system, Case 2. Dotted lines indicate values of ideal parameters while solid lines represent parameter estimates.

6.1 Recursive Least Squares

Since a linear approximation to the ball-beam model must be used, and since there are two states in the ball-beam model (i.e. the system is second order), consider a second order model

$$H(z) = \frac{\hat{a}z^2 + \hat{b}z + \hat{c}}{z^2 + \hat{d}z + \hat{e}} \tag{21}$$

where the parameters $\hat{a}$, $\hat{b}$, $\hat{c}$, $\hat{d}$, and $\hat{e}$ are to be determined. $H(z)$ is a transfer function from the input u (the angle) to an output (the ball position) which is denoted by y. This model can be written in terms of a difference equation

$$y(k) = \hat{a}u(k) + \hat{b}u(k-1) + \hat{c}u(k-2) + \hat{d}y(k-1) + \hat{e}y(k-2)\ . \tag{22}$$

The signs for $\hat{d}$ and $\hat{e}$ have been reversed for simplicity in writing the difference equation.

For the recursive least squares algorithm, consider

$$y_k = y(k) , \tag{23}$$

$$\phi_k = \begin{bmatrix} u(k) \\ u(k-1) \\ u(k-2) \\ y(k-1) \\ y(k-2) \end{bmatrix} ,$$

and

$$\hat{\theta} = \begin{bmatrix} \hat{a} \\ \hat{b} \\ \hat{c} \\ \hat{d} \\ \hat{e} \end{bmatrix} .$$

For this system, $\tilde{\alpha} = 25000.0$ was used to initialize the $P(0)$ matrix in the RLS algorithm, and all parameters in $\hat{\theta}(0)$ were initialized to zero. The value of λ_f was chosen to be 1, corresponding to a "standard" RLS algorithm. For the sake of comparison, a weighted RLS algorithm with $\lambda_f = 0.9$ was also used, but the results were much more noisy than those of the standard RLS algorithm, indicating that it was not very useful to adjust λ_f away from 1 in this case. The training input was a noise signal with a range of ± 6.0 degrees ($\pm\frac{\pi}{30}$ radians), and the sampling period T was chosen to be 0.01 seconds. Every 0.1 seconds, a new random value from that range was used as the angle input to the ball-beam system. The initial ball position was 15.0 units.

An hour of ball-beam operation was simulated (see Figure 10), and the following estimates were obtained: $\hat{a} \approx 0$, $\hat{b} = 0.00003$, $\hat{c} = -0.0515$, $\hat{d} = 1.7612$, and $\hat{e} = -0.7612$. The validity of these estimates was tested as the approximate linear system was compared to the nonlinear system with GA estimated parameters in Figure 13.

For the purpose of verification, four test input signals were used. The first was a sinusoid with a frequency of 0.15 Hz, an amplitude of $\frac{\pi}{60}$, and a phase of 180 degrees. The second test signal was a sinusoid with a frequency of 0.5 Hz, an amplitude of $\frac{\pi}{40}$, and a phase of 180. The third test signal was a square wave with a period of 0.24 seconds and an amplitude of $\frac{\pi}{35}$. The

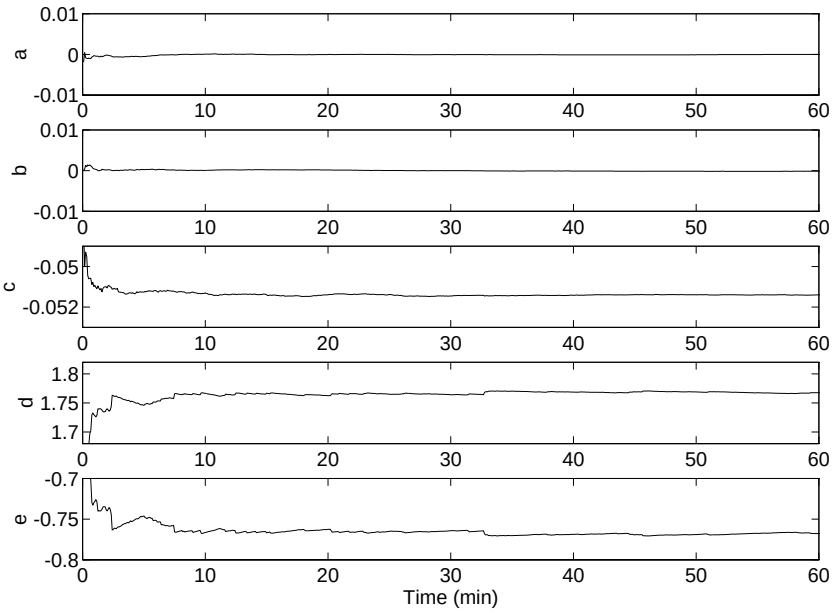


Figure 10: On-line RLS estimation of the parameters for the approximate ball-beam model.

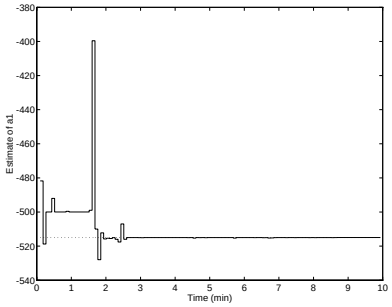
first half-cycle of this signal (which is negative) was shortened from 0.12 seconds to 0.094 seconds. The final test signal was another square wave with a period of 0.9 seconds and an amplitude of $\frac{\pi}{90}$. The first half-cycle of this signal (which is also negative) was shortened from 0.45 seconds to 0.425 seconds. Note that the RLS estimated model behaved rather poorly for all test signals.

