Genetic Optimization Using Derivatives*

by

Jasjeet S. Sekhon[†]

and

Walter R. Mebane, Jr.[‡]

Draft: January 6, 2000

* *Political Analysis*, 1998.

[†] Assistant Professor, Department of Government, Harvard University, jsekhon@fas.harvard.edu, HTTP://data.fas.harvard.edu/jsekhon/. Jasjeet Sekhon's research is supported in part by the Social Sciences and Humanities Research Council of Canada grant number 752-95-0380.

[‡] Associate Professor, Department of Government, Cornell University, wrm1@cornell.edu, HTTP://macht.arts.cornell.edu/wrm1/.

Abstract

Genetic Optimization Using Derivatives

We describe a new computer program that combines evolutionary algorithm methods with a derivative-based, quasi-Newton method to solve difficult unconstrained optimization problems. The program, called GENOUD (GENetic Optimization Using Derivatives), effectively solves problems that are nonlinear or perhaps even discontinuous in the parameters of the function to be optimized. When a statistical model's estimating function (for example, a log-likelihood) is nonlinear in the model's parameters, the function to be optimized will usually not be globally concave and may contain irregularities such as saddlepoints or discontinuous jumps. Optimization methods that rely on derivatives of the objective function may be unable to find any optimum at all. Or multiple local optima may exist, so that there is no guarantee that a derivative-based method will converge to the global optimum. We discuss the theoretical basis for expecting GENOUD to have a high probability of finding global optima. We conduct Monte Carlo experiments using scalar Normal mixture densities to illustrate this capability. We also use a system of four simultaneous nonlinear equations that has many parameters and multiple local optima to compare the performance of GENOUD to that of the Gauss-Newton algorithm in SAS's PROC MODEL.

Introduction

We report on technology we have developed to estimate statistical models that present difficult optimization problems. Optimization difficulties often arise when a model's estimating function (for instance, the log-likelihood) is a nonlinear function of the parameters. In such cases the function to be optimized is usually not globally concave. For example, a log-likelihood that is not globally concave may have multiple local optima, saddle points, boundary solutions or discontinuous jumps. In such cases, methods of optimization that depend entirely on derivatives can be unreliable and often are virtually unusable. Newton-Raphson and quasi-Newton methods are among the commonly used optimization methods that rely completely on derivatives. Such methods work well when the function to be optimized is regular and smooth over the domain of parameter values that is of interest, but otherwise the methods often fail (Gill, Murray and Wright 1981). Even in models where such methods can be expected to work most of the time, resampling techniques such as the bootstrap (Efron and Tibshirani 1993; Shao and Tu 1995) can generate resamples in which derivative-based optimization algorithms encounter severe difficulties. This is unfortunate because the methods most frequently used for optimization in problems of statistical estimation are entirely based on derivatives.

Many statistical models used in the social sciences have estimating functions that are nonlinear functions of the parameters. If one wishes to use such a model, significant optimization difficulties may need to be overcome. For example, the estimating functions of all of the following models are not in general globally concave: models with multivariate qualitative variables, censoring or endogenous switching (Maddala 1983; Amemiya 1985); linear covariance structures (Bollen 1989); models for durations or transitions with unobserved heterogeneity (Heckman and Singer 1985; Lancaster 1990); quasi-likelihood and extended quasi-likelihood models (McCullagh and Nelder 1992) and systems based on such models (Fahrmeir and Tutz 1994); and nonlinear simultaneous equation models (Gallant 1987). For such models, optimization methods driven entirely by derivatives are often unreliable. For some versions of some of these models, algorithms such as EM that involve data augmentation have been successfully applied to produce optimization problems that derivative-based methods can solve (e.g., Lancaster 1990, 197).

The optimizing computer program that we have developed combines evolutionary algorithm methods with a quasi-Newton method. The quasi-Newton method is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method (Gill, Murray and Wright 1981, 119). When the BFGS is being used, our program offers the option of using either built-in numerical derivatives or user-supplied analytical derivatives. Our program can also work without the BFGS, in which case no derivatives are needed and the optimizer will work even when the function is discontinuous. Appropriate use of the BFGS can make the algorithm converge to the global optimum much more quickly. But premature or excessive use of the BFGS can prevent convergence to the global optimum. Our program does not eliminate the need for judgment, testing and patience.

Our program, written in C (Kernighan and Ritchie 1988), extends and greatly modifies a constrained optimization program created by Michalewicz (1992) called GENOCOP. The acronym for the optimizing core of our program is GENOUD (GENetic Optimization Using Derivatives).[1]

We and others have successfully used GENOUD to solve optimization problems arising in the course of estimating and statistically testing hypotheses about a number of empirical models. Examples include a highly nonlinear system of four simultaneous equations (Mebane 1998), an endogenous switching tobit regression model (Mebane and Wawro 1996), a model for mobility in the U.S. House (Wawro 1996), and Markov Chain models for rolling cross-section data (Mebane and Wand 1997). The most widely used statistical model to which GENOUD has been applied is the covariance structure model known as the LISREL model (Jöreskog 1973). Mebane and Sekhon's (1998) program, called GENBLIS, computes bootstrap estimates for the distribution of a covariance structure model's parameter estimates, and also for the distribution of the statistic used to test whether the model fits the data.[2] It would have been highly unreliable to implement the bootstrap resampling method in the absence of an optimizer as effective as GENOUD. With most data, there are usually optimization difficulties in a number of the bootstrap resamples. Resamples in which the effort to optimize fails cannot be discarded or ignored, because they may be indicating that the covariance structure model is misspecified. It is important to have confidence that an optimization failure is not merely a numerical artifact brought on by using an ineffective optimization method.

As Gill, Murray and Wright observe, "there is no guaranteed strategy that will resolve every difficulty" (1981, 285). In this article, we discuss the theory of random search algorithms that supports the assertion that GENOUD has a high probability of finding global optima when such exist. We present results from Monte Carlo experiments using some simple (but fiendish) scalar Normal mixture models to illustrate this capability and to compare GENOUD's abilities with those of an enhanced derivative based algorithm. We then compare the performance of GENOUD with a Gauss-Newton algorithm by using both to estimate a least-squares variant of Mebane's (1998) nonlinear simultaneous equation system.

Evolutionary Nonlinear Optimization

An evolutionary algorithm (EA) uses a collection of heuristic rules to modify a population of trial solutions in such a way that each generation of trial values tends to be on average better than its predecessor. The EA in GENOUD is intended to work for cases where a solution is a vector of real numbers, each number being a value for a scalar parameter of a function to be optimized. Each heuristic rule, or *operator*, acts on one or more trial solutions from the current population to produce one or more trial solutions to be included in the new population. In order to find the global optimum of a function, the EA portion of GENOUD does not require that derivatives exist or even that the function be continuous.

The EA in GENOUD is fundamentally a genetic algorithm (GA) in which the code-strings are vectors of floating point numbers rather than bit strings, and the GA operators take special forms tuned for the floating-point vector representation. A GA uses a set of randomized genetic operators to evolve a finite population of finite code-strings over a series of generations (Holland 1975; Goldberg 1989; Grefenstette and Baker 1989). The operators used in GA implementations vary (Davis 1991; Filho, Treleaven and Alippi 1994), but in an analytical sense the basic set of operators can be defined as reproduction, mutation, crossover and inversion. The wide variety of implementations of these operators reflects the variety of codings that are best suited for different applications. Reproduction entails selecting a code-string with a probability that increases with the code-string's fitness value. Crossover and inversion use pairs or larger sets of the selected code-strings to create new code-strings. Mutation randomly changes the values of elements of a single selected code-string.

