Doherr, Thorsten; Czarnitzki, Dirk

Working Paper

Genetic algorithms: a tool for optimization in econometrics - basic concept and an example for empirical applications

ZEW Discussion Papers, No. 02-41

Provided in cooperation with:
Zentrum für Europäische Wirtschaftsforschung (ZEW)

Suggested citation: Doherr, Thorsten; Czarnitzki, Dirk (2002) : Genetic algorithms: a tool for optimization in econometrics - basic concept and an example for empirical applications, ZEW Discussion Papers, No. 02-41, http://hdl.handle.net/10419/24644

Discussion Paper No. 02-41

Genetic Algorithms: A Tool for Optimization in Econometrics – Basic Concept and an Example for Empirical Applications

Dirk Czarnitzki and Thorsten Doherr

Discussion Paper No. 02-41

Genetic Algorithms: A Tool for Optimization in Econometrics – Basic Concept and an Example for Empirical Applications

Dirk Czarnitzki and Thorsten Doherr

Non–technical Summary

Many new inventions in the field of engineering sciences are based on the knowledge of structures in nature. These highly efficient structures are the results of an optimization process called evolution. Evolution is the strategy used by nature to optimize the adaptation of life according to the environment.

In this paper, we present a tool for optimization of econometric models based on evolutionary programming: a genetic algorithm. Genetic algorithms simulate evolution for a population of candidate solutions in an artificial environment representing a specific problem. Examples are the optimization of movement patterns for artificial life–forms, the determination of weights for neural networks or, not surprisingly, the emergence of markets in the economy. These by no means exhaustive examples demonstrate the versatility of genetic algorithms.

We briefly describe the fundamental concept of genetic algorithms and explain the design of a specifically developed algorithm which we have designed to estimate econometric models, especially those models where the criterion function is not continuously differentiable. This genetic algorithm is implemented in the statistical software package STATA (Version 7.0).

To demonstrate its performance, we apply it to a difficult econometric problem: the semiparametric estimation of a censored regression model. We carry out some Monte Carlo simulations and compare the genetic algorithm with another estimation technique, which is called "iterative linear programming algorithm" (ILPA), to run the censored least absolute deviation (CLAD) estimator. It turns out that both algorithms lead to similar results in this case, but that the proposed method is computationally more stable than its competitor.

Genetic algorithms: A tool for optimization in econometrics — Basic concept and an example for empirical applications[1]

by

Dirk Czarnitzki and Thorsten Doherr

July 2002

Abstract

This paper discusses a tool for optimization of econometric models based on genetic algorithms. First, we briefly describe the concept of this optimization technique. Then, we explain the design of a specifically developed algorithm and apply it to a difficult econometric problem, the semiparametric estimation of a censored regression model. We carry out some Monte Carlo simulations and compare the genetic algorithm with another technique, the iterative linear programming algorithm, to run the censored least absolute deviation estimator. It turns out that both algorithms lead to similar results in this case, but that the proposed method is computationally more stable than its competitor.

Keywords: Genetic Algorithm, Semiparametrics, Monte Carlo Simulation
JEL–Classification: C14, C25, C45, C61, C63

Address: Centre for European Economic Research (ZEW)
Department of Industrial Economics and International Management
P.O.Box 10 34 43
68034 Mannheim
Germany
Phone: +49/621/1235–158, –291
Fax: +49/621/1235–170
E-Mail: `czarnitzki@zew.de, doherr@zew.de`

[1]Helpful comments by François Laisney and James L. Powell are gratefully acknowledged. Moreover, we would like to thank the participants of the "Econometrics Lunch" of the University of California at Berkeley, where an earlier version of this paper has been presented.

1 Introduction

Many new inventions in the field of engineering sciences are based on the knowledge of structures in nature. These highly efficient structures are the results of an optimization process called evolution. Evolution is the strategy used by nature to optimize the adaptation of life according to the environment. The basic principles are crossover, mutation and selection. Evolution theory explains these principles on the basis of whole populations. Genetic science takes a much closer look at the individual aspects of evolution: the genes. It explains the meaning of crossover and mutation at a molecular level. The knowledge of both worlds is combined in genetic algorithms to use the problem solving capabilities of evolution for a large number of scientific and engineering problems or models. Genetic algorithms (GAs) simulate evolution for a population of candidate solutions in an artificial environment representing a specific problem. Examples are the optimization of movement patterns for artificial life–forms, the determination of weights for neural networks or, not surprisingly, the emergence of markets in the economy. These by no means exhaustive examples demonstrate the versatility of GAs.