6.2 Genetic Algorithm

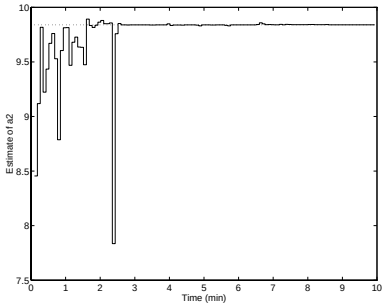
An advantage for the GA in this case is that a nonlinear model structure can be used, and its parameters can be tuned (i.e. no linear approximations need to be made). Assuming the structure of the ball-beam model is known to be the structure in Equations 19 and 20, the parameters a_1, a_2, a_3, and a_4 can be identified.

The fitness function of Equations 4 and 5 can be used here, again with $e_k = y(k) - \hat{y}(k)$ for $k > 0$ and $e_k = 0$ for $k \leq 0$. The value of y_k is the current ball position of the actual ball-beam system, and $\hat{y}_k$ is the current ball

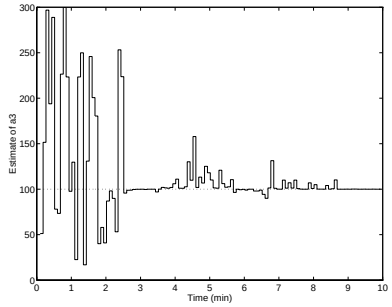
position of the identified model (using $\hat{a}_1$, $\hat{a}_2$, $\hat{a}_3$, and $\hat{a}_4$). If the structure of the model is known, it can be assumed that the physical laws used to determine this model would give some insight into the ranges of $\hat{a}_1$, $\hat{a}_2$, $\hat{a}_3$, and $\hat{a}_4$ that should be chosen for GA operation. For this reason, $\hat{a}_1$ was allowed in the range $[-2000.0,0.0]$, $\hat{a}_2$ was in the range $[0.0,30.0]$, $\hat{a}_3$ was in the range $[0.0,300.0]$, and $\hat{a}_4$ was in the range $[0.0,20000.0]$.



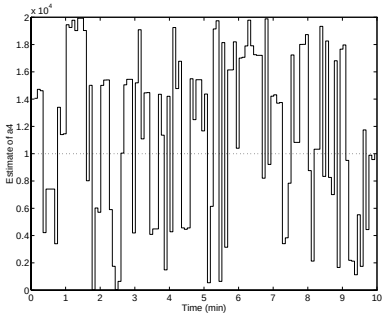
(a) Estimation of a_1.



(b) Estimation of a_2.



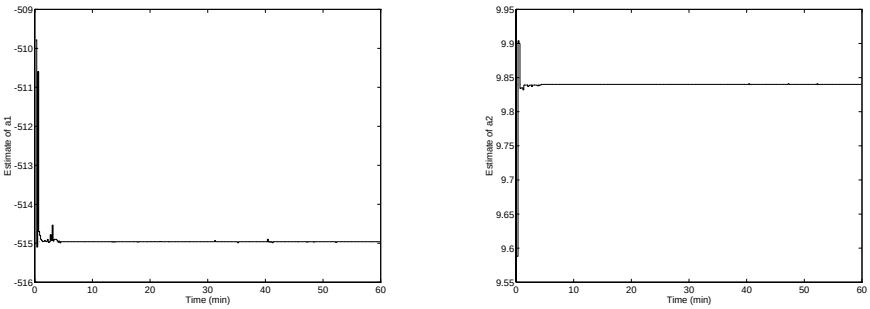
(c) Estimation of a_3.



(d) Estimation of a_4.

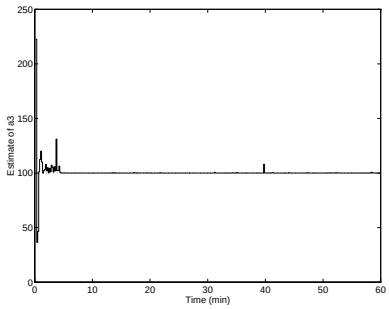
Figure 11: GA parameter estimation for the ball-beam model (ten minute execution). Dotted lines indicate values of the ideal parameters while solid lines represent parameter estimates.

The GA parameters used here were as follows: $N = 70$, $p_m = 0.2$, $D = 20$, $\rho \cdot D = 2$, $W = 10$, and $g_t = 1$. These parameters were initially chosen to be small, for computational efficiency. Then they were tuned according to the procedure described in Section 4.2. A sampling period T of 0.1 seconds was used, and the noise signal used was the same training signal used with the RLS algorithm. The parameter ranges for $\hat{a}_1$, $\hat{a}_2$, $\hat{a}_3$, and $\hat{a}_4$ were identical to those used by the GA in the off-line case. Two separate GA executions are shown in Figures 11 and 12. Figure 11 corresponds to a

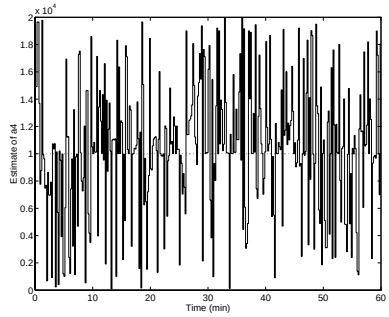


(a) Estimation of a_1.

(b) Estimation of a_2.



(c) Estimation of a_3.



