Curve Fitting Using Genetic Algorithms

DTIC
ELECTE
FEB 27, 1992
S
B
D

K. Messa
Loyola University
Department of Mathematical Sciences
New Orleans, LA 70118

M. Lybanon
Remote Sensing Branch
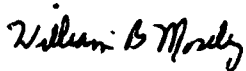Ocean Sensing and Prediction Division
Ocean Science Directorate

92-04806

92 2 25 050

Foreword

The principle of least squares is applied in every field of science and engineering, but least-squares algorithms often suffer from convergence problems in complex applications. The present work was motivated by the difficulty encountered in fitting a mathematical model to altimetric sea-surface height residuals. The presence of mean dynamic topography in the reference surface used to calculate the residuals leads to significant difficulty in interpretation. The goal of the fitting of a mathematical representation is the estimation and subsequent removal of the error. However, standard least-squares procedures have difficulty unless the initial estimates of model parameters are very good.

This report describes research performed at the Naval Oceanographic and Atmospheric Research Laboratory on the application of genetic algorithms to the task of fitting mathematical models to data. Genetic algorithms are search techniques that are based upon the mechanics of natural selection. They have been used successfully in a number of optimization problems, but this is apparently their first application to curve fitting. The genetic algorithm approach is easily implemented, is accurate, and provides consistent results. It also has application to curve-fitting problems in general, not just to the problem that motivated the research. Therefore, this new approach to least-squares curve fitting has the potential to make a valuable contribution in a number of scientific fields.

W. B. Moseley
Technical Director

L. R. Elliott, Commander, USN
Commanding Officer

Executive Summary

Genetic algorithms are search techniques based on the mechanics of natural selection. They have been used successfully in many applications because of their robustness and because of their ability to search in a noisy problem space. In particular, genetic algorithms are used in curve-fitting. The genetic algorithm selects the coefficients of a particular curve that most closely matches a given set of data.

Candidate solutions are vectors of real numbers that represent the coefficients of the curve to be modeled. Thus, every candidate solution corresponds to a new function. As such, each candidate solution is evaluated using the sum of the squares of the residuals. The evaluation of each of these curves with respect to its fit of the data guides the genetic algorithm toward the solution with the greatest merit.

Several examples of the application of genetic algorithms to curve-fitting problems are presented. Convergence to the optimal solution is rapid when knowledge of the coefficients is available. When little is known about the coefficients, a degree of experimentation helps obtain the optimal solution.

Acknowledgments

Contents

Curve Fitting
Using Genetic Algorithms

I. Introduction
Techniques to fit curves to data are plentiful. Linear least squares/ regression is one of the more commonly used methods. There are also nonlinear techniques that work well when the data set is small and when the function is relatively simple. However, many of these techniques often fail when large data sets are used or when complicated functions need to be modeled. Singular value decomposition, QR* decomposition, Marquardt's algorithm, and steepest descent techniques are sometimes used. This report describes an alternate method of curve fitting that can be used for complex functions and for large data sets and that employs genetic algorithms.

Genetic algorithms are search techniques based on the mechanics of natural selection. Genetic algorithms are related to "generate-and-test" search techniques. In pure "generate and test," a candidate solution is generated and then sent to an evaluator for testing. If the candidate solution is not optimal, then the procedure is repeated. In genetic algorithms, the generate-and-test procedure is repeated iteratively over a large set of candidate solutions. Because this set can be large, a significant number of possible solutions can be tested simultaneously. Holland (1975) refers to this as implicit parallelism. Genetic algorithms seem to be unique among optimization techniques in employing implicit parallelism.

The terminology of genetic algorithms is taken from genetics. Each candidate solution is called an *organism*. A *chromosome* is a list of elements called *genes*. In the simplest case, an organism consists of a single chromosome (haploid), although there are cases when the organism consists of dual-strand chromosomes (diploid). Chromosomes usually consist of linear lists of genes. A gene can assume any of a number of values called *alleles*, which are taken from the base set, for example, {0, 1}. Generally, problem solutions are encoded as strings of alleles (most commonly, strings of 0's and 1's).

Many organisms are grouped together into a set called a *population*. The genetic algorithm evaluates a population and generates a new one iteratively. Each successive population is called a *generation*. Thus we have an initial population, $G(0)$, and for each generation $G(t)$, the genetic algorithm generates a new one, $G(t + 1)$. An algorithm to implement a genetic algorithm is given by

generate initial population, $G(0)$;
evaluate $G(0)$;
$t : = 1$;
repeat
    generate $G(t)$ using $G(t - 1)$;
    evaluate $G(t)$;
    $t : = t + 1$;
until solution is found.

---

*A method developed by Francis (1961; 1962) for finding the real and complex eigenvalues of an arbitrary matrix.

Like all generate-and-test methods, the genetic algorithm requires the two main steps of generation and evaluation. To evaluate a population, a *fitness* function is needed. In nature, a species responds in some way to environmental pressure. The genetic algorithm analog to this pressure is the fitness function. The fitness function is built from domain-specific information and returns the relative merit, or fitness, of the organism.

The operation of generating a new population distinguishes the genetic algorithm from the other techniques. To obtain the next generation, pairs of organisms are selected based on their fitness. The pairs are combined to form new organisms that are added to the next generation. This procedure is the genetic algorithm analog to "survival of the fittest." As with population genetics, the pair selected are called the *parents*. Their mating produces new organisms called *offspring*. Goldberg (1989) discusses genetic algorithms in detail.

Genetic algorithms also differ in other ways from traditional search and optimization methods:

- Genetic algorithms search large numbers of candidate solutions simultaneously.
- Genetic algorithms are probabilistic—they use random search and/or selection rather than deterministic methods.
- The objective function used by genetic algorithms is based on actual, problem-specific information, rather than auxiliary information, such as derivatives.

As is the case in natural selection, organisms that are highly fit mate and produce offspring. On the average their offspring are more favorably adapted than offspring of less highly fit parents. Generally, the offspring of the highly fit are themselves highly fit, some even more so than their parents.

The genetic algorithm is based on the same principle as natural selection. The first operator used by genetic algorithms is called *selection*. Like its natural selection counterpart, the selection operation selects pairs of highly fit organisms for mating. This focus toward the highly fit individuals is what drives genetic algorithms.