GAs have already been used in economic research, especially in industrial economics. For example, Arifovic (1994) applied a GA to the cobweb model and compares its performance with three other learning algorithms (least squares learning algorithm, algorithm in which agents form cobweb expectations as well as sample averages of past prices). She concludes that the GA converges to the rational expectations equilibrium for a wider range of parameter values than the competitors. One main result is that the GA requires less prior information: while other algorithms assume that agents know how to maximize their objective function, this is not necessary for the GA. Instead, the agents learn how to maximize their objective functions in view of their environment. Arifovic points out that this feature of the GA behaviour is similar to the characteristics of human behaviour observed in experiments. Price (1997) uses GAs to model strategic behaviour in several standard industrial organization games. He demonstrates that GA performs well as a modeling tool and that evolutionary programming has a potential role in applied economics when detailed market simulations are required. Cooper (2000) applies a GA to some design problems of firms' research and development activities. He explores stochastic learning curves, patent design and the importance of technodiversity in the introduction of new technologies to developing countries. An empirical application has been conducted by Varetto (1998), who introduces a GA to analyze the insolvency risk of 3,840 industrial Italian companies. Varetto compares linear discriminant analysis as a traditional statistical methodology for bankruptcy classification and prediction and a genetic algorithm. He concludes that genetic algorithms are a very effective instrument for insolvency diagnosis, although the results obtained with linear discriminant analysis were superior to those obtained with the GA. Varetto notes that "the results of the GA were obtained in less time and with more limited contributions from the financial analyst than the linear dicriminant analysis."

The use of GAs in econometrics is rare. Especially, for semiparametric estimations it may be a useful alternative to existing optimization algorithms. The only study we are aware of is Dorsey and Mayer (1995). They apply GAs to several optimization problems in econometric estimation and carry out Monte Carlo simulations. Among other methods, Dorsey and Mayer compare a GA to the Maximum Score estimator developed by Manski (1975). However, they use data from a study of McManus (1985) on the effect of capital punishment which consist of only 44 observations. Dorsey and Mayer (1995) focus their study on the impact of varying the tuning parameters of a GA but they do not present results on the six coefficients that are estimated. In line with Dorsey and Mayer, we provide Monte Carlo results to demonstrate that a GA works well. We estimate a censored least absolute deviation (CLAD) model using a GA for optimization. We report results for different sample sizes and analyze the convergence of the estimates.

The next section explains the concept of GA and introduces basic terminology used in evolutionary programming. Section 3 describes the design of a genetic algorithm which we have developed for the estimation of econometric models. In section 4, we apply this GA to a specific econometric model and carry out simulations to study the performance of the GA.

2 Genetic algorithms

The concept of genetic algorithm is very appealingly described by Cooper (2000). He calls it a concept of partial imitation and refers to an approach which is familiar to every economist: "[...] an effective method for creating innovative new models is to combine the successful features of two or more existing models" (Cooper, 2000: 403). That is exactly what Nature does and what is known as evolution. Learning from Nature and finding improved elements of a complex space is incorporated into a formal method of optimization called genetic algorithm.

2.1 Terminology

The terminology of GAs is mainly borrowed from biology and evolution theory to underline the analogies. Each term represents the artificial implementation of biological or evolutionary concepts, though on a much simpler level. Because there are always different views on the same item, a multitude of more biological or more evolutionary inspired realizations exists.

Population denotes a set of specific entities. In biological terms these entities are organisms of the same species. In the world of GAs the organisms are from the species "candidate solution" — living in an artificial environment defined by a given problem. The first generation of candidate solutions is a sample of possible parameter combinations for the problem.

Fitness is a term from evolution theory. It is a measure of the survival and the reproduction probability and fertility of an entity. The fitness function of the GA defines the environment for the artificial evolution. Each candidate solution in the population will be evaluated by the fitness function. A higher fitness results in a higher survival chance and a higher reproduction rate. A GA maximizes the fitness of a population.

Selection is the evolutionary term for "survival of the fittest", referring to the probability for an organism to survive and reproduce. Most GAs described in the literature have been "generational" — at each generation the new population consists entirely of offsprings formed by parents in the previous generation (Mitchell 1996). The parent generation is completely discarded. These GAs rely only on selection for reproduction. Other GAs additionally implement the struggle for survival. The environment can only support a given number of entities. An evolutionary step of a GA consists of reproduction to create an intermediate population of parents and offspring, and of evaluation and selection of the fittest entities to re–establish the original population size. This and similar selection methods are based on a concept called "elitism", first introduced by De Jong (1975). Many researchers have found that elitism significantly improves the GAs' performance (Mitchell 1996).

Reproduction is another basic principle of evolution. In GAs the fertility of a candidate solution is determined by the relative fitness. A fitter solution will reproduce more often than a less fit entity.

Deoxyribonucleic Acid (DNA) is the carrier of genetic information for all complex organisms. The DNA is organized in strings which are called chromosomes that serve as a "blueprint" for the organism. Specific sections of chromosomes define the genes: functional blocks of DNA, each of which encodes a particular protein. On a more abstract level — which we call the "evolutionary view" from now on — the genes encode different traits, such as the color of the skin. The most biologically inspired form of artificial DNA encoding for GAs is the use of segmented bit strings. Each segment represents a parameter of the fitness function. Mutation and crossover can easily be achieved by simple binary operations. The drawback of this method is that a small change in the bit pattern can double or halve the real value of the segment. Although there have been many extensions to the basic binary encoding schema, such as gray coding (Bethke, 1980, Caruna and Schaffer, 1988) to circumvent this problem, binary encoding is considered unnatural and unwieldy for many problems (Mitchell, 1996). A more evolution inspired method is the use of real valued encodings. These are much more natural to use because the fitness functions of many problems require real numbers as parameter inputs. A candidate solution is more a set of traits than a chromosome.