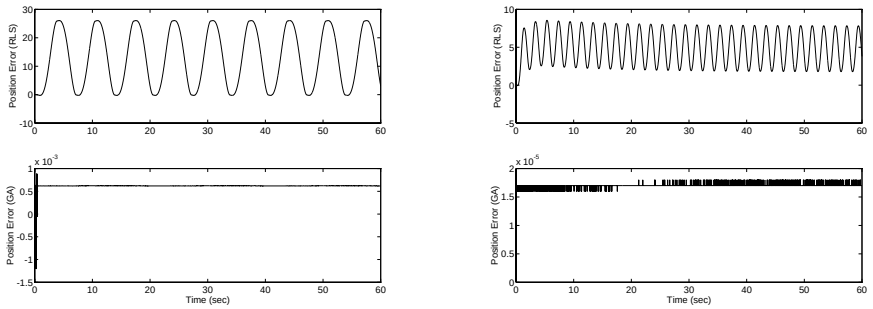
(d) Estimation of a_4.

Figure 12: GA parameter estimation for the ball-beam model (one hour execution). Dotted lines indicate values of the ideal parameters while solid lines represent parameter estimates.

GA execution over a ten minute period and Figure 12 corresponds to a GA execution over a period of one hour.

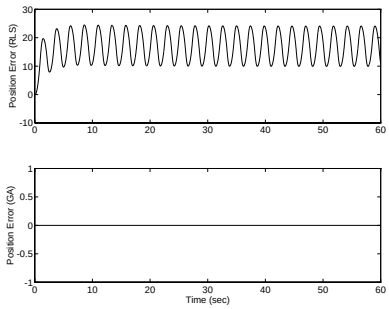
In both cases, $\hat{a}_1(k)$, $\hat{a}_2(k)$, and $\hat{a}_3(k)$ converge to the correct values of -514.96, 9.84, and 100.0, respectively. Also in both cases, the behavior of $\hat{a}_4(k)$ is quite erratic and does not seem to converge to any value. It is interesting to see that, although the GA seems unable to cause convergence for $\hat{a}_4(k)$, it is able to effect exact convergence for the other three parameters. Note that the general behavior shown in Figures 11 and 12 is typical of any GA execution, although each execution will differ slightly from another (due to the stochastic nature of the GA).

To compare the GA estimation with that of the RLS algorithm, the one-hour GA execution was used. For $\hat{a}_4(k)$, an average was taken over the entire one hour estimation and a value of 10427.32 was used. The same four test signals used for the RLS algorithm were used for the GA. The results

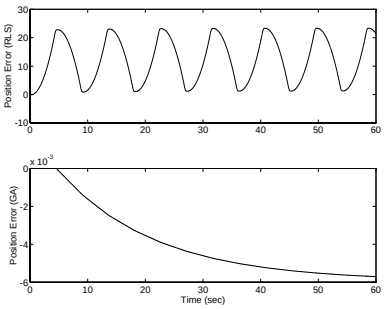


(a) Test sinusoid 1.

(b) Test sinusoid 2.



(c) Test square wave 1.



(d) Test square wave 2.

Figure 13: Testing GA and RLS parameter estimates for the ball-beam model. The position error is the error between the ball position for the actual ball-beam model and that of the estimated model, using the same test signal and initial conditions.

of the comparison are seen in Figure 13. The linear approximation with the RLS estimated parameters exhibited poor performance in all four tests, while the GA estimated system showed very good performance in all four tests. This should be expected, again noting that the GA could directly estimate parameters of the nonlinear model, while the RLS algorithm could only estimate parameters of an approximate linear model.

This provides some evidence that a GA based estimator may be a viable candidate as a parameter estimator for nonlinear systems. It is important to point out, however, that this comparison of GA to the RLS algorithm is somewhat weak in that the GA was unable to find convergence for the parameter $\hat{a}_4(k)$. Testing the system (even with an average value of $\hat{a}_4(k)$) implies that $\hat{a}_4(k)$ did converge to some value. If a different range with respect to this parameter was chosen for the GA, a different average value

may have been obtained and the results of the comparison tests may have been quite different. The real value of the GA is seen in its ability to estimate the other three parameters, indicating that the GA may provide an alternative method of parameter estimation for nonlinear systems in some (but not all) cases.

7 An Automated Highway System

To solve some of the problems caused by the current rise in traffic congestion, research is being done to develop "automated highway systems" (AHS) that are more safe (see e.g., [8, 9]). In the AHS, a "platoon" of vehicles can be driven automatically with onboard controllers. These controllers enable each vehicle to track the velocity of the vehicle directly ahead while maintaining a specific following distance.

Consider the state variables δ, v, and f, where δ_i refers to the distance between the i^{th} and $i-1^{st}$ vehicles (also called the intervehicle spacing), v_i refers to the velocity of the i^{th} vehicle, and f_i refers to the driving or braking force applied to the i^{th} vehicle. The i^{th} vehicle follows the $i-1^{st}$ vehicle. Using these state variables, the model for the i^{th} vehicle can be written

$$\begin{aligned} \dot{\delta}_i &= v_i - v_{i-1} \\ \dot{v}_i &= \frac{1}{m_i}(-A_\rho v_i^2 - d_i + f_i) \\ \dot{f}_i &= \frac{1}{\tau_i}(-f_i + u_i) \end{aligned} \tag{24}$$

where m_i is the mass of the i^{th} vehicle, A_ρ is an aerodynamic drag constant, d_i is a constant frictional force for the i^{th} vehicle, τ_i is the engine/brake time constant for the i^{th} vehicle, and u_i is the control input for the i^{th} vehicle [8]. Nominal values for the constants are $A_\rho = 0.3Ns^2/m^2$, $d_i = 100N$, and $\tau_i = 0.2s$. The controller used to specify u_i in this study is a proportional-derivative (PD) controller with $y_i = -(\delta_i + \lambda v_i)$ used as an input and a goal of driving y_i to zero. The value $\lambda = 0.9$ is used, corresponding to a desired intervehicle spacing of roughly one vehicle length per 10 mph.

