

UNIVERSITÀ DEGLI STUDI DI PADOVA
Facoltà di Scienze Statistiche
Corso di Laurea Triennale in Statistica e Gestione delle Imprese



Tesi di Laurea

ALGORITMI GENETICI PER LA STATISTICA

Relatore: Ch.mo prof. MONICA CHIOGNA

Laureanda: GIULIA COSTANTINI

Anno Accademico 2006-2007

Contents

Introduction	7
Chapter 1 - Genetic algorithms: structure and functioning	11
1.1 - Genetic algorithms	11
1.1.1 Definition	11
1.1.2 History	11
1.1.3 Holland's GA	12
1.1.4 GAs' applications	13
1.2 - The basic elements of GAs	14
1.2.1 Chromosomes' population	14
1.2.2 Fitness function	15
1.2.3 Selection	16
1.2.4 Single point crossover	19
1.2.5 Mutation	21
1.3 - The simple genetic algorithm (SGA)	21
Chapter 2 - A first, simple, application of SGA	27
2.1 - Maximization of a univariate function	27
2.1.1 Problem's definition	28
2.1.2 Chromosome's encoding	29
2.1.3 Fitness function	29
2.2 Description of a generation	30

2.2.1	Step 1 (initialization)	30
2.2.2	Step 2 (fitness)	31
2.2.3	Step 3 (selection)	31
2.2.4	Step 4 (crossover)	32
2.2.5	Step 5 (mutation)	33
2.3	Results	33
2.3.1	Changing population size	35
2.3.2	Changing number of generations	36
2.3.3	Changing crossover rate	36
2.3.4	Changing mutation rate	37
Chapter 3 - Variable selection for regression models		39
3.1	- Variable selection	39
3.2	- AIC	41
3.3	- Problem's definition	43
3.4	- Chromosome's encoding	49
3.5	- Fitness function	49
3.6	- A comparison term	51
3.7	- Description of a generation	52
3.8	- Results	52
3.8.1	- Changing population size	59
3.8.2	- Changing crossover rate	74
3.8.3	- Changing mutation rate	83
3.9	- Solution's analysis	88
3.9.1	- Stepwise regression and GAs in comparison	95
3.9.2	- Variables' subsets comparison	103
Chapter 4 - Maximization of skew-normal's likelihood		107
4.1	- The skew-normal distribution	107

4.1.1 - A very brief history	107
4.1.2 - Formulation of the univariate form	108
4.1.3 - Examples of the univariate form	109
4.1.4 - Likelihood and Log-Likelihood functions	110
4.1.5 - Problems of the likelihood function	111
4.2 - An interesting dataset	112
4.3 - Problem's definition	113
4.3.1 - Chromosomes' encoding	113
4.3.2 - Fitness function	114
4.4 - Results	115
4.4.1 - The "one parameter" case $(\lambda, 0, 1)$	116
4.4.2 - The "three parameters" case $(\lambda, \lambda_1, \lambda_2)$	123
4.4.3 - The "two parameters" case $(\lambda, \lambda_1, 1)$	126
4.4.4 - The "two parameters" case $(\lambda, 0, \lambda_2)$	131
4.4.5 - Summary	134
4.5 - A method to acknowledge the problem	134
4.5.1 - Sliding-ranges method applied to our problem	136
4.5.2 - Sliding-ranges method applied to another problem	138
Conclusions	141
A Dispersion graphs	145
B Dataset used in paragraph 4.5.2	149

Introduction

Genetic Algorithms (GAs) are adaptive heuristic search algorithms premised on the evolutionary ideas of natural selection and genetic. The basic concept of GAs is designed to simulate processes in natural systems necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such, they represent an intelligent exploitation of a random search within a defined search space to solve a problem.

Genetic algorithms have been applied in a vast number of ways: in fact, nearly everyone can gain benefits from them, provided that:

- he/she can encode solutions of a given problem to chromosomes;
- he/she can compare the relative performance (fitness) of solutions.

An effective GA representation and meaningful fitness evaluation are the keys of the success in GA applications.

The appeal of GAs comes from their simplicity and elegance as robust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems. GAs are useful and efficient when:

- the search space is large, complex or poorly understood;
- domain knowledge is scarce or expert knowledge is difficult to encode to narrow the search space;
- no mathematical analysis is available;

- traditional search methods fail.

GAs have been used for problem-solving and for modelling. GAs are applied to many scientific, engineering problems, in business and entertainment, including optimization, automatic programming, machine and robot learning, economic models and ecological models.

The purpose of this thesis is to apply genetic algorithms in statistics and to study their performances in that field. Since statistics is a large area, we narrowed our focus on two of the most significant problems; in particular, we will consider the two problems of **variable selection** and of **likelihood maximization**.

In the first chapter genetic algorithms are introduced and their working is explored, in order to better comprehend the tools that we will be using.

In the second chapter we will see a first, simple, application of genetic algorithms: the maximization of a univariate function. During this task, we will observe in greater detail the single elements which compose a GA. While analyzing the results, we will explore some variations of the algorithm's parameters, and this should help understanding the role of each one.

In the third chapter we will start with “real” problems, and in particular with the variable selection one. We will consider a dataset with many explicative variables, too many to be all included in the regression model and most of them probably not significant enough. The task of the GA will be to choose a subset of (hopefully) meaningful variables from the original set.

In the fourth chapter we will tackle the likelihood maximization problem. We will consider a particular distribution, the **skew-normal**, and try to maximize its likelihood with a problematic dataset.

At the end we will try to draw some conclusions about the use of genetic algorithms in statistics.

Chapter 1 - Genetic algorithms: structure and functioning

1.1 - Genetic algorithms

1.1.1 Definition

A genetic algorithm (often called GA) can be defined as an heuristic and adaptive search technique, whose goal is to find true or approximate solutions to optimization and search problems. Genetic algorithms belong to a broader family of evolutionary algorithms and they lay their foundation on evolutionary biology and genetics.

1.1.2 History

Computer simulations of evolution began to spread in the 1950s. In the 1960s, Rechenberg (1965, 1973) introduced the so-called *evolutionary strategy*, and the topic was expanded further on by Schwefel (1975, 1977). Fogel, Owen and Walsh (1966) developed *evolutionary programming*, a technique whose candidate solutions were finite state machines. During those years, many other scientists worked with automatic learning and algorithms inspired by evolution: amongst them, we can find Box (1957), Friedman (1959), Bledsoe (1961), Bremermann (1962), and Reed, Toombs and Barricelli (1967).

However, the father of genetic algorithms (GAs) is surely John Holland, who invented them in the 1960s and developed them with a group of students and colleagues

in the 1960s and 1970s. Holland presented his creation in his book *Adaptation in Natural and Artificial Systems* (1975). Still, GAs' research was mainly theoretical until the mid-1980s, when the First International Conference on Genetic Algorithms was held (1985). Moreover, Holland's student David Goldberg did a great part in spreading these new methods, thanks to his studies which climaxed in the publication of *Genetic Algorithms in Search, Optimization and Machine Learning* (1989).

In the course of time, along with the involvement in these issues grew also the machines' power, making possible a constantly wider practical application of those methods.