Crossover is the recombination of the subsequences from two chromosomes to create two offsprings. The new chromosomes share the genes of both parents. From the evolutionary point of view, crossover secures the continuation of successful traits. In GAs the method to simulate crossover depends on the method of DNA encoding. Binary encoding requires only the splitting and recombination of the parent bit streams at randomly chosen crossover points — simulating the biological process. Real value encoded GAs use more complex methods to simulate the crossover of traits. Since traits are represented by real numbers the trait of the offspring can be calculated as a randomly weighted arithmetic mean of the corresponding trait numbers of the parents. This allows the "child" to have the hair of its mother, the eyes of its father and a nose with a little bit of both parents.

Mutation is a very important evolutionary aspect for GAs. While crossover can produce many new variants of existing solutions, mutation has the power to produce completely new solutions. It is randomly applied after crossover to mutate one or more genes in an offspring. GAs using binary encoding just have to apply the logical "not" operator at randomly chosen bit positions in the string. The mutation of randomly selected real value encoded traits is resolved by the multiplication with a random factor within a specific interval — termed for further reference "radiation level".

2.2 Implementation of a genetic algorithm

There is no golden rule to implement a GA. The biological and evolutionary concepts of GA leave much room for interpretation. Many additional concepts can be introduced to optimize the GA for a specific problem. We choose a very straightforward approach using real valued encoding and elitism — rooted more in the evolutionary than in the biological soil. The target of this implementation was to devise a reliable and versatile tool for many statistical optimization problems. The GA has been developed in STATA Version 7.0, a statistical software package with a flexible programming language. The fitness function can be dynamically linked to the GA, so that there is no need to reprogram the GA for a specific problem. The integration of the GA into other STATA programs can easily be achieved. The implementation is described below:

1. Creation of the initial population
   The initial population consists of s "survivor" vectors representing the candidate solutions. A vector β has k elements corresponding to the parameters of the fitness function
   $$f_i = f(\beta_{i1}, \dots, \beta_{ik}) \qquad \text{with} \quad i = 1, \dots, s. \tag{1}$$
   The elements β_{ij} $(j = 1, \dots, k)$ are initialized by a random value in a particular interval $[a_j, b_j]$ which has to be chosen by the user. A first evaluation of the fitness f of each vector is then performed.

2. Main loop
   The main loop runs the artificial evolution. It repeats steps 3 to 5 until a maximum number of generations T is reached or the GA stagnates. Stagnation occurs when the current generation equals the previous

generation over a given number of subsequent generations τ (with $\tau \leq T$) .

3. Determination of the mutation probability and radiation level
The values of the mutation probability γ and the radiation level δ both shrink according to the general half–life formula:

$$\begin{aligned} \gamma_t &= \gamma_0 \exp\left(\frac{-\ln(2)}{\lambda_\gamma} t\right) \qquad \text{with} \quad t = 1, \ldots, T, && (2) \\ \delta_t &= \delta_0 \exp\left(\frac{-\ln(2)}{\lambda_\delta} t\right) \qquad \text{with} \quad t = 1, \ldots, T, && (3) \end{aligned}$$

where γ_0 and δ_0 are the initial values and λ_γ and λ_δ are the half–life durations. Since mutation is a probability, the initial value must be in the interval $[0, 1]$. The absolute value of the radiation level and its negative counterpart define the interval limits for the random mutation factor.

4. Determination of the selection probability
A selection probability ω_i has to be associated for each vector β_i. The first step is the determination of the minimum and the maximum of the fitness values to calculate an offset for the following normalization scheme:

$$\text{offset} = \frac{\max(f_1, \ldots, f_s) - \min(f_1, \ldots, f_s)}{s} \tag{4}$$

The offset is required to give the lowest fitness a reasonable probability greater than zero. Therefore, we compute a rescaled fitness

$$h_i = f_i - \min(f_1, \ldots, f_s) + \text{offset}. \tag{5}$$

Then, the selection probability is defined as

$$\omega_i = \frac{h_i}{\sum_{i=1}^{s} h_i}. \tag{6}$$

In rare occasions, it may be possible that users want to decrease the importance of the current fitness for selection, for example, as an additional tool to minimize the danger of getting trapped in local extrema. Therefore, we include an optional weight W for the selection probability. By default, $W = 1$ which means that the selection probability for crossovers is fully determined by the fitness of the candidate solutions. If W is set to values smaller than 1, the importance of the individual fitness decreases. If $W = 0$, the selection probability is independent of the fitness, so that the chance of being chosen for crossover would be equal for every candidate solution. If W is specified, the selection probability is calculated as a the convex combination

$$\omega_i^* = (1 - W)\frac{1}{s} + W\omega_i. \tag{7}$$

5. Evolution

   - Two different candidate solutions are drawn out of the population according to the selection probabilities ω_i (optionally weighted by W).
   - Crossover is applied by a randomly weighted mean for each element pair of the drawn candidates. The weight is uniformly distributed on the interval $[0; 1]$.
   - An element of the resulting offspring vector is mutated according to the mutation probability γ.
   - The radiation level determines the interval $[-\delta; \delta]$ of the random mutation factor which is uniformly distributed.
   - The offspring is evaluated by the fitness function.
   - This process is repeated until the number of offsprings reaches a given number o. This intermediate population is sorted by the fitness of its members to determine the s survivors for the next generation.
   - The criterion of stagnation (see step 2 above) is fulfilled when the new generation contains no offspring.