Using a forward looking difference, the model for the i^{th} vehicle is then written

$$\begin{aligned} \delta_i(k) &= \delta_i(k-1) + T\Big(v_i(k-1) - v_{i-1}(k-1)\Big) \\ v_i(k) &= v_i(k-1) + \frac{T}{m_i}\Big(-A_\rho v_i^2(k-1) - d_i + f_i(k-1)\Big) \end{aligned} \tag{25}$$

$$f_i(k) = f_i(k-1) + \frac{T}{\tau_i}\Big(-f_i(k-1) + u_i(k-1)\Big) \ ,$$

with the following control law

$$\begin{aligned} y_i(k) &= -\Big(\delta_i(k) + \lambda v_i(k)\Big) \\ u_i(k) &= K_{p_i} y_i(k) + \frac{K_{d_i}}{T}\Big(y_i(k) - y_i(k-1)\Big) \ . \end{aligned} \tag{26}$$

The PD controller parameters for the i^{th} vehicle are K_{p_i} and K_{d_i}, and T is the sampling period (here $T = 0.01$ is chosen).

In this example, it would be useful if certain parameters of the model could be estimated. One such parameter is the vehicle mass m_i, as each vehicle in a platoon would have a different mass. This is, of course, due to the fact that the number of passengers in a vehicle can vary, as can the payload carried by that vehicle. Having an accurate estimate of the mass of a vehicle can be useful in tuning a controller for that vehicle, as stopping distance and required braking force depend on that mass (a higher mass would indicate a larger stopping distance and required braking force). The recursive least squares algorithm is not as useful here as the AHS is a nonlinear system. Any parameters estimated by the RLS algorithm would be parameters of an approximate linear system and may not make sense in terms of the actual parameters to be estimated (especially at highway speeds where the term $A_\rho v_i^2$ will have a significant influence).

7.1 Vehicle Mass Estimation

For this AHS study, a platoon of four vehicles is used. Actually, a fifth vehicle is involved, as the lead vehicle follows a "virtual" vehicle. That is, the controller for the lead vehicle uses its own velocity as well as a virtual intervehicle spacing for generating a control input to govern the behavior of the lead vehicle.

The masses of the four vehicles are $m_1 = 1576.0kg$, $m_2 = 1061.0kg$, $m_3 = 1413.0kg$, and $m_4 = 1234.0kg$. The d_i and τ_i parameters for all vehicles are set to the nominal values for this mass estimation study. After some tuning, the following PD controller values were used: $K_{p_1} = 1088.0$, $K_{d_1} = 1525.9$, $K_{p_2} = 1411.0$, $K_{d_2} = 1654.2$, $K_{p_3} = 714.1$, $K_{d_3} = 1493.1$, $K_{p_4} = 1523.5$, and $K_{d_4} = 1352.5$. The operation of the AHS can be seen in Figures 14 and 15, where the thin solid line in Figure 14 indicates the velocity profile of the "virtual" vehicle ahead of the lead vehicle. The dotted lines in both figures indicate the behavior of the lead vehicle (or first vehicle

in the platoon). The dashed lines indicate the behavior of the second vehicle. The dash-dot lines indicate the behavior of the third vehicle. And the thick solid lines indicate the behavior of the fourth and final vehicle in the platoon. The intervehicle spacing errors are shown in Figure 15. Notice that they are all less than 0.8 meters, so safe AHS operation is achieved.

Each vehicle employs its own GA for the mass estimation problem. That is, four separate GAs are operating in this study. Each GA uses the fitness function found in Equations 4 and 5, with $e_k = y(k) - \hat{y}(k)$ for $k > 0$ and $e_k = 0$ for $k \leq 0$. In this case, $y(j)$ and $\hat{y}(j)$ have the following form for the i^{th} vehicle

$$\begin{aligned} y(j) &= \begin{bmatrix} \delta_i(j) \\ v_i(j) \end{bmatrix} \\ \hat{y}(j) &= \begin{bmatrix} \hat{\delta}_i(j) \\ \hat{v}_i(j) \end{bmatrix} \quad . \end{aligned} \tag{27}$$

The states $\hat{\delta}_i(j)$, and $\hat{v}_i(j)$ that correspond to the beginning of a window of data are initialized to the values of $\delta_i(j)$ and $v_i(j)$ at the beginning of that window. This procedure is the same as that used in the previous windowed GA applications.

However, there is a difference from previous windowed GA applications in that the state f_i is assumed not to be directly measurable. One must find some way to initialize $\hat{f}_i(j)$ at the beginning of a data window using only the measurable quantities $\delta_i(j)$ and $v_i(j)$. Referring to Equations 25, a possible equation for $\hat{f}_i(j)$ initialization (for the data window at time k) can be written

$$\hat{f}_i(j) = \frac{\hat{m}_i(k)}{T}\Big(v_i(j+1) - v_i(j)\Big) + A_\rho v_i^2(j) + d_i \quad , \tag{28}$$

where $\hat{m}_i(k)$ is the current mass estimate for the i^{th} vehicle at time k. Since a window of *past* data values is used, $v_i(j+1)$ is available for this initialization procedure. Note that as $\hat{m}_i(k)$ converges to m_i, this initialization becomes more accurate. Equation 28 has been found to work quite well, as will be seen below.

The parameters chosen for the GA vehicle mass estimation were as follows: $N = 30$, $p_m = 0.3$, $D = 20$, $\rho \cdot D = 2$, $W = 50$, and $g_t = 1$. Again, they were found by tuning the GA according to the procedure discussed in Section 4.2. The mass estimates $\hat{m}_i$ were allowed in the range [1000.0,1600.0]. The results of the estimation are shown in Figure 16. Note how quickly the mass estimate for each vehicle was able to converge to the exact mass

of that vehicle. In all cases, complete convergence occurred in less than 1 second. This seems to provide additional evidence for the utility of the GA in nonlinear system parameter estimation.