Nowadays the GAs' diffusion is so large that a majority of *Fortune 500* companies use them to solve a wide range of problems, like scheduling, data fitting, trend spotting and budgeting problems (from Wikipedia).

1.1.3 Holland's GA

Holland's original goal, citing M. Mitchell, "*was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems*".

Holland's GA, as presented in his book (1975), is a method to pass from an initial population of chromosomes to a new one, more *fit* to the environment, using a natural selection mechanism and the genetic operators of crossover, mutation and inversion.

Each **chromosome** is made of a certain number of genes (i.e. bits); each gene is in turn the instance of an allele (i.e. 0 or 1). So, a 7-bits chromosome could present itself like this: 1011010.

The **selection** operator decides which chromosomes can reproduce, so that the best chromosomes (the fitter to the environment) produce a number of offspring larger than the worst chromosomes (the less fit).

The **crossover** operator swaps subsets of two chromosomes, with a procedure similar to that of biological recombination between two haploid (single-chromosome) organ-

isms.

The **mutation** operator randomly alters allele's value in some locations of the chromosomes.

The **inversion** operator inverts the order of a contiguous section of the chromosome.

1.1.4 GAs' applications

GAs can be successfully used in a huge range of fields, mainly computer science, engineering, economics, chemistry, physics and, last but not least, mathematics and statistics.

Two distinct purposes which can be aimed for through GAs are the following:

- *optimizing or improving the performance of operating systems*, such as a gas distribution pipeline system, traffic lights, travelling salesmen, allocation of funds to projects, scheduling, etc.... The choice of particular values for the system's parameters will lead the system to a certain performance, which can be measured through some relevant objective or fitness function. Since in realistic problems the interaction amongst the parameters aren't analitically encodable in a simple way, GAs are widely used in this field.
- *testing and fitting quantitative models*, that is searching for the parameters' values which minimize the discrepancy between a model and the data on which the model is built. In this case, the objective function is more an error function than a fitness function, and it will be a function of the difference between the observed data values and the data values that would be predicted from the model (fitted values).

The distinction between these two areas of potential use for GAs could seem a distinction between maximization (of a system's fitness) and minimization (of the discrepancy between data and a model). We have to remember, though, that maximization and minimization are always interchangeable.

1.2 - The basic elements of GAs

Despite the great confusion about the exact meaning of “genetic algorithm”, most of the methods called with that name have in common at least the following elements:

- a population composed by chromosomes;
- selection based on the fitness;
- crossover for the procreation;
- random mutation in the offspring.

Inversion (see paragraph 1.0.3) is rarely used in current implementations.

1.2.1 Chromosomes' population

Generally the chromosomes of a GA's population are bit sequences (but they could be also characters sequences, integers sequences, etc...). Each chromosome's site (bit) can assume the values (alleles) 0 or 1.

Each chromosome represents a point in the search space of the candidate solutions. In fact a generic chromosome encodes a possible solution to the problem taken into account; therefore it exists a correspondence between the search space of the solutions and the chromosomes' space. To make a very simple example about this, we can consider the following problem: finding the greatest k -bits number. The relationship between chromosomes and solutions is easy: a k -bits string (chromosome) corresponds to the number represented in binary by that string (solution). If $k = 4$, a possible chromosome is 1011, and the corresponding candidate solution will be:

$$1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 8 + 0 + 2 + 1 = 11.$$

The GA replaces each population with another one, obtained through the selection, crossover and mutation operators. Hopefully, the new population will be more fit to the environment than the previous one.

1.2.2 Fitness function

The selection operator needs (and we will see this later) an evaluation of the current population. This evaluation is performed through the so-called **fitness function**, which assigns to each population's chromosome a score (the fitness). The fitness depends on how the chromosome works out in the problem at issue; it could be a mathematical function, or an experiment, or a game. Recalling the example seen in paragraph 1.1.1, the fitness of a chromosome is simply the value of the number represented by that chromosome. In that problem, the goal was to maximize that value and therefore to maximize the fitness. A possible population (of size $n = 6$), equipped with the fitness of each chromosome, is this:

Population's individual (chromosome)	Corresponding candidate solution	Corresponding fitness value
0001	1	1
0010	2	2
0101	5	5
1001	9	9
1011	11	11
1101	13	13

Table 1: Example of chromosomes with associated candidate solutions and fitnesses

Remember two things.

- The fitness can be also a minimization, as well as a maximization. In fact the fitness is only a objective function and the goal could be to minimize it (in this case, though, it could be more appropriate to call it *cost function*, but we will refer to it always using its most common name).

- In this example, the candidate solution and its fitness value were the same “object” (a number), and, even more, they had the same numerical value. However, they can be (and in most cases they are) very different things: as we will see in Chapter 3, the candidate solution could be a linear regression model, while its associated fitness could be a goodness of fit indicator (R-squared, AIC, etc...).

1.2.3 Selection

This operator selects the population’s chromosomes for the reproduction. The basic idea is very simple: the largest is the chromosome’s fitness, the higher is its probability to procreate. In practice, though, the selection operator can present itself in wide-ranging shapes. The most common are:

- **roulette wheel selection** (or proportional selection): consider a roulette in which each population’s individual occupies a space proportional to its fitness (the largest the fitness, the higher the space taken up, therefore the bigger the probability of being extracted as parent). If f_i = fitness value of individual i , $F = \sum_{i=1}^n f_i$ = total fitness, and $f_i^c = \sum_{j=1}^i f_j$ = cumulated fitness of individual i , a possible implementation of roulette wheel selection is this:

- extract a random number r in range $(0, F)$;
- select the first population’s individual whose $f_i^c \geq r$.

It is easy to see that each individual has probability $\frac{f_i}{F}$ of being selected to be parent;

- **tournament selection**: this selection method randomly picks j population’s individuals (usually $j = 2$) and chooses amongst them the individual with largest fitness to become parent.

The procedures that we just saw select only one parent; obviously, they will have to be repeated as many times as the number of needed parents.

There are also some variations that can be applied before carrying out the “real” selection. The most common are:

- **rank selection:** the population’s individuals are ranked by fitness; then roulette wheel selection is performed on ranks. This expedient is used when the fitnesses’ values differ greatly (for example, if the best individual occupies 90% of the roulette) to avoid selecting always the same individual.
- **elitism:** the best individual (or the first j best individuals) is copied without alterations in the next population, in this way preserving its genetic code from crossover and mutation. In addition, the corresponding j worst individuals can be deleted. After that, the actual selection is performed.

Recalling again the example from paragraph 1.1.2, suppose that we want to perform roulette wheel selection (without variations). Firstly, we have to compute the individuals’ cumulated fitness, and then divide it, for each individual, by the total fitness. The probability of extraction of individual i is easily calculated:

$$p_i = \frac{f_i^c}{F} - \frac{f_{i-1}^c}{F} = \frac{f_i}{F}$$

where $F = \sum_{j=1}^n f_j$ and $f_0^c = 0$.