3 An example of implementing genetic algorithms in econometrics

This section presents a small Monte–Carlo study of the estimation of a semiparametric econometric model: the censored least absolute deviation (CLAD) model proposed by Powell (1984). We compare the genetic algorithm with an estimation technique called iterative linear programming algorithm (ILPA) suggested by Buchinsky (1994). Note that there is no particular reason for choosing this model for demonstration of the GA. The

GA can be applied to every numerical criterion function. However, as we encountered problems using the CLAD estimator in empirical studies, we thought it might be useful to think about the GA as another option for practical use or at least as a supplementary method.

Consider the econometric model

$$y_i^* = \beta' \mathbf{x}_i + \varepsilon_i, \tag{8}$$

where $\mathbf{x}_i$ is a set of regressors, β the corresponding coefficient vector to be estimated and ε_i a stochastic error term. y_i^* is an unobserved variable and we only observe a left censored variable

$$y_i = \begin{cases} y_i^* & \text{if} \quad y_i^* > 0, \\ 0 & \text{if} \quad y_i^* \leq 0. \end{cases} \tag{9}$$

A special case is the Tobit model which is a fully parametric model and can be estimated with the common maximum likelihood (ML) techniques. It is derived from the additional assumption that $y_i^* \sim N(\mu, \sigma^2)$. If the assumptions of homoscedasticity or normality are violated, the ML estimates may be inconsistent. In case of heteroscedasticity, researchers can attempt to model heteroscedasticity as a function of some observable variables. However, the true functional form is usually unknown and the choice of variables determining the heteroscedasticity function is arbitrary. Tobit estimates are sensitive to different choices of the heteroscedasticity term.

Powell has developed different semiparametric models to relax the strict assumptions which were needed to estimate censored regression models. Among others, he proposed the CLAD model, where he suggests to estimate $MED(y_i|x_i)$ instead of its expectation. This quantile regression can be expressed by the minimization problem

$$\beta_{CLAD} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left| y_i - \max(0, \beta' \mathbf{x}_i) \right|. \tag{10}$$

This estimator is consistent and asymptotically normally distributed even in case of heteroscedastic and/or non–normally distributed error terms (see Powell 1984, 1994). However, as this criterion function is not continuously differentiable, estimation is not easy. Buchinsky (1994) has proposed an iterative technique (ILPA) which uses the idea of sample trimming to estimate CLAD models. His procedure can be summarized by the following steps:

1. Estimate a median regression (least absolute deviation: LAD) with the entire sample and generate the estimated $\hat{y}_i$ for this initial step.

2. Subsequently, the sample is trimmed, i.e. the observations for which $\hat{y}_i < 0$ are dropped. It is noteworthy that we keep observations for which $\hat{y}_i \geq 0$. In Buchinsky's original procedure, he suggested to keep only those observations for which $\hat{y}_i > 0$. However, Fitzenberger (1994) has shown that the modified version (using the subset of observations if $\hat{y}_i \geq 0$) is more likely to converge than Buchinsky's original proposal. Fitzenberger calls it modified iterative linear programming algorithm (MILPA).

3. Repeat the estimation of a median regression for the trimmed sample and predict $\hat{y}_i$ again (for the entire sample).

4. Return to step 2 and keep iterating until the estimated coefficients do not change during the iterations.

Buchinsky's algorithm has the advantage that it is quite easy to implement using standard econometric software packages (which provide median regressions as implemented command). Fitzenberger (1994) shows that ILPA is less likely to converge than his modification "MILPA". Moreover, he finds that both ILPA and MILPA are outperformed by another algorithm called BRCENS: Fitzenberger adapts an algorithm (Barrodale–Roberts) for standard LAD regressions and extends it to the CLAD model. It outperforms the others with respect to the frequencies that the global optimum is reached. However, conditional on convergence of ILPA and BRCENS there is no clear ranking between the algorithms (see also Fitzenberger 1997 and Fitzenberger and Winker 1999).

The practical problem of non–convergence of the Buchinsky algorithm which occurs in applications is that the iteration procedure may jump between two (or more) trimmed states of the sample. If it is running in circles it is unclear, what the researcher can do to either achieve convergence or how to decide whether one of these circling states represents a minimum. Of course, one could record the value of the criterion function and choose the minimum but there is no reason which ensures that this value is a global optimum. It is noteworthy that this circling of the algorithm is not unlikely to happen. Due to these phenomenon, we compare the performance of the (modified) Buchinsky algorithm called MILPA with the genetic algorithm in censored regression models. First, we choose a very simple case of a model and carry out Monte–Carlo simulations. Consider the true model

$$y_i^* = \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i}, \qquad i = 1, \ldots, N, \tag{11}$$

with the constant term $x_{3i} = 1 \quad \forall \quad i$. The explanatory variable x_1 is standard normally distributed and x_2 is uniformly distributed on the interval [0,1]. For simplicity, we set $\beta_1 = \beta_2 = \beta_3 = 1$. The observed dependent variable is

$$y_i = \begin{cases} y_i^* + \varepsilon_i & \text{if} \quad y_i^* + \varepsilon_i > 0 \\ 0 & \text{if} \quad y_i^* + \varepsilon_i \leq 0 \end{cases} \tag{12}$$

where ε_i is a normally distributed heteroscedastic error term. The heteroscedasticity is modeled as

$$\sigma_i = \exp(0.2 x_4), \qquad \text{with} \quad x_4 \sim N(0, 1). \tag{13}$$

Table 1 displays the required tuning parameters for running the GA to estimate β_1, β_2 and β_3. The tuning parameters have to be set by the user. The right column shows their values for the following Monte Carlo study. The initialization interval for all three β–parameters was set to $[-2; 2]$.