7.2 Multiple Parameter Estimation

Although the GA worked quite well for the estimation of a single parameter within the AHS, it is important to realize that parameters such as d_i and τ_i may exhibit variations that one may want to estimate. Such variations could result in faulty estimates $\hat{m}_i$. In addition, if the engine or brakes undergo any type of degradation or failure there will be noticeable differences in the constant frictional force d_i or in the engine/brake time constant τ_i (or both). Clearly, being able to estimate these additional parameters can be useful, not only in maintaining accurate estimates of m_i, but also in the early detection of brake or engine problems. Therefore, the GA should be used to estimate these parameters as well.

However, instead of using a single GA to estimate all three parameters, three parallel GAs are used on each vehicle, each estimating one parameter. One reason for this is that additional parameters to be estimated by a single GA might lead to a larger required population. Another reason is that adding parameters to a GA raises the dimension of the GA's search space, reducing the likelihood for quick convergence. In addition, if it is possible that the fitness function used by GA favors any of the parameters to be estimated, the convergence of the remaining parameters may suffer. Finally, since the fitness function seeks the *combination* of the parameter estimates that best describes the system, there is no guarantee that the individual parameters will converge to their correct values. Since the individual parameters (not their combination) are of interest in this case, the choice is made to use a separate GA for each parameter. Thus, twelve GAs are in operation, three GAs on each of the four vehicles in the platoon.

For this case, the same values of m_i used in the single parameter mass estimation are used. The values of d_i and τ_i may differ from the nominal values used in the single parameter mass estimation. In fact, the values used for those parameters are as follows: $d_1 = 103.4N$, $\tau_1 = 0.22s$, $d_2 = 84.1N$, $\tau_2 = 0.21s$, $d_3 = 100.0N$, $\tau_3 = 0.42s$, $d_4 = 119.6N$, and $\tau_4 = 0.35s$.

The fitness function used for each GA is identical to that for the single parameter mass estimation. The $\hat{f}_i(j)$ initialization equation used is Equation 28 with d_i replaced by $\hat{d}_i(k)$. In addition, each GA for a given vehicle has access to the current estimates from the other two GAs for that vehicle. For example, the GA generating the current estimate $\hat{m}_1(k)$ has access to the most recent estimates $\hat{d}_1(k)$ and $\hat{\tau}_1(k)$ from the GAs generating those

estimates, and vice versa. For the first generation, before estimates of the other two parameters are available, the nominal values are used (for $\hat{m}_i$, the nominal value of $1300.0kg$ is used).

The following parameters were used for all GAs: $N = 40$, $p_m = 0.4$, $D = 20$, $\rho \cdot D = 2$, $W = 1000$, and $g_t = 1$ (with twenty time samples elapsing between successive generations). The simplest method of tuning the parameters for twelve GAs was to use the same parameter value for every GA, as if only one GA needed to be tuned. The actual tuning procedure used was the same as that used in previous sections. Successive generations were chosen to be twenty time samples apart in order to improve the likelihood that such a window length W could be implemented in real time, as it was found that this large a window length improved the convergence of all estimates. The estimates $\hat{m}_i$ were allowed in the range [1000.0,1600.0], the estimates $\hat{d}_i$ were allowed in the range [80.0,120.0], and the estimates $\hat{\tau}_i$ were allowed in the range [0.2,0.5]. A sampling period T of 0.01 seconds was used, and the same K_{p_i} and K_{d_i} parameters from the single parameter mass estimation were used.

The results are shown in Figures 17 through 19. Note that all of the GAs performed quite well. A slight fluctuation appears in the estimates of vehicle number 3 at the end of the simulation, but note from Figure 14 that the velocities and intervehicle spacings stay constant at this point. This would indicate that the estimation works best when the states of the system are varying (i.e. some excitation is provided). To overcome this problem, the GA could be redefined in such a way that it would not adapt during steady state operation. At any rate, there seems to be enough evidence here for the GA as a possible alternative for nonlinear system parameter estimation.

To verify the validity of Figures 17 through 19, 100 simulations were executed. In each simulation, a "squared error sum" was calculated for each of the three estimates on each vehicle. This squared error sum is a summation, over the entire simulation, of the error between the parameter estimate and the actual parameter. In other words,

$$E_{s_\kappa} = \sum_{k=1}^{m_g} \Big(\kappa - \hat{\kappa}(k) \Big)^2 \quad , \tag{29}$$

where κ is the parameter in question, $\hat{\kappa}(k)$ is the estimate of κ at time sample k, and m_g is the total number of time samples in the simulation.

Average, minimum, and maximum squared error sums over all 100 simulations were recorded for all of the parameters. These results are given in Tables 1 through 3. From most of the maximum and minimum values given, it seems that the corresponding parameters exhibit squared error

sums that remain within a "reasonable range" of the average values. For a few of the parameters, the average values are somewhat close to the minimum values but far away from the maximum values. This implies that for those parameters, nearly all of the simulations provided squared error sums within a reasonable range of the average values, with a few simulations generating squared error sums well outside reasonable range. Finally, for a few parameters, the average values were somewhat far away from minimum *and* maximum values. For these parameters no conclusions can be made as to whether the squared error sums generally remained within a reasonable range of the average values.

The squared error sums corresponding to Figures 17 through 19 resemble the average squared error sums and are given in Table 4. Upon examining several executions of the GA*, it would seem improbable for any given execution to generate a squared error sums table that almost *exactly* matches the table of average values. Figures 17 through 19 (and Table 4) correspond to a typical case. In this representative case, the $E_{s_{m_i}}$ and $E_{s_{d_i}}$ values fall reasonably close to the average values, while the $E_{s_{\tau_i}}$ values may tend to stray from those average values. Examining the figures, it seems that even with this behavior, the parameter estimates can be expected to converge reasonably to the ideal parameter values. This serves as further evidence for the GA as a reasonable parameter estimation tool for nonlinear systems.