Chromosome i	f_i	f_i^c	$\frac{f_i^c}{F}$	p_i
0001	1	1	0.024	0.024
0010	2	3	0.073	0.049
0101	5	8	0.195	0.122
1001	9	17	0.415	0.220
1011	11	28	0.683	0.268
1101	13	41	1.000	0.317

Table 2: Example of chromosomes with associated fitnesses and probabilities of extraction

We can see from Table 2 that the chromosomes with bigger associated values have higher probability of being extracted. In particular, the chromosome with the largest associated values (13) has the highest probability (0.317) of becoming parent.

At this point, we should extract random numbers in range $(0, 41)$ and compare them with cumulated fitnesses (fourth column of the previous table); alternatively we could extract random numbers in range $(0, 1)$ and compare them with the values in the fifth column (the relative cumulated fitnesses). Suppose that we extract this random numbers' sequence (in range $(0, 1)$):

Random number	Selected chromosome
0.6299	1011
0.6012	1011
0.9456	1101
0.1812	0101
0.8907	1101
0.3367	1001

Table 3: Example of selection

We have obtained a set of parents; now we have to pair them to form the offspring (with the crossover operator). We can simply take them in order, therefore forming these three couples:

- (1011, 1011);
- (1101, 0101);
- (1101, 1001).

Each couple will generate offspring (two chromosomes), reaching a total of 6 children chromosomes, which will form the new population. Note that the two worse individuals did not reach the status of “parents” and they will not procreate.

1.2.4 Single point crossover

The **single point crossover** operator chooses a random point on the two chromosomes (the parents), and genetic material is exchanged around this point in order to form two children chromosomes. To clarify this concept, we can see an example:

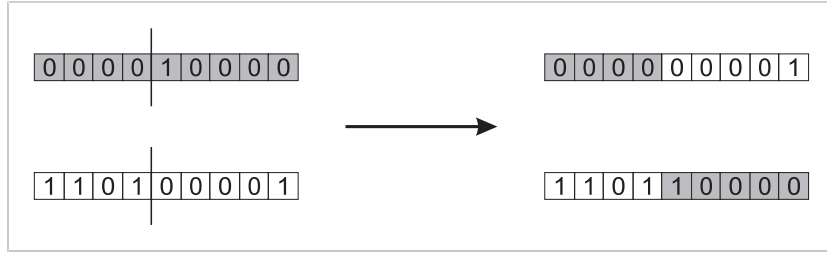


Figure 1: Example of single point crossover

Recalling, as usual, the problem from paragraph 1.1.3, consider the second parents couple that we selected, that is (1101, 0101) and suppose that we randomly extracted the second locus. The children will be 1101 and 0101. Note that, in this case, we have obtained children with the same genetic code of their parents.

There are also other crossover types, like **two point crossover** (it chooses two locus instead of one) and **uniform crossover** (each bit is copied randomly from one parent or the other). We can see examples of these crossovers:



Figure 2: Example of two point crossover

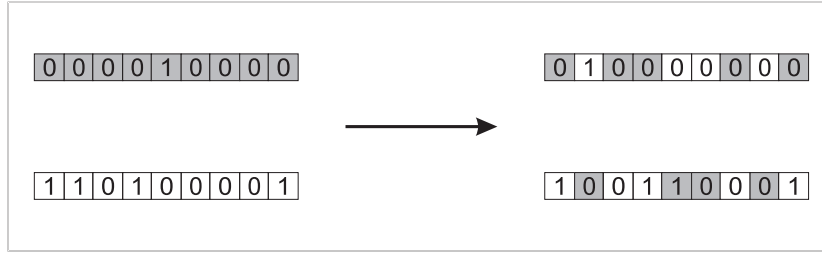


Figure 3: Example of uniform crossover

However, the so-called *simple GA* (we will see it in paragraph 1.2) uses single point crossover.

1.2.5 Mutation

The **mutation** operator randomly changes some chromosome's bits. Mutation can happen in any string's position, with a very low probability (0.001 is commonly used). Consider one of the children generated in the example of paragraph 1.1.4, that is, the chromosome 1101, and suppose that a random mutation happens in the third locus: the resulting chromosome would be 1111.

1.3 - The simple genetic algorithm (SGA)

We have seen the basic elements with which a GA is composed. Now we can see *how* such elements are composed to effectively form a GA.

GAs are classified in three different classes ([15]), depending on the operators (and possible variations) that they employ.

- **Simple GA** (or SGA): single point crossover, mutation, roulette wheel selection.
- **Refined GA**: uniform crossover, mutation, roulette wheel selection plus possible variations (such as elitism).

- **Crowding GA:** uniform crossover, mutation, random selection plus possible variations.

Now we will consider in greater details the simple genetic algorithm (from now on SGA), whose definition and formalisation are due to Goldberg (1989).

Given a well defined problem and a bit-string representation for the candidate solutions, a SGA works like this:

1. randomly generate a population of n individuals (chromosomes composed by l bits, that is the candidate solutions);
2. compute, for each population's chromosome x , its fitness $f(x)$;
3. repeat the next steps until n children chromosomes have been created:
 - choose a couple of chromosomes (parents) from the current population. The largest the fitness of one chromosome, the higher its chances of being selected as parent. The selection is carried out with replacement, that is, the same chromosome can be selected more than once to become parent.
 - with probability p_c (crossover probability or *crossover rate*), crossover is carried out (between the two selected parents) in a random locus (chosen with uniform probability). If crossover does not happen, the created children are precise copies of their parents (the genetic codes remain immutated).
 - mute the children chromosomes in each locus with probability p_m (mutation probability or *mutation rate*) and position the resulting chromosomes in the new population.

If n is odd, a new member of the population can be randomly discarded.

4. swap the current population with the new one;
5. go to step 2.

A complete iteration of this procedure (from step 2 to step 4) is called **generation**. A GA is typically made of a number of generations between 50 and 500 (or even more). The complete set of generations is called **run**. At the end of a run, the current population should contain one or more chromosomes highly fit to the environment. Considering, though, the role played by randomness in each run of the algorithm, it's clearly visible that two different runs (built with different random seeds) will obtain different results. Because of this issue, it is better to execute some runs for the same problem (instead of only one), and then compare their results, or combine them somehow.

To better visualize the procedure that we illustrated, we can see some graphical illustrations.