Used in suitable combinations, the genetic operators tend to improve the average fitness of each successive generation. There is no guarantee that the average fitness will improve between every pair of successive generations. The average fitness may well decline. And in general, the code-string with the greatest fitness in one generation will not appear in the next one.[3] But theorems exist to prove that code-substrings that have above average fitness values for the current population are sampled at an exponential rate for inclusion in the subsequent population (Holland 1975, 139–140). As Grefenstette (1993, 77) points out, however, such theorems describe only how a GA treats individual substrings in the transition from one generation to the next. The theorems do not imply that GAs converge to global optima. Each generation's population contains a biased sample of code-strings, so that a substring's performance in that population is a biased estimate of its average performance over all possible populations (De Jong 1993; Grefenstette 1993). The simple average of a substring's performance in any one generation is therefore not in general a good indication of how well the substring will perform over many generations.

Better indications regarding the long-run properties of a GA can be gained by thinking of the GA as a Markov chain. A state of the chain is a code-string population of the size used in the GA. For code-strings of finite length and GA populations of finite size, the state space is finite. If such a GA uses random reproduction and random mutation, the Markov chain is aperiodic and all states always have a positive probability of occurring. A finite GA with random reproduction and mutation is therefore an aperiodic and irreducible Markov chain.[4] An aperiodic, irreducible, finite Markov chain converges at an exponential rate to a unique stationary distribution (Billingsley 1986, 128). This means that the probability that each population occurs rapidly converges to a constant, positive value. Nix and Vose (1992; Vose 1993) use a Markov chain model to show that in a GA where the probability that each code-string is selected to reproduce is proportional to its observed fitness, the stationary distribution strongly emphasizes populations that contain code-strings that have high fitness values. They show that asymptotically in the population size—i.e., in the limit for a series of GAs with successively larger populations—populations that have suboptimal average fitness have probabilities approaching zero in the stationary distribution, while the probability for the population that has optimal average fitness approaches one. If $k > 1$ populations have optimal average fitness, then in the limiting stationary distribution the probability for each approaches $1/k$.

The crucial practical implication from the theoretical results of Nix and Vose is that a GA's success as an optimizer depends on having a sufficiently large population of code-strings. If the GA population is not sufficiently large, then the Markov chain that the GA implements is converging to a stationary distribution in which the probabilities of optimal and suboptimal states are not sharply distinguished. Suboptimal populations can be as likely or even more likely to occur than optimal ones. Because the Markov chain is irreducible, the GA will necessarily generate an optimal code-string if it is allowed to run for an unlimited number of generations.[5] But if the stationary distribution is not favorable, the run time in terms of generations needed to produce an optimal code-string will be excessive. If π_j is the probability of an optimal code-string s_j, then the expected number of generations until s_j occurs (from an arbitrary starting population) is reasonably approximated by the recurrence time, $\mu_j = 1/\pi_j$ (Feller 1970, 393). For all but trivially small state spaces, an unfavorable stationary distribution can easily imply an expected running time in the millions of generations. But if the stationary distribution strongly emphasizes optimal populations, relatively few generations may be needed to find an optimal code-string. In general, the probability of producing an optimum in a fixed number of generations increases with the GA population size.

In GENOUD, the probability that each n-dimensional parameter vector is selected for reproduction is a tunable decreasing function of its rank as determined by the value of the objective function: the probability is $p = Q(1-Q)^{r-1}$ where $Q \in (0,1)$ is the specified tuning value and $r \in \{1, 2, \ldots, K\}$ is the rank of a vector from the population of size K, with $r = 1$ being the rank of the best vector. The best vector in each generation is carried over into the next one. All other vectors are replaced with the results from application of eight operators that either mutate single vectors or cross two or more vectors.

The eight operators are listed in Table 1. The operators extend and modify the set of seven operators that Michalewicz (1992; Michalewicz and Logan 1993; Michalewicz, Swaminathan and Logan 1993) used in GENOCOP. GENOUD uses four mutation and four crossover operators.[6]

*** Table 1 about here ***

In *uniform mutation*, *boundary mutation* and *non-uniform mutation*, a single element of the selected vector is chosen at random. Uniform mutation changes the value of the element to a value chosen from the uniform distribution on the interval between the lower and upper bounds specified for the element. Boundary mutation replaces the element with one of the bounds. Non-uniform mutation shrinks the element toward one of the bounds, with the amount of shrinkage decreasing as the generation count approaches the specified maximum number. *Whole non-uniform mutation* does non-uniform mutation for all the parameters in the vector.

Heuristic crossover uses two vectors, $\mathbf{x}$ and $\mathbf{y}$, to produce a vector located at a random increment from $\mathbf{x}$ in a direction pointing away from $\mathbf{y}$. *Polytope crossover* computes one vector which is a convex combination of as many vectors as there are parameters. The input vectors are selected with replacement from the current population. Over generations this operator works as a kind of randomized polytope (or simplex) search method (Gill, Murray and Wright 1981, 94–95). *Multiple point simple crossover* computes two vectors from two input vectors by replacing a randomly selected set of elements with convex combinations of the input vectors' values for those elements. *Local-minimum crossover* computes a new vector in two steps. First, starting from an input vector, it does a preset number of BFGS iterations. Then it computes a convex combination of the input vector and the vector generated by the BFGS iterations.

If the parameter values that define the global optimum are within the ranges over which GENOUD is allowed to search, GENOUD's operator set is very good at finding a neighborhood of the global optimum within which the objective function is concave (assuming the function is concave at the optimum). But the EA search operators can be quite slow to move from an arbitrary point in that neighborhood to the optimum point itself. To expedite final convergence, GENOUD by default applies BFGS optimization to the best trial solution in each generation. In place of analytical derivatives, the BFGS optimizations can use the built-in numerical derivatives. The numerical derivatives automatically use finite differences based on optimal intervals, computed using the procedure described by Gill, Murray and Wright (1981, 337–345).

Normal Mixture Densities

Our first example of GENOUD involves finding maxima for three normal mixture densities. Because they have many local maxima, such densities can be extremely difficult to optimize. We run a Monte Carlo sampling experiment to compare the performance of GENOUD with an algorithm based on BFGS optimization from the best of many random starting points. The name of the modified BFGS algorithm that we compare with GENOUD is enhanced BFGS (EBFGS).

The EBFGS first generates a large number of random starting values and then runs the BFGS on the best one.[7] This modification is an improved implementation of what is often done to enhance the performance of derivative based optimization systems.[8] The EBFGS, with its multiple starting values, optimizes the mixture densities significantly better than does the BFGS. The BFGS implementation in the EBFGS is the same as the one in GENOUD. In both cases, the BFGS uses the analytical first derivative.[9]

The following three equations define the normal mixture densities we use. The first is known as the Claw density, the second as the Asymmetric Double Claw and the third as the Discrete Comb density (Marron and Wand 1992):

$$f_{\mathrm{C}} = \frac{1}{2}\mathrm{N}(0,1) + \sum_{m=0}^{4} \frac{1}{10}\mathrm{N}\left(\frac{m}{2}-1, \frac{1}{10}\right) \tag{1}$$

$$f_{\mathrm{ADC}} = \sum_{m=0}^{1} \frac{46}{100}\mathrm{N}\left(2m-1, \frac{2}{3}\right) + \sum_{m=1}^{3} \frac{1}{300}\mathrm{N}\left(\frac{-m}{2}, \frac{1}{100}\right) + \sum_{m=1}^{3} \frac{7}{300}\mathrm{N}\left(\frac{m}{2}, \frac{7}{100}\right) \tag{2}$$

$$f_{\mathrm{DC}} = \sum_{m=0}^{2} \frac{2}{7}\mathrm{N}(\frac{(12m-15)}{7}, \frac{2}{7}) + \sum_{m=0}^{2} \frac{1}{21}\mathrm{N}(\frac{2m+16}{7}, \frac{1}{21}) \ . \tag{3}$$

In each equation, $\mathrm{N}(a,b)$ denotes the normal density with mean a and standard deviation b. The second and third of these densities are particularly difficult to optimize. Graphs of the densities appear in Figures 1 through 4.