Table 1: Average squared error sums for GA AHS parameter estimation.

	avg $E_{s_{m_i}}$	avg $E_{s_{d_i}}$	avg $E_{s_{\tau_i}}$
Vehicle 1	2300749.2	5365.180	0.18766
Vehicle 2	458925.6	357.520	0.00230
Vehicle 3	2059526.6	5498.766	0.66574
Vehicle 4	295475.8	1574.246	0.06249

8 Concluding Remarks

In this study, we introduced a novel genetic adaptive parameter estimation technique and studied its performance for four cases—two linear cases, a nonlinear ball on a beam system, and a nonlinear automated highway system (AHS) application. For the linear cases and for the ball on a beam system, the performance of an average GA execution was compared with that of the

Table 2: Minimum squared error sums for GA AHS parameter estimation.

	min $E_{s_{m_i}}$	min $E_{s_{d_i}}$	min $E_{s_{\tau_i}}$
Vehicle 1	1421365.2	4506.865	0.05075
Vehicle 2	409574.4	295.361	0.00066
Vehicle 3	734650.1	4450.025	0.15376
Vehicle 4	148538.2	1569.669	0.02341

Table 3: Maximum squared error sums for GA AHS parameter estimation.

	max $E_{s_{m_i}}$	max $E_{s_{d_i}}$	max $E_{s_{\tau_i}}$
Vehicle 1	15885661.0	6226.874	2.54984
Vehicle 2	528768.0	560.942	0.01153
Vehicle 3	7951851.3	7273.442	2.31489
Vehicle 4	1415352.3	1589.753	0.93605

Table 4: Squared error sums for GA AHS parameter estimation shown in Figures 17 through 19.

	$E_{s_{m_i}}$	$E_{s_{d_i}}$	$E_{s_{\tau_i}}$
Vehicle 1	1436288.9	5218.439	0.05944
Vehicle 2	456692.8	339.943	0.00111
Vehicle 3	2276631.3	5754.204	1.04031
Vehicle 4	317209.5	1580.793	0.03016

recursive least squares technique. The GA appeared able to perform the task of parameter estimation for almost all of the linear examples, although it was generally outperformed by recursive least squares. For the ball on a beam system, the GA easily outperformed recursive least squares, as was to be expected because recursive least squares was forced to estimate an approximate linear model. For the AHS application, the GA alone was studied (since specific parameters were of interest and since it is a nonlinear system), and except for a few easily correctable instances, the parameter tracking was quite good. From these three cases, evidence can be gathered that may point to the existence of the GA as an alternative method for

parameter estimation, particularly in nonlinear cases.

Clearly, however, further research of the GA as a state estimation tool is needed. Among this research, possible topics could include

- Alternative fitness function formulations,
- Mathematical stability, convergence, and robustness analysis,
- Actual implementation issues (e.g., processor speed, memory resources, possible hardware operations, etc.), and
- Tuning procedures for the GA.

Due to the computational complexity of the GA, it seems logical to attempt conventional approaches for parameter estimation *before* employing the GA, especially for linear systems. From the results in this study, however, there is evidence that the GA does provide a viable alternative when other techniques encounter difficulty, particularly when dealing with nonlinear systems.

References

[1] L. Yao and W. A. Sethares, "Nonlinear parameter estimation via the genetic algorithm," *IEEE Transactions on Signal Processing*, vol. 42, pp. 927–935, April 1994.

[2] K. Kristinsson and G. Dumont, "System identification and control using genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 5, pp. 1033–1046, 1992.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[4] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1992.

[5] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, pp. 17–26, 1994.

[6] W. K. Lennon and K. M. Passino, "Intelligent control for brake systems," in *IEEE International Symposium on Intelligent Control*, 1995.

[7] P. A. Ioannou and J. Sun, *Robust Adaptive Control*. Prentice-Hall, 1996.

[8] J. T. Spooner and K. M. Passino, "Adaptive control of a class of decentralized nonlinear systems," tech. rep., Department of Electrical Engineering, The Ohio State University, February 1995.

[9] R. E. Fenton and R. J. Mayhan, "Automated highway studies at the ohio state university—an overview," *IEEE Transactions on Vehicular Technology*, vol. 40, pp. 100–113, February 1991.

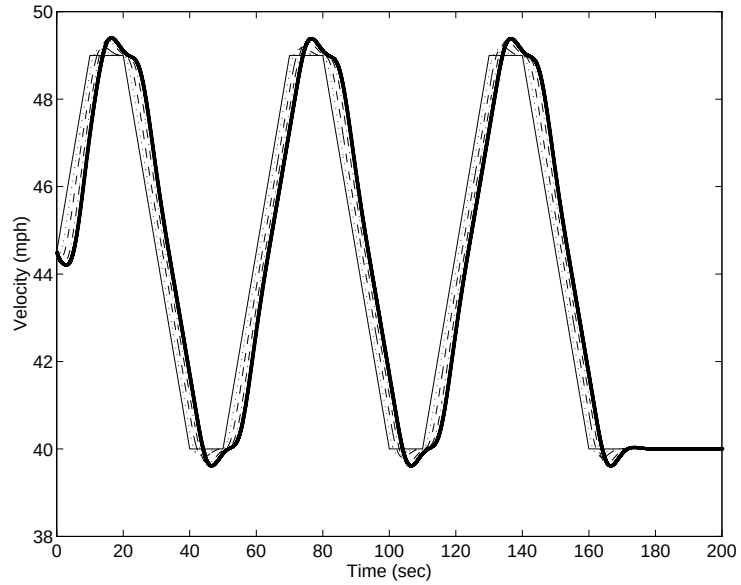


Figure 14: Velocity profiles for a four car AHS platoon.

