

An Evolutionary Algorithm with Applications to Statistics
Author(s): Mary C. Meyer
Source: *Journal of Computational and Graphical Statistics*, Vol. 12, No. 2 (Jun., 2003), pp. 265-281
Published by: Taylor & Francis, Ltd. on behalf of the American Statistical Association, Institute of Mathematical Statistics, and Interface Foundation of America
Stable URL: https://www.jstor.org/stable/1391195
Accessed: 27-02-2019 20:21 UTC



# An Evolutionary Algorithm With Applications to Statistics

Mary C. MEYER

Evolutionary algorithms are used to find maxima of functions. Their claim to fame is an ability to find a global maximum in the presence of local maxima. The computations do not require derivatives or convexity, but still may be fairly computationally intensive in larger dimensions. This article presents a new type of evolutionary algorithm that works well in many dimensions, with the added advantage that linear equality constraints are implemented in a natural way. Bounds on the coordinates of the solution are also easy to implement. Several examples are presented and the new algorithm is compared with standard versions of evolutionary algorithms. The first group consists of functions in one or two dimensions, chosen to be difficult to maximize by gradient or simplex-based methods. The second set of examples are from statistics problems that are known to be computationally difficult. The first is least absolute deviations nonlinear regression with bootstrapped confidence bounds on the mean response, the second is smoothed nonparametric unimodal density estimation requiring both a linear equality constraint and linear inequality constraints. The Fortran subroutine is available for the user.

**Key Words:** Genetic algorithm; Global extremum; Stochastic optimization.

## 1. EVOLUTIONARY ALGORITHMS

Evolutionary algorithms are used to optimize functions by simulating natural evolution in a "population" of candidate solutions. Suppose we want to find $\mathbf{x} \in \mathcal{R}^n$ to maximize $f : \mathcal{R}^n \to \mathcal{R}$. We initialize a population $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, where each $\mathbf{x}_i$ is a point in $\mathcal{R}^n$, called an "individual" having "fitness" $f(\mathbf{x}_i)$. These individuals are allowed to reproduce and mutate, with a selection scheme that mimics the evolutionary concept of "survival of the fittest." If the less fit individuals are more likely to die, or less likely to reproduce, or both, then the tendency is for successive generations to have higher overall fitness. Mutation fosters exploration of the function domain. After "many" generations, the optimal fitness (the function maximum) may be reached.

---

Mary C. Meyer is Assistant Professor, Statistics Department, The University of Georgia, Athens, GA 30602 (E-mail: mmeyer@stat.uga.edu).



*Journal of Computational and Graphical Statistics, Volume 12, Number 2, Pages 265–281*
*DOI: 10.1198/1061860031699*

The genetic algorithm is a special case of an evolutionary algorithm. In the original form, Holland (1992) used "bits" to represent the genetic structure. Chatterjee, Laudato, and Lynch (1996) applied the genetic algorithm to several statistics problems. To illustrate the idea, consider the problem of maximizing the function $f(x) = \exp(-x^2)\sin(x)$ over an interval $(-20, 20)$, as shown in Figure 1(a). This function has many local maxima, with a global maximum at $x \approx 0.653$. The points in the interval are coded as binary numbers with $D$ digits, with $000\cdots000$ representing $-20$ and $111\cdots111$ representing $+20$. The best of the $2^D$ values of $x$ is found by initializing a population of $N$ individuals with genetic material represented by the digit strings. To initialize the population, the genetic bits are randomly set to zeros or ones. The fitness of the individual corresponds to the value of the function evaluated at the individual's coordinate, represented by the bits. Pairs of individuals reproduce by the "crossover" method of combining bit strings. A digit $d$ between 1 and $D$ is randomly chosen, and the new individual takes the first $d$ bits from one individual, and the second $D - d$ bits from the other. Individuals mutate by having a fixed probability that a digit will flip from zero to one or vice versa in every cycle. Finally, deaths occur in such a way that the less fit individuals are more likely to die than more fit individuals.

Although there are many forms, all evolutionary algorithms have elements in common. First, the population is randomly initialized. The representation of the individuals may be in bits to represent genes, as in the genetic algorithm, or as real-valued vectors, where the coordinates are considered to carry the "genetic information." Then we have in a loop: recombination, mutation, and selection. For real-valued vectors, many of these methods are listed systematically in Bäck, Fogel, and Michalewicz (2000). The "geometric," "crossover," and "heuristic crossover" recombination methods will be discussed here.

Suppose $\mathbf{x}_m = (x_{m1}, x_{m2}, \ldots, x_{mn})'$ and $\mathbf{x}_d = (x_{d1}, x_{d2}, \ldots, x_{dn})'$ are members of the population. They may recombine, or reproduce, to form $\mathbf{x}_b$. For geometric recombination, $\mathbf{x}_b = (\sqrt{x_{m1}x_{d1}}, \sqrt{x_{m2}x_{d2}}, \ldots, \sqrt{x_{mn}x_{dn}})'$. For the crossover method, a digit $n_0$ is randomly chosen between 1 and $n$, and the new individual takes the first $n_0$ coordinates of one individual, and the second $n - n_0$ coordinates from the other. For the heuristic crossover method, a uniform random number $u$ is generated, and $\mathbf{x}_b = u(\mathbf{x}_m - \mathbf{x}_d) + \mathbf{x}_m$ if $\mathbf{x}_m$ is more fit than $\mathbf{x}_d$, and $\mathbf{x}_b = u(\mathbf{x}_d - \mathbf{x}_m) + \mathbf{x}_d$ otherwise.

Mutation for real-valued vectors is often accomplished simply by adding a random number to one of the coordinates of an individual. Usually this random number has mean zero and is a from a familiar distribution such as normal or Cauchy.

Selection methods can be deterministic or stochastic. A straight-forward deterministic method eliminates the least fit in each generation to make room for the next generation; this method requires sorting on the fitness variable. The "tournament" selection method requires randomly selecting pairs of individuals, and eliminating the less fit of the two, or having a higher probability of the more fit individual surviving. The "proportionate" method assigns probability proportional to $f_i / \sum f_i$ of surviving to the $i$th individual, where $f_i$ is the fitness of this individual. Neither the tournament nor the proportionate method requires sorting.

These ideas for recombination, mutation, and selection can be "mixed and matched" according to the problem at hand, or new ideas may be tried. It seems likely that many programmers will implement their own ideas for at least one of the elements. The author's

idea is described in the Section 2. The greatest improvement over other algorithms is that linear constraints are easily implemented, as well as bounds on the coordinates, and the speed of implementation should be favorable or at least comparable to other evolutionary methods.