The genetic algorithm analogy to mating is called *crossover*. The crossover operator provides a mixing of the genes from the parents, and globally it mixes the genetic material of the whole population. It is the mixing of the genes, the stirring of the pot of genetic material, that gives robustness to the genetic algorithm. The two organisms chosen by selection are combined to form a new individual with similarities to both parents. If the mixing is done carefully, then a large amount of genetic material can be tested. Although selection focuses on the genetic algorithm, it is crossover that adds variety.

The simplest version of crossover consists of randomly selecting a single crossover point and then combining the left side of one parent chromosome, up to the crossover point, to the right side of the other parent chromosome (Fig.1). In addition to the single-point crossover, there are quite a few variations of the crossover operator. Each is designed to produce a new offspring from the parents in such a way that the offspring produced will further sample the solution space of organisms.

While selection and crossover are the chief operators used in genetic algorithms, there are numerous other minor operators proposed to

Figure 1. Crossover of two binary-alleled chromosomes after the fifth gene was selected at random as the crossover points.
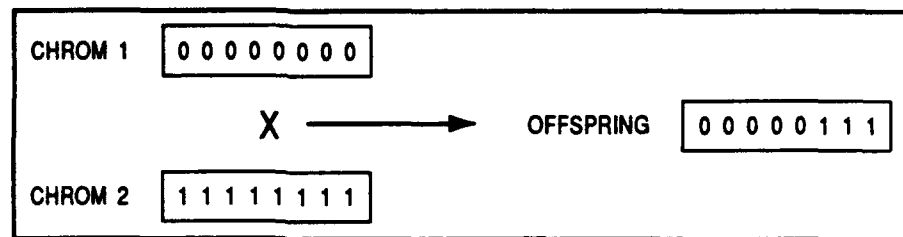
strengthen genetic algorithms under certain circumstances. For certain applications, these minor operators can add to the genetic algorithm's efficiency or prevent it from converging to a local optimum rather than a global optimum. For example, it sometimes happens that the genetic algorithm converges to a solution prematurely because crossover only mixes the genetic material that is present in the initial population; it does not introduce new material. In nature, new genes are introduced into a species through mutation. Analogously in genetic algorithms, a *mutation* operator is used occasionally to modify a chromosome to add new genes into the population and to prevent premature convergence.

De Jong (1975) observed the need to scale the fitness function values of the organisms, since, as the genetic algorithm nears a maximum value, most of the organisms will have fitness values that lie within a very small range. Organisms whose fitness values are near the bottom of this narrow range still have a fairly large probability of being selected for mating. By scaling these low values to near 0, the probability of the less fit being selected is markedly decreased. Scaling has become a widely accepted practice. If an organism has a high fitness value, there is a good probability that it will be selected for mating, thus contributing its genetic material into later generations. However, by chance, it may not be selected, which could delay the genetic algorithm for several generations in finding this individual again. To overcome this problem, an elitist strategy could be used. This strategy allows a few of the fittest organisms to be placed unchanged into the next generation.

II. Representation

The organisms in a genetic algorithm represent solutions to the problem. In this case, solutions are real values assigned to each coefficient in the curve model. There is also a measurement of the goodness of fit with respect to the data $\mathcal{D}$. Thus, if $f = f(a_1, a_2, \ldots a_n; x)$ is the curve to be fit, then real number values for the coefficients $a_1, a_2, \ldots, a_n$ are searched for. Thus, vectors $<r_1, r_2, \ldots, r_n>$, where a_i is replaced by real numbers r_i, $i = 1, 2, \ldots, n$, are candidate solutions. Therefore, organisms for the genetic algorithm used in curve fitting are vectors of real numbers, $<r_1, r_2, \ldots r_n>$.

This view of the representation is useful at the higher level of the curve-fitting problem. However, the genetic algorithm works at a lower level—the level of bits or alleles. To successfully use the genetic algorithm consider a representation of the real numbers r_i at the allele level. Given upper and lower bounds for each r_i, u_i and l_i respectively, we can look at r_i as an unsigned binary integer with m bits and calculate its value with respect to l_i and u_i.

Given a binary integer b, where b is in $[0, 2^{k-1}]$, its corresponding real value is given by the formula

$$r = b/2^m * (u - l) + l,$$ (1)

where u and l are the upper and lower bounds, respectively. Combining these two levels, a candidate solution is constructed:

$$\theta = <b_{11}\ b_{12}\ b_{13}\ .\ .\ .\ b_{1m},\ b_{21}\ b_{22}\ .\ .\ .\ b_{2m},\ .\ .\ .\ ,\ b_{n1}\ b_{n2}\ .\ .\ .\ b_{nm}>,$$

where each binary integer $b_{i1}\ b_{i2}\ .\ .\ .\ b_{im}$ corresponds to a real number r_i which lies in the interval $[l_i,\ u_i]$. The correspondence is given in equation 1.

Computing the value of the fitness function on a candidate solution θ requires the two steps: converting each binary integer $b_{i1}\ b_{i2}\ .\ .\ .\ b_{im}$ into its corresponding real value r_i and then evaluating the curve $f = f(r_1, r_2, \ .\ .\ .\ , r_n;\ x)$ at the data points of $\mathcal{D}$.

III. Fitness Function

The fitness function is a measurement of how well the candidate solution fits the data. It is the entity that focuses the genetic algorithm toward the solution. The fitness used in this problem is built in several stages.

- The candidate solution θ is converted into a vector of real numbers $r = <r_1, r_2, \ .\ .\ .\ , r_n>$ and the value of $f(r; x) = f(r_1, r_2, \ .\ .\ .\ r_n;\ x)$ is calculated for each x_i where the pair (x_i, y_i) is in $\mathcal{D}$.
- The sum of the squares of the differences between the $f(r; x_i)$ and y_i (that is, the residuals) yields a measurement where 0 indicates an exact fit. This value is called the prefitness value of θ. Thus,

$$\text{prefitness } (\theta) = [\Sigma\ (f(r;\ x_i) - y_i)^2],$$ (2)

where the summation is taken over all data points (x_i, y_i) of $\mathcal{D}$.