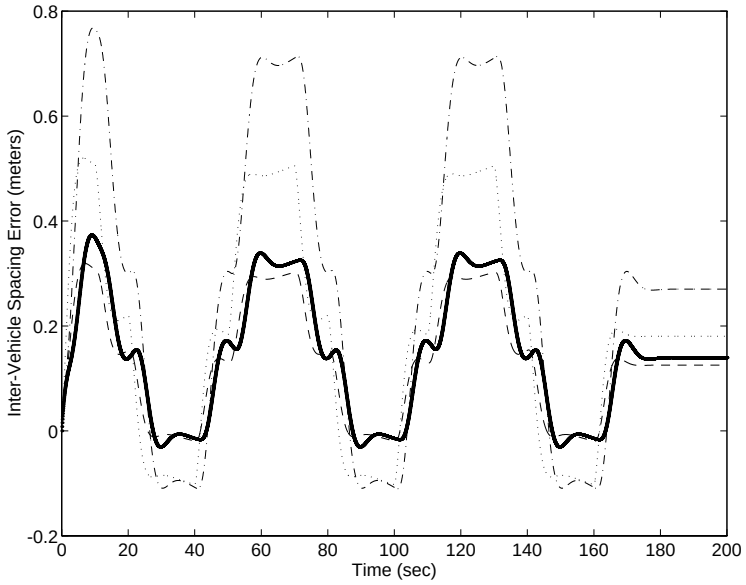
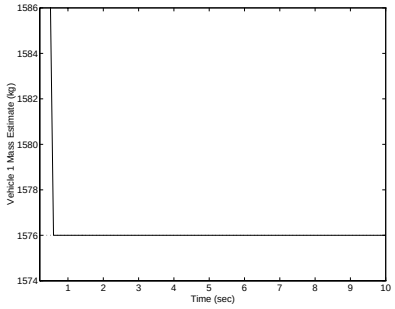
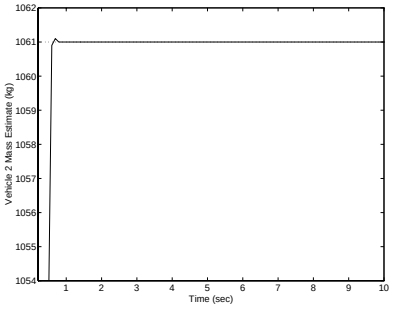


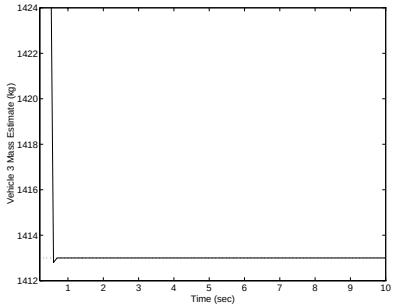
Figure 15: Intervehicle spacing errors for a four car AHS platoon.



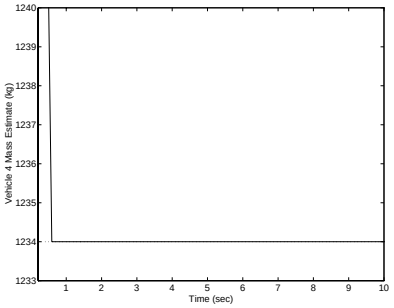
(a) Estimation of m_1.



(b) Estimation of m_2.

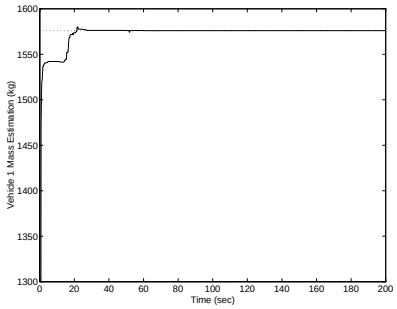


(c) Estimation of m_3.

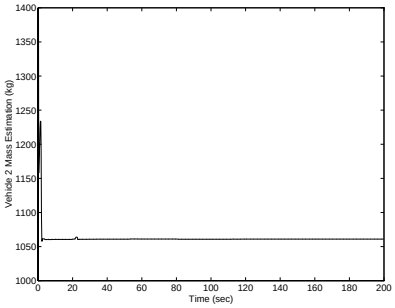


(d) Estimation of m_4.

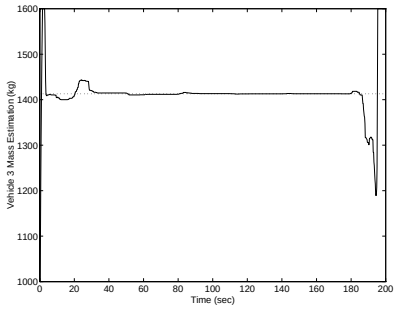
Figure 16: GA vehicle mass estimation for AHS. All estimates remain constant over the full 200 second simulation. Dotted lines indicate ideal masses while solid lines represent mass estimates.



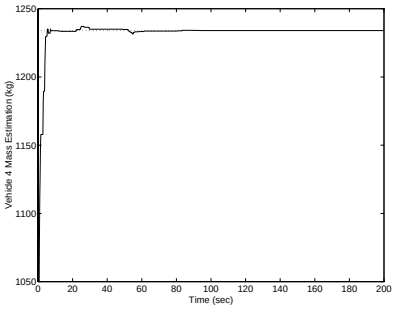
(a) Estimation of m_1.



(b) Estimation of m_2.