Table 1:
Set of tuning parameters for running the genetic algorithm

Parameter	Description	Parameter value
s	Population size	30
o	Number of offsprings	60
γ_0	Mutation probability	0.5
δ_0	Radiation level	1
λ_γ	Half–life duration of mutation	15
λ_δ	Half–life duration of radiation	15
T	Maximum number of generations	250
τ	Number of subsequent stagnations	10
W	Weight for the selection probability	1.0

Initially, we start with a sample of 400 observations and carry out 200 replications of each regression (Buchinsky and GA) to obtain distributions of the estimated coefficients. Afterwards, we repeat the simulation with samples of 800, 1,600 and 3,200 observations to investigate the convergence behavior of the algorithms.

The results suggest that both the Buchinsky algorithm and the genetic algorithm produce consistent estimates, although the findings are not exactly numerically identical. Table 2 displays the mean squared errors of both methods for all simulations.

Table 2:
Mean squared errors of estimated coefficients (multiplied with 100)

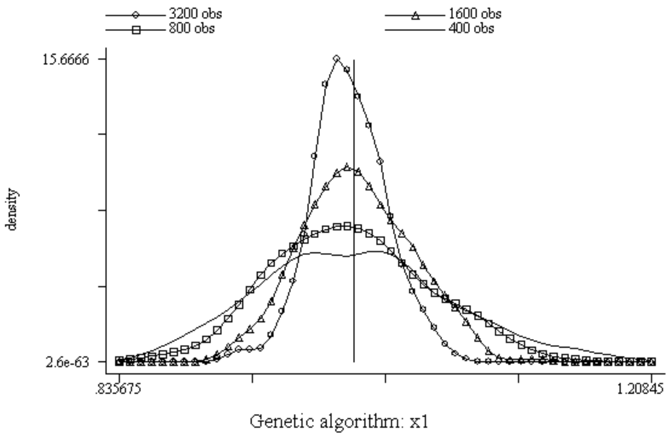
Regressor		number of observations in sample			
		400	800	1,600	3,200
x_1	GA	0.4244	0.2885	0.1468	0.0686
	Buchinsky	0.4207	0.2870	0.1468	0.0689
x_2	GA	5.8185	2.7720	1.2521	0.6561
	Buchinsky	5.9725	2.2818	1.3054	0.6497
x_3	GA	1.7721	1.0356	0.4634	0.2209
	Buchinsky	1.8326	0.9096	0.4833	0.2158

However, it is noteworthy that the Buchinsky algorithm fails to converge in about 12% of all estimations. In such cases, we have dropped 1% of the observations of the initial sample and repeated the estimation. Usually, the Buchinsky algorithm converges after dropping a few observations. However, in empirical applications with "real" data, this may cause problems as information is lost.

Figures 1 to 3 display kernel density estimates of the distributions of β_1, β_2 and β_3. Testing for normality of the estimates shows that the hypothesis "both algorithms produce asymptotically normal distributed results" has not to be rejected.[2] Figure 4 shows the same results as presented in Figures 1 to 3, but displayed as box plots of the coefficients' distributions. The line in the middle of each box represents the median of the data. The box extends from the 25th percentile to the 75th percentile, the interquartile range (IQR). Additionally, STATA computes *whiskers* which extend to the upper and lower *adjacent values*. "The upper adjacent value is defined as the largest data point less than or equal to 75th percentile + 1.5 $\times$ IQR. The lower adjacent value is defined as the smallest data point greater than or equal to 25th percentile − 1.5 $\times$ IQR. Observed points more extreme than the adjacent values, if any, are referred to as outside values and are individually plotted." (STATA Graphics Manual, Release 6, p. 32).

[2]Test results are not presented here, but are available upon request.

