Genetic algorithms for econometric optimization[1]

Dirk Czarnitzki[a,b,c] and Thorsten Doherr[c]

[a] *K.U.Leuven, Dept. of Managerial Economics, Strategy and Innovation, Belgium*
[b] *Centre for R&D Monitoring (ECOOM) at K.U.Leuven*
[c] *Centre for European Economic Research (ZEW), Mannheim, Germany*

This Version: May 2009
First Version: July 2002

Abstract

This paper discusses a tool for optimization of econometric models based on genetic algorithms. Due to the increasing popularity of semi–parametric estimators, researchers often have to optimize non–differentiable, non–smooth criterion functions that cannot be solved with conventional gradient methods. Genetic algorithms constitute an alternative opportunity for optimization. So far, however, applications are not wide–spread, as there is no software readily available yet. This paper overcomes this limitation by describing a genetic algorithm implemented in STATA that can easily be dynamically linked to any criterion function to be optimized. We describe the algorithm, provide two example applications, and provide a step–by–step guide on how to use the software.

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

Address:	Dirk Czarnitzki K.U.Leuven Dept. of Managerial Economics, Strategy and Innovation Naamsestraat 69 3000 Leuven Belgium	Thorsten Doherr Centre for European Economic Research (ZEW) Dept. of Industrial Economics and International Management P.O.Box 10 34 43 68034 Mannheim Germany
Phone:	+32 16 326 906	+49 621 1235–291
Fax:	+32 16 326 732	+49 621 1235–170
E-Mail:	dirk.czarnitzki@econ.kuleuven.be	doherr@zew.de

[1]Helpful comments by François Laisney, James L. Powell, two anonymous referees and the editor 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 are the results of an optimization process called evolution. 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 for using 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 emergence of markets in an economy, transport or packaging problems in logistics, or the determination of weights for neural networks. These, by no means exhaustive examples, demonstrate the versatility of GAs. Goldberg (1989) provides a survey of existing applications of GAs in different fields. Among many other fields, GAs have already been used in economic research[2] and econometrics. In the latter field, Dorsey and Mayer (1995) were the first to apply GAs to 11 "test problems" taken from econometric literature. These consider highly non–smooth and non–differentiable criterion functions where commonly used algorithms are likely to fail. Among others, they use a non–linear least squares example of Judge et al. (1985) where conventional methods like the Newton–Raphson algorithm fail to converge to the global minimum (the function has two local minima). Furthermore, they consider disequilibrium models as used by Maddala and Nelson (1974) and Mayer (1989), and also demonstrate the usefulness of Genetic Algorithms to estimate, for instance, semi–parametric econometric models, such as the Maximum Score estimator of Manski (1975, 1985). Östermark (1999) extends the concepts of GAs to hybrid GAs where a GA is combined with features of constrained non–linear programing techniques.[3]

While the literature clearly demonstrates the usefulness of genetic algorithms in the field of econometrics, heuristic techniques, such as genetic

[2]Examples are Axelrod, R. (1987), Marimon et al. (1990), Arifovic (1994), Price (1997), Varetto (1998), or Cooper (2000).

[3]See Corana et al. (1987) and Goffe et al. (1994) for a related non–gradient iterative method. They both apply simulated annealing to four different estimation problems, but Goffe et al. extend the original method of Corana et al. in three different dimensions.

algorithms, are not commonly used yet. For instance, Cameron and Trivedi (2005) mention the upcoming importance of genetic algorithms and simulated annealing, but only briefly reference to these techniques in their recent textbook. An important reason for the scarce use of such procedures may the lack of readily available software for the implementation of these techniques for a broad range of optimization problems. So far, genetic algorithms have been implemented by scholars only for their specific problems in various different programming languages. The goal of our contribution is not an application of a genetic algorithm to a specific econometric problem, but the introduction of a research tool for the implementation of genetic algorithms to a wide range of econometric problems. We have programmed a highly flexible genetic algorithm with the environment of the econometric software package STATA. Our program code is available as an "ado" file that can be freely downloaded from the internet. Scholars can dynamically link any desired criterion function to be optimized to our algorithm so that the estimation of highly complex models only requires little more programming than a standard regression command (the criterion function has to be specified by the researcher).