*** Figure 1 about here ***

Each figure contains lines showing the path that GENOUD took to the solution in an example run. Because GENOUD's evolutionary algorithm has stochastic components, other runs may follow different paths.[10] What is most interesting is the way GENOUD jumps among the local maxima in its search for the solution. Unlike gradient based methods, GENOUD is not restricted to searching just the concave region—i.e., near the peak—in which it started. As can be seen in Figures 2–4, GENOUD often jumps back and forth between concave regions. This behavior illustrates the fact that GENOUD accumulates and maintains information about the whole parameter space.

*** Figure 2 about here ***

Algorithms based solely on derivatives (Levenberg-Marquardt, Newton-Raphson, quasi-Newton) cannot reliably find the global maxima displayed in these figures. Each of the densities has several local maxima. The discrete comb density, in Figures 3 and 4, has a global maximum that exceeds the highest local maximum only in the third decimal place.

*** Figure 3 about here ***

For each of the three normal mixtures we run GENOUD and the EBFGS with two different boundary conditions: narrow and wide. The narrow boundaries range from -3 to 3 and the wide ones from -20 to 20. Figures 1–3 show the range within which the densities are significantly positive. In practice one rarely knows where the solution is, and hence the case with the wide boundaries is the most realistic. We also manipulate the population size. The population refers to the number of random starts that we allow the EBFGS to perform and to the number of trial solutions that GENOUD employs. We use a small population of 701 random starts (or trial solutions) and a large population of 1402. We execute 1000 Monte Carlo replications for each combination of boundary width and population size. For each replication, we randomly draw starting values from a uniform distribution defined on the interval between the lower and upper boundary values.

*** Figure 4 about here ***

Although we expect GENOUD to outperform the EBFGS, we do not expect GENOUD to find the correct solution in all cases. For any finite number of generations, there is always a positive probability that GENOUD will not find the correct solution. As the Monte Carlo results in Tables 2 and 3 show, that probability is small for the mixture densities. The probability that GENOUD fails is far smaller than the failure rate of the EBFGS.

*** Tables 2 and 3 about here ***

The Claw density (equation (1) and Figure 1) is the easiest of the three mixtures to maximize. Regardless of the population size or width of the boundaries, GENOUD always correctly optimizes the function. The EBFGS always correctly optimizes the function only with the narrow boundaries. When it has to contend with the wide boundaries, its error rate rises sharply. With the small population of 701 random starts, the EBFGS gets the wrong answer in 24.4% of the replications. With 1402 random starts the EBFGS still fails 9.5% of the time.

The Asymmetric Double Claw (equation (2) and Figure 2) is a more difficult function. With the small population, GENOUD almost always finds the solution regardless of the boundary width. Its worst performance is an error rate of 0.9%, which occurs with the small population and wide boundaries. The EBFGS again performs much worse than GENOUD, and worse than with the Claw density. The EBFGS's best result is an error rate of 12.4%, achieved with the large population and narrow boundaries. For this density, the EBFGS's error rate ranges from 12 to 21 times as large as GENOUD's.

The Discrete Comb density (equation (3) and Figures 3–4) is also more difficult than the Claw density to optimize. GENOUD again finds the correct solution almost all of the time, while the EBFGS performs even more poorly. The worst performance for GENOUD is the error rate of 0.4% for the small population and the wide boundaries. The EBFGS's error rate ranges from a low of 23.1% to a high of 88.7%. When the EBFGS uses the large population, its performance does improve but is still far inferior to GENOUD with the *small* population. To achieve the inferior performance, the EBFGS consumes more time.

If the number of random starts were large enough, both GENOUD and the EBFGS would almost always find the solution. For fixed boundaries that include the solution, the probability of each algorithm finding the correct solution goes to 1.0 as the number of random starts goes to infinity. As the number of random starts increases, so does the probability of having at least one starting value arbitrarily near the correct mode of each density. GENOUD, however, vastly outperforms the EBFGS when both are given the same population. GENOUD outperforms the EBFGS even when the EBFGS is given twice the number of random starts. The EBFGS takes significantly longer to run (2.5 to 4.9 times as long), and yet performs dramatically worse than GENOUD.

In practice, people often run derivative-based systems with only a handful of different starting values. We find that even with a large number of random starts, in half of the experimental conditions the EBFGS has an error rate of over 19%. There is no reason to believe that other gradient-based optimization algorithms would perform substantially better. All such algorithms share the same vulnerability. They optimize locally. They all climb the nearest hill. If the nearest hill is not the best one, the algorithms will return the wrong answer. Even derivative-based optimization with multiple random starts is not reliable unless one has high confidence about the location of the true solution or the problem is easy. An easy problem is one with a function to optimize that is globally concave or one where the area of the concave neighborhood that contains the solution is large relative to the area of the space being searched.

The Four-dimensional Hopf Model

In this section we compare the performance of GENOUD to that of a Gauss-Newton optimizer built into SAS's PROC MODEL (SAS Institute Inc. 1988; 1989), which is a well-known program for estimating nonlinear systems by least squares. The main point is to show with real data how GENOUD can outperform derivative-based optimization when the function to optimize has many parameters and many local optima. For this purpose we use a version of Mebane's (1998) *four-dimensional Hopf* (4DH) model. We also illustrate how variation in the use of derivatives in GENOUD can affect its performance.

The 4DH model is a nonlinear simultaneous equation system for four observed variables. Mebane (1998) uses the model to test certain predictions from a game theoretic model of congressional campaigns. Denoting the observed variables by v, w, x and y, and using $v = \mathsf{v} - \tilde{v}$, $w = \mathsf{w} - \tilde{w}$, $x = \mathsf{x} - \tilde{x}$ and $y = \mathsf{y} - \tilde{y}$, for unknown parameters $\tilde{v}$, $\tilde{w}$, $\tilde{x}$ and $\tilde{y}$, the four-equation specification is

$$\begin{aligned} u_v = v + [&-c_{vy}y + c_{vw}w + c_{vx}x + (a_{vy}v - b_{vy}y)(v^2 + y^2 + e_{vy}vy) \\ &+ (a_{vw}v + b_{vw}w)(v^2 + w^2 + e_{vw}vw) + (a_{vx}v + b_{vx}x)(v^2 + x^2 + e_{vx}vx)]/3 \end{aligned} \tag{4a}$$

$$\begin{aligned} u_w = w + [&-c_{wy}y - c_{wx}x - c_{vw}v + (a_{wy}w - b_{wy}y)(w^2 + y^2 + e_{wy}wy) \\ &+ (a_{wx}w - b_{wx}x)(w^2 + x^2 + e_{wx}wx) + (a_{vw}w - b_{vw}v)(w^2 + v^2 + e_{vw}wv)]/3 \end{aligned} \tag{4b}$$

$$\begin{aligned} u_x = x + [&-c_{xy}y + c_{wx}w - c_{vx}v + (a_{xy}x - b_{xy}y)(x^2 + y^2 + e_{xy}xy) \\ &+ (a_{wx}x + b_{wx}w)(x^2 + w^2 + e_{wx}xw) + (a_{vx}x - b_{vx}v)(x^2 + v^2 + e_{vx}xv)]/3 \end{aligned} \tag{4c}$$

$$\begin{aligned} u_y = y + [&c_{vy}v + c_{wy}w + c_{xy}x + (a_{vy}y + b_{vy}v)(v^2 + y^2 + e_{vy}vy) \\ &+ (a_{wy}y + b_{wy}w)(w^2 + y^2 + e_{wy}wy) + (a_{xy}y + b_{xy}x)(x^2 + y^2 + e_{xy}xy)]/3 \end{aligned} \tag{4d}$$

The vector $\mathbf{u} = (u_v, u_w, u_x, u_y)^\top$ is assumed to be normally distributed with mean $E\mathbf{u} = 0$ and covariance matrix $E\mathbf{u}\mathbf{u}^\top = \mathbf{\Sigma}$. Parameters to be estimated are $\tilde{v}$, $\tilde{w}$, $\tilde{x}$, $\tilde{y}$, the elements of $\mathbf{\Sigma}$, and

a_{ij}, b_{ij}, c_{ij} and e_{ij} for $ij \in \{vy, xy, wy, wx, vw, vx\} \equiv \mathfrak{I}$, with $-2 < e_{ij} < 2$.