Figure 1: Gauss kernel density estimates of β_1



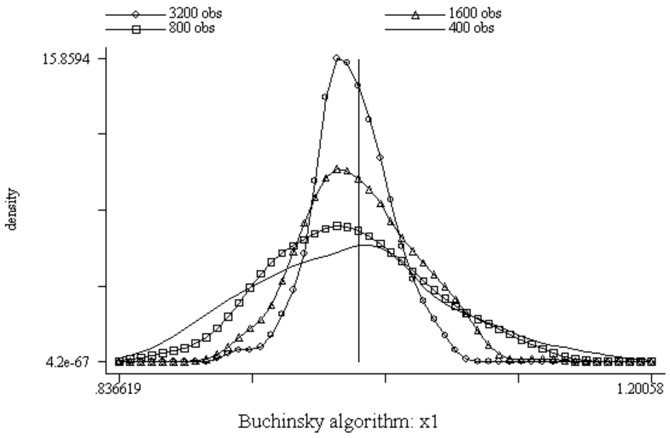
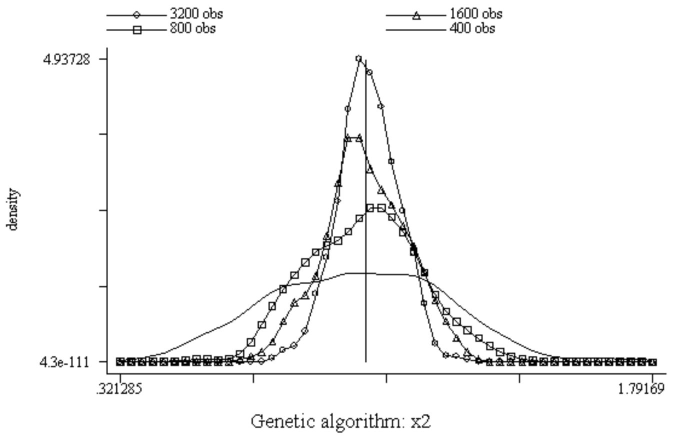
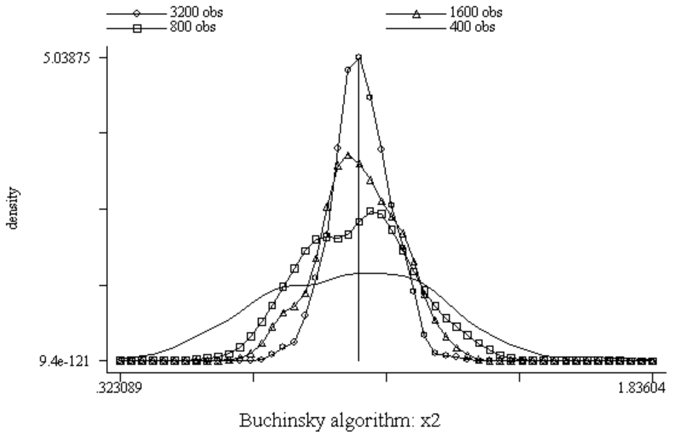


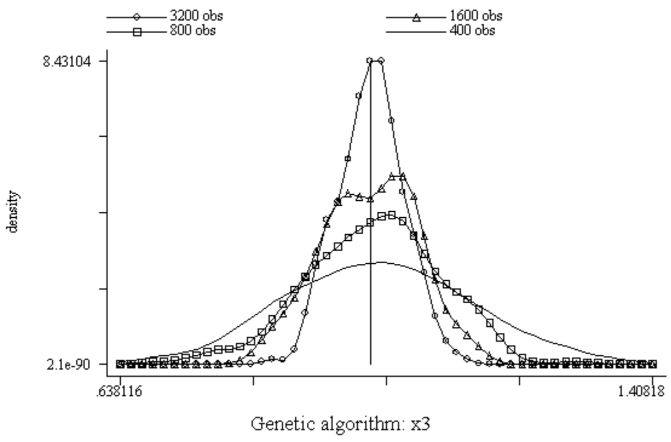
Figure 2: Gauss kernel density estimates of β_2 *





* We have dropped one data point from the results of the genetic algorithm. For the sample with 800 observations one estimated value was -.0027. As this would influence the graphical presentation of all other results, we have decided to exclude it in this figure.

Figure 3: Gauss kernel density estimates of β_3



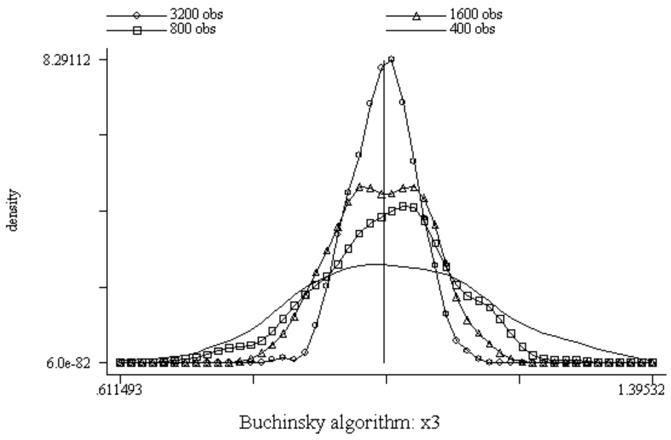
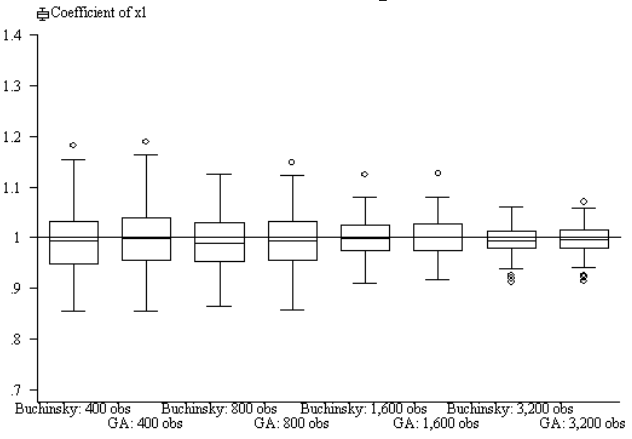
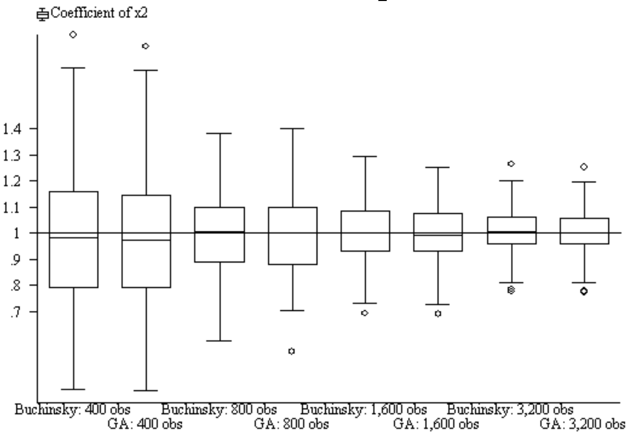