The next section explains the concept of GAs and describes the design of a genetic algorithm which we have developed for the estimation of econometric models. In section 3, we apply this GA to censored least absolute deviation models for illustrative purpose. The performance of the GA is compared to the iterative linear programming algorithm that is frequently used for the computation of this estimator. In section 4, we apply the GA to compute a maximum rank correlation estimator for which currently no ready-to-use software exists to the best of our knowledge. Section 5 concludes. Appendix A describes a step–by–step guide for using our algorithm in STATA. Appendix B includes a classic textbook example of a non–linear regression, where the conventional Newton–Raphson algorithm fails to find the global optimum. We provide an example program where we show that the genetic algorithm always converges to the global optimum. This example may be useful for interested researchers who would like to explore the trade–off between convergence and computation time with different sets of tuning parameters.

2 A genetic algorithm

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). A process similar to this is known as evolution in nature. Learning from nature and finding improved elements of a complex space is incorporated into a formal method of optimization called genetic algorithm.

This section describes the concept of GAs and explains our particular implementation of the method. It should be noted that 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. 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, a statistical software package with a flexible programming language.[4] A major advantage of our software over existing applications of genetic algorithms is that any criterion function to be optimized can be dynamically linked to the GA. In contrast to other studies using GAs for optimization, there is no need to re–program the GA for a specific problem. The integration of the GA into other STATA programs can easily be achieved. The implementation is described below.

The starting point is a criterion function to optimized with respect to some unknown parameter values. A well known example is the linear regression model,

$$y_i = \beta' \mathbf{x}_i + \varepsilon_i, \quad \text{with } i = 1, \ldots, N, \tag{1}$$

where y is the vector of the dependent variable, $\mathbf{x}$ is a set of covariates, β the corresponding parameters to be estimated and ε is an i.i.d. error term. The ordinary least estimator is a convenient method to estimate β. In this case, the criterion function to be optimized with regard to β is

$$\beta = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \beta' \mathbf{x}_i)^2 \tag{2}$$

[4] The software is compatible with STATA version 9.0 and above.

In the language of GAs, this is called the fitness function. A sketch of the procedure performed by a genetic algorithm can be described as follows:[5] The algorithm starts with a fixed number of different random candidate solutions for β (the population) and evaluates the fitness function with respect to each candidate solution. Depending on the "fitness" (the fit) of the different solutions, pairs of vectors are chosen to generate a new population. An example of such a "crossover" would be taking the mean value of each pair of elements in the two selected vectors and use the mean vector as one new candidate solution in a new set of "offsprings". Once again the fitness function is evaluated for the initial population and its offsprings. According to the fitness of each solution, the group of "survivors" is determined, and the crossover and offspring creation is repeated. The GA converges either when a certain number of iterations have been computed (generations) or no offsprings enter the survivors for a number of trials set by the researcher (stagnation). Let us now consider each step in more detail:

– Creation of the initial population

The "population" denotes a set of candidate solutions for a particular optimization problem, e.g. a set of vectors of different parameter values. 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

$$f_i = f(\beta_{i1}, \ldots, \beta_{ik}) \qquad \text{with} \quad i = 1, \ldots, s. \tag{3}$$

The elements β_{ij} $(j = 1, \ldots, k)$ 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.

– 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 $\tau \leq T$) .

[5]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.

– Determination of the mutation probability and radiation level

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 elements in an offspring. 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".

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

$$\gamma_t = \gamma_0 \exp\left(\frac{-\ln(2)}{\lambda_\gamma} t\right) \qquad \text{with} \quad t = 1, \ldots, T, \tag{4}$$

$$\delta_t = \delta_0 \exp\left(\frac{-\ln(2)}{\lambda_\delta} t\right) \qquad \text{with} \quad t = 1, \ldots, T, \tag{5}$$

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.

Changing the mutation and radiation levels within the iteration process is based on following idea: initially it is desirable to search a broad space for candidate solutions even outside the interval of parameter starting values. During the optimization, however, potential candidates are found quickly, and then is is desirable to achieve convergence fast. Reducing the mutation and radiation level over time facilitates these two goals. Thus after an initial broad search the algorithm focuses on fine tuning the best parameter solutions. Note, however, that our software does also allow to fix mutation and radiation levels throughout the optimization process at any desired level if requested by the researcher.

– Determination of the selection probability

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 (cf. Mitchell 1996), and this is followed here, too.