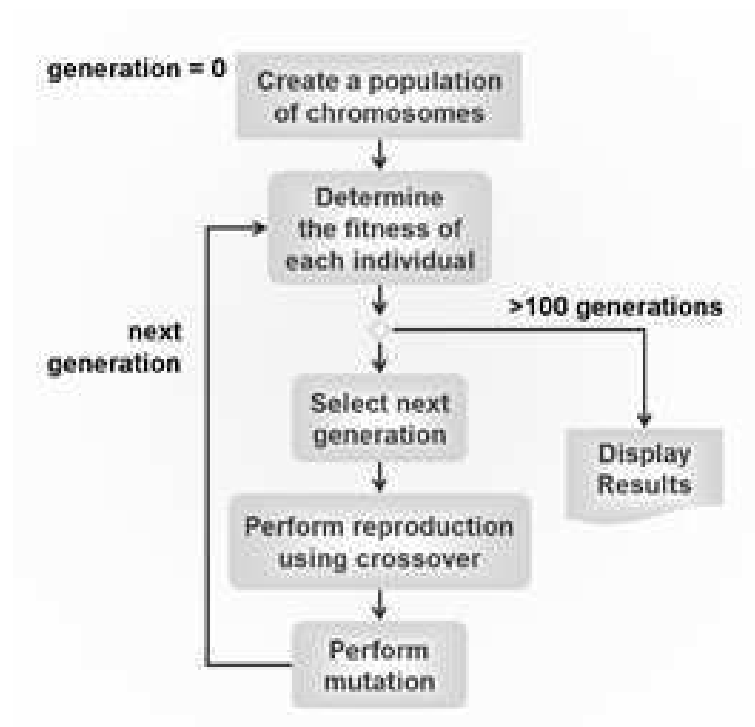


Figure 4: Cycle of a SGA

This figure shows the main loop of the SGA. The figure also assumes a predefined number of generations (100): this means that after 100 generations, the run is con-

sidered complete. However, instead of stopping the algorithm after a certain number of generations, we can fix more general termination criteria, as we can see in the next figure.

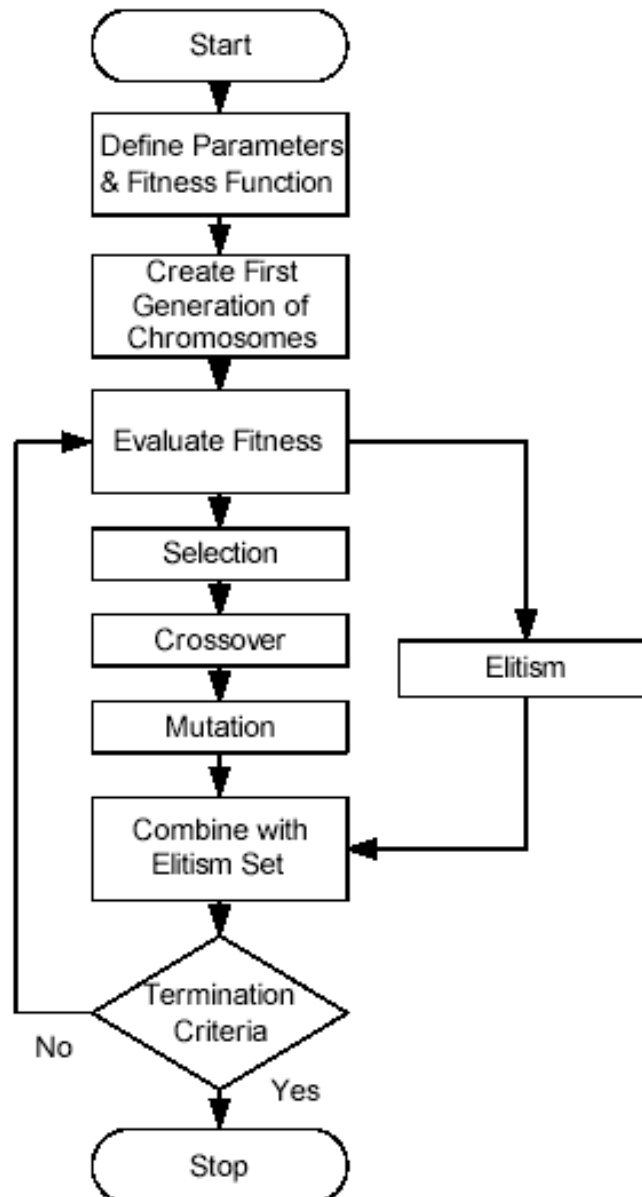


Figure 5: Cycle of an elitist GA

This figure, besides considering unspecified termination criteria, inserts in the algorithm the elitist variation. In Chapter 3, we will talk more specifically about elitist GA.

Figure 5 also reminds us that, before entering the main loop of the algorithm, we have to define parameters and fitness function. In fact, in order to effectively execute the algorithm, we will need to fill the blanks left from the algorithm, i.e.:

- population's size (how much chromosomes compose the population?);
- crossover rate (probability of doing the crossover);
- mutation rate (probability of carrying out a mutation);
- fitness function (when a chromosome is highly fit? when not?);
- codification process (from candidate solutions to chromosomes and viceversa).

All these decisions are not to be taken superficially, because the GA's performance greatly depends on these details.

Chapter 2 - A first, simple, application of SGA

After seeing the GAs foundations and the basic theory behind them, it is time to move towards their practical applications and to see them in action. We will start with a simple task, that is, the maximization of a univariate and unimodal function. We will see the performance of the SGA on this problem, and then try some alterations of the algorithm's parameters (population size, number of generations, crossover rate, mutation rate).

2.1 - Maximization of a univariate function

The first things that anyone using a GA has to do are:

- clearly define the problem;
- decide the encoding for the chromosomes (that is, decide the structure of the individuals of the population);
- define the fitness function.

These three issues constitute the only input that the human has to give to the computer (as well as the algorithm's parameters, but those can be easily modified, as we will see later), and they have to be thought of very carefully.

2.1.1 Problem's definition

Our problem is to determine the global optimum of the univariate function:

$$f(x) = y = 100 - (x - 4)^2$$

where $x \in [0, 13)$.

The function graphically presents itself like this:

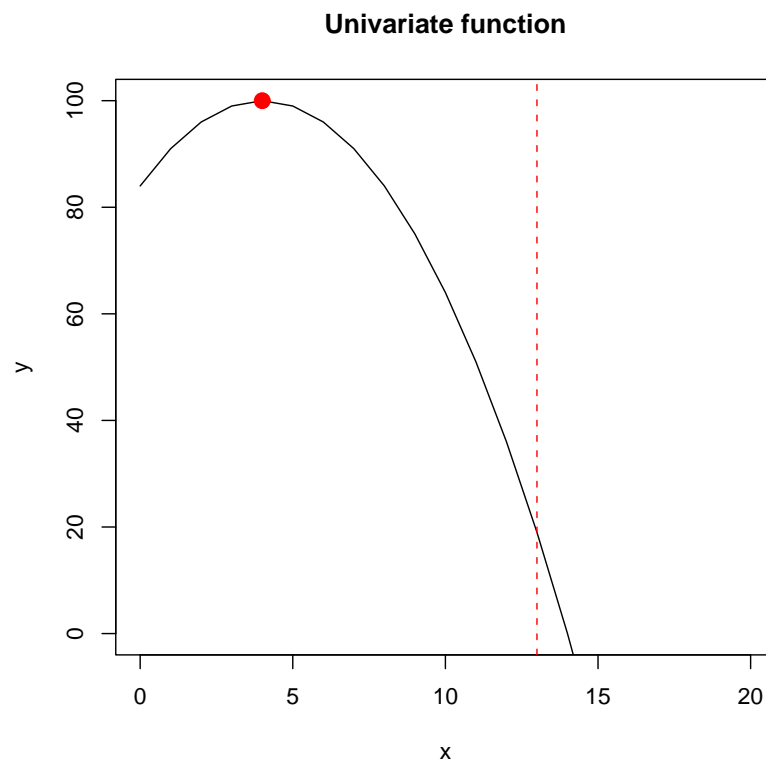


Figure 6: Univariate function

The red dashed line indicates the end of our domain (remember that $x \in [0, 13)$); the black line is our function and the red dot points towards the function's only maximum, located in $x = 4$.