Figure 4: Box plots of estimated coefficients' distributions

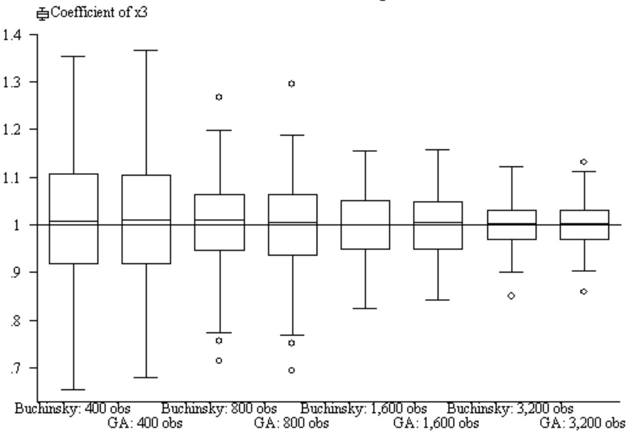
Coefficient of x_1



Coefficient of x_2



Coefficient of x_3



As the figures above show, both algorithms produce very similar distributions of results. With growing sample size, the estimates seem to converge to the true parameter values. From econometric theory, it is known that the CLAD model should exhibit a rate of convergence of $N^{-\frac{1}{2}}$ (Powell, 1984). However, for the genetic algorithm this behavior is unclear a priori as no theoretical econometric foundation exists. To investigate the convergence behavior in more detail, we calculate the estimates' variances $\hat{\sigma}_{N,\beta_k}$ and relate them to the sample sizes. Searching for α_k such that

$$\frac{\hat{\sigma}_{N,\beta_k}}{N^{\alpha_k}} = \text{constant} \qquad \forall\, k = 1,\ldots,3, \qquad N = 400, 800, 1600, 3200,$$

we can estimate the rate of convergence with the following regression model

$$\ln(\hat{\sigma}_{N,\beta_k}) = \alpha_k \ln(N) + \text{constant} + \varepsilon_k.$$

For all three α_k — the rate of convergence — the estimate should be about -0.5. To analyze the rate of convergence of the model empirically, we compare the findings of both methods. The regressions yield the following coefficients:

Table 3:
Empirical rate of convergence of both methods*

convergence α_k for	Genetic algorithm		Buchinsky	
β_1	$-.4466$	(.0442)	$-.4425$	(.0454)
β_2	$-.5286$	(.0115)	$-.5210$	(.0331)
β_3	$-.5068$	(.0257)	$-.5065$	(.0147)

* Standard errors in parentheses.

For all six cases, tests show that the hypothesis that the slope coefficient is -0.5 does not have to be rejected.[3] Hence, we conclude that both algorithms exhibit the same behavior of convergence.

4 Some practical experience

In this section we discuss some of our experience in using the GA for econometric applications. First, the tuning parameters are an important feature of the GA which may influence its performance.

1. **Population size and number of offsprings**
   In the beginning of our study, we used to create a large population, e.g.

[3]The reliability of these results may be questioned because we have only four observations from our simulations for each regression and we estimate two parameters, the constant term and the slope coefficient which represents the rate of convergence α_k.

of 100 candidate solutions and chose a smaller number of offsprings, e.g. 50. This makes the process of evolution sluggish as only a few crossovers take place, which means that it may take several generations until the fitness of the best candidate solution improves significantly. We recommend using a smaller population size, e.g. 30, and to create a larger number of crossovers, e.g. 60, as done in the presented simulation. This may require more generations until the evolution stagnates, but it is rapidly computed and the best fitness is found quickly due to a large number of trials (offsprings) to find improvements.

2. **Mutation and radiation**
   The mutation probability is an important feature to avoid local extrema. So it should not be set to zero or close to zero. Although we do not provide simulations on the effect of changing the radiation level and the half–life time of radiation and mutation, we have experimented with these parameters. The half–life time had no perceptible influence on the performance of the GA. However, we recommend using a reasonable value for each parameter, because this reduces the danger of getting stuck in local extrema. A higher level of radiation yields many trials which may be further away from possible good solutions and thus increases the number of necessary generations until stagnation.