- Since genetic algorithms work by searching for maximum values and since the prefitness function has its minimum for the best fit, max and min must be reversed. For each population p, let max_p represent the largest prefitness value (that is, the worst fit). (Also let avg_p be the average of the fitnesses of p.) Compute the raw fitness as the difference

$$\text{raw fitness } (\theta) = max_p - \text{prefitness } (\theta).$$ (3)

This computation is not precisely the fitness function used in the genetic algorithm. Most genetic algorithms require a scaling of these fitness values. Scaling amplifies the distinction between good and very good organisms. In the simplest case, linear scaling was used:

$$\text{fitness } (\theta) = m_p * \text{raw fitness } (\theta) + b_p,$$ (4)

where m_p and b_p are constants calculated for each population p. Equations 3 and 4 can be combined to form a linear relationship between the fitness and the prefitness functions:

$$\text{fitness}\,(\theta) = \alpha_1{}^*\,\text{prefitness}\,(\theta) + \alpha_2, \tag{5}$$

where α_1 and α_2 are derived constants for each population. Thus, the final fitness of an organism θ is linearly related to the sum of the squares of the residuals.

IV. Genetic Operators

A. Selection

The method used here is stochastic sampling without replacement, called "expected value" by Goldberg (1989). To implement this method, given an organism θ with fitness f_i, the fitness is weighted with respect to the population's average f^*. The truncated division of f_i by f^* yields the expected number of offspring formed by θ. The expected number of offspring of each organism in the population is calculated. Should any slots in the new population be unfilled after this process is completed, the fractional parts of each fitness, truncated by the previous division, are used to determine remaining positions (De Jong, 1975).

In addition to this selection method, De Jong's elitist strategy (De Jong, 1975) was used, whereby the single best organism from one generation is placed unchanged into the next generation. This strategy gives a little more weight to the best organism than might be achieved from selection alone and prevents the possibility that the best organism might be lost early through crossover or mutation.

B. Crossover

Simple crossover was used and it worked effectively. For problems with many coefficients or where many bits are needed for greater precision, two-point crossover yielded better results than single-point crossover. Ninety percent of the selected pairings were crossed, and the remaining 10% were added unchanged to the next generation. This is referred to as a 90% crossover rate.

C. Mutation

The mutation method most commonly used is to change the value of a bit at a gene position with a frequency equal to the mutation rate. Numerous mutation rates were tested, but good results were found infrequently while using the bit-change method. High mutation rates have the effect of introducing a large amount of new material into the population. Thus, good solutions were frequently found. However, a high mutation rate also has the effect of frequently changing organisms, including those that have a good fitness. So while high rates often find good solutions, they tend to be disruptive to the population as a whole. There were other times that, even with a high rate, the genetic algorithm failed to find an acceptable solution. With low mutation rates, the genetic algorithm also often failed.

A problem with this mutation method is related to the so-called "Hamming cliffs" (Carruana and Schaffer, 1988). Suppose an optimal coefficient has value of 0.5 and is represented as 100000 using 6-bit binary integers and a boundary interval of [0, 1]. A candidate solution of 011111 (= 0.46875) might yield fairly good results because of its nearness to the optimum. However, once the genetic algorithm has begun converging to this suboptimal value, it is very unlikely for it to move away from this near-optimal solution toward a better solution. Only

chan_ing a single bit or two would never enable a population of such organisms to move to the optimal solution.

Instead of the bit-change mutation, a method that adds a real value ε to (or subtracts it from) the organism's converted value was introduced. The value ε is a power of 2 ranging from 1 to 2^m. If $\varepsilon_i = 0 \ldots 0 1 0 \ldots 0$ with 1 in the ith position and 0's elsewhere, then this mutation method adds ε to (or subtracts ε from) the organism to be mutated, where ε is in $\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m\}$. We will refer to this method of mutation as $\pm\varepsilon$.

The method proceeds as follows: If randomness has determined that the allele in the relative position i (the ith gene of some coefficient in θ) is to be mutated, then ε_i is either added to or subtracted from θ. If the ith gene is 0, then adding ε_i is identical to a bit change. Or if the ith gene is 1, then subtracting ε_i is just the bit-change mutation. The improvement in this method over bit-change mutation occurs when ε is added to a gene with value of 1 (or subtracted from a gene with a value of 0). Given the example referred to, adding 1 to the sixth gene in 011111 yields the optimal value of 100000.

These two mutation methods are compared in Table 1. The results are from two runs; one run uses a genetic algorithm with bit-change mutation, and the other uses the same genetic algorithm parameters, but with $\pm\varepsilon$ mutation. The runs were chosen because their initial populations had similar best organisms. The values listed are the errors in the fit of each curve. The error used here is the square root of the prefitness value of the organism. That is, error $(\theta) = SQRT(\Sigma\ (f(r; x_i) - y_i)^2)$, where $\theta = <r>$ and (x_i, y_i) range over $\mathcal{D}$. Results were taken from Problem 1 (Section V) for illustration purposes here. In this case, the ideal organism was predetermined to have error of 0.411911.

As can be seen, both methods allow convergence toward the optimum value. However, the bit-change mutation method often does not approach the optimum as rapidly because of the Hamming cliffs.

The $\pm\varepsilon$ mutation allows faster convergence than bit-change mutation in several runs with otherwise identical genetic algorithm parameters. Therefore, not only does $\pm\varepsilon$ provide better accuracy, it reaches its answer faster. Since the ideal organism has an error of 0.411911, a value of 0.42 was chosen as the threshold for measuring convergence (Table 2). Note that convergence in Problem 1 was measured by comparing the error to the threshold. Only in this case was the ideal organism known in advance. This luxury was not present in the other problems.

Table 1. Comparison of errors of best organisms from each generation.

Bit Change vs. Mutation Techniques		
Generation	Bit Change	$\pm\varepsilon$
0	2.097375	2.074479
10	0.663834	0.412049
20	0.530236	0.412039
30	0.427454	0.411977
40	0.421900	0.411972
50	0.418273	0.411947

Table 2. Generation in which best organism had an error less than 0.42.

Convergence Comparison of Bit Change and $\pm\varepsilon$ Mutation		
Run	Bit Change	$\pm\varepsilon$
1	18	6
2	*	40
3	41	5

*Did not reach the 0.42 threshold in 50 generations.