A reproduction probability ω_i has to be associated with 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:

$$\text{offset} = \frac{\max(f_1, \ldots, f_s) - \min(f_1, \ldots, f_s)}{s} \tag{6}$$

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

$$h_i = f_i - \min(f_1, \ldots, f_s) + \text{offset}. \tag{7}$$

The selection probability is defined as

$$\omega_i = \frac{h_i}{\sum_{i=1}^{s} h_i}. \tag{8}$$

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 reproduction probability. By default, $W = 1$ which means that the reproduction 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 reproduction probability is calculated as a the convex combination

$$\omega_i^* = (1 - W)\frac{1}{s} + W\omega_i. \tag{9}$$

– **Evolution**

1. Two different candidate solutions are drawn out of the population according to the selection probabilities ω_i (optionally weighted by W).

2. Crossover is applied by a randomly weighted mean for each element pair of the drawn candidates. The weight is uniformly distributed on the interval $[0; 1]$.[6]

3. Each element of the resulting offspring vector is mutated according to the mutation probability γ.

4. For every element selected for mutation, the radiation level determines the interval $[-\delta; \delta]$ of the random uniformly distributed mutation factor. Thus, the parameters are adjusted as follows:

$$\tilde{\beta}_{ik} = \beta_{ik} + \beta_{ik}(\delta - 2\delta U),$$

where $U \sim U(0, 1)$.

5. The offspring is evaluated by the fitness function.

6. 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.

7. The criterion of stagnation (see "main loop" above) is fulfilled when the new generation contains no offspring.

3 An application using the CLAD estimator

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).[7] Consider the econometric model

$$y_i^* = \beta' \mathbf{x}_i + \varepsilon_i, \tag{10}$$

[6] A simpler version would be to attach a weight of 0.5 to all cases. This would slightly improve computation time. The trade–off is, however, that the risk of getting stuck in a local optimum would be larger. Suppose two "parent vectors" are close to two local optima that are of similar fit. If we would always take a mean, the offspring vectors would always be further away from the best solution and be discarded immediately. In addition, the parent vectors would not be fine–tuned around the local optima which may result in non–convergence.

[7] 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.

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

$$y_i = \begin{cases} y_i^* & \text{if} \quad y_i^* > 0, \\ 0 & \text{if} \quad y_i^* \leq 0. \end{cases} \tag{11}$$

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

$$\beta_{CLAD} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left| y_i - \max(0, \beta' \mathbf{x}_i) \right|. \tag{12}$$

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).[8]. The practical problem of non–convergence of the MILPA 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 global minimum.[9] 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 this phenomenon, we compare the performance of the MILPA with the GA in CLAD regression models. We choose a simple case of a model and carry out Monte–Carlo simulations. Consider the true model

$$y_i^* = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i}, \qquad i = 1, \ldots, N, \tag{13}$$

The explanatory variable x_1 is standard normally distributed and x_2 is uniformly distributed on the interval [0,1]. For simplicity, we set $\beta_0 = \beta_1 = \beta_2 = 1$. The observed dependent variable is

$$y_i = \begin{cases} y_i^* + \varepsilon_i & \text{if} \quad y_i^* + \varepsilon_i > 0 \\ 0 & \text{if} \quad y_i^* + \varepsilon_i \leq 0 \end{cases} \tag{14}$$

[8]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)

[9]Fitzenberger (1997: 180) states: " When ILPA does not converge, it typically oscillates between two or three coefficient vectors. If ILPA does not converge, it must oscillate, since a finite sample allows only for a finite number of subsamples which a standard quantile regression can be based upon. Oscillation arises, since an observation, for which the current standard quantile regression implies a censored fitted value, still contributes to the distance function. In the next iteration, this observation is excluded from the sample, which can result in a new estimate for which the fitted value at the aforementioned observation is now uncensored."

where ε_i is the error term.

Table 1 displays the required tuning parameters for running the GA to estimate β. 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 GA tuning parameters for the CLAD estimation

Parameter	Description	Parameter value
s	Population size	30
o	Number of offsprings	60
γ_0	Mutation probability	1
δ_0	Radiation level	1
λ_γ	Half–life duration of mutation	40
λ_δ	Half–life duration of radiation	40
T	Maximum number of generations	500
τ	Number of subsequent stagnations	20
W	Weight for the selection probability	1.0

We carry out Monte–Carlo simulations for four different types of error terms ε_i,

1. a standard normally distributed error term: $\varepsilon_i \sim N(0, 1)$;
2. a heteroscedastic error term, where the standard error is modeled as

$$\sigma_i = \exp(0.2x_4), \qquad \text{with} \quad x_4 \sim N(0, 1); \tag{15}$$

3. a skewed error term, $\varepsilon_i = \exp(z)$ where $z \sim N(0, 1)$;
4. and a uniformly distributed error term, such that $\varepsilon_i \sim U[-0.5; 0.5]$

where we always impose the restriction that $MED(\varepsilon_i) = 0$ to satisfy the CLAD model's assumptions.