2.1.2 Chromosome's encoding

The candidate solutions for this problem are values in the domain, that is $[0, 13]$; therefore our chromosomes need to be able to represent values in that range. One way to accomplish this is to make the chromosomes encode values in the range $[0, 1)$ and then multiply their value by 13, hence obtaining a value in the desired domain.

We can accomplish this by selecting a length, n , for a chromosome and making each bit of the chromosome carrying a value. For example, we might decide that the i -th bit carries the value of 0 if it's a 0 or 2^{-i} if it's a 1. In this case, the chromosome 1000100100 represents the value:

$$1 * 2^{-1} + 0 * 2^{-2} + 0 * 2^{-3} + 0 * 2^{-4} + 1 * 2^{-5} + 0 * 2^{-6} + 0 * 2^{-7} + 1 * 2^{-8} + 0 * 2^{-9} + 0 * 2^{-10} = 0.53515625$$

To transform this value in the range $[0, 13)$ we multiply it by 13:

$$0.53515625 * 13 = 6.95703125 \cong 6.957.$$

For our problem, we do not need such level of precision and so many digits after the decimal point; this means we can restrict ourselves to a relatively small number of bits for the candidate solutions. In our tests, we will limit chromosomes' length to 8 bits, resulting in a distance of 0.05078125 ($2^{-8} * 13$) between any two contiguous values of our representation.

2.1.3 Fitness function

Now that we have clearly defined the problem and encoded the individuals of the population, it is time to think about the fitness function. Remember that the fitness function is a function that accepts a chromosome as input and returns as output the "goodness" of that chromosome in terms of our problem.

In this problem the formulation is explicitly based on the maximization of some function, so the fitness function is exactly that function. This means that there is no noise or other factor that makes the evaluation of the chromosome's fitness fuzzy.

In fact, our chromosomes (according to the encoding established in the previous para-

graph) represent values of the independent variable (x) of our univariate function. So, each chromosome is connected (through the univariate function) to one and only one value of the dependent variable (y), the quantity that we want to maximize.

Let us see the fitness function in pseudo-code:

```

Fitness function( chromosome  $k$  )
  calculate the decimal value ( $h$ ) that the chromosome  $k$  represents
  multiply  $h$  by 13, obtaining  $x$ 
  calculate the value  $y = f(x)$ 
  return  $y$ 

```

It is simple to see that the bigger the value of y is, the higher is the fitness of the chromosome. In other words, the chromosomes which have high values of their “connected” y are fit to the environment of our problem. The GA assumes that they carry a good genetic code, so they are good for reproduction: they will be selected as parents with higher probability than the chromosomes with lower fitness.

2.2 Description of a generation

We have defined the problem, the encoding and the fitness function.

Now we can describe an entire generation of the SGA for this problem.

2.2.1 Step 1 (initialization)

We randomly generate the individuals of the first population. In pseudo-code, it looks like this:

randomFirstPopulationCreation

```

{
   $\forall$  population's chromosome (individual)
     $\forall$  chromosome's bit

```

```

    extract a random number  $r \in (0, 1)$ 
    if (  $r < 0, 5$  )
        put 0 in the bit
    else
        put 1 in the bit
}

```

2.2.2 Step 2 (fitness)

We compute the fitness of each population's individual through the fitness function (the one described in the previous paragraph).

2.2.3 Step 3 (selection)

Now that the existing population has been evaluated, we can choose the parents, that is, the chromosomes fit to reproduce. We have what we need to do roulette wheel selection, the selection method used in SGA. We proceed as described in paragraph 1.2.3.

The pseudo-code of the part that selects a parent is the following:

parentsRouletteWheelSelection

```

{
     $f_i$  = fitness of individual  $i$ 
     $F = \sum_{i=1}^{popSize} f_i$  = total fitness
     $f_i^c = \sum_{j=1}^i f_j$  = individual's cumulated fitness
     $f_i^{cp} = f_i^c / F$  = individual's cumulated fitness scaled down in range (0,1)
    extract a random number  $r \in (0, 1)$ 
     $\forall$  population's individual (with increasing  $f_i^{cp}$ )
        if(  $f_i^{cp} \geq r$  ) {
            select individual  $i$  as a parent
        }
}

```

```

        return
    }
}

```

2.2.4 Step 4 (crossover)

For each couple of parents selected with STEP 3, we have to make them to reproduce, in order to generate the so-called *offspring* (two children which will be individuals of the following population).

The pseudo-code of this part (which uses single point crossover) is:

```

crossover( parent A, parent B )
{
    extract a random number  $r \in (0, 1)$ 
    if(  $r > \text{crossoverRate}$  )
        copy the parents A and B (as they are) in the next population
    else
         $k = \text{chromosome size (in bits)}$ 
        extract a random number  $h \in (0, k)$  and cast the number to integer
         $\forall$  bit of the chromosomes of children C and D
            if( bit position  $< h$  )
                C [ bit position ] = A [ bit position ]
                D [ bit position ] = B [ bit position ]
            else
                C [ bit position ] = B [ bit position ]
                D [ bit position ] = A [ bit position ]
        put C and D in the next population
    return
}

```

We repeat this procedure (choose two parents and generate offspring) until we have

completely filled the next population (the population size has to remain the same).

2.2.5 Step 5 (mutation)

We now randomly mutate some bits of the new population's individuals. The pseudo-code is:

```
mutatePopulation
{
     $\forall$  population's individual (chromosomes)
         $\forall$  bit
            extract a random number  $r \in (0, 1)$ 
            if(  $r < \text{mutationRate}$  )
                flip the value of the bit
}
```

After this, we have completed the first generation and we can go back to STEP 2 with the new population.

The SGA is considered complete after a large number of generations (approximately between 50 and 500) is done.

Remember that:

- the crossover rate is the probability of combining the genetic code of two parents (in opposition to leave their genetic code unchanged);
- the mutation rate is the probability of changing the value of an individual's bit.

2.3 Results

Finally, we can execute the SGA for our problem. But first we have to fix the values of the algorithm's parameters. We will begin using the values shown in the following table.

Parameter	Population size	Number of generations	Crossover rate	Mutation rate
Value	50	100	0,75	0,001

Table 4: Parameters' values

In Chapter 1 we said that each run of a SGA will have different results, because of the central role of randomness in the algorithm. So, in order to make more general speculations, for each set of parameters we will execute 10 runs of the SGA and compute the mean of their results. We consider the result of a run the mean fitness of the final population.

In the following table, we report the mean fitness of the final population of each performed run; in the end we calculate the average result on the 10 runs.

Run 1	Run 2	Run 3	Run 4	Run 5	Run 6
98.6794	99.5364	97.4253	99.9203	99.9944	98.3326
Run 7	Run 8	Run 9	Run 10	Average	
99.5713	98.4759	96.658	99.8301	98.8424	

Table 5: Results of the 10 runs

The average result of the 10 runs is 98.8424: this is very good, if we think that our goal was 100!