For the simpler problems, the $\pm\varepsilon$ mutation method was used. However, decaying mutation worked better for more complex problems. Decaying mutation proceeds as follows:

Let $\mathcal{E} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m\}$ be as described above. Initially, ε is chosen at random from $\mathcal{E}$. After some specified number of generations (for example, 10 or 20), choose ε from $\mathcal{E} - \{\varepsilon_1\}$. Later, choose ε from $\mathcal{E} - \{\varepsilon_1, \varepsilon_2\}, \ldots$. In this manner, the change affected by $\pm\varepsilon$ is declining over time. The frequency of change remains fixed, but the degree of change is gradually reduced. This method improved convergence, especially when looking at the average of the population rather than the best organism.

The solution for Problem 1 uses the original fixed $\pm\varepsilon$, while other solutions use the decaying version. However, when convergence is rapid, as it is with Problem 1, the two methods are essentially identical. Hence, references to $\pm\varepsilon$ will be that of the decaying version.

There are several approaches to genetic algorithm convergence. One approach examines the best organism in the population; another uses the average of the population. Holland (1975) refers to two other approaches, where the best of all the generations are averaged (offline) or the average of all the generations are averaged (online). If p_t is the population of generation t, then

$$\text{offline} \ (p_t) = 1/t \ \Sigma \ \max_{p_n} \tag{6}$$

and

$$\text{online} \ (p_t) = 1/t \ \Sigma \ \text{avg}_{p_n}, \tag{7}$$

where the summation is taken over all generations $n = 1, 2, \ldots, t$. Since the goal is to find the best fit for a curve, attention is generally restricted to the best of the generations so far. However, the errors of the best and the average of several generations are compared (Figs. 2 and 3).

A sustained error at a certain value could indicate convergence. Convergence is faster using the best of each generation, since it may take several generations for the average of a population (hence, almost all the members) to be near the optimal value. Convergence is more uniform using the average of the generations. If the best of the generations is used, then a suboptimal organism may stand out for several generations before a better organism is found. If the population is thought to have converged at this suboptimal value, then the genetic algorithm may be stopped prematurely before the optimal organism has a chance to emerge. Table 3 for a comparison of these concepts.

V. Experimental Results

Population size was kept at 100 for all except the most complex functions. For these, convergence was too slow and a larger population of 200 improved performance considerably.

The mutation rate adopted was 1% for the examples below. Scaling was crucial, since so many prefitness values were near each other. The fitness of an organism is the new value after scaling is applied
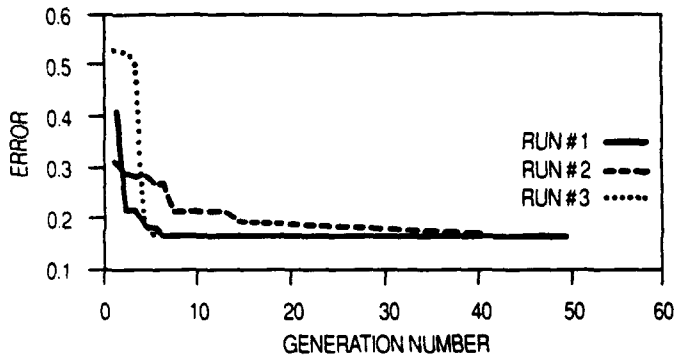
Figure 2. Error (square root of prefitness) of the best of each generation. The ideal organism has value 0.1697.
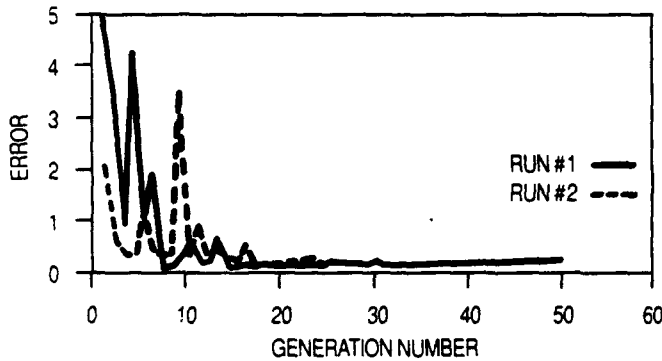

Figure 3. The average error from each generation. Ideal value is 0.1697.

Table 3. Generation at which the threshold is maintained.

Comparison of Using "Best of Generation" and "Average of Generation" for Convergence				
Run	Best	Avg.	Best	Avg.
1	6	40	19	74
2	40	54	78	91
3	5	43	28	77
	Threshold: 0.42		Threshold: 0.412	

to the raw fitness value. See equation 4. For the problems in this analysis, scaling proceeded as follows: After the raw fitness of each organism in a population p was calculated, the average raw fitness avg_p was computed. For any organism with a value equal to avg_p, scaling made no change. Likewise, for organisms whose raw fitnesses were less than avg_p, no change was made. The organism with the largest raw fitness max_p was scaled to $avg_p *$ scale_factor, where scale_factor is a predetermined parameter. Organisms with values intermediate between avg_p and max_p are scaled proportionately. Thus,

$$\text{fitness }(\theta) = \begin{cases} \text{raw\_fitness }(\theta), & \text{if raw\_fitness }(\theta) < avg_p \\ m_p * \text{raw\_fitness }(\theta) + b_p, & \text{otherwise}, \end{cases} \quad (8)$$

where m_p and b_p are from equation 4. To achieve the accuracy desired, very large scale factors were needed. They ranged from 1,000 to 100,000.

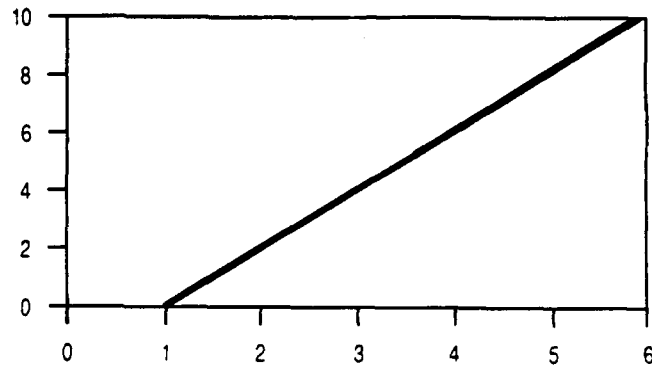*Figure 4. Linear function which best fits the data set from Problem 1.*

Table 4. Generation at which the threshold is maintained.