Mebane estimates the model by maximum likelihood using 24 combinations of congressional district-level data from 1983–1985 and from 1985–1987. Variable v has four realizations in each period, corresponding to four types of intergovernmental transfers from the federal government to local governments in each district. Variables w and x each have three realizations, corresponding to campaign contributions to incumbents and to challengers from three kinds of political action committees (PACs). Variable y is the logit of the proportion P of all general election votes that were cast for the incumbent. See Mebane (1998) for further details about the data.

To facilitate comparison between GENOUD and PROC MODEL, we estimate the 4DH model using the ordinary least squares criterion of minimizing the sum of the variances in the diagonal of $\mathbf{\Sigma}$. Let $\sigma_+ = \sigma_{vv} + \sigma_{ww} + \sigma_{xx} + \sigma_{yy}$ denote that sum. Minimizing σ_+ gives solutions different from those obtained from the multivariate normal 4DH model likelihood, especially because the term $\log\det\mathbf{\Sigma}$ of the log-likelihood is omitted. This change in the model is inconsequential for our current, purely computational interests. To enforce the range restrictions $-2 < e_{ij} < 2$, we use a penalty function. The penalty added to σ_+ is $\sum_{ij\in\mathfrak{I}} \left[e^{15(e_{ij}-2.07)} + e^{15(-e_{ij}-2.07)}\right]^4$. Penalty functions are not necessary for GENOUD, but are the only feasible technique for imposing the range restrictions on PROC MODEL.

We compare results from Gauss-Newton estimation using PROC MODEL to results from four configurations of GENOUD. The Gauss-Newton algorithm is run using analytical gradients while GENOUD is run using the built-in numerical gradients. All the Gauss-Newton runs start with $\tilde{v}$, $\tilde{w}$, $\tilde{x}$ and $\tilde{y}$ set equal to the sample means of the observed variables and $\mathbf{\Sigma}$ equal to the sample covariance matrix. For GENOUD, those values are included once in the initial population of trial solutions, with all the other trial solutions being randomly generated; identical random values are used for all populations of the same size. The Gauss-Newton runs are allowed up to 1000 iterations. The GENOUD runs use trial solution populations of size 3051, corresponding to 500 instances of Table 1's operators 1, 3, 4, 5, 6 and 7, 50 instances of operator 2 and none of operator 8. Two GENOUD runs have an upper limit of 50 generations and two have an upper limit of 100 generations. In one of each of these runs, GENOUD applies the BFGS to the best trial solution in every generation. For the other two runs the BFGS is used only after half the allowed number of generations have been completed.

Table 4 shows that GENOUD almost always outperforms the Gauss-Newton optimizer for this problem. For all 24 sets of data, both Gauss-Newton and GENOUD find local-minimum solutions that give residual variance σ_+ considerably less than the sum of the sample variances of the observed variables (denoted "original variance"). Only two of the Gauss-Newton runs converge within 1000 iterations by the default criterion (for which see SAS Institute Inc. 1988, 347), but elements of the gradient of the estimating function are always small, except for e_{ij} parameters with near-boundary values. Most of the GENOUD solutions have gradient vectors in which all elements have magnitude less than 10^{-8}. Exceptions are marked as not being local minima. For 22 of the 24 data sets, the best of the GENOUD solutions has residual variance smaller than the residual variance of the Gauss-Newton solution. Inspection of the parameter estimates shows differences large enough to matter for substantive conclusions in 16 of the 22 cases. Of the two instances where the Gauss-Newton solution residual variance is smaller, only for the case of labor PACs and highways transfers in 1986 are the differences in the parameter estimates consequentially large.

*** Table 4 about here ***

Table 5, which reports the CPU time required for the various runs, shows that GENOUD's superior performance does not come at the price of inordinate demand for computing resources. Indeed, in only two instances do the GENOUD runs require more time to complete than PROC MODEL.

*** Table 5 about here ***

The use of the BFGS and the number of generations affect GENOUD's performance, though not in a uniform manner. Variations occur across the GENOUD configurations for nine of the 24 sets of data. In all but one of those nine instances, the configurations that apply the BFGS after every generation do at least as well as the configurations that start using the BFGS only after half the allowed number of generations have completed. Increasing the number of generations from 50 to 100 never hurts performance when the BFGS is being applied to all generations. But when the BFGS is not used for the early generations, the final result is better with 100 generations than with 50 generations one time, but two times it is worse.

The one instance in Table 4 where the Gauss-Newton algorithm clearly outperforms GENOUD presents the most interesting case for exploring the choice that can arise between locally efficient optimization, which is best performed by the BFGS, and effective global search by means of the EA part of GENOUD. Table 6 shows results from running GENOUD on the 1986 data for labor PACs and highways transfers, using four different generation count limits. The results in the table are for GENOUD configurations in which the BFGS is applied only after the halfway point. If the BFGS is used for all generations, GENOUD is unable to improve on the best GENOUD solution reported for these data in Table 4, i.e., $\sigma_+ = 2.9336$. The prematurely reached local minimum is reproduced excessively in the population of trial solutions, to such an extent that the population contains insufficient variety for GENOUD to be able to improve on it in a reasonable number of generations. GENOUD finds better solutions only when it is allowed to search globally for several generations, without the strong pressure to converge locally that the BFGS applies.

*** Table 6 about here ***

Table 6 shows that as the number of generations increases, the solutions found improve. A solution better than that found with Gauss-Newton appears when 400 generations are allowed.

The residual variance is slightly smaller than the Gauss-Newton solution, but the parameter values do not substantially differ.

The four-dimensional Hopf model example shows that GENOUD consistently performs better than SAS's Gauss-Newton optimizer—significantly so in 16 out of 24 cases. The example also illustrates that one needs to be careful when using the BFGS portion of GENOUD. The BFGS's local hill-climbing prowess can cause premature convergence and hence ineffective global optimization, as occurred in 1 of the 24 cases. For maximum assurance, one should use a large population of trial solutions, turn off the BFGS portion of GENOUD and run the program for a very large number of generations. For most problems this would be overly conservative, but one must keep the tradeoff in mind.

Conclusion

In our Monte Carlo experiments even an enhanced derivative based algorithm, the EBFGS, has an error rate as high as 88.7% while GENOUD's error rate is always below 1%. In one problem, the Discrete Comb Density, the EBFGS's error rate ranges from 23.1% to 88.7% while GENOUD's error rate ranges from 0.0% to 0.4%. Too often users may not try more than a few distinct starting values in part because derivative based algorithms are not optimized for a large number of such starts. Therefore, in practice, derivative based systems probably perform even worse than one would infer from our experimental results.

Notwithstanding the power of the GENOUD optimizer, the user must still apply good judgment based on understanding of the mathematical properties of the function to be optimized, the substantive meaning and statistical properties of the model, and the form and content of the data. As the four-dimensional Hopf model example illustrates, there may be important tradeoffs to consider between local hill-climbing efficiency and global optimization. We hope that the availability of an effective, efficient and simple to use program for global optimization will encourage the use of the large class of statistical models whose estimating functions are a nonlinear function of the parameters.

Notes

[1]GENOUD source code, documentation and working papers are available at `http://data.fas.harvard.edu/jsekhon/genoud/`.

[2]GENBLIS is an acronym for GENetic optimization and Bootstrapping of LInear Structures. The GENBLIS source code, documentation and working papers are available at `http://data.fas.harvard.edu/jsekhon/genblis/`.

[3]In GENOUD, however, the code-string with the best fitness in one generation will appear in the next.

[4]Feller (1970, 372–419) and Billingsley (1986, 107–142) review the relevant properties of Markov chains.