We run the four set–ups with samples of $N = 50, 100, 1000, 10000$ and carry out 200 replications for each simulation using the MILPA and the GA to estimate the model. Instead of reporting the coefficient estimates, we just report wether the algorithm converged to the global minimum and the average computation time in seconds. We declare convergence to be achieved if the found minimum is in tolerance of 10^{-7} of the true minimum.

Table 2 shows the simulation results. The GA results in less failures with respect to achieving convergence than the MILPA in all simulations. For instance, in the simulation with a skewed error term and $N = 50$, the GA fails to converge once out of 200 trials, whereas the MILPA fails 48 times, i.e. it does not converge in 24% of the simulations. However, 43 out of the 48 failures are due to the aforementioned problem of oscillating. Conditional on the not oscillating the performance of the GA and the ILPA becomes comparable, but the GA fails still less, on average. Therefore, we can conclude that the GA performs very well with respect to finding the global minimum when compared to a more conventional algorithm. That does not come without cost, however.

As can be seen in Table 2, the computation time of the GA is always larger then that of the MILPA. In the simulation with the skewed error term, the average computation time of the GA increases from 2.4 seconds for $N = 50$ to 94 seconds for $N = 10,000$, whereas the corresponding numbers of the MILPA are only 0.35 seconds to 14 seconds.

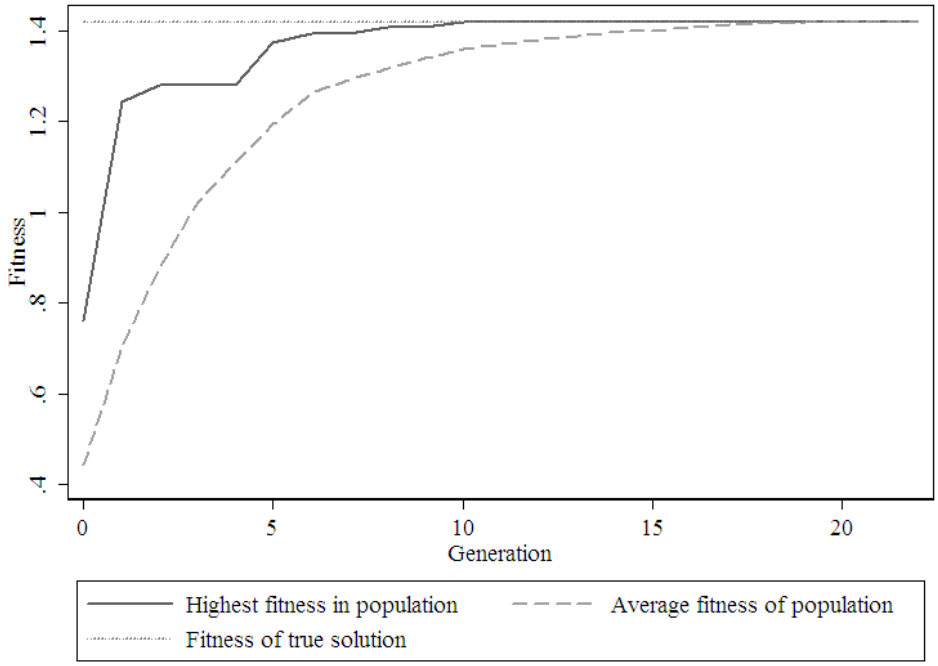
Table 2:
Simulation results for the CLAD estimator