Convergence Comparisons Based on the Width of the Domain Intervals				
	Best of Each Generation			
Run	Interval Width		Interval Width	
	2	16	2	16
1	3	19	1	6
2	37	76	3	40
3	10	26	6	5
Average	17	40	3	17
	Threshold = 0.412		Threshold = 0.42	

The first example is designed to illustrate only the concepts. Should a problem of this simple genre be encountered, other more traditional strategies, such as least squares regression, might be employed to solve it. However, the model described here works well.

Problem 1. Given the data $\mathcal{D} = \{(1.0, 0.0), (2.01, 2.15), (3.08, 4.14), (4.22, 6.08), (5.0, 8.17)\}$ find the linear function $f = Ax + B$ that fits $\mathcal{D}$ most closely (Fig. 4).

The genetic algorithm used in curve fitting requires bounds for the coefficients. If the length of the boundary interval is small, then convergence is generally rapid. For larger intervals, convergence proceeds more slowly. In this problem, the answers were known in advance: $\cong A = 2.0$ and $B = -2.0$. By choosing intervals [1,3] and [-3,-1] for A and B, respectively, very rapid convergence is obtained. For the wider interval [-8, 8], convergence is somewhat slower (Table 4).

Assume that A and B lie within the boundary interval [-8,8]. The number of bits assigned to the representation is a function of the interval width and the degree of precision needed. The representation of the solution was somewhat arbitrarily chosen as 24 bit integers. This number gave the accuracy to the fifth decimal position. For example, 1010100 . . . 0 would represent 2.5 (calculated from 0.65625*16 – 8) and –5.75 would be represented as 00100100 . . . 0 (as 0.140625*16 – 8).

Since the curve to be fitted is linear, the results can be compared to the solution using linear regression (Table 5).
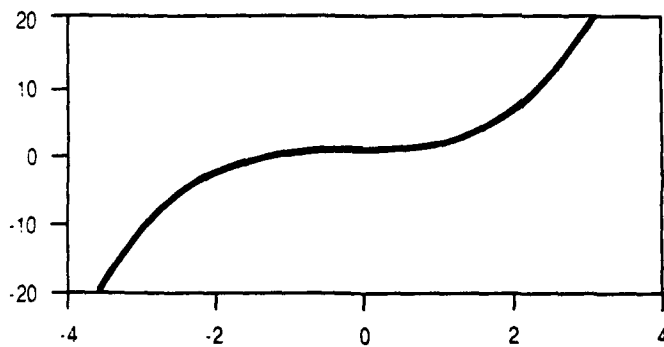
Figure 5. Cubic polynomial which best fits the data set from Problem 2.

Table 5. Comparison of several genetic algorithm methods to linear regression.

Comparison of Several Curve Fitting Methods Best of the 50th Generation			
Method	A	B	Error
Regression/Least Square	1.97852	-1.95024	0.411911
Genetic Algorithm/Small Intervals	1.97849	-1.95012	0.411948
Genetic Algorithm/Large Intervals	1.97860	-1.95047	0.411945
Genetic Algorithm/Large Intervals*	1.97853	-1.95028	0.411944

*100 generations

Table 6. Results of three runs of genetic algorithm on the cubic equation $f = Ax^3 + Bx^2 + Cx + D$ and the data from Figure 5.

Coefficients Found by Genetic Algorithm Using ±ε Mutation			
	Best of Each Generation		
Coefficient	Run 1	Run 2	Run 3
A	0.499999	0.500000	0.500000
B	0.249999	0.250000	0.250000
C	0.250008	0.250000	0.249999
D	0.749999	0.749999	0.750000
Error	2.14 E-5	5.05 E-6	5.05 E-6

Even though the results in Table 5 are the best of several runs, the worst results are not far off either. When allowed to run to 100 generations, all three runs returned the same error: 0.411944.

Problem 2. The seven data points pictured in Figure 5 lie roughly on a cubic, $f = Ax^3 + Bx^2 + Cx + D$. Find the coefficients that yield the best fit.

The solution to this problem is approached by using the same parameters and strategies as those of Problem 1, except simple crossover with 2 points and decaying ±ε mutation is employed.

The coefficients and the resulting error were found by three runs of 300 generations each. The genetic algorithms converged quite rapidly to reasonably good answers; however, the area of interest was in a greater degree of accuracy than they achieved at first (Table 6).

It may appear that 300 generations is too many. However, if the precision needed, is relaxed, then good results are obtained sooner.

Table 7. Genetic algorithm runs with bit change mutation.

Coefficients Found by Genetic Algorithm Using Bit Change Mutation			
	Best of Each Generation		
Coefficient	Run 1	Run 2	Run 3
A	0.500001	0.499999	0.499999
B	0.249999	0.250000	0.250000
C	0.249994	0.250008	0.250004
D	0.750004	0.750000	0.749996
Error	1.73 E-5	1.49 E-5	2.30 E-5

Using an error threshold of 0.001, the best of the generations were within the threshold as early as generation 115 but no later than the generation 181. The average of the population yielded similar values in the generation 300. For these cases, with the weakened threshold, the values of the coefficients were accurate to about three decimal places with an average error in the range of 10^{-5}.

For comparison, Table 7 is a list of the coefficients of three runs where traditional bit change mutation is used. Here, the results are nearly as good, and these results were achieved almost as quickly as with $\pm\varepsilon$. Organisms whose values were within the 0.001 threshold were found as early as generation 120, but no later than generation 209. Even after 300 generations, however, the average of the population had not converged. In this case, the average coefficient was accurate to only two decimal places.

After several runs of the genetic algorithm on Problem 2, it became clear that the curve being searched would fit the data exactly or nearly so, since the errors were near 0. Because the fitness values of the organisms were near 0, a variation of the fitness function was tried that might give better results. After calculating the prefitness function as before, that value was raised to a small power p $(0 < p < 1)$. The idea was that organisms near each other in fitness and also with a near-0 prefitness could be separated by this maneuver but not far enough to make their differences too extreme. This separation effectively distanced the good organisms from the very good ones (just as scaling does), but did not place the good organisms too far from the very good ones and cause them to be eliminated from further selection (as scaling sometimes does).