The new algorithm is compared with several others in the third section, for some functions in one and two dimensions chosen so that the global maximum is difficult to find, as well as some statistics applications. The discussion in the last section concerns some convergence issues and user-defined parameter settings for the new algorithm.

## 2. A NEW EVOLUTIONARY ALGORITHM USING LINEAR CONSTRAINTS

For the algorithm introduced in this article, the recombination and mutation steps are formulated so that the objective function may be maximized under linear equality constraints, in a very natural way. The computational expense of each step is comparable to other methods so that the constraint capabilities do not detract from the speed of execution. In fact, since the recombination step entails a good deal of exploration of the function domain, for many problems the mutation step is minimized or even unnecessary for convergence in a reasonable amount of time. Both recombination and mutation use the Cauchy distribution, chosen because the heavy tails facilitate exploration of the parameter space and discourage premature convergence to a local maximum. Furthermore, the cumulative distribution function for this density is invertible, so that random deviates from the Cauchy density and the truncated Cauchy density are simple to calculate from uniform random deviates.

**Notation and Parameters:** Let $N$ be the number of individuals in the population, let $n$ be the number of dimensions in the search set, and let $m$ be the number of mutations per cycle. Choices of these parameters will be discussed in the examples section, but typically $N$ is a few hundred.

**Initialization and Evaluation:** The algorithm starts by randomly generating $N$ points (individuals) in the search set of the function domain. If the set in $\mathcal{R}^n$ over which to maximize is bounded, the initial points may be uniformly generated over this set. If unbounded, an initial guess is provided, and for each initial individual, random Cauchy deviates are added to each of the coordinates, until $N$ points have been generated. If there are linear constraints, the initial individuals must begin on the affine space defined by the constraints. For instance, suppose the constraints are of the form $\mathbf{Ax} = \mathbf{b}$ for full row-rank $m \times n$ matrix $\mathbf{A}$, and $\mathbf{x}_0$ is a solution to the matrix equation. If the columns of the $n \times m$ matrix $\mathbf{Z}$ form a basis for the nullspace of $\mathbf{A}$, then any $\mathbf{x} = \mathbf{x}_0 + \mathbf{Zy}$ is in the desired affine space. If $\mathbf{x}_0$ is an initial guess, the initial population may be generated using Cauchy random deviates for the coordinates of $\mathbf{y} \in \mathcal{R}^m$.

The algorithm allows for optional bounds on the search space. If some of the coordinates are bounded, appropriately truncated Cauchy deviates are generated. The truncation is sim-

ple to code because the cumulative distribution function of the Cauchy density is invertible. The fitness of each individual is the value of the function evaluated at its coordinates.

**Selection:** A version of the tournament selection scheme is implemented in such a way as to eliminate exactly half of the population in each cycle. The fitness of the $i$th individual is compared to that of individual $i + N/2$, for $i = 1, 2, \ldots, N/2$, and the $i$th individual is replaced by the more fit of the two. Then the second half of the array is open for new members of the population created through recombination.

**Recombination:** Two parents are randomly selected from the population remaining after selection. According to the new scheme, the "babies" are randomly generated on the line connecting the parents, according to a Cauchy distribution. If the parents are the points $\mathbf{x}_m$ (mom) and $\mathbf{x}_d$ (dad), then the $i$th coordinate of the baby $\mathbf{x}_b$ is

$$x_{bi} = \frac{x_{mi} + x_{di}}{2} + C\frac{x_{mi} - x_{di}}{2}, \quad i = 1, \ldots, n,$$

where $C$ is a standard Cauchy random variable. Note that if $C = 0$, the baby is exactly between the parents, if $C = 1$, the baby has the same coordinates as the mother, and if $C = -1$, the baby's coordinates are identical to the father's. A simple integration shows that the baby's coordinates are between the parents' with probability 0.5. If there are bounds on the parameters, an appropriately truncated Cauchy random variate is generated. Since the cumulative distribution function for the Cauchy density is invertible, this is easily done. The Cauchy density was chosen to facilitate exploration of the parameter space and to discourage premature convergence to a local maximum.

**Mutation:** The mutation scheme is chosen so as to effectively explore the function domain and discourage premature convergence to a local maximum. Although the recombination scheme has a good bit of exploration built in, the mutation adds more randomness to the direction of the search, while staying in the affine space defined by the linear constraints. To this end, four members of the population are randomly chosen for the mutation method. The first two are compared and the less fit individual will be replaced by a mutated version of the more fit individual. The second two members will simply determine the direction of exploration. Let $\mathbf{x}_L$ be the less fit of the first two members, let $\mathbf{x}_U$ be the more fit, and let $\mathbf{x}_1$ and $\mathbf{x}_2$ be the other (distinct) randomly chosen members. Let $d = \sqrt{\sum_{i=1}^{n}(x_{2i} - x_{1i})^2}$, the distance between $\mathbf{x}_1$ and $\mathbf{x}_2$. Then the new member replaces $\mathbf{x}_L$ and is:

$$x_{Li} = x_{Ui} + \frac{C(x_{2i} - x_{1i})}{d}.$$

**Immigration:** For very large dimensional problems that require many generations for convergence, there is a danger of the set of solutions (current population) collapsing into a smaller dimensional set. The immigration method is meant to prevent this, by introducing a small number of new members to the population in each generation, using the initialization method. For the examples in this article, immigration is used only in the largest ($n = 50$).

The methods for recombination and mutation allow for linear equality constraints. All subsequent generations will inhabit the same affine space as the initial population and

immigrants; the babies and "mutants" will inherit the constraints from their parents (or progenitors). Of the other methods discussed in Section 1, the heuristic crossover recombination will preserve equality constraints, but the mutation methods do not.

## 3. EXAMPLES

This section has four parts. The first part presents examples of objective functions in one dimension, chosen to be difficult to maximize because of local maxima. The second part presents difficult examples in two dimensions. A robust regression example is given third part, and the fourth part presents an example in smooth density estimation that requires linear equality and inequality constraints.

The performance of three "standard" algorithms is compared with the new algorithm. The first (GA) is a genetic algorithm with the coordinate information encoded into bits as described in Chatterjee et al. (1996), and the other two (E1 and E2) are evolutionary algorithms using real-valued vectors. E1 uses geometric recombination Cauchy mutation, and deterministic selection; E2 uses heuristic crossover recombination, Cauchy mutation, and tournament selection. Cauchy mutation was chosen as the best of the options for the type of examples presented, which require a great deal of exploration of the sample space. The proportionate method of selection was found to be inadequate for the more difficult examples, since the most fit of the population might not be preserved over the generations.