Moreover, we have to remember that this result cannot be *exactly* 100. The reason for this has to be found in our codification/decodification process. Our chromosomes represent floating point numbers with a specific precision; that value has then to be multiplied by 13, and this operation makes almost impossible to reach the precise value of 4.0 for x (and therefore of 100.0 for y).

2.3.1 Changing population size

So far, we have seen that our SGA (with the parameters shown in Table 4) works fine. Now, we can see what happens if we modify some of the parameters, one at a time (to better comprehend the difference in the results). We shall start by modifying the **population size**.

Population size	Average fitness
20	96.3063
30	98.3643
40	98.9429
50	98.7512
60	99.0199
70	99.5567
80	99.0659
90	98.9602
100	98.7167
120	98.9487

Table 6: Results of SGA with different population sizes

We can see from Table 6 that the population size has a little role (though not vital) in this problem's results. In fact, the average fitness is good in all instances, but is a bit worse with very small population. In particular, the worst result is obtained with *population size* = 20, the best with *population size* = 70.

2.3.2 Changing number of generations

We shall try to modify another parameter, i.e., the **number of generations** of each run. We preserve the initial values of the other parameters (population size = 50, crossover rate = 0.75, mutation rate = 0.001).

# generations	Average fitness
10	95.0670
20	96.1546
35	96.9471
50	98.3094
70	99.1944
80	98.8897
100	98.6132
150	98.0872
200	98.5816

Table 7: Results of SGA with different generations' numbers

The *number of generations* parameter has an evident behaviour, as we can see from Table 7. The SGA needs a certain number of generations to reach the “convergence” on the solution for our problem; after that it maintains the goodness of the results, with some alterations due to random mutations.

2.3.3 Changing crossover rate

We shall now see how the algorithm reacts by changing the **crossover rate**, that is, the probability of effectively doing the crossover when making individuals procreate (instead of copying their genetic code in the next population). As usual, we preserve

the initial values of the other parameters (generations number = 100, population size = 50, mutation rate = 0.001).

Crossover rate	Average fitness
0.25	98.8613
0.50	97.8205
0.70	98.5307
0.75	98.1522
0.80	98.1505
0.85	98.3973
0.90	97.7203
1.00	98.4599

Table 8: Results of SGA with different crossover rates

In this case the parameter's modification has not had valuable effects: the SGA always finds a good solution to our problem. This could be caused by the simplicity of our problem.

2.3.4 Changing mutation rate

We can try varying the last parameter left, the **mutation rate**. This parameter represents the probability that each locus of each chromosome flips its value. We will preserve the initial values of the other parameters, that is:

- population size = 50;
- number of generations = 100;
- crossover rate = 0.75;

Mutation rate	Average fitness
0.0001	99.2635
0.0005	98.2141
0.001	98.5449
0.005	97.9856
0.01	96.5034
0.05	92.3261
0.1	87.1942

Table 9: Results of SGA with different mutation rates

With these changes we can see very serious alterations in the results: by increasing the mutation rate, the precision of the solution drastically becomes worse. This happens because a high mutation rate excessively shuffles the chromosomes currently present in the population.

Chapter 3 - Variable selection for regression models

After seeing that the SGA works fine with a simple problem (the maximization of a univariate function), we can move on to more complicated (and more interesting) problems; in particular we will focus on problems of statistical kind. To begin with, we will see a problem of variable selection for linear regression models. We will consider a moderately large dataset, with various independent variables. The goal is to select the most important and meaningful variables. In order to compare the goodness of fit of different regression models, we shall make use of the Akaike Information Criterion. In fact, the R-squared wouldn't be appropriate, because this would certainly take us to the regression model which includes all the variables present in the dataset. With the AIC criterion, instead, we consider the trade-off between the performance of the regression model and the number of independent variables included in the predictors.

3.1 - Variable selection

The problem that we are considering in this section, variable selection, is very well known in literature and it is part of a more general and complex issue (model selection); it is also known as subset selection (we will soon understand why).

Suppose that Y (a variable of interest) and X_1, X_2, \dots, X_p (a set of potential explanatory variables) are vectors of n observations. The problem of variable (or subset)

selection comes into existence when one wants to model the relationship between Y and a subset of X_1, X_2, \dots, X_p , but is not sure about the subset to use. This issue is particularly significant if p is very large and if a lot of X_i could be redundant or irrelevant variables.

Usually, to choose the variables to include in a model, one has to consider some different formulations of the model and make a comparison based on some significant indicator. Some possible indicators are:

- R^2 , the coefficient of determination. It represents the proportion of variability explained by the model. It can be expressed in many ways, the most common of which are: $R^2 = \frac{SS_R}{SS_T} = 1 - \frac{SS_E}{SS_T}$, where $SS_T = \sum_i (y_i - \bar{y})^2$ (total sum of squares), $SS_R = \sum_i (\hat{y}_i - \bar{y})^2$ (regression sum of squares) and $SS_E = \sum_i (y_i - \hat{y}_i)^2$ (sum of squared errors). Hence, R^2 is a descriptive measure about the goodness of fit of the model and it assumes values in the range $(0, 1)$. The closer to one, the better the model. However, it has a drawback: its value can only increase (and not decrease) as we increase the number of variables in the model, causing overfitting. Using this measure to compare models in our problem would take us to include *all* the variables. In conclusion, R^2 is not an appropriate measure for variable selection.
- **Adjusted R^2** , that is a modification of R^2 which takes into account the number of variables in the model. When increasing the number of explanatory terms, adjusted R^2 increases *only* if the new term significantly improves the model. The adjusted R^2 is defined as: $1 - (1 - R^2) \frac{n-1}{n-k-1}$ where k is the number of regressors of the considered model. Hence, this measure could be used in variable selection.
- **AIC** (Akaike Information Criterion), which we will properly discuss in the next paragraph.
- **BIC** (Bayesian Information Criterion) $= -2 * \ln(L) + k * \ln(n)$, where L is the maximized value of the likelihood function for the estimated model, k is the

number of regressors (including the constant) and n is the number of observations. Under the assumption that the model errors are normally distributed, we can rewrite the expression as: $n * \ln(\frac{SS_E}{n}) + k * \ln(n)$. Between two models, one should choose the one with lower value of BIC. The BIC penalizes free parameters more strongly than the Akaike Information Criterion does.

- “**Mallows Cp** = $\frac{SS_E}{\hat{\sigma}_{FULL}^2} + 2 * k - n$, where SS_E is the residual sum-of squares for the model at issue and $\hat{\sigma}_{FULL}^2$ is the usual unbiased estimate of σ^2 based on the full model. Motivated as an unbiased estimate of predictive accuracy of the model at issue, Mallows (1973) recommended the use of Cp plots to help gauge subset selection, see also Mallows (1995). Although he specifically warned against minimum Cp as a selection criterion (because of selection bias), minimum Cp continues to used as a criterion.” [9]

Another option to consider for the variable selection problem is the method of **stepwise regression**. Stepwise regression is a greedy algorithm that adds the best feature (or deletes the worst feature) at each iteration. In stepwise regression there are two possibilities: *forward selection* (find the best one-variable model, then the best two-variable model with that first variable included, and so on until no addition is statistically significant) or *backward selection* (begin with the model with all variables, remove the least significant one, and so on until all variables are statistically significant). Several criticisms have been made about stepwise regression, which does not necessarily find the best model among all the possible variable combinations.