Results comparable to those in Table 6 were achieved, and these values were arrived at sooner than anticipated. These results were reached in less than 200 generations using $p = 0.1$, compared to 300 generations without p.

Problem 3. This problem is designed to test the genetic algorithm's ability to fit data to a nonpolynomial. Fit 10 points to the curve $f = A + B \tanh(C\ (x - D))$, where tanh is the hyperbolic tangent. (This function was chosen because it approximates a cross section of the sea surface topographic expression of a front.) The set of data points and the curve whose coefficients are being searched for are given in Figure 6.

Using information that can be gained from knowledge of the data, the conclusions were that A was between −0.05 and 0.0, B was between
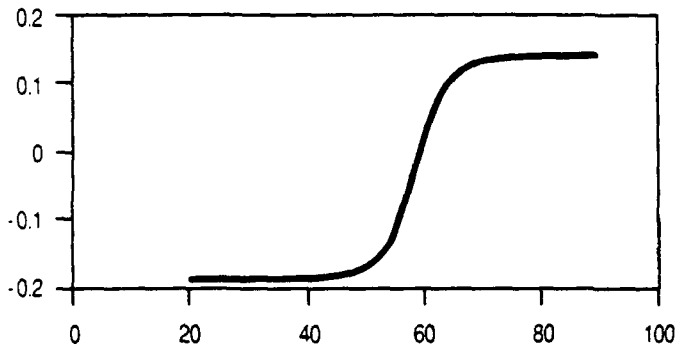
Figure 6. The curve $f = A + B$ tanh[C (x − D)] which best fits the data set from Problem 3.

Table 8. Genetic algorithm runs for nonpolynomial data.

Coefficients Found by Genetic Algorithm Using Narrow Intervals			
	Best of Each Generation		
Coefficient	Run 1	Run 2	Run 3
A	-0.025789	-0.025789	-0.025789
B	-0.162363	-0.162363	0.162363
C	-0.184057	-0.184053	0.184053
D	58.8945	58.8944	58.8945
Error	0.002728	0.002728	0.002728

0 and 0.2, C was between 0 and 1, and D was between 55 and 60. Begin with intervals for A, B, and C of $[-1,1]$. Broaden the range somewhat to allow for errors. Values of $[55,60]$ were used for $\mathcal{D}$. The results from the genetic algorithm are given in Table 8. Note that the values for the coefficients B and C in run 3 are the opposite sign of those in runs 1 and 2. This opposition can be explained by using the property of hyperbolic tangents: $-\tanh(x) = \tanh(-x)$.

When small intervals are used for the domain of each coefficient, convergence is faster and more accurate. The ranges of the coefficients can be derived from the function itself, from knowledge about the problem from which the function arose, and from experimentation. All three techniques were used in choosing the intervals for the functions found here. Because of the size, the interval was much larger (100), and the genetic algorithm described above was adjusted by increasing the length of a chromosome to maintain the same level of accuracy. The genetic algorithm was also allowed to run for additional generations (500) to allow it more time to converge. The results were comparable (Table 9).

Problem 4. The nine data points displayed in Figure 7 lie on the curve, $f = Ax^4 + Bx^3 + Cx^2 + Dx + E$.

As expected, when the boundary interval was large the degree of accuracy was much less than when it was small. An alternate approach to Problem 4 exploited this situation. Initially, large boundaries were chosen for the coefficients. Once convergence was achieved, the genetic algorithm was redone with smaller intervals attained from the earlier runs. For instance, $[-10, 10]$ was initially chosen as the boundary interval for the coefficient A. After several runs, it became clear that A was in the interval $[0.20, 0.30]$. By repeating the genetic algorithm with these smaller intervals, a good degree of accuracy was obtained. The

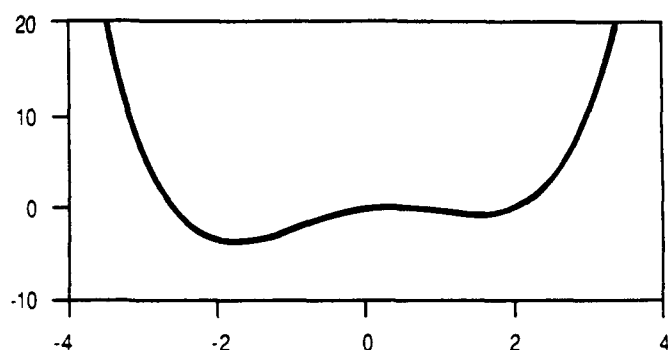*Figure 7. Quartic polynomial which best fits the data set from Problem 4.*

Table 9. Runs for Problem 3 with wider intervals and more generations.

Coefficients Found by Genetic Algorithm Using Wider Interval			
	Best of Each Generation		
Coefficient	Run 1	Run 2	Run 3
A	-0.025792	-0.025793	-0.025785
B	-0.162351	-0.162359	0.162362
C	-0.184088	-0.184062	0.184058
D	58.8448	58.8942	58.8946
Error	0.002728	0.002728	0.002728

Table 10. Genetic algorithm runs for Problem 4. Earlier experiments allowed determination of narrow boundary intervals.

Quartic Polynomial Coefficient Result			
	Best of Each Generation		
Coefficient	Run 1	Run 2	Run 3
A	0.257026	0.257026	0.257033
B	-0.047801	-0.047803	-0.047792
C	-1.50491	-1.50492	-1.50500
D	1.04891	1.04891	1.04833
E	0.185663	0.185687	0.185813
Error	0.967972	0.967972	0.967972

results of this experiment are listed in Table 10. Similar results could probably have been achieved by allowing the original genetic algorithm to run longer and by using longer chromosomes. Further experimentation will determine which approach yields better results.

Problem 5. This last problem is the one that motivated this work. Without developing the source of the problem (Lybanon et al., 1990), take the function $f = A$ $\tanh(B \ (x - D - E)) - F$ $\tanh(C \ (x - E)) + G$ along with 100 data points in $\mathcal{D}$. The methods described above were employed to find the seven coefficients that support the best fit. As with Problem 4, the desired accuracy could not be achieved using large intervals—those whose approximate length was 100. However, the range of possible solutions was reduced considerably. As the genetic algorithm

was repeated with the same parameters, but with the much smaller intervals, the optimal values that were sufficiently accurate could be focused on.