All algorithms were coded by the author in Fortran 90, and run on a Sun Enterprise 4000 with 3 Ultrasparc processors and 1GB of RAM. The pseudo random number generator used for uniform random variables was taken from Press, Teukolsky, Vetterling, and Flannery (1992).

### 3.1 PART ONE: EXAMPLES IN ONE DIMENSION

The first examples are functions in one variable that are constructed to be difficult to maximize with non-evolutionary optimization schemes such as gradient or simplex-based methods. The first plot in Figure 1 shows a "hilly" function with many local maxima; the second shows two widely separated maxima with a flat spot between.

$$1(\text{a}) \qquad f(x) = \exp(-x^2)\sin(x),$$

$$1(\text{b}) \qquad f(x) = 0.9\exp(-(x-4)^2) + \exp(-(x-20)^2)$$

The genetic algorithm requires the user to define bounds for the search space. The bounds used here are $(-20, 20)$ for Function 1(a), and $(-10, 30)$ for Function 1(b), with $N = 300$, $m = 10$, and $D = 20$. For both functions, GA converges very quickly (usually within 20 generations), largely because in one dimension, it is very likely that one of the individuals in the initial population is very close to the global maximum, so that the evolution part of the algorithm does not have much to accomplish. Of course, the user must set the bounds to capture the global maximum if it is to be found.

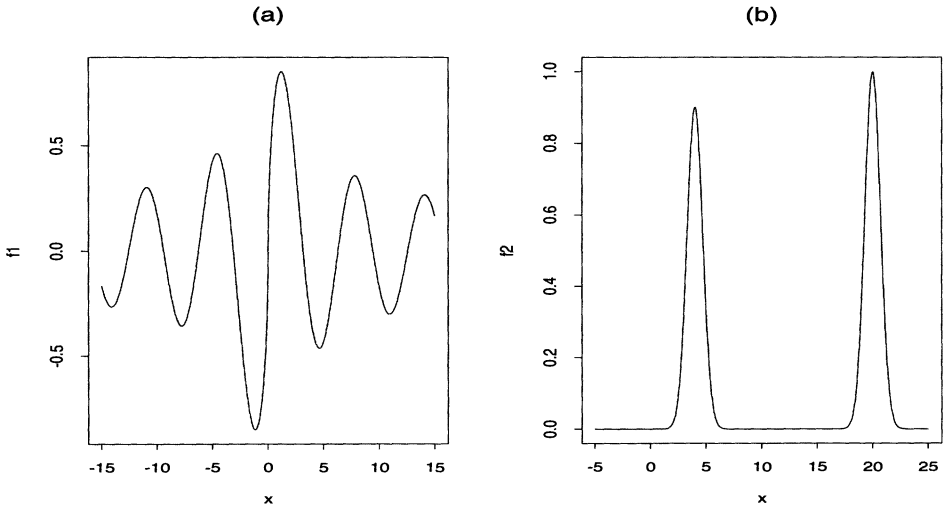
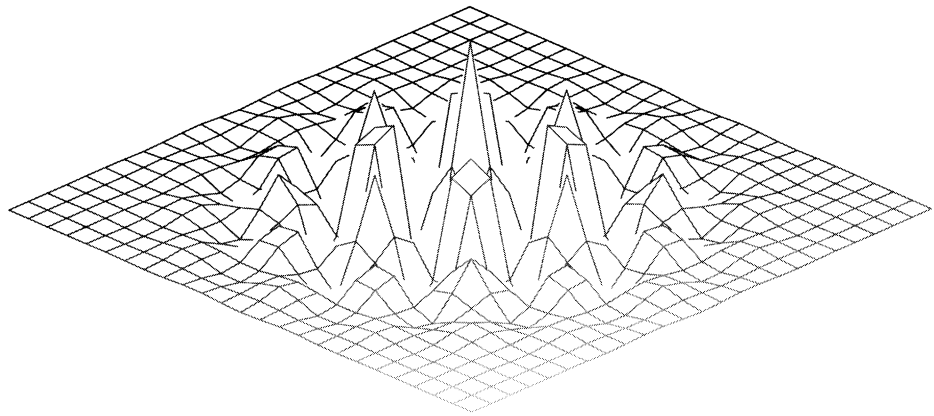



*Figure 1. Examples of one-dimensional objective functions.*

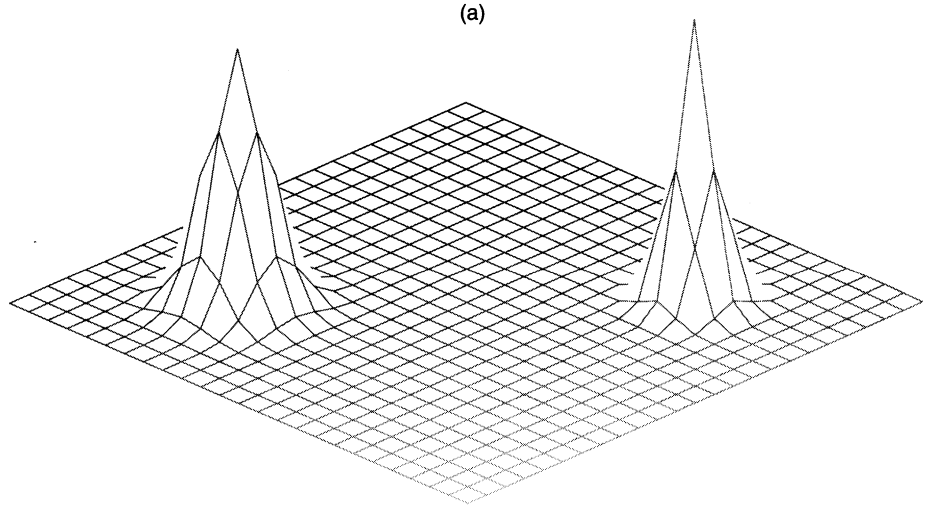
None of the three evolutionary algorithms has a bounded search space for these examples. The initial guess for Function 1(a) is set at $-20$, and at $-4$ for Function 1(b). The E1 algorithm works well for Function 1(a) (the $x$-coordinates must be shifted to the positive half-line in order to employ geometric recombination), but for Function 1(b), the geometric recombination is inadequate for exploring the parameter space. Using $N = 400$ and $m = 130$, the algorithm fails to find the global maximum in more than 60% of the runs. The heuristic crossover recombination of E2 has no problem with 1(a) and also does well with 1(b). For both functions, $N = 400$, $m = 130$, and convergence is always well within a few hundred generations.