	$N = 50$	$N = 100$	$N = 1,000$	$N = 10,000$
Normally distributed, homoscedastic error term				
GA: # of failures	0	1	0	0
GA: computation time	2.295	2.715	8.605	97.080
MILPA: # of failures (loop)	38(30)	17(17)	22(21)	25(25)
MILPA: computation time	0.340	0.265	0.895	12.950
Normally distributed, heteroscedastic error term				
GA: # of failures	1	1	0	1
GA: computation time	2.430	2.605	8.880	94.410
MILPA: # of failures (loop)	30(29)	27(27)	30(30)	29(29)
MILPA: computation time	0.350	0.370	1.225	14.425
Skewed error term				
GA: # of failures	1	2	0	1
GA: computation time	2.355	2.595	9.100	93.720
MILPA: # of failures (loop)	48(43)	36(33)	36(35)	29(29)
MILPA: computation time	0.500	0.475	1.515	19.765
Univariately distributed error term				
GA: # of failures	3	1	3	2
GA: computation time	2.330	2.580	9.695	95.900
MILPA: # of failures (loop)	20(18)	6(6)	17(17)	17(17)
MILPA: computation time	0.225	0.070	0.720	8.360

Notes: Each result is based on 200 replications. Computation time is reported as average of the 200 replications in seconds. The MILPA number of failures in brackets report the number of failures out of total failures that are due to oscillating around a minimum. The simulations were performed using STATA 10 on a machine with a Pentium D 3.20GHz processor.

For an explanation of the longer computation time, Figure 1 shows a typical convergence behavior of the GA during the simulation with $N = 1000$ and a normally distributed homoscedastic error term. The health of the population, measured as the average fitness, follows a growing group of dominating solutions towards the global maximum. Even if there is no distinguishable difference between the health of the population and the highest fitness, an exchange happens between the population and the offspring, considerable delaying the stagnation of the GA. Even so we defend the usage of the stagnation as a stopping criterion for the GA because of its problem independency. Furthermore, the stagnation tendency can always be increased by reducing the precision of the fitness. A solution space with a lower precision is less prone to time-consuming fine tuning of the population.

Figure 1: Typical pattern of GA convergence



4 The maximum rank correlation estimator

In addition to the CLAD example, this section presents a small simulation study on a binary choice model for which no readily available optimization algorithm exists so far. Han (1987) proposed the maximum rank correlation (MRC) estimator and shows a variety of possible applications. We choose the binary choice model in this case, because the criterion function is highly non–smooth and there is so far no standard software package which offers MRC estimation in this context, to the best of our knowledge. Let

$$y_i^* = x_i'\beta + \varepsilon_i, \tag{16}$$

where we observe

$$y_i = \begin{cases} 1 & \text{if} \quad x_i'\beta + \varepsilon_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

The MRC estimator chooses β to maximize

$$\begin{aligned} Q_H(\beta) = \frac{1}{N(N-1)} \sum_{j=1}^{n} \sum_{i \neq j} & I(y_i > y_j) I(x_i'\beta > x_j'\beta) \\ & + I(y_i < y_j) I(x_i'\beta < x_j'\beta), \end{aligned} \tag{18}$$

where the summation is taken over all combinations of two distinct elements (i, j) from $i = 1, \ldots, N;\ j = 1, \ldots, N$. The idea of the estimator is the following: $P(y_i \geq y_j | x_i, x_j) \geq P(y_i \leq y_j | x_i, x_j)$ if $x_i'\beta_0 \geq x_j'\beta_0$, making it likely that $y_i \geq y_j$ whenever $x_i'\beta_0 \geq x_j'\beta_0$, so that a high correlation between the rank of y_i and the rank of $x_i'\beta$ should be observed when $\beta = \beta_0$, whereas a lower value would be exhibited when β departs from β_0 (see Pagan and Ullah, 1999). It has been shown that this estimator is $\sqrt{N}$–consistent and asymptotically normal (Sherman, 1993).

We choose the same setup for our simulation as given in eq. (13), and also use the same four scenarios regarding the error term (see Section 3). Note, however, that the constant term is not identified in the MRC estimator, as it is scale invariant. The intercept is thus simply dropped here. Table 3 displays the GA's tuning parameters used to produce the simulation results shown in Table 4. Convergence is achieved when the GA predicts an equal number (or more) correct ranks than the number of correct predictions obtained with the true coefficient values.

Table 3:
Set of GA tuning parameters for the maximum rank correlation estimation

Parameter	Description	Parameter value
s	Population size	30
o	Number of offsprings	60
γ_0	Mutation probability	1
δ_0	Radiation level	2
λ_γ	Half–life duration of mutation	20
λ_δ	Half–life duration of radiation	25
T	Maximum number of generations	500
τ	Number of subsequent stagnations	20
W	Weight for the selection probability	1.0
	Interval of starting values	[-2,2]

As can be seen in Table 4, the GA converges to the global minimum in all simulation runs except in three cases. The computation time becomes substantial in the simulations with 10,000 observation, though. A single estimation roughly amounts to three minutes then. Nevertheless, it becomes obvious that the GA offers an opportunity to successfully estimate parameter values for models with non–differentiable, highly non–smooth criterion functions where no other readily available optimization algorithm exists. As the MRC estimator has not been applied frequently yet, we also report the mean squared errors of the coefficient estimates which are in line with the theoretically expected rate of convergence.