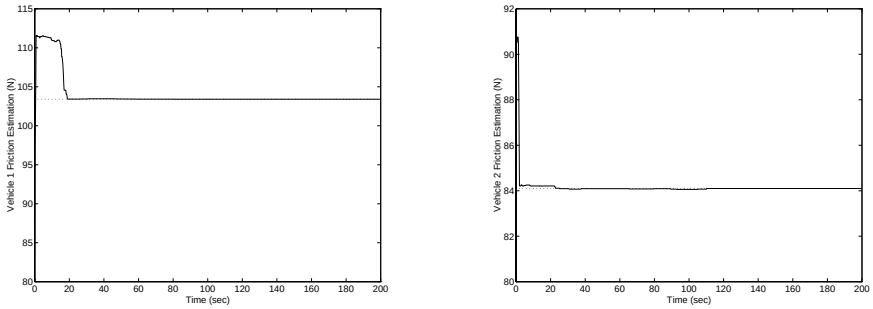


(c) Estimation of m_3.



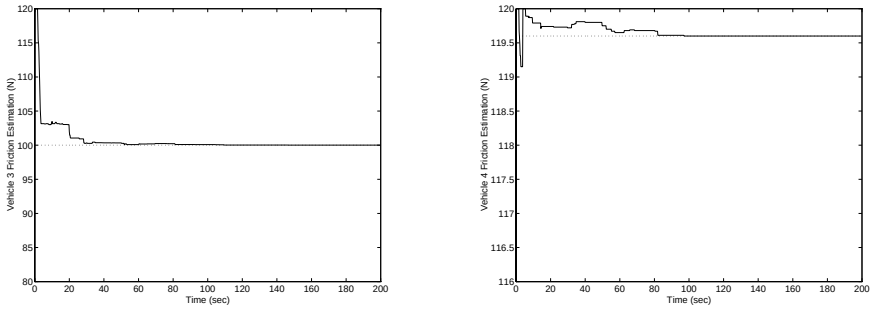
(d) Estimation of m_4.

Figure 17: GA multiple parameter mass estimation for AHS. Dotted lines indicate ideal mass values while solid lines represent mass estimates.



(a) Estimation of d_1.

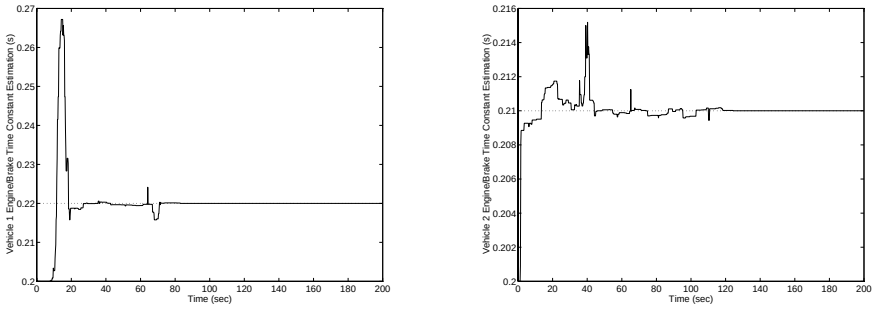
(b) Estimation of d_2.



(c) Estimation of d_3.

(d) Estimation of d_4.

Figure 18: GA multiple parameter d estimation for AHS. Dotted lines indicate ideal friction values while solid lines represent friction estimates.



(a) Estimation of τ_1.

(b) Estimation of τ_2.

(c) Estimation of τ_3.

(d) Estimation of τ_4.

Figure 19: GA multiple parameter τ estimation for AHS. Dotted lines indicate ideal time constant values while solid lines represent time constant estimates.

James R. Gremling received both a B.Sånd an M.S. in Electrical Engineering at The Ohio State University in 1994 and 1996, respectively. He has worked at Battelle Memorial Institute assisting on a contract with the United States Air Force for its F-16 Radar Electro/Optic (REO) display set. He currently works for Lucent Technologies, at the Columbus Works factory in Ohio, providing test engineering support for the manufacture of the AirLoop (registered trademark) wireless local access product. His interests include intelligent control, wireless telecommunications, computer programming, bicycling, and Christian community involvement.

Kevin M. Passino received his Ph.D. in Electrical Engineering from the University of Notre Dame in 1989. He has worked on control systems research at Magnavox Electronic Systems Co. and McDonnell Aircraft Co. He spent a year at Notre Dame as a Visiting Assistant Professor and is currently an Associate Professor in the Dept. of Electrical Engineering at The Ohio State University. He has served as an elected member of the IEEE Control Systems Society Board of Governors; was Chair for the IEEE CSS Technical Committee on Intelligent Control; has been an Associate Editor for the IEEE Transactions on Automatic Control and also for the IEEE Transactions on Fuzzy Systems; served as the Guest Editor for the 1993 IEEE Control Systems Magazine Special Issue on Intelligent Control; and a Guest Editor for a special track of papers on Intelligent Control for IEEE Expert Magazine in 1996; and was on the Editorial Board of the Int. Journal for Engineering Applications of Artificial Intelligence. He is currently the Vice President for Technical Activities of the IEEE Control Systems Society. He was a Program Chairman for the 8th IEEE Int. Symp. on Intelligent Control, 1993 and was the General Chair for the 11th IEEE Int. Symp. on Intelligent Control. He is co-editor (with P.J. Antsaklis) of the book "An Introduction to Intelligent and Autonomous Control," Kluwer Academic Press, 1993; co-author (with S. Yurkovich) of the book "Fuzzy Control," Addison Wesley Longman Pub., 1998; and co-author (with K.L. Burgess) of the book "Stability Analysis of Discrete Event Systems," John Wiley and Sons, 1998. He has authored over 130 technical papers and his research interests include intelligent systems and control, adaptive systems, stability analysis, and fault tolerant control.