The new algorithm is also successful in finding the global maximum for both one-dimensional examples. For Function 1(a), the population size was set to $N = 200$ with no mutations ($m = 0$). The Cauchy recombination sufficiently explores the search space without need for the extra searching capacity of the mutation routine, because all the "action" is taking place in a bounded area. For Function 1(b), mutation is needed to effectively explore the space and find the global maximum. For the new algorithm, we use $N = 400$ and $m = 130$ to match the choices for E2.

Both the new algorithm and E2 were surprisingly successful with a more extreme version Function 1(b), with the global maximum moved farther to the right. If the second peak is moved all the way out to 130, with the local maximum remaining at 4, with initial guess at $-5$, both algorithms successfully find the global in about 75% of runs of 500 generations. (These runs differ only by the seed of the random number generator.) Furthermore, the 1,000 runs of 500 generations took less than half an hour of real time, showing that in one dimension, showing that these evolutionary algorithms are quick as well as efficient in finding the global maximum.



(a)



(b)

*Figure 2. Examples of two-dimensional objective functions. Example 2(b) is shown for $c = 10$.*

## 3.2 PART TWO: FUNCTIONS IN TWO DIMENSIONS

To illustrate the performance of the algorithms in a two dimensional search space, consider the functions

$$2(\text{a}) \qquad f(x_1, x_2) = \exp(-(x_1^2 + x_2^2)/50)\cos(x_1)\cos(x_2),$$

$$2(\text{b}) \qquad f(x_1, x_2) = 0.9 \times \exp(-x_1^2/2 - x_2^2/2) + \exp(-(x_1 - c)^2 - (x_2 - c)^2/2)$$

as shown in Figure 2. For Function 2(a), the true maximum is at the origin, but there are many local maxima nearby. Function 2(b) has two widely separated maxima, with a local maximum at $(0, 0)$, and the global maximum at $(c, c)$; in the figure, $c = 10$.

For the genetic algorithm, we increase the population size to $N = 800$ and the mutations to $m = 20$. If the bounds are $(-20, 20)$ for both dimensions (the range shown in Figure 2(a)), then the algorithm successfully finds the global maximum in all of 100 runs.

Bounds are not used in any of the three evolutionary algorithms. For Example 2(a), the initial guess is at the point $(-20, -20)$, which is "on the outskirts," where the function is rather flat. The new algorithm is consistently successful in finding the global maximum when $N = 200$, and no mutation is necessary. This is again because all of the "action" is taking place in a group of "mountains," and the Cauchy recombination is sufficient to explore the function domain. For these parameters and starting values, the new algorithm was seeded 1,000 times; it finds the global maximum to four decimal places in fewer than 100 generations in 976 of these runs; in all but six runs the global maximum is found within 500 generations. The E2 algorithm is also consistently successful with $N = 200$ and no mutation, with the global maximum found to four decimals in fewer than 100 generations, in 980 out of 1,000 runs.

Algorithm E1 is also consistently successful in finding the global maximum at $N = 200$ and $m = 20$, but when the initial guess is $(3, 3)$ instead of $(-20, -20)$, this version tends to get stuck at the local maximum at about $(3.02, 3.02)$. Increasing the number of mutations to $m = 80$ gives about a 98% success rate in 1000 runs from either starting value.

The dependence of the speed of evolution of the coordinates of Function 2(a) on population size is depicted in Figure 3 for the new algorithm. Convergence to the global maximum typically happens well within the allowed 100 generations when the population size is as large as $N = 200$, but is noticeably slower for smaller populations. For the runs shown, no mutation is used.

Finding the global maximum of Function 2(b) is a more difficult task for all four algorithms. Using GA with $N = 800$ and $m = 20$, setting the bounds to be $(-50, 50)$ in both dimensions, and allowing 500 generations for convergence, the algorithm gets stuck at the local maximum in one or two percent of the runs, and gets within four decimal places of the global maximum in about 40% of the runs. In most of the runs with GA, the algorithm terminates after 500 generations within one or two decimal places of the global maximum in both coordinates, but is not very precise. A larger population size ($N$ about 4,000) is necessary for consistently better precision within 500 generations.

To test the evolutionary algorithms, bounds are not used, and we start at $(-5, -5)$, which is closer to the local maximum at $(0, 0)$ than to the global maximum at $(c, c)$. The geometric recombination of E1 is not efficient for exploration. If $c = 6$, $N = 800$ and $m = 200$, the local maximum is found in 97% of 1,000 runs of E1 at 500 generations each, and the global maximum is found in only 3%. The recombination methods of the new algorithm and E2 are more efficient, but unlike the Function 2(a) simulations, rather a lot of mutation is necessary to ensure converge to the global maximum. If the population size is $N = 600$, with $m = 200$, the new algorithm finds the true maximum within 500 generations, when $c = 6$, for every one of 1,000 attempts. The E2 algorithm is less successful, finding the global maximum in 80% of attempts at $N = 600$, $m = 200$, and 500 generations. A comparison of the performance of E2 and the new algorithm, for different settings of $c$, is shown in Table 1.

The plots in Figure 4 show the distribution of the coordinates over the generations