VI. Conclusions/Summary

Genetic algorithms form a basis for another method of curve fitting. Once the genetic algorithm testbed is built, only minor modifications in the parameters and strategies are needed to achieve a fair degree of accuracy. When some knowledge of the coefficients is available, the genetic algorithm can quickly and accurately determine the optimal fit. When there is little knowledge, some experimentation with the genetic algorithm may be necessary to achieve a high degree of accuracy. In any case, genetic algorithms were used successfully to fit curves to data.

VII. References

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, 14–21.

Calman, Jack (1987). Introduction to sea-surface topography from satellite altimetry. *Johns Hopkins APL Technical Digest* 8(2):206–211.

Carruana, R. A. and J. D. Schaffer (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. *Proceedings of the 5th International Conference on Machine Learning*, 153–161.

Daniel, Cuthbert and Fred S. Wood (1980). *Fitting Equations to Data*, 2nd Ed. New York (NY): John Wiley & Sons.

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

De Jong, Kenneth A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. dissertation, University of Michigan. Dissertation Abstracts International 36(10):5104B.

Eshelman, Larry J., Richard A. Carvana, and David J. Schaffer (1989). Biases in the crossover landscape. *Proceedings of the Third Internationa! Conference on Genetic Algorithms*, San Mateo (CA): Morgan Kaufmann, 10–20.

Francis, J. G. F., (1961/62). The QR transformation – a unitary analogue to the LR transformation, Parts 1 & 2. *Comp. Journal* 4:265–271 and 332–345,

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading (MA): Addison-Wesley.

Grefenstette, John J. (1986). Optimization of control parameters of genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics* SMC-16(1):122–128.

Guest, P. G. (1961). *Numerical Methods of Curve Fitting*. Bristol (England): Cambridge University Press.

Holland, John H. (1975). *Adaption in Natural and Artificial Systems*. Ann Arbor (MI): University of Michigan Press.

Lybanon, Matthew, Richard Crout, Conrad Johnson, and Pavel Pistek (1990). Operational altimeter-derived oceanographic information: The NORDA GEOSAT ocean applications program, *Journal of Atmospheric and Oceanic Technology* 7(3):357–376.

Lybanon, Matthew, D. R. Johnson, and R. S. Romalewski (1988). Separation of the mean Gulf Stream topography from an altimeter-derived reference surface. *EOS Transactions,* American Geophysical Union 69(44):1281.

Lybanon, Matthew and Richard L. Crout (1987). The NORDA GEOSAT ocean applications program. *Johns Hopkins APL Technical Digest* 8(2):212–218.

Schaffer, J. David (1985). Multiple objective optimization with vector evaluated genetic algorithms. *Proceedings of an International Conference on Genetic Algorithms and Their Applications,* 93–100.

Siedlecki, W. and J. Sklansky (1989). A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters* 10(5):335–347.

Suh, Jung Y. and Dirk Van Gucht (1987). Incorporating information into genetic search. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms,* 100–107.

Whitley, D. and D. Shaner (1988). Representation Issues in Genetic Algorithms. Department of Computer Science, Colorado State University, Technical Report #CS-88-102.

Distribution List

Applied Physics Laboratory
Johns Hopkins University
Johns Hopkins Road
Laurel MD 20707

Applied Physics Laboratory
University of Washington
1013 NE 40th St.
Seattle WA 98105

Applied Research Laboratory
Pennsylvania State University
P.O. Box 30
State College PA 16801-0030

Applied Research Laboratories
University of Texas at Austin
P.O. Box 8029
Austin TX 78713-8029

Assistant Secretary of the Navy
Research, Development & Acquisition
Navy Department
Washington DC 20350-1000

Chief of Naval Operations
Navy Department
Washington DC 20350-2000
Attn: OP-71
        OP-987

Chief of Naval Operations
Oceanographer of the Navy
U.S. Naval Observatory
34th & Massachusetts Ave. NW
Washington DC 20392-1800
Attn: OP-096
        OP-0961B
        Dr. R. Feden

David W. Taylor Naval Research Center
Bethesda MD 20084-5000
Attn: Commander

Defense Mapping Agency
Systems Center
8613 Lee Hwy.
Mail Stop A-13
Fairfax VA 22031-2137
Attn: Code PRN

Fleet Antisub Warfare Tng Ctr-Atl
Naval Station
Norfolk VA 23511-6495
Attn: Commanding Officer

Fleet Numerical Oceanography Center
Monterey CA 93943-5005
Attn: Commanding Officer
        M. Clancy
        J. Cornelius

Louisiana State University
Department of Computer Science
Baton Rouge LA 70803-4020
Attn: Dr. S. S. Iyengar

Loyola University
6363 St. Charles Ave.
New Orleans LA 70118
Attn: Dr. K. W. Messa (40)
        Dept. of Mathematical Sciences

NOAA
Rockville MD 20852
Attn: Dr. R. E. Cheney, N/CG11

National Ocean Data Center
1825 Connecticut Ave., NW
Universal Bldg. South, Rm. 206
Washington DC 20235

Naval Air Development Center
Warminster PA 18974-5000
Attn: Commander

Naval Air Systems Command HQ
Washington DC 20361-0001
Attn: Commander

Naval Civil Engineering Laboratory
Port Hueneme CA 93043
Attn: Commanding Officer

Naval Coastal Systems Center
Panama City FL 32407-5000
Attn: Commanding Officer

Naval Eastern Oceanography Center
McAdie Building, U117
Naval Air Station
Norfolk VA 23511
Attn: C. A. Weigand

Naval Facilities Engineering
Command HQ
200 Stovall St.
Alexandria VA 22332-2300
Attn: Commander

Naval Oceanographic Office
Stennis Space Center MS 39522-5001
Attn: Commanding Officer
        Library
        W. Austin, Code MTA
        L. J. Bernard, Code TD
        A. A. Johnson, Code MT

Naval Oceanography Command
Stennis Space Center MS 39529-5000
Attn: Commander
        Dr. D. Durham
        Program Integration Dept.
        Dr. P. F. Moersdorf
        Space Oceanography Programs

Naval Oceanographic & Atmospheric
Research Laboratory
Atmospheric Directorate
Monterey CA 93943-5006
Attn: Code 400
        Dr. P. Tag
        Dr. J. Hovermale