Table 4:
Simulation results for the maximum rank correlation estimator

	$N = 50$	$N = 100$	$N = 1,000$	$N = 10,000$
Normally distributed, homoscedastic error term				
# of failures	0	0	0	0
computation time	1.380	2.110	16.405	190.185
$MSE(\beta_1)$	1.1389	0.4237	0.0471	0.0042
$MSE(\beta_2)$	0.9726	0.1491	0.0102	0.0010
Normally distributed, heteroscedastic error term				
# of failures	0	0	1	0
computation time	1.445	2.190	16.205	190.225
$MSE(\beta_1)$	1.1328	0.4612	0.0427	0.0039
$MSE(\beta_2)$	0.9990	0.1878	0.0080	0.0007
Skewed error term				
# of failures	0	1	0	0
computation time	1.455	2.160	15.740	182.600
$MSE(\beta_1)$	0.8889	0.4889	0.0403	0.0036
$MSE(\beta_2)$	0.3324	0.0959	0.0106	0.0012
Univariately distributed error term				
# of failures	0	0	0	1
computation time	1.335	1.945	14.975	171.505
$MSE(\beta_1)$	0.2846	0.1384	0.0093	0.0007
$MSE(\beta_2)$	0.0781	0.0307	0.0022	0.0002

Notes: Each result is based on 200 (=R) replications. Computation time is reported as average of the 200 replications in seconds. The simulations were performed using STATA 10 on a machine with a Pentium D 3.20GHz processor. The Mean Squared Error was calculated as $MSE = 1/R \sum_{r=1}^{R} (\beta_r - \beta)^2$.

5 Conclusions

This paper describes a genetic algorithm implemented in STATA (compatible with Version 9.0 and above) that can be downloaded from the Internet or be obtained from the authors directly. On the background of the growing development and applications of semiparametric and non–parametric estimators in econometrics, heuristic methods such as genetic algorithms become increasingly more important in empirical research. Our paper tries to bridge the gap between available (theoretical) knowledge on genetic algorithms and its advantages and its actual availability in common software packages. We provide an implementation of a fairly general concept of genetic algorithms that can be linked easily to any criterion function to be optimized by a researcher. A set of various tuning parameters allows researchers to adjust the optimization procedure to a wide range of specific combinations of data structure and a particular estimation problem.

Our small Monte–Carlo simulation examples mainly meant for illustrative purpose of the power and flexible use of the GA demonstrate the performance of the algorithm. The following appendix describes a step–by–step guide for using our research tool in STATA along with some anecdotal recommendations based on our own experience with the algorithm that has not entered the simulations presented above.

For further research on the performance of our research tool it would be desirable to collect evidence on the algorithm's performance on a broad set of econometric problems, and more detailed simulations on the importance and implications of choosing particular values for the tuning parameters.

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.

Cameron, A.C. and P.K. Trivedi (2005), *Microeconometrics: methods and applications*, Cambridge University Press.

Corana, A., M. Marchesi, C. Martini and S. Ridella (1987), Minimizing multimodal functions of continuous variables with the 'simulated annealing algorithm', *ACM Transactions on Mathematical Software* 13, 262–280.

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.

Goffe, W.L., G.D. Ferrier and J. Rogers (1994), Global optimization of statistical functions with simulated annealing, *Journal of Econometrics* 60, 65–99.

Goldberg, D.E. (1989), *Genetic algorithms in search, optimization and machine learning*, Reading, MA: Addison–Wesley.

Han, A.K. (1987), Non–parametric analysis of a generalized regression model: the maximum rank correlation estimator, *Journal of Econometrics* 35, 303–316.

Judge, G.G., W.E. Griffiths, R.C. Hill, H. Ltkepohl and T.C. Lee (1985), *The theory and practice of econometrics*, 2nd ed., New York: John Wiley.

Maddala, G.S. and F.D. Nelson (1974), Maximum likelihood methods for models for markets in disequilibrium, *Econometrica* 42, 1013–1030.