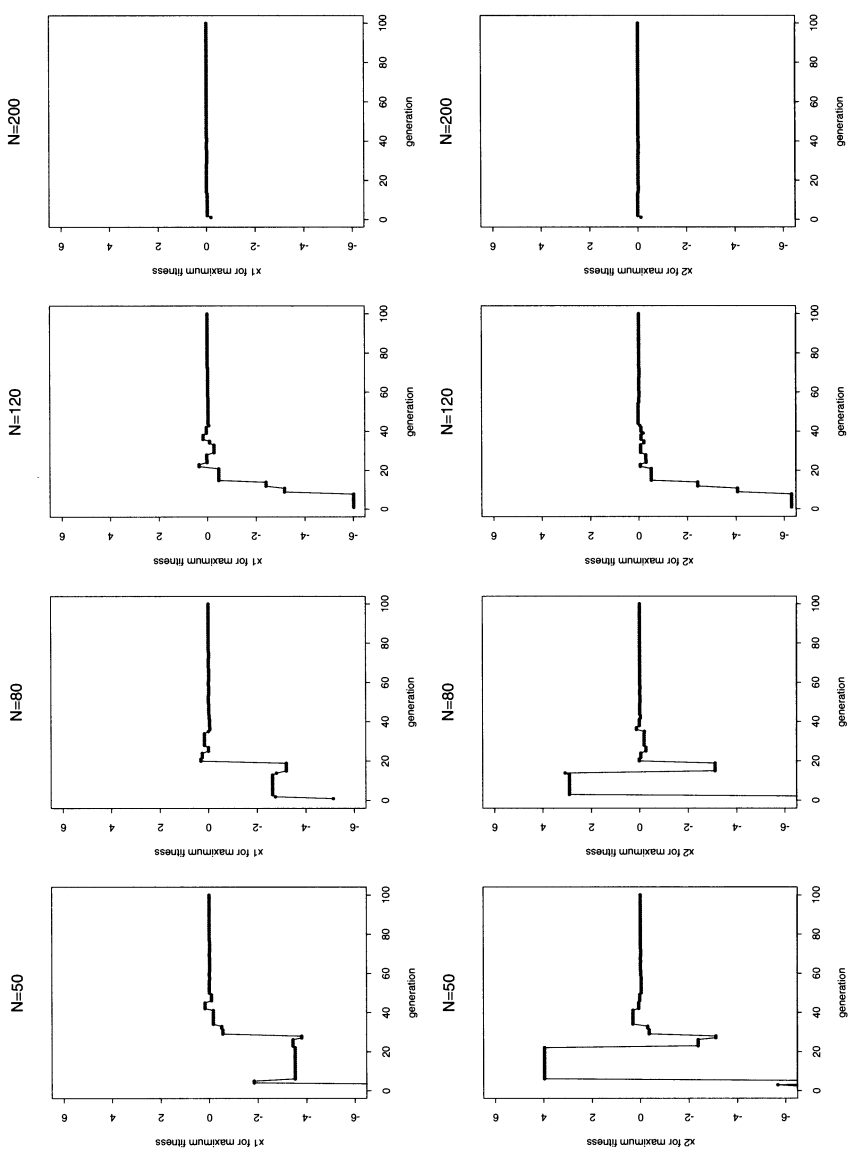



*Figure 3. Comparing convergence to the global maximum for typical runs of the new algorithm and Function 2(a), for varying population size. The first row depicts the evolution of the first coordinate of the individual of maximum fitness, the second row depicts the evolution of the second coordinate for this example.*

Table 1. Numbers of Successful Runs Out of 1000 Starts, for Algorithm E2 and the New Algorithm, for Objective Function 2(b). Both algorithms use $N = 600$, $m = 200$, and 500 generations.

| | *Number of successes out of 1,000* | |
|---|---|---|
| *Value of c* | *Algorithm E2* | *New algorithm* |
| 6 | 800 | 1000 |
| 10 | 94 | 847 |
| 15 | 9 | 424 |

for the function shown in Figure 2(b). For this run, $N = 300$ and $m = 100$. We see that the original population is clustered around the initial guess, but within a few cycles the local maximum at (0,0) is found. The Cauchy mutations and recombinations find the global maximum and the population clusters there after a few hundred generations.

### 3.3 Part Three: Robust Regression

Robust regression is chosen as the first example in statistics. The problem is to fit a Gompertz model for sigmoidal growth, as discussed in the book by Lawrence and Arthur (1990, chap. 3). The model is

$$Y_i = \alpha \exp[-\exp(\beta - \gamma x_i)] + \epsilon_i,$$

for $i = 1, \ldots, 15$; where the $\alpha$, $\beta$, and $\gamma$ are three parameters to fit, and the errors are assumed to be independent double-exponential. The data (given in the book) are depicted in the scatterplot of Figure 5.

The maximum likelihood estimator for the parameters minimizes the sum of absolute deviations, so that the function to maximize over the three parameters is

$$-\sum_{i=1}^{15} |y_i - \alpha \exp[-\exp(\beta - \gamma x_i)]|,$$

where the parameters are constrained to be nonnegative.

The fit is shown in Figure 5. The best fit parameters are known to be $\hat{\alpha} = 22.37$, $\hat{\beta} = 2.14$, and $\hat{\gamma} = 0.395$. The discussion by Lawrence and Arthur (1990) concerns the starting points for the optimization algorithms. "Because a nonlinear regression problem is essentially a nonlinear programming problem, the sensitivity of the final solution to the initial estimates must be considered." We show here that the new evolutionary algorithm finds the best fit parameters regardless of how bad the starting point is. Several starting points were chosen: (1, 1, 1), (10, 10, 10), (1, 20, 10). The algorithm was run 1,000 times for each starting point (with a different seed for the random number generator), with $N = 200$, $m = 0$. Convergence to the correct solution was obtained within 500 cycles for every implemenation. Furthermore, the 1,000 implementations of 500 cycles took less than 20 minutes of real time on a Sun Ultra 5 workstation.

Because the convergence is very quick and reliable, confidence bounds on the response $y$ can be obtained by bootstrapping. A bootstrap sample of size $n$ is obtained by sampling the