Drawing some conclusions, the best catches are probably AIC and BIC. In particular, we will focus on the first one.

3.2 - AIC

AIC (Akaike Information Criterion), introduced by Hirotugu Akaike in his seminal 1973 paper (“*Information Theory and an Extension of the Maximum Likelihood Principle*”),

is considered the first model selection criterion and is derived as an estimator of the expected *Kullback discrepancy* between the true model and a fitted model. AIC has limited demands, in fact it requires only a large sample size (n); moreover, when the candidate model is relatively small (k is small) its estimation of Kullback discrepancy is approximately unbiased. We shall now see AIC derivation.

Suppose we have:

- $f(y|\theta_0)$, the true or generating model (unknown);
- $f(y|\theta_k)$, the candidate or approximating model.

For two arbitrary parametric density $f(y|\theta)$ and $f(y|\theta^*)$, the **Kullback-Leibler information** or **Kullback's directed divergence** between $f(y|\theta)$ and $f(y|\theta^*)$ with respect to $f(y|\theta)$, is defined like this:

$$l(\theta, \theta^*) = E_\theta[\ln \frac{f(y|\theta)}{f(y|\theta^*)}]$$

where E_θ denotes the expectation under $f(y|\theta)$.

$l(\theta, \theta^*)$ is a measure of distance between the two models, although it is not a formal distance measure. Note that:

- $l(\theta, \theta^*) \geq 0$
- $l(\theta, \theta^*) = 0 \iff \theta = \theta^*$

In particular, we will consider $l(\theta_0, \theta_k)$, which is the Kullback-Leibler information between $f(y|\theta_0)$ and $f(y|\theta_k)$ with respect to $f(y|\theta_0)$.

Now, we define the **Kullback discrepancy** as:

$$d(\theta_0, \theta_k) = E_{\theta_0}\{-2 \ln f(y|\theta_k)\}.$$

It can be proved that there is a relationship between the Kullback discrepancy and Kullback-Leibler information, that is:

$$2l(\theta_0, \theta_k) = d(\theta_0, \theta_k) - d(\theta_0, \theta_0)$$

Since $d(\theta_0, \theta_0)$ does not depend on θ_k , we can consider (for our purposes of ranking candidate models) $l(\theta_0, \theta_k)$ equivalent to $d(\theta_0, \theta_k)$.

We have a problem, though: we cannot evaluate $d(\theta_0, \theta_k)$, since θ_0 is unknown. Here arrives Akaike with his work (1973). He suggests, as biased estimator of $d(\theta_0, \theta_k)$, the quantity $-2 \ln f(y|\hat{\theta}_k)$. He also asserts that the bias adjustment can often be asymptotically estimated by twice the dimension of θ_k , i.e., k . Therefore Akaike defines the criterion:

$$\text{AIC} = -2 \ln f(y|\hat{\theta}_k) + 2k$$

whose expected value should, under appropriate conditions, asymptotically approach the expected value of $d(\theta_0, \hat{\theta}_k)$.

So, AIC provides us with an asymptotically unbiased estimator of $E_{\theta_0}\{d(\theta_0, \hat{\theta}_k)\}$ (the so-called **expected Kullback discrepancy**), in settings where n is large and k is comparatively small. AIC is derived on the assumption that the fitted model is either correctly specified or overfitted; regardless, AIC can still be effectively used in case of underspecified models.

In the special case of least squares estimation with normally distributed errors, AIC can be expressed as:

$$\text{AIC} = n \ln (\hat{\sigma}^2) + 2k$$

where

$$\hat{\sigma}^2 = \frac{\sum_i (\hat{\epsilon}_i)^2}{n}.$$

and the $\hat{\epsilon}_i$ are the estimated residuals from the fitted model. Note that k must be the total number of parameters in the model, including the intercept and σ^2 .

3.3 - Problem's definition

In this section, we want to solve a variable selection problem. We will use a dataset coming from *Baseball data from M.R. Watnik (1998)*, “Pay for Play: Are Baseball

Salaries Based on Performance", *Journal of Statistics Education*, Volume 6, number 2. This dataset contains salaries in 1992 and 27 performance statistics for 337 baseball players (no pitchers) in 1991. The metric unit of the salary is \$1000s. The performance statistics are:

- average = batting average
- obp = on base percentage
- runs = runs scored
- hits
- doubles
- triples
- homeruns
- rbis = runs batted in
- walks
- sos = strike outs
- sbs = stolen bases
- errors
- freeagent (or eligible for free agency)
- arbitration (or eligible for arbitration)
- runsperso = runs/sos
- hitsperso = hits/sos
- hrsperso = homeruns/sos

- $\text{rbisperso} = \text{rbis}/\text{sos}$
- $\text{walksperso} = \text{walks}/\text{sos}$
- $\text{obppererror} = \text{obp}/\text{errors}$
- $\text{runspererror} = \text{runs}/\text{errors}$
- $\text{hitspererror} = \text{hits}/\text{errors}$
- $\text{hrspererror} = \text{homeruns}/\text{errors}$
- $\text{soserrors} = \text{sos} * \text{errors}$
- $\text{sbsobp} = \text{sbs} * \text{obp}$
- $\text{sbsruns} = \text{sbs} * \text{runs}$
- $\text{sbshits} = \text{sbs} * \text{hits}$

We would like to model a relationship (if any) between the salary and the performance statistics of the baseball players. The explicative variables are many, and probably a lot of them are redundant or unimportant. So we would like to use only a subset of those variables; choosing this subset is our problem.

We can do a brief exploratory analysis about the explicative variables' correlation. With 27 variables the dispersion graphs are too many to be presented here. Instead, we insert here an “information summary” of the dispersion graphs, that is, the correlation matrix; in the (i, j) box of the following table we have the correlation between x_i and x_j . The correlation values are in the range $(-1, 1)$. The maximum degree of correlation corresponds to a value of 1 or -1; the minimum one to a value of 0. Obviously $\text{cor}(x_i, x_i) = 1$. Since $\text{cor}(x_i, x_j) = \text{cor}(x_j, x_i)$, the matrix is symmetrical.