Naval Oceanographic & Atmospheric
Research Laboratory
Stennis Space Center MS 39529-5004
Attn: Code 100
        Code 105
        Code 115
        Code 125L (10)
        Code 125P
        Code 200
        Code 300
        Code 321, M. Lybanon (40)

Naval Ocean Systems Center
San Diego CA 92152-5000
Attn: Commander

Naval Postgraduate School
Monterey CA 93943
Attn: Superintendent
        Dr. Chin-Hwa Lee,
        Associate Professor,
        Dept. of Electrical & Computer
        Engineering

Naval Research Laboratory
Washington DC 20375
Attn: Library (2)
        R. P. Shumaker

Naval Sea Systems Command HQ
Washington DC 20362-5101
Attn: Commander

Naval Surface Warfare Center Det
Silver Spring
White Oak Laboratory
10901 New Hampshire Ave.
Silver Spring MD 20903-5000
Attn: Officer in Charge
        Library

Naval Surface Warfare Center
Dahlgren VA 22448-5000
Attn: Commander

Naval Underwater Systems Center
Newport RI 02841-5047
Attn: Commander

Naval Underwater Systems Center Det
New London Laboratory
New London CT 06320
Attn: Officer in Charge

Office of Naval Research
800 N. Quincy St.
Arlington VA 22217-5000
Attn: Code 10D/10P, Dr. E. Silva
        Code 112, Dr. E. Hartwig
        Code 12
        Code 10

Office of Naval Research
Detachment
Stennis Space Center MS 39529
Attn: E. D. Chaika

Office of Naval Research
ONR European Office
PSC 802 Box 39
FPO AE 09499-0700
Attn: Commanding Officer

Office of Naval Technology
800 N. Quincy St.
Arlington VA 22217-5000
Attn: Code 20, Dr. P. Selwyn
        Code 228, Dr. M. Briscoe
        Code 234, Dr. C. Votaw

Planning Systems, Inc.
PSI Science Center
115 Christian Lane
Slidell LA 70458
Attn: Dr. R. L. Crout,
        Senior Scientist

Planning Systems, Inc.
Naval Systems Division
7925 Westpark Drive
McLean VA 22102
Attn: Dr. E. Molinelli
        Director, Environmental
        Acoustic Group

Scripps Institution of Oceanography
University of California
291 Rosecrans St.
San Diego CA 92106-3505

Space & Naval Warfare Sys Com
Director of Navy Laboratories
SPAWAR 005
Washington DC 20363-5100
Attn: Commander

Space & Naval Warfare Sys Com
Crystal Park 5, Room 301
2451 Crystal Drive
Arlington VA 22202
Attn: CDR P. Ranelli, PMW-165
        CAPT J. Jensen

Tulane University
Dept. of Computer Science
New Orleans LA 70118
Attn: Dr. B. P. Buckles
        Dr. F. E. Petry

University of Miami
RSMAS
Miami FL 33149-1098
Attn: Dr. O. B. Brown
        Dr. D. B. Olson

University of North Carolina
at Greensboro
College of Arts and Sciences
Greensboro NC 27412-5001
Attn: Dr. S. Lea
        383 Bryan Building
        Dept. of Mathematics

University of Rhode Island
Graduate School of Oceanography
Narragansett RI 02882
Attn: Dr. P. Cornillon

University of Tennessee
Computer Science Department
Knoxville TN 37996-2100
Attn: Dr. M. G. Thomason
        Room 107, Ayres Hall
        Dr. M. M. Trivedi
        Electrical and Computer
        Engineering Dept.,
        Ferris Hall

Woods Hole Oceanographic Institution
P.O. Box 32
Woods Hole MA 02543
Attn: Director

REPORT DOCUMENTATION PAGE

1. Agency Use Only *(Leave blank).*	2. Report Date. October 1991	3. Report Type and Dates Covered. Final

4. Title and Subtitle.

Curve Fitting Using Genetic Algorithms

5. Funding Numbers.

Work Unit No.
93210T   13210X
Program Element No.
0603704N   0602435N
Project No.
00101   3587
Task No.
100   —
Accession No.
DN394464   DN256010

6. Author(s).

K. Messa* and M. Lybanon

7. Performing Organization Name(s) and Address(es).

Naval Oceanographic and Atmospheric Research Laboratory
Ocean Science Directorate
Stennis Space Center, Mississippi 39529-5004

8. Performing Organization Report Number.

NOARL Report 18

9. Sponsoring/Monitoring Agency Name(s) and Address(es).

Space and Naval Warfare Systems Command
Washington, D.C.

Naval Oceanographic and Atmospheric Research Laboratory
Ocean Science Directorate
Stennis Space Center, MS 39529-5004

10. Sponsoring/Monitoring Agency Report Number.

11. Supplementary Notes.

*Loyola University, Department of Mathematical Sciences, New Orleans, LA 70118

12a. Distribution/Availability Statement.

Approved for public release; distribution is unlimited. Naval Oceanographic and Atmospheric Research Laboratory, Stennis Space Center, Mississippi 39529-5004.

12b. Distribution Code.

13. Abstract *(Maximum 200 words).*

Genetic algorithms are search techniques based on the mechanics of natural selection. They have been used successfully in many applications because of their robustness and because of their ability to search in a noisy problem space. In particular, genetic algorithms are used in curve-fitting. The genetic algorithm selects the coefficients of a particular curve that most closely matches a given set of data.

Candidate solutions are vectors of real numbers that represent the coefficients of the curve to be modeled. Thus, every candidate solution corresponds to a new function. As such, each candidate solution is evaluated using the sum of the squares of the residuals. The evaluation of each of these curves with respect to its fit of the data guides the genetic algorithm toward the solution with the greatest merit.

Several examples of the application of genetic algorithms to curve-fitting problems are presented. Convergence to the optimal solution is rapid when knowledge of the coefficients is available. When little is known about the coefficients, a degree of experimentation helps obtain the optimal solution.

14. Subject Terms.

remote sensing, artificial intelligence, lagrangian drifter, microbubbles, satellite data processing, environment, information extraction

15. Number of Pages.
18

16. Price Code.

17. Security Classification of Report.	18. Security Classification of This Page.	19. Security Classification of Abstract.	20. Limitation of Abstract.
Unclassified	Unclassified	Unclassified	Same as report