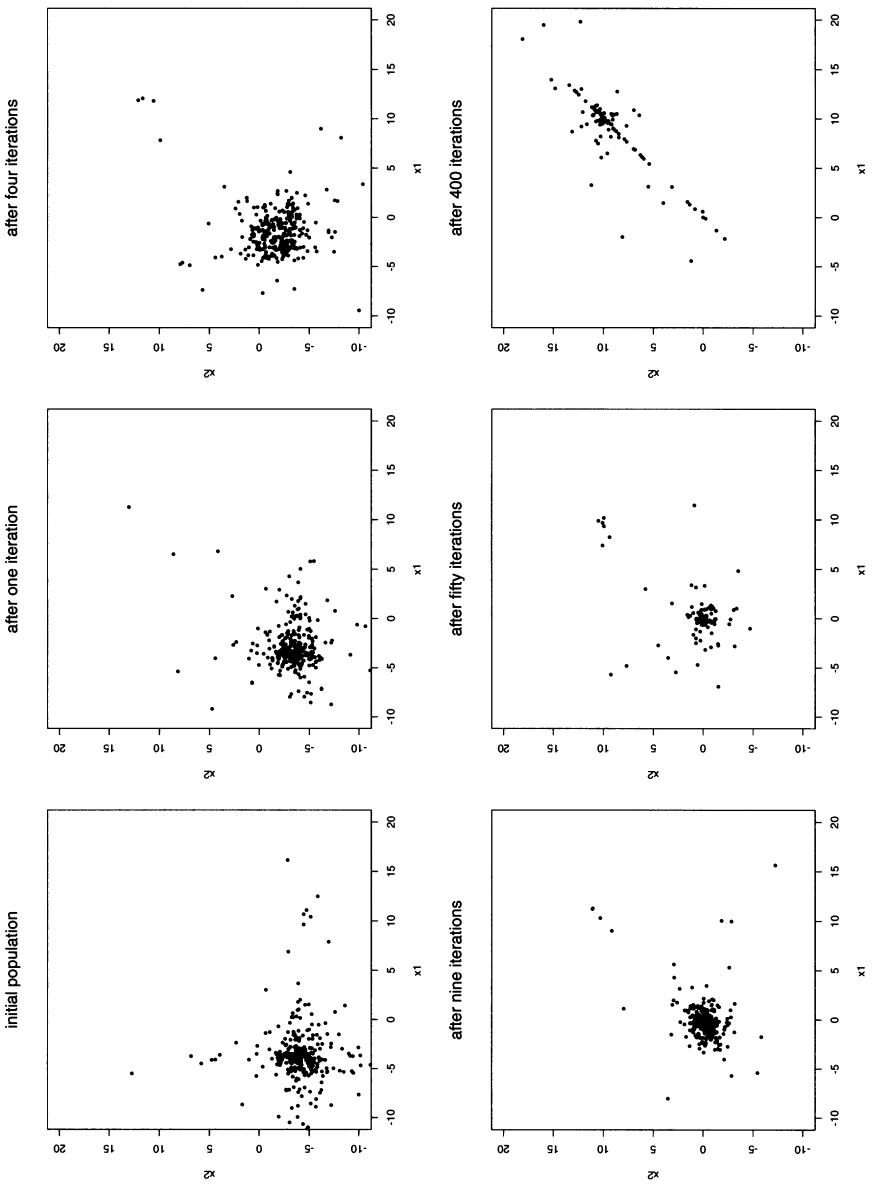



*Figure 4. Depiction of population evolving over generations, using the new algorithm for Function 2(b).*

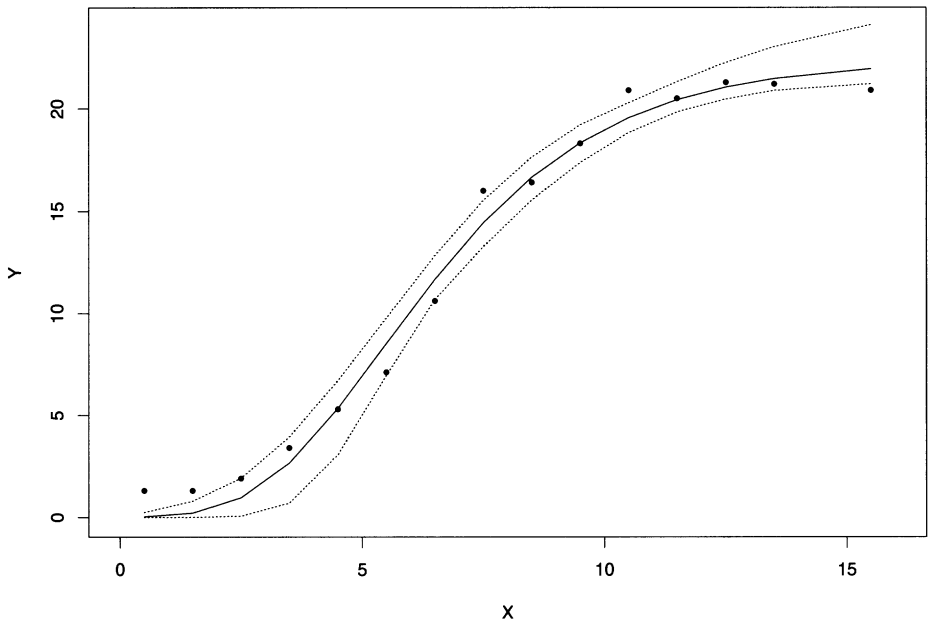



*Figure 5. BEANS dataset from Lawrence and Arthur (1990); best fit Gompertz model using least absolute deviations regression (solid line) and bootstrapped confidence bounds (dotted lines).*

data points with replacement. If 1,000 such samples are obtained, the parameter estimates and corresponding predicted responses can be found for each sample. A 95% bootstrap confidence interval for the mean response at each design point is given by the 2.5 and 97.5 percentiles of the 1,000 predicted responses. These are shown as dotted lines in Figure 5. We see that perhaps the model does not fit the data well at the smaller values of the predictor variable.

The genetic algorithm was implemented for this problem, with bounds of (15,25), (0,5), and (0,1) for the parameters. These are reasonable bounds given the scatterplot and the interpretation of the parameters. Though the population size was large ($N = 3{,}000$) with 2,400 mutations per cycle, and 500 cycles, with 20 digits in the bit strings, the convergence was not nearly as good as the new evolutionary algorithm, and the execution was slower. Figure 6 shows the histograms for 1,000 implementations of the algorithm (with different seeds for the random number generator each time). These results are similar to those given in a regression problem with three parameters in the Chatterjee et al. (1996) article.

The performance of E2 equaled that of the new algorithm, but E1 could not find the global maximum consistently with the any precision, even with larger population and more mutations.

### 3.4 PART FOUR: DENSITY ESTIMATION

Smooth unimodal density estimation is chosen as the second example in statistics. Let

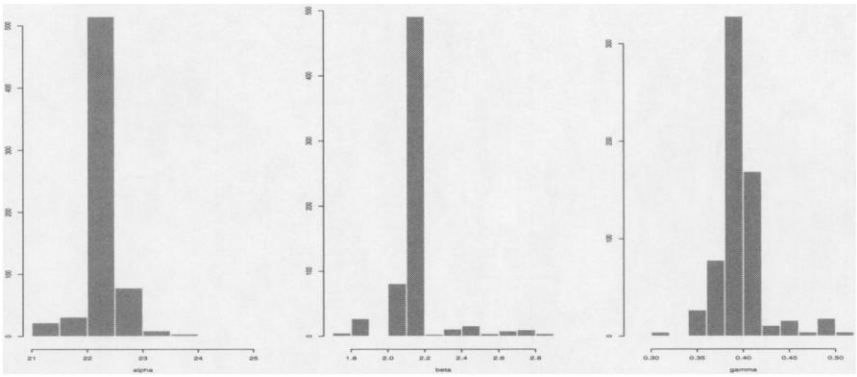


*Figure 6. Parameter estimates using genetic algorithm for the robust regression example.*