[5]Note that the theory regarding the optimality properties of a GA assumes that the random aspects of the GA's behavior are occurring truly at random. In practice, where "random" behavior is actually determined by an algorithmic pseudorandom number generator that ultimately cycles, the theoretical optimality results are not guaranteed. When its theoretically random aspects are actually being driven by a pseudorandom cycle, the Markov chain that the GA implements is periodic. Using a good pseudorandom number generator can make the period extremely long, but nonetheless the periodicity implies that an optimal code-string need not ever occur even if one exists. GENOUD uses a Tausworthe-Lewis-Payne (TLP) generator that has a cycle length of $2^{1279} - 1$ bits (Lewis and Payne 1973; Bright and Enison 1979).

[6]GENOUD differs from GENOCOP in numerous ways other than those described in the text. Among the differences that affect optimization performance are operator changes, the TLP pseudorandom number generator, new methods for seeding the initial population of trial solutions, and new rules of selection for reproduction and for replacement to produce new generations. GENOCOP also did not include any derivative-based optimization.

[7]The starting values are picked by drawing from a uniform distribution. This is exactly how GENOUD determines its starting values.

[8]For instance, PROC NLIN of SAS provides an option for executing a grid search to choose a starting value. Searching a very fine grid would match the performance of the random procedure. But fine grid search does not readily scale up for multiple parameters.

[9]The use of a Newton algorithm does not significantly alter the results. Newton methods are sometimes inferior. Far from a maximum, methods which use the analytical Hessian—such as Newton methods—face significant problems because the analytical Hessian may not be positive definite.

[10]The path would be the same if the operator set, population size, starting values and pseudorandom number generator seeds were all the same.

References

Amemiya, Takeshi. 1985. *Advanced Econometrics.* Cambridge: Harvard University Press.

Billingsley, Patrick. 1986. *Probability and Measure.* New York: Wiley.

Bollen, Kenneth A. 1989. *Structural Equations with Latent Variables.* New York: Wiley.

Bright, H. S., and R. L. Enison. 1979. "Quasi-random Number Sequences from a Long-period TLP Generator with Remarks on Application to Cryptography." *Computing Surveys* 11:357–370.

Davis, Lawrence, ed. 1991. *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold.

De Jong, Kenneth A. 1993. "Genetic Algorithms Are Not Function Optimizers." In *Foundations of Genetic Algorithms 2*, ed. L. Darrell Whitley. San Mateo, CA: Morgan Kaufmann.

Efron, Bradley, and Robert J. Tibshirani. 1993. *An Introduction to the Bootstrap.* New York: Chapman and Hall.

Fahrmeir, Ludwig, and Gerhard Tutz. 1994. *Multivariate Statistical Modeling Based on Generalized Linear Models.* New York: Springer-Verlag.

Feller, William. 1970. *An Introduction to Probability Theory and Its Applications.* Vol. 1, 3d ed., revised printing. New York: Wiley.

Filho, Jose L. Ribeiro, Philip C. Treleaven and Cesare Alippi. 1994. "Genetic Algorithm Programming Environments." *Computer* 27:28–43.

Gallant, A. Ronald. 1987. *Nonlinear Statistical Models.* New York: Wiley.

Gill, Philip E., Walter Murray and Margaret H. Wright. 1981. *Practical Optimization.* San Diego: Academic Press.

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

Grefenstette, John J. 1993. "Deception Considered Harmful." In *Foundations of Genetic Algorithms 2*, ed. L. Darrell Whitley. San Mateo, CA: Morgan Kaufmann.

Grefenstette, John J., and James E. Baker. 1989. "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism." *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 20–27. San Mateo, CA: Morgan Kaufmann.

Heckman, James J., and Burton Singer, ed. 1985. *Longitudinal Analysis of Labor Market Data.* New York: Cambridge University Press.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Jöreskog, Karl G. 1973. "A General Method for Estimating a Linear Structural Equation System." In Arthur S. Goldberger and Otis Dudley Duncan, eds. *Structural Equation Models in the Social Sciences.* New York: Seminar Press.

Kernighan, Brian W., and Dennis M. Ritchie. 1988. *The C Programming Language.* 2d ed. Englewood Cliffs, NJ: Prentice Hall.

Lancaster, Tony. 1990. *The Econometric Analysis of Transition Data.* New York: Cambridge University Press.

Lewis, T. G., and W. H. Payne. 1973. "Generalized Feedback Shift Register Pseudorandom Number Algorithm." *Journal of the Association for Computing Machinery* 20:456–468.

Maddala, G.S. 1983. *Limited-Dependent and Qualitative Variables in Econometrics.* Cambridge: Cambridge University Press.

Marron, J. P. and M. P. Wand. 1992. "Exact Mean Integrated Squared Error." *Annals of Statistics.* 20:712–736.

McCullagh, P. and J.A. Nelder. 1992. *Generalized Linear Models.* New York: Chapman & Hall.

Mebane, Walter R. Jr. 1998. "Congressional Campaign Contributions, District Service and Electoral Outcomes in the United States: Statistical Tests of a Formal Game Model with Nonlinear Dynamics." In *Political Complexity: Nonlinear Models of Politics*. Diana Richards, ed. Ann Arbor: University of Michigan Press.

Mebane, Walter R. Jr., and Jasjeet Sekhon. 1996. *GENOUD: GENetic Optimization Using Derivatives*. Computer program.

Mebane, Walter R. Jr., and Jasjeet Sekhon. 1998. *GENBLIS: GENetic optimization and Bootstrapping of LInear Structures*. Computer program.

Mebane, Walter R. Jr., and Gregory J. Wawro. 1996. "*Chadha*, Gramm-Rudman-Hollings and Impoundments: Testing a Spatial Game Model of Congressional Delegation and Institutional Change." Manuscript.

Mebane, Walter R. Jr., and Jonathan Wand. 1997. "Markov Chain Models for Rolling Cross-section Data: How Campaign Events and Political Awareness Affect Vote Intentions and Partisanship in the United States and Canada." Paper presented at the 1997 Annual Meeting of the Midwest Political Science Association, April 10–12, Palmer House Hilton, Chicago, IL.

Michalewicz, Zbigniew. 1992. *Genetic Algorithms + Data Structures = Evolution Programs.* New York: Springer-Verlag

Michalewicz, Zbigniew, and Thomas D. Logan. 1993. "GA Operators for Convex Continuous Parameter Spaces." Manuscript.

Michalewicz, Zbigniew, Swarnalatha Swaminathan and Thomas D. Logan. 1993. *GENOCOP* (version 2.0). C language computer program source code.

Nix, Allen E., and Michael D. Vose. 1992. "Modeling Genetic Algorithms with Markov Chains." *Annals of Mathematics and Artificial Intelligence* 5:79–88.

SAS Institute Inc. 1988. *SAS/ETS User's Guide, Version 6, First Edition.* Cary, NC: SAS Institute Inc.

SAS Institute Inc. 1989. "SAS (r) PROC MODEL." Proprietary Software Release 6.09. Cary, NC: SAS Institute Inc.

Shao, Jun, and Dongsheng Tu. 1995. *The Jackknife and Bootstrap.* New York: Springer-Verlag.

Wawro, Gregory J. 1996. *Legislative Entrepreneurship in the U.S. House of Representatives.* Unpublished Ph.D. dissertation, Cornell University.

Vose, Michael D. 1993. "Modeling Simple Genetic Algorithms." In *Foundations of Genetic Algorithms 2*, ed. L. Darrell Whitley. San Mateo, CA: Morgan Kaufmann Publishers.

Table 1: GENOUD Operators

1. Uniform Mutation. At random choose $i \in \mathbf{N}$. Select a value $\tilde{x}_i \sim U(\underline{x}_i, \overline{x}_i)$. Set $X_i = \tilde{x}_i$.