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.

Mayer, W.J. (1989), Estimating disequilibrium models with limited a priori price–adjustment information, *Journal of Econometrics* 41, 303–317.

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.

Östermark, R. (1999), Solving irregular econometric and mathematical optimization problems with a genetic hybrid algorithm *Computational Economics* 13, 103–115.

Pagan, A. and A. Ullah (1999), *Nonparametric econometrics*, Cambridge: Cambridge University Press.

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.

Sherman, R.C. (1993), The limiting distribution of the maximum rank correlation estimator, *Econometrica* 61, 123–137.

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

A A step–by–step guide for the GA's STATA syntax

When using our GA for the first time, download the GA at

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

.
The `zip` file contains `dna.ado` and `dna.hlp` which have to be copied into `c:\ado`.

Furthermore, the file contains three sample progams that run

1. an example of the Censored Least Absolute Deviation estimation as shown in Section 3 of this paper;
2. an example of the Maximum Rank Correlation estimation as shown in Section 4 of this paper;
3. and a classical example taken from Judge et al. (1985). It demonstrates a simple nonlinear optimization problem where a standard algorithm like Newton–Raphson fails to find the global optimum but gets stuck in a local minimum. This example is described in some more detail in Appendix B of this paper.

If you encounter problems with the download, please do not hesitate to contact us by e-mail, either `dirk.czarnitzki@econ.kuleuven.be` or `doherr@zew.de`.

The STATA GA ado–file allows for an easy integration of self written commands into its syntax. In the case of the GA this command is called `dna`. The parametrization of the GA is achieved over multiple calls of this command followed by a sub command and its parameters separated by blanks. All settings are saved in global macros starting with the letters "DNA". If a macro does not exist it will be generated with a default setting. The complete syntax and additional information on the macros and default settings is provided in the obligatory help file that can be evoked by typing `help dna` at the STATA command line.

This step by step guide uses our actual parametrization of the GA for the CLAD estimation example. We start with the definition of the fitness function:

```
cap program drop cladfit
program define cladfit
    tempvar score xb
    qui gen double ‘xb’ = x1*dna[1,1] + x2*dna[1,2] + dna[1,3]
    qui replace ‘xb’ = max(‘xb’,0)
    qui egen double ‘score’ = sum(abs(y-‘xb’))
    scalar hit = _N/‘score’
    qui matrix dna[1,$DNAFIT] = hit
end
```

This is the fitness function for the CLAD estimation. A potential solution is copied into the row vector `dna` to be evaluated by the function. The last element off this vector is always reserved for the fitness value of the solution. The global macro `DNAFIT` contains this row number for your convenience.

```
dna clear
```

This command deletes all `DNA` macros, resulting in the usage of the default settings.

```
dna function cladfit
```

The `function` command associates the fitness function to the GA.

```
dna length 3
```

It is required to define the size of a candidate vector. The function `cladfit` defines the regression model with 3 coefficients.

```
dna population 30
```

The population defines the number of candidate vectors in the base population.

```
dna offspring 60
```

60 offspring vectors will be created during each evolutionary step. In this example the intermediate population will consist of 90 candidates. Only the best 30 candidates as evaluated by the fitness function remain in the base population.

From our own experience, we recommend to choose a relatively small size of the population. If one uses a large population, e.g. of 100 candidate

solutions and chooses a smaller number of offsprings, e.g. 50, the process of evolution becomes 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 small 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.

```
dna mutation 1 40
```

The command `mutation` specifies the probability of a mutation. The first parameter sets the starting value of this probability to 1. Every element of an offspring will be mutated. The second parameter defines the half–life. After 40 generations the mutation probability is reduced to 0.5. If the second parameter is omitted or zero the mutation probability stays constant.

```
dna radiation 1 40
```

The effect of a mutation is determined by the radiation level (the first parameter). A level of 1.0 results in a positive or negative shift by up to 100% of the original value. Like the `mutation` command the `radiation` command has an additional parameter to specify the half–life.

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. Higher values for each parameter reduce the danger of getting stuck in local extrema. However, a higher level of radiation yields many trials that can be relatively far away from "good" solutions and thus increases the number of necessary generations until stagnation.

```
dna define 1 -2 2
dna define 2 -2 2
dna define 3 -2 2
```

With the `define` command it is possible to specify a separate initilization range for every element of the candidate vectors in the base population. The first parameter is the column index of the element. The other two

parameters define the low and high range for the starting value.