$x_1 < x_2 < \cdots < x_n$ be a (sorted) random sample from a density from the class of unimodal densities, and suppose the mode is known to be at the origin. The maximum likelihood linear spline density estimate (see Bickel and Fan 1996) is obtained by maximizing $\prod_{i=1}^{n} f(x_i)$, subject to the constraints that $f$ is piecewise linear with knots at the data, that $f$ is unimodal with mode at the origin, that $f$ is nonegative, and that the area under the curve is unity. Bickel and Fan (1996) showed that the solution is provided by a generalized isotonic regression. However, this solution is not very smooth, and tends to spike at the mode. Smoothing can be added by penalizing the log-likelihood. If big changes in slope are to be discouraged, the function to maximize may be defined as

$$\sum_{i=1}^{n} \log(f(x_i)) - \alpha \sum_{i=2}^{n-1} \left| \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} - \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \right|,$$

where $\alpha$ is the smoothing parameter. Larger values of $\alpha$ result in a larger penalty and smoother fits. However, the solution can no longer be obtained using generalized isotonic regression.

The estimate may be represented as a vector $\boldsymbol{\theta}$, where $\theta_i = f(x_i)$, $i = 1, \ldots, n$. To put the problem in a form amenable to the new evolutionary algorithm, note that the linear spline estimator $\boldsymbol{\theta}$ can be represented by a linear combination of $n$ vectors with nonnegative coefficients. Define vectors $\boldsymbol{\delta}^{(1)}, \ldots, \boldsymbol{\delta}^{(n)}$ where the values of the vectors are given by the rows of the following matrix (where here $n = 8$ and the mode is between the fourth and fifth observations):

$$\Delta = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

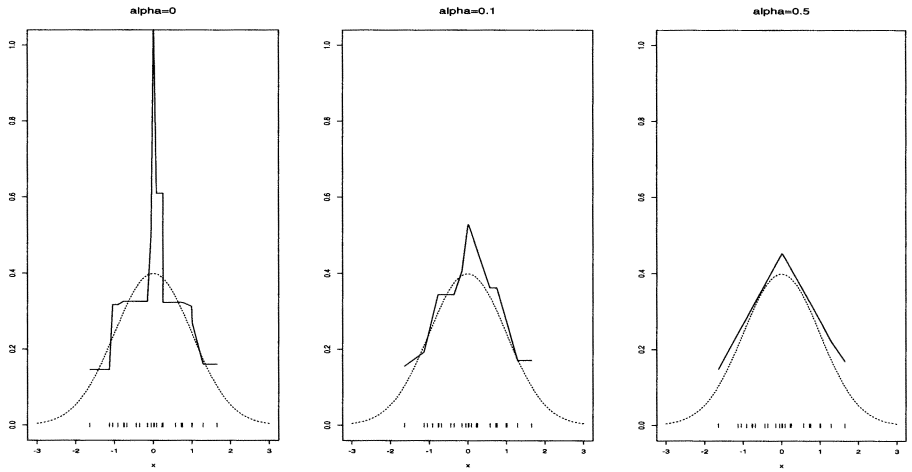



*Figure 7. Penalized maximum likelihood density estimates (solid lines) for a random sample from a normal density. The true density is indicated by dashed lines, and the tick marks indicate the sample.*

The polyhedral convex cone defined by the nonnegative linear combinations of the rows define the set of linear spline functions unimodal over $(x_1, x_n)$, so that if

$$\boldsymbol{\theta} = \sum_{j=1}^{n} b_j \boldsymbol{\delta}^{(j)},$$

the problem is to determine the $b_j \geq 0, j = 1, \ldots, n$ which result in a maximum penalized likelihood subject to the constraints. Upper bounds $u_j$ for each of the $b_j$ can be obtained from the inequality constraining the area under $b_j \boldsymbol{\delta}^{(j)}$ to be at most unity for each $j$. That is, $u_j = 1/\sum_{i=1}^{n} b_j \delta_i^{(j)}$.

The area constraint is

$$\sum_{i=1}^{n} c_i \theta_i = 1, \tag{3.1}$$

where $c_i = x_{i+1} - x_{i-1}$ for $i = 1, \ldots, n$, with $x_0 = x_1$ and $x_{n+1} = x_n$. This becomes $\sum_{i=1}^{n} b_j a_j = 1$, where $a_j = \sum_{i=1}^{n} c_i \delta_i^{(j)}$.

The new evolutionary algorithm is initialized to search for the optimum $b_j$ coefficients by first randomly generating $N$ points in the box determined by $0 \leq b_j \leq u_j$, $j = 1, \ldots, n$, and subsequently projecting these points onto the hyperplane defined by the area contraint (3.1). All subsequent generations and mutations will automatically lie on this plane.

This is a constrained optimization of a function in $n$ dimensions. The new algorithm can not be compared with the others, since those are not capable of constrained optimization. A comparison of estimates with different smoothing parameters is shown in Figure 7, for a sample of size $n = 25$ taken from a standard normal density. The dotted line in the plots represents the actual density, and the observations are marked as "ticks" at the bottom of

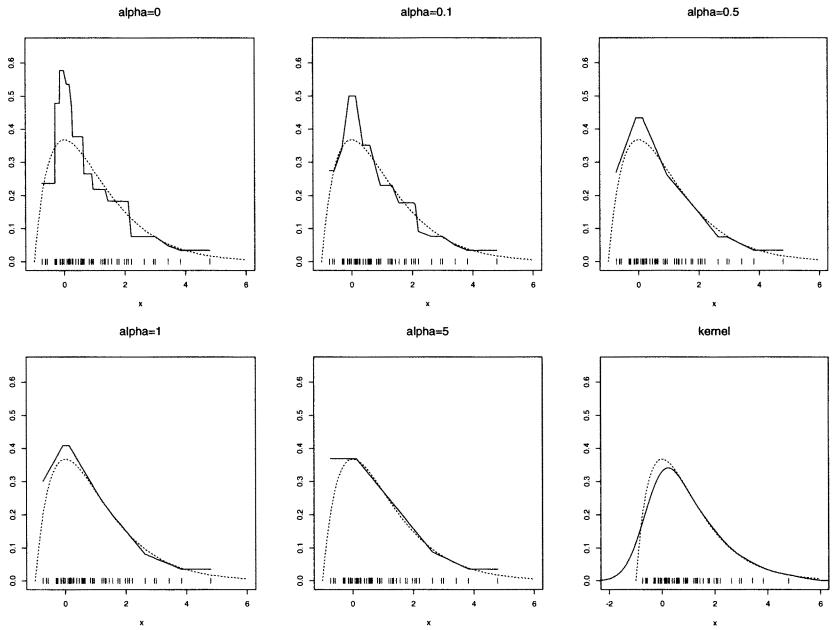



*Figure 8. Penalized maximum likelihood density estimates (solid lines) for a random sample from a gamma density. The true density is indicated by dashed lines, and the tick marks indicate the sample. The last plot shows the kernel estimator for comparison.*

the plot. We see that the unsmoothed estimate is very rough with a spike at the mode, and the smoothed estimators are closer to the target density. The parameters chosen here are $N = 2{,}000$ and $m = 100$. The algorithm converges within 4,000 generations and takes between one and two hours of real time. The convergence was checked by rerunning the algorithm several times with different starting points. The resulting density estimates were identical to five decimal places.