2. Boundary Mutation. At random choose $i \in \mathbf{N}$. Set either $X_i = \underline{x}_i$ or $X_i = \overline{x}_i$, with probability 1/2 of using each value.

3. Non-uniform Mutation. At random choose $i \in \mathbf{N}$. Compute $p = (1 - t/T)^B u$, where t is the current generation number, T is the maximum number of generations, $B > 0$ is a tuning parameter and $u \sim U(0,1)$. Set either $X_i = (1-p)x_i + p\underline{x}_i$ or $X_i = (1-p)x_i + p\overline{x}_i$, with probability 1/2 of using each value.

4. Polytope Crossover. Using $m = \max(2, n)$ vectors $\mathbf{x}$ from the current population and m random numbers $p_j \in (0,1)$ such that $\sum_{j=1}^{m} p_j = 1$, set $\mathbf{X} = \sum_{j=1}^{m} p_j \mathbf{x}_j$.

5. Multiple Point Simple Crossover. Choose a random number $m \in \mathbf{N}$ of distinct integers i from $\mathbf{N}$. Using two parameter vectors, $\mathbf{x}$ and $\mathbf{y}$, for each i set $X_i = px_i + (1-p)y_i$ and $Y_i = py_i + (1-p)x_i$, where $p \in (0,1)$ is a fixed number.

6. Whole Non-uniform Mutation. Do non-uniform mutation for all the elements of $\mathbf{X}$.

7. Heuristic Crossover. Choose $p \sim U(0,1)$. Using two parameter vectors, $\mathbf{x}$ and $\mathbf{y}$, compute $\mathbf{z} = p(\mathbf{x} - \mathbf{y}) + \mathbf{x}$. If $\mathbf{z}$ satisfies all constraints, use it. Otherwise choose another p value and repeat. Set $\mathbf{z}$ equal to the better of $\mathbf{x}$ and $\mathbf{y}$ if a satisfactory mixed $\mathbf{z}$ is not found by a preset number of attempts. In this fashion produce two $\mathbf{z}$ vectors.

8. Local-minimum Crossover. Choose $p \sim U(0,1)$. Starting with $\mathbf{x}$, run BFGS optimization up to a preset number of iterations to produce $\tilde{\mathbf{x}}$. Compute $\mathbf{z} = p\tilde{\mathbf{x}} + (1-p)\mathbf{x}$. If $\mathbf{z}$ satisfies boundary constraints, use it. Otherwise shrink p by setting $p = p/2$ and recompute $\mathbf{z}$. If a satisfactory $\mathbf{z}$ is not found by a preset number of attempts, return $\mathbf{x}$. This operators is extremely computationally intensive, use sparingly.

$\mathbf{X} = \left[X_1, \ldots, X_n\right]$ is the vector of n parameters X_i. $\underline{x}_i$ is the lower bound and $\overline{x}_i$ is the upper bound on values for X_i. x_i is the current value of X_i, and $\mathbf{x}$ is the current value of $\mathbf{X}$.
$\mathbf{N} = \{1, \ldots, n\}$. $p \sim U(0,1)$ means that p is drawn from the uniform distribution on the $[0,1]$ interval.

Table 2: Normal Mixture Experiment Results, Small Population

	GENOUD Narrow Boundaries	**GENOUD** Wide Boundaries	**EBFGS** Narrow Boundaries	**EBFGS** Wide Boundaries
Figure 1 Claw Density				
% Error†	0.0	0.0	0.0	24.4
Time‡	3.8	5.3	2.9	2.9
Figure 2 Asymmetric Double Claw				
% Error	0.4	0.9	13.3	19.2
Time	4.0	5.7	3.1	3.2
Figures 3–4 Discrete Comb Density				
% Error	0.0	0.4	49.2	88.7
Time	4.7	8.5	2.6	2.5

Notes: 1000 replications were completed for each category. There were 701 random starting values for the small population.

† The percentage of the 1000 replications in which the correct maximum was *not* found.

‡ The mean time in seconds that it took to finish each replication.

Table 3: Normal Mixture Experiment Results, Large Population

	GENOUD		EBFGS	
	Narrow Boundaries	Wide Boundaries	Narrow Boundaries	Wide Boundaries
Figure 1 Claw Density				
% Error†	0.0	0.0	0.0	9.5
Time‡	11.5	11.5	9.6	25.8
Figure 2 Asymmetric Double Claw				
% Error	0.2	0.7	12.4	13.6
Time	12.1	13.2	9.9	26.9
Figures 3–4 Discrete Comb Density				
% Error	0.0	0.3	23.1	80.1
Time	16.1	19.9	9.1	22.7

Notes: 1000 replications were completed for each category. There were 1402 random starting values for the large population.

† The percentage of the 1000 replications in which the correct maximum was *not* found.
‡ The mean time in seconds that it took to finish each replication.

Table 4: GENOUD versus Gauss-Newton Optimization of the Least Squares 4DH Model

				Residual Variance				
					GENOUD			
	Type of Transfer/		Original	Gauss-	25+25^a		50+50^b	
Year	PACs	N	Variance	Newton	B+B^c	no+B^d	B+B	no+B
1984	education							
	corporate	162	7.3920	1.9141$^\diamond$	1.8314	1.7766	1.8314	1.7766
	labor	114	9.4636	1.8909$^{\diamond\circ}$	1.8895°	1.8895°	1.8895°	1.8895°
	non-connected	186	7.2649	2.3035$^\diamond$	2.1997	2.2497*	2.1997	2.1997
	highways							
	corporate	92	12.8424	2.6218$^\diamond$	2.6197	2.7219	2.6197	2.7219
	labor	63	12.4288	1.9982$^{\diamond\circ}$	1.9948°	1.9948°	1.9948°	1.9948°
	non-connected	103	12.5410	2.7457$^\diamond$	2.7755	2.8487*	2.7319	2.7455
	social welfare							
	corporate	162	7.1987	1.7356$^\diamond$	1.7224	1.7224	1.7224	1.7224
	labor	114	9.6753	1.8133$^{\diamond\circ}$	1.8133°	1.8133°	1.8133°	1.8133°
	non-connected	186	6.8500	2.1068$^\diamond$	1.9988	1.9988	1.9988	1.9988
	other							
	corporate	162	6.6478	1.6062$^\diamond$	1.4716	1.4716	1.4716	1.5357
	labor	114	8.6963	1.5894$^\diamond$	1.5884	1.5884	1.5884	1.5884
	non-connected	186	6.4171	1.7610$^\diamond$	1.6848	1.6848	1.6848	1.6848
1986	education							
	corporate	150	5.5344	1.5042$^{\diamond\circ}$	1.5041°	1.5041°	1.5041°	1.5041°
	labor	94	8.6402	1.4757$^{\diamond\circ}$	1.4727°	1.4727°	1.4727°	1.4727°
	non-connected	167	7.0023	1.6522$^\diamond$	1.6498	1.6498	1.6498	1.6498
	highways							
	corporate	101	8.5653	1.7595$^\diamond$	1.7085	1.7085	1.7085	1.7123*
	labor	67	14.3379	2.7857$^\diamond$	2.9336	3.0225*	2.9336	2.9611*
	non-connected	119	8.9918	1.8046$^{\diamond\circ}$	1.8053°	1.8053°	1.8053°	1.8053°
	social welfare							
	corporate	150	4.9803	1.2975	1.2389	1.2389	1.2389	1.2389
	labor	94	7.6896	1.1788$^\diamond$	1.1332	1.1355	1.1332	1.1355
	non-connected	167	6.3375	1.4766$^{\diamond\circ}$	1.4724°	1.5157	1.4724°	1.5157
	other							
	corporate	150	4.5346	1.1132$^\diamond$	1.0966	1.0966	1.0966	1.0966
	labor	94	7.3780	1.2868	1.1493	1.1493	1.1493	1.1493
	non-connected	167	5.8507	1.3282$^\diamond$	1.2503	1.2503	1.2503	1.2503

Notes: The Gauss-Newton algorithm (PROC MODEL of SAS) ran for 1000 iterations. The operator counts for GENOUD were $(500, 50, 500, 500, 500, 500, 500, 0)$. ^{a} 25 generations with or without BFGS then 25 generations with BFGS. ^{b} 50 generations with or without BFGS then 50 generations with BFGS. ^{c} Both sets of generations used BFGS. ^{d} Only the second set of generations used BFGS. $^\diamond$ Not converged. * Not a local minimum. $^\circ$ No substantial differences between Gauss-Newton and best GENOUD solution parameter values.