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 still find the global optimum due to the mutation feature, but the convergence is slower.

```
dna generations 500
```

One way to stop the iterations of the algorithm is to define a maximum number of generations. In this example the GA stops at a maximum of 500 generations.

```
dna stagnation 20
```

This is another way the stop the algorithm. After 20 generations without any offsprings that are fit enough to claim a place in the base population the artificial evolution is stagnated and stops.

```
dna selection 1
```

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.

```
dna info
```

```
-------------------------------------------------------------------
          Description | Macro    | Value
----------------------+----------+-----------------------------------
            parameter | DNALEN   | 3
           population | DNAPOP   | 30
            offspring | DNAOFF   | 60
            selection | DNASEL   | 1
```

```
             mutation | DNAMUT   | 1
  half-life: mutation | DNAMHL   | 40
           radiation | DNARAD   | 1
 half-life: radiation | DNARHL   | 40
        mutation mode | DNAMOD   | auto
     max. generations | DNAGEN   | 500
           stagnation | DNASTA   | 20
         display mode | DNADIS   | display
----------------------+----------+----------------------------------
Parameter 1       low | DNAL1    | -2
                 high | DNAH1    | 2
                group | DNAG1    | 0
----------------------+----------+----------------------------------
Parameter 2       low | DNAL2    | -2
                 high | DNAH2    | 2
                group | DNAG2    | 0
----------------------+----------+----------------------------------
Parameter 3       low | DNAL3    | -2
                 high | DNAH3    | 2
                group | DNAG3    | 0
-------------------------------------------------------------------
```

This command displays all settings, the corresponding macros and their values. As stated above, additional information about the syntax and macros can be retrieved by consulting the provided help file.

`dna go`

Up to this point the GA only consists of global macros for the settings. This command starts the actual iteration process using these settings. The population resides in the matrix `pop` consisting of DNAPOP rows and DNALEN+1 columns. The last column contains the fitness of the corresponding row. After completition `pop` is sorted in descending order by the fitness. The interface vector to the fitness function `dna` contains a copy of the top solution. Both matrices can easily be accessed by using the STATA `matrix` command, i.e. `matrix list dna`.

B The Judge et al. (1985) example

Judge et al. (1985) illustrate the potential pitfalls of the Newton Algorithm with a simple nonlinear statistical model. Let

$$y_i = \theta_1^* + \theta_2^* x_{i2} + \theta_2^{*^2} x_{i3} + e_i, \quad i = 1, \ldots, 20. \tag{19}$$

$$y = f(\theta^*) + e, \tag{20}$$

where $y = (y_1, y_2, \ldots, y_{20})'$, $\theta^* = (\theta_1^*, \theta_2^*)'$ is the true parameter vector, $e = (e_1, e_2, \ldots, e_{20})'$ and

$$f(\theta) = \begin{bmatrix} \theta_1 + \theta_2 x_{12} + \theta_2^2 x_{13} + e_i \\ \theta_1 + \theta_2 x_{22} + \theta_2^2 x_{23} + e_i \\ \vdots \\ \theta_1 + \theta_2 x_{20,2} + \theta_2^2 x_{20,3} + e_i \end{bmatrix} \tag{21}$$

The data used for this example are given in Judge et al. (1985: 956) and is also included in the file `newton.txt` (tab–delimited). The true parameters are $\theta_1^* = \theta_2^* = 1$.

To determine the least squares estimate of θ^*, we have to minimize

$$H(\theta) = [y - f(\theta)]'[y - f(\theta)]. \tag{22}$$

If the Newton–Raphson algorithm is used, it terminates at two different points in the parameter space, which is not surprising as $H(\theta)$ has two different local minima. The estimates found by the Newton–Raphson may either be

1. $\theta_1 = 0.864787$, $\theta_2 = 1.235748$ with $H(\theta) = 16.0817$,
2. $\theta_1 = 2.354471$, $\theta_2 = -0.319186$ with $H(\theta) = 20.9805$.

Judge et al. (1985) also show that this problem cannot be circumvented with different starting values for the optimization process when the Newton algorithm is used.

The file `newton.do` runs the estimation using he GA, and the interested user will find that the GA does not terminate at the local optimum $H(\theta) = 20.9805$, but always converges to the global optimum at $H(\theta) = 16.0817$.