Because the convergence requires so many generations, there is a danger that the population starts to stray from the affine space, due to the limits of the machine precision. This is avoided by readjusting the population to the affine space every 500–1,000 generations, which does not appreciably slow the algorithm.

When the number of dimensions is doubled to $n = 50$, we are "pushing the limits" of the algorithm but it is still successful. We have a random sample from a gamma distribution with shape parameter 2. For this example, some immigration is employed to speed convergence. When $N = 10{,}000$, $m = 3{,}000$, and 100 immigrants per cycle, the solution is reproducible to five decimal places from different starting points, within 4,000 generations, but the algorithm is run overnight. Figure 8 shows the results of the density estimation for several smoothing parameters, and compares these to the standard kernel estimator.

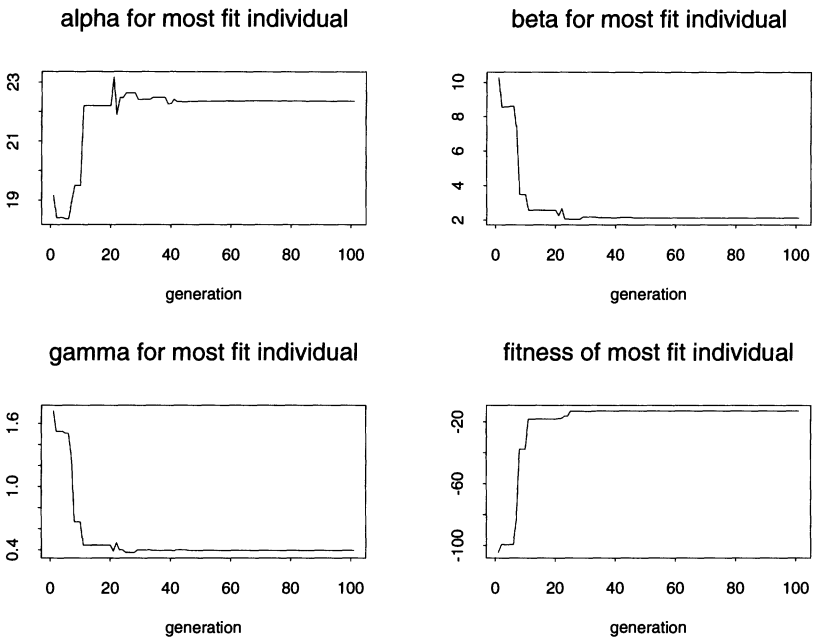



*Figure 9. Graphical assessment of convergence for robust regression example.*

# 4. DISCUSSION

The new algorithm has been compared to a genetic algorithm and two versions of evolutionary algorithms for several examples. The speed of the new algorithm is as good or better than the others, depending on the problem. Example 2(b) (p. 271) provides some evidence that the exploration capability provided by the Cauchy recombination and mutation is as good or better than existing methods. However, the real benefit to the new algorithm is its ability to handle linear constraints, as demonstrated in the density estimation example. The algorithm is coded in Fortran; a subroutine with documenation is provided at the Web site http://www.stat.uga.edu/~mmeyer/absgen.html.

## 4.1 CONVERGENCE TO THE GLOBAL OPTIMUM

Since the algorithm is stochastic, it cannot be guaranteed that it will converge with probability one. Consider the one-dimensional example represented in Figure 1(b) (p. 270). There is a finite probability that the algorithm will terminate at $x = 4$ rather than the true optimum at $x = 20$, since there are scenarios in which the exploration never happens to land near enough to $x = 20$ to pull the generations in that direction. This probability seems to be very small for the parameters chosen, but gets larger with longer flat spots between the maxima.

There are two types of lack of convergence. First, it may be that the algorithm was

terminated after too few generations; this is easy to fix by running the algorithm longer. Plots of the coordinates of the most fit individual over the generations will help assess this lack of convergence. For example, Figure 9 shows a convergence plot for the robust regression example. The values of the coordinates of the most fit individual in every generation are shown for each coordinate, along with the value of the objective function over the generations. We see that although the starting values are not very near the optimal values, convergence is rather fast, with the final values (within several decimals places accuracy) are reached by the 50th generation.

The second possible problem is convergence to a local maximum. In this case, the plots will show that the coordinates of the most fit individual have stopped changing significantly over the generations, but the population is stuck in a local maximum or flat spot. The user may assess this type of lack of convergence by restarting the algorithm with different starting values and random number seeds, and comparing the resulting solutions. If these are farther apart than a given tolerance, this indicates a problem with convergence. A larger population size and/or more mutation should be tried. For large dimensional problems some immigration may help with both types of lack of convergence.

*[Received February 2001. Revised December 2001.]*

## REFERENCES

Bäck, T., Fogel, D. B., and Michalewicz, Z. (2000), *Evolutionary Computation 1: Basic Algorithms and Operators*, Philadelphia, PA: Institute of Physics Publishing.

Bickel, P., and Fan, J. (1996), "Some Problems of the Estimation of Unimodal Densities," *Statistica Sinica*, 6, 23–45.

Chatterjee, S., Laudato, M., and Lynch, L. A. (1996), "Genetic Algorithms and Their Statistical Applications; An Introduction, *Computational Statistics and Data Analysis*, 22, 633–651.

Holland, J. H. (1992), "Genetic Algorithms," *Scientific American*, July, 66–72.

Lawrence, K. D., and Arthur, J. L. (1990), *Robust Regression*, New York: Marcel Dekker, Inc.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992), *Numerical Recipes in Fortran 77, Second Edition: The Art of Scientific Computing*, Cambridge: Cambridge University Press.