Table 5: GENOUD versus Gauss-Newton Optimization CPU Times

Year	Type of Transfer/ PACs	Gauss-Newton	GENOUD 25+25[a] B+B[c]	GENOUD 25+25[a] no+B[d]	GENOUD 50+50[b] B+B	GENOUD 50+50[b] no+B
1984	education					
	corporate	1252.57	189.1	357.4	370.4	697.4
	labor	799.89	176.2	334.6	337.6	645.0
	non-connected	1590.19	202.0	372.2	388.1	734.4
	highways					
	corporate	746.96	170.3	293.0	326.7	617.4
	labor	455.02	157.4	301.0	304.9	397.5
	non-connected	631.07	338.6	323.7	660.3	629.3
	social welfare					
	corporate	1113.15	190.1	367.3	366.3	714.0
	labor	766.37	341.6	326.7	202.9	644.9
	non-connected	1380.91	291.1	446.5	473.2	435.1
	other					
	corporate	1760.47	213.8	279.2	392.0	369.4
	labor	763.27	179.2	203.9	338.6	650.4
	non-connected	1224.39	393.0	384.1	396.0	732.3
1986	education					
	corporate	1156.68	187.1	349.5	356.4	678.0
	labor	672.00	154.4	233.6	155.4	334.8
	non-connected	1599.94	199.0	363.3	377.2	708.6
	highways					
	corporate	725.58	179.2	325.7	345.5	638.8
	labor	451.02	311.9	303.9	317.8	602.9
	non-connected	833.49	121.8	329.7	121.8	658.1
	social welfare					
	corporate	436.87	184.1	228.7	355.4	674.2
	labor	822.33	165.3	316.8	321.8	617.8
	non-connected	1302.72	199.0	378.2	376.2	699.9
	other					
	corporate	1279.36	185.1	346.5	357.4	412.8
	labor	258.35	170.3	246.5	324.7	340.6
	non-connected	998.65	179.2	230.7	179.2	701.9

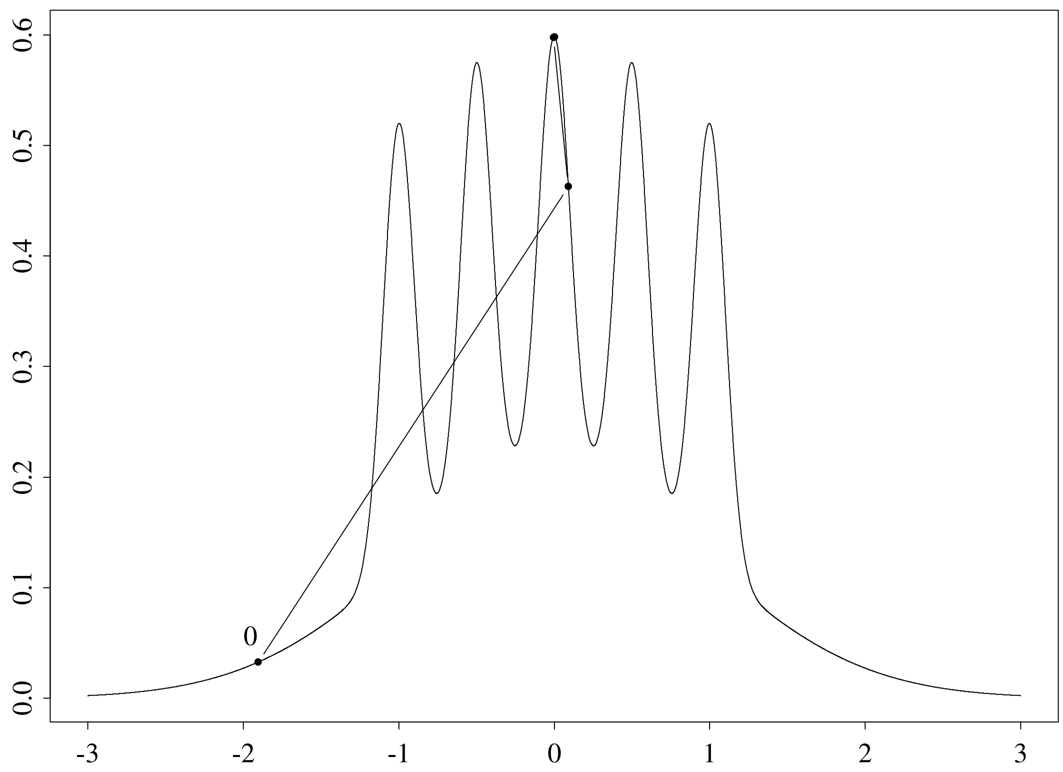
Notes: Times (user plus system) are in seconds. [a] 25 generations with or without BFGS then 25 generations with BFGS. [b] 50 generations with or without BFGS then 50 generations with BFGS. [c] Both sets of generations used BFGS. [d] Only the second set of generations used BFGS.

Table 6: GENOUD for 1986 Labor PACs and Highways Transfers Data

Generations	3051^a
$25+25^b$	3.0225*
$50+50^b$	2.9611*
$100+100^b$	2.8527*
$200+200^b$	2.7828

Notes: The operator counts for GENOUD were ^{a} $(500, 50, 500, 500, 500, 500, 500, 0)$. * Not a local minimum. ^{b} Only the second set of generations used BFGS.

Figure 1: Normal Mixture: The Claw Density

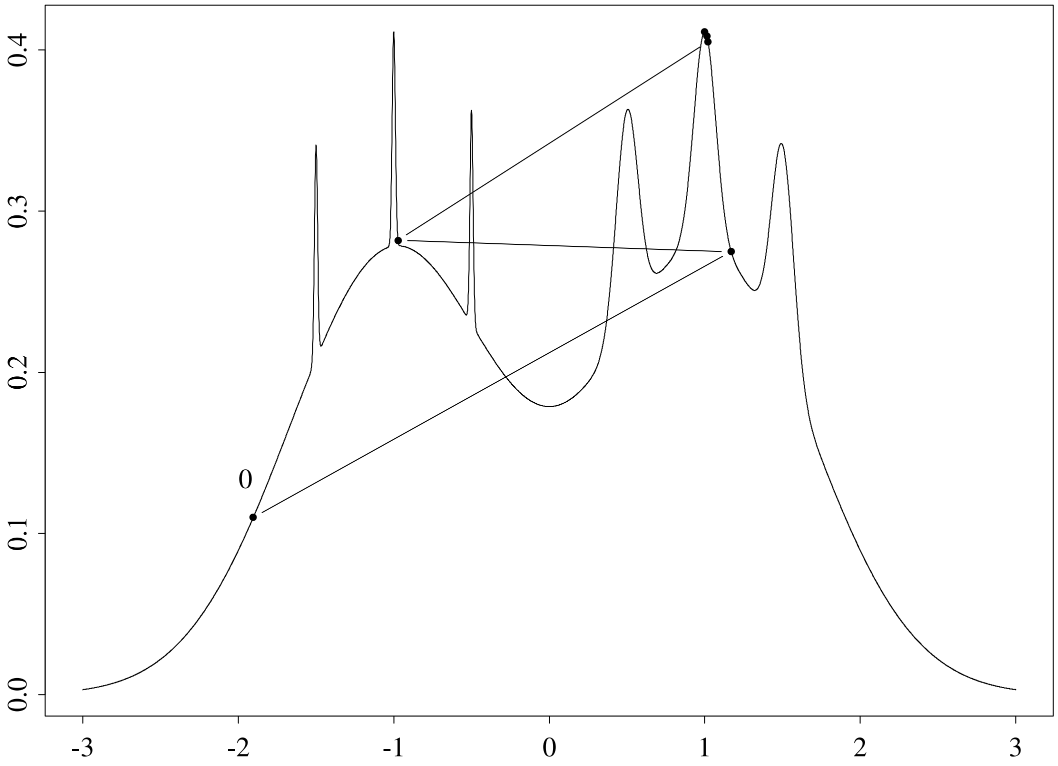


Mixture Density: $\frac{1}{2}\mathrm{N}(0,1) + \sum_{m=0}^{4} \frac{1}{10}\mathrm{N}(\frac{m}{2} - 1, (\frac{1}{10}))$

The straight lines show the path that GENOUD took, in an example run, to find the solution.