3. **Initial parameter interval**
   The initial parameter intervals are needed to create the first population. Although we found that the mutation feature is a powerful option to get out of wrong starting intervals, we recommend to set the initial intervals to meaningful positions and widths. If the true solutions are outside the initial interval the GA will usually still find the global optima due to the mutation feature, but the convergence is slower.

4. **Weight of the selection probability**
   The weight of the selection probability W should be a rarely used feature. By default, W is set to 1. In our case, there was no need to scale down the importance of the current fitness for the selection probability. We did not encounter problems of ending up in local extrema. Usually, the evolutionary process finds the global solution. However, there may be some special cases, where the selection is desired to be completely random. For example, if multiple solutions are expected.

5 Conclusions and Perspectives

In this paper, we present the concept of a genetic algorithm. This method of optimization is based on Nature and simulates the evolution of artificial life–forms. We propose to use genetic algorithms as an alternative tool for optimizing criterion functions, especially those which are not continuously differentiable because many alternative techniques run into numerical problems in this case.

We present the implementation of a genetic algorithm developed by us and programmed in STATA (Version 7.0). As an example of its performance, we do some Monte Carlo simulations on the semiparametric censored least absolute deviations model (CLAD). We compare the results of the genetic algorithm with those of the algorithm suggested by Buchinsky (1994). We conclude that both results are similar: the mean squared errors are of equal size, the distributions of estimated coefficients look similar and the rate of convergence is almost identical in our simulations. However, the genetic algorithm is numerically more stable than its competitor.

For future applications, it would be interesting to compare the genetic algorithm with other estimation methods: for example, a simulation on the difficult optimization of maximum score models (Manski 1975, 1985, Manski and Thompson 1986) including the comparison between the genetic algorithm and the original technique for optimization suggested by Manski. Finally, an application with non–simulated data would be desirable.

Notice for interested researchers

The genetic algorithm is programmed in STATA Version 7.0 as an ado–file. If you are interested in using it yourself, the genetic algorithm is available for download at

`ftp://ftp.zew.de/pub/zew-docs/div/genetic.zip`

If you encounter problems with the download, please do not hesitate to contact us by e-mail, either `czarnitzki@zew.de` or `doherr@zew.de`. We will send the ado–file and a simple example for running the genetic algorithm.

References

Arifovic, J. (1994), Genetic Algorithm learning and the cobweb model, *Journal of Economic Dynamics and Control* 18, 3–28.

Bethke, A.D. (1980), *Genetic algorithms as function optimizers*, Ph.D. thesis, University of Michigan, Ann Arbor.

Buchinsky, M. (1994), Changes in the U.S. wage structure 1963–1987: Application of quantile regression, *Econometrica* 62(2), 405–458.

Caruana R.A. and J.D. Schaffer (1988), Representation and hidden bias: gray versus binary coding for genetic algorithms, Proceedings of the fourth international conference on machine learning.

Cooper, B. (2000), Modelling research and development: How do firms solve design problems?, *Journal of Evolutionary Economics* 10, 395–413.

De Jong, K.A. (1975), An analysis of the behavior of a class of genetic adaptive systems, Ph.D. thesis, University of Michigan, Ann Arbor.

Dorsey, R.E. and W.J. Mayer (1995), Genetic algorithms for estimations problems with multiple optima, nondifferentiability, and other irregular features, *Journal of Business & Economic Statistics* 13(1), 53–66.

Fitzenberger, B. (1994), A note on estimating censored quantile regressions, Discussion Paper 14, University of Konstanz.

Fitzenberger, B. (1997), A guide to censored quantile regressions, in: G. Maddala and C. Rao (eds.), *Handbook of statistics, Vol. 15: robust inference*, Amsterdam, 405–437.

Fitzenberger, B. and P. Winker (1999), Improving the Computation of censored quantile regressions, Discussion Paper 568–99, University of Mannheim.

Manski, C.F. (1975), Maximum score estimation of the stochastic utility model of choice *Journal of Econometrics* 3, 205–228.

Manski, C.F. (1985), Semiparametric analysis of discrete response: Asymptotic properties of the maximum score estimator, *Journal of Econometrics* 27, 313–333.

Manski, C.F. and T.S. Thompson (1986), Operational characteristics of maximum score estimation, *Journal of Econometrics* 32, 85–108.

McManus, W.S. (1985), Estimates of the deterrent effect of capital punishment: the importance of researcher's prior beliefs, *Journal of Political Economy* 93, 417–425.

Mitchell, M. (1996), *An introduction to genetic algorithms*, Cambridge.

Powell, J.L. (1984), Least absolute deviations for the censored regression model, *Journal of Econometrics* 25, 303–325.

Powell, J.L. (1994), Estimation of semiparametric models, in: R.F. Engle and D.L. McFadden, *Handbook of econometrics IV*, New York.

Price, T.C. (1997), Using co–evolutionary programming to simulate strategic behaviour in markets, *Journal of Evolutionary Economics* 7, 219–254.

STATA Corporation (1999), *STATA Graphics Manual, Release 6*, College Station.

Varetto, F. (1998), Genetic algorithms applications in the analysis of insolvency risk, *Journal of Banking and Finance* 22, 1421–1439.