See Tables 2,3 for the Monte Carlo results.

Figure 2: Normal Mixture: Asymmetric Double Claw

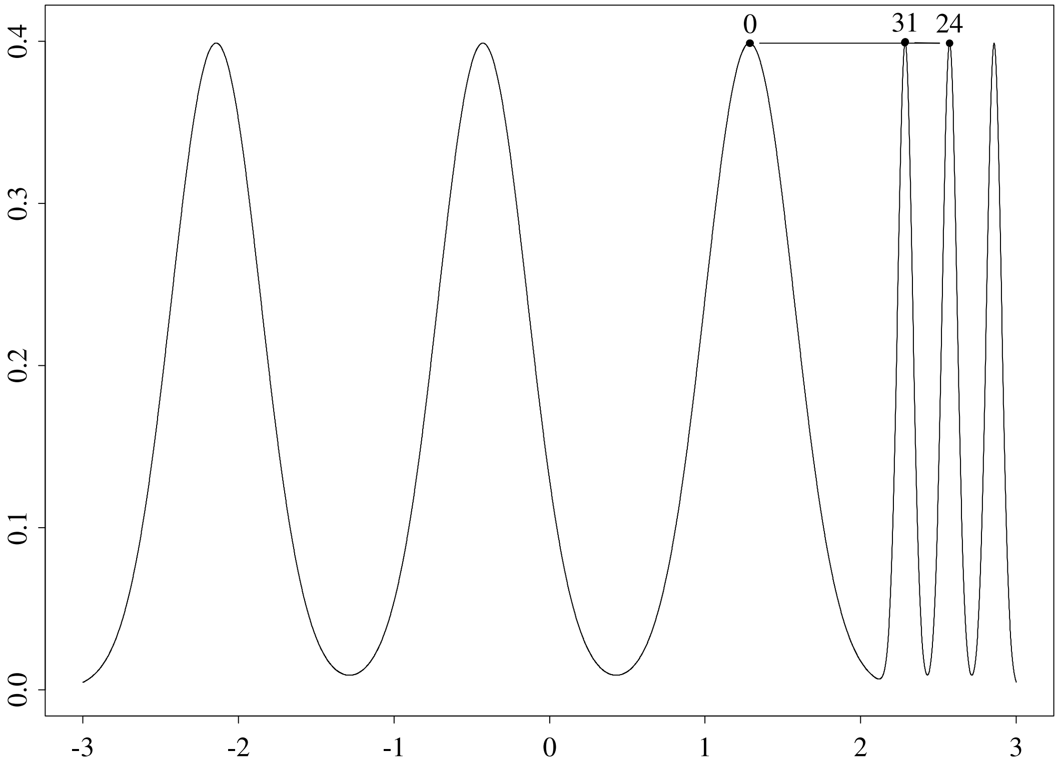


Mixture Density: $\sum_{m=0}^{1} \frac{46}{100}\mathrm{N}(2m-1, \frac{2}{3}) + \sum_{m=1}^{3} \frac{1}{300}\mathrm{N}(\frac{-m}{2}, \frac{1}{100}) + \sum_{m=1}^{3} \frac{7}{300}\mathrm{N}(\frac{m}{2}, \frac{7}{100})$

The straight lines show the path that GENOUD took, in an example run, to find the solution.

See Tables 2,3 for the Monte Carlo results.

Figure 3: Normal Mixture: Discrete Comb Density



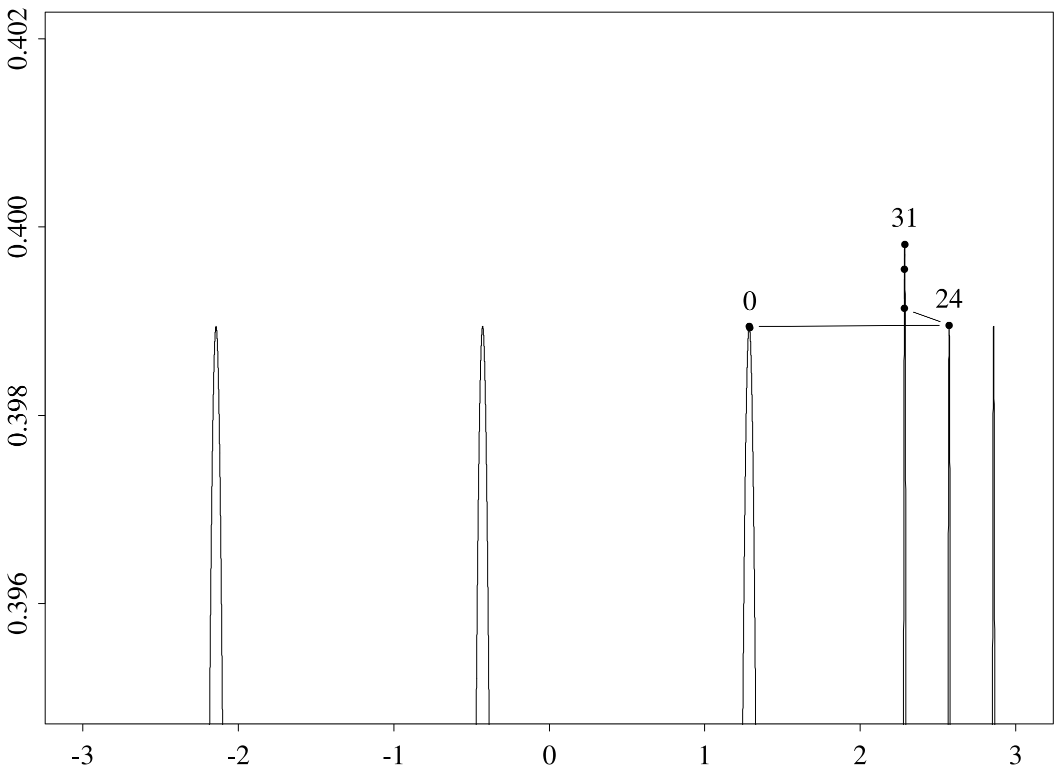
Mixture Density:

$$\sum_{m=0}^{2} \frac{2}{7}\mathrm{N}\left(\frac{(12m-15)}{7}, \frac{2}{7}\right) + \sum_{m=0}^{2} \frac{1}{21}\mathrm{N}\left(\frac{2m+16}{7}, \frac{1}{21}\right)$$

The straight lines show the path that GENOUD took, in an example run, to find the solution.

See Tables 2,3 for the Monte Carlo results.

Figure 4: Normal Mixture: Discrete Comb Density, Magnified Peaks



Mixture Density:
$\sum_{m=0}^{2} \frac{2}{7}\mathrm{N}(\frac{(12m-15)}{7}, \frac{2}{7}) + \sum_{m=0}^{2} \frac{1}{21}\mathrm{N}(\frac{2m+16}{7}, \frac{1}{21})$

The straight lines show the path that GENOUD took, in an example run, to find the solution.

See Tables 2,3 for the Monte Carlo results.