cor	y	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
y	1.0	0.27	0.32	0.64	0.62	0.57	0.23	0.59	0.66	0.56	0.4
x1	0.27	1.0	0.8	0.43	0.5	0.45	0.26	0.21	0.36	0.27	0.07
x2	0.32	0.8	1.0	0.51	0.45	0.4	0.19	0.31	0.39	0.59	0.2
x3	0.64	0.43	0.51	1.0	0.92	0.83	0.54	0.68	0.83	0.82	0.68
x4	0.62	0.5	0.45	0.92	1.0	0.88	0.54	0.61	0.85	0.72	0.64
x5	0.57	0.45	0.4	0.83	0.88	1.0	0.41	0.63	0.82	0.64	0.6
x6	0.23	0.26	0.19	0.54	0.54	0.41	1.0	0.12	0.33	0.3	0.32
x7	0.59	0.21	0.31	0.68	0.61	0.63	0.12	1.0	0.87	0.62	0.74
x8	0.66	0.36	0.39	0.83	0.85	0.82	0.33	0.87	1.0	0.72	0.74
x9	0.56	0.27	0.59	0.82	0.72	0.64	0.3	0.62	0.72	1.0	0.66
x10	0.4	0.07	0.2	0.68	0.64	0.6	0.32	0.74	0.74	0.66	1.0
x11	0.25	0.2	0.23	0.52	0.42	0.29	0.52	0.07	0.2	0.35	0.27
x12	0.12	0.14	0.09	0.34	0.41	0.33	0.19	0.15	0.29	0.24	0.3
x13	0.56	0.05	0.17	0.3	0.29	0.22	0.05	0.29	0.3	0.35	0.22
x14	0.12	0.09	0.01	0.14	0.18	0.2	0.15	0.03	0.12	0.03	0.03
x15	0.23	0.41	0.38	0.37	0.29	0.27	0.26	-0.01	0.13	0.2	-0.21
x16	0.2	0.54	0.35	0.22	0.33	0.26	0.2	-0.1	0.12	0.07	-0.31
x17	0.47	0.26	0.27	0.43	0.41	0.44	0.01	0.71	0.6	0.34	0.27
x18	0.35	0.48	0.35	0.29	0.38	0.37	0.12	0.22	0.41	0.19	-0.12
x19	0.26	0.3	0.57	0.33	0.27	0.22	0.12	0.0	0.15	0.5	-0.15
x20	-0.13	0.04	0.14	-0.21	-0.3	-0.27	-0.16	-0.13	-0.24	-0.11	-0.22
x21	0.25	0.14	0.27	0.4	0.28	0.24	0.18	0.26	0.27	0.4	0.27
x22	0.26	0.19	0.25	0.35	0.33	0.27	0.17	0.26	0.3	0.37	0.28
x23	0.29	0.07	0.19	0.32	0.25	0.28	-0.01	0.56	0.44	0.39	0.43
x24	0.19	0.09	0.11	0.47	0.48	0.43	0.25	0.4	0.47	0.39	0.63
x25	0.28	0.22	0.27	0.54	0.43	0.3	0.5	0.09	0.21	0.38	0.27
x26	0.32	0.23	0.27	0.6	0.49	0.36	0.51	0.16	0.29	0.43	0.31
x27	0.32	0.26	0.25	0.59	0.54	0.4	0.56	0.15	0.31	0.41	0.33

cor	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
y	0.25	0.12	0.56	0.12	0.23	0.2	0.47	0.35	0.26	-0.13
x1	0.2	0.14	0.05	0.09	0.41	0.54	0.26	0.48	0.3	0.04
x2	0.23	0.09	0.17	0.01	0.38	0.35	0.27	0.35	0.57	0.14
x3	0.52	0.34	0.3	0.14	0.37	0.22	0.43	0.29	0.33	-0.21
x4	0.42	0.41	0.29	0.18	0.29	0.33	0.41	0.38	0.27	-0.3
x5	0.29	0.33	0.22	0.2	0.27	0.26	0.44	0.37	0.22	-0.27
x6	0.52	0.19	0.05	0.15	0.26	0.2	0.01	0.12	0.12	-0.16
x7	0.07	0.15	0.29	0.03	-0.01	-0.1	0.71	0.22	0.0	-0.13
x8	0.2	0.29	0.3	0.12	0.13	0.12	0.6	0.41	0.15	-0.24
x9	0.35	0.24	0.35	0.03	0.2	0.07	0.34	0.19	0.5	-0.11
x10	0.27	0.3	0.22	0.03	-0.21	-0.31	0.27	-0.12	-0.15	-0.22
x11	1.0	0.17	0.05	0.1	0.26	0.1	-0.03	-0.02	0.17	-0.11
x12	0.17	1.0	-0.01	0.21	0.06	0.12	0.04	0.07	0.05	-0.61
x13	0.05	-0.01	1.0	-0.39	0.1	0.1	0.24	0.17	0.23	-0.05
x14	0.1	0.21	-0.39	1.0	0.13	0.11	0.04	0.12	0.01	-0.12
x15	0.26	0.06	0.1	0.13	1.0	0.75	0.28	0.64	0.68	0.04
x16	0.1	0.12	0.1	0.11	0.75	1.0	0.21	0.82	0.6	-0.09
x17	-0.03	0.04	0.24	0.04	0.28	0.21	1.0	0.56	0.2	-0.05
x18	-0.02	0.07	0.17	0.12	0.64	0.82	0.56	1.0	0.52	-0.07
x19	0.17	0.05	0.23	0.01	0.68	0.6	0.2	0.52	1.0	0.06
x20	-0.11	-0.61	-0.05	-0.12	0.04	-0.09	-0.05	-0.07	0.06	1.0
x21	0.19	-0.37	0.2	-0.04	0.23	0.0	0.13	0.03	0.18	0.59
x22	0.13	-0.4	0.22	-0.05	0.07	0.05	0.14	0.09	0.13	0.57
x23	-0.05	-0.27	0.21	-0.04	-0.06	-0.14	0.38	0.06	0.0	0.4
x24	0.25	0.83	0.02	0.15	-0.08	-0.11	0.12	-0.04	-0.08	-0.44
x25	0.99	0.15	0.07	0.1	0.28	0.11	-0.01	0.0	0.2	-0.09
x26	0.95	0.17	0.08	0.11	0.27	0.12	0.03	0.02	0.19	-0.08
x27	0.95	0.2	0.07	0.13	0.23	0.15	0.03	0.05	0.16	-0.12

cor	x21	x22	x23	x24	x25	x26	x27
y	0.25	0.26	0.29	0.19	0.28	0.32	0.32
x1	0.14	0.19	0.07	0.09	0.22	0.23	0.26
x2	0.27	0.25	0.19	0.11	0.27	0.27	0.25
x3	0.4	0.35	0.32	0.47	0.54	0.6	0.59
x4	0.28	0.33	0.25	0.48	0.43	0.49	0.54
x5	0.24	0.27	0.28	0.43	0.3	0.36	0.4
x6	0.18	0.17	-0.01	0.25	0.5	0.51	0.56
x7	0.26	0.26	0.56	0.4	0.09	0.16	0.15
x8	0.27	0.3	0.44	0.47	0.21	0.29	0.31
x9	0.4	0.37	0.39	0.39	0.38	0.43	0.41
x10	0.27	0.28	0.43	0.63	0.27	0.31	0.33
x11	0.19	0.13	-0.05	0.25	0.99	0.95	0.95
x12	-0.37	-0.4	-0.27	0.83	0.15	0.17	0.2
x13	0.2	0.22	0.21	0.02	0.07	0.08	0.07
x14	-0.04	-0.05	-0.04	0.15	0.1	0.11	0.13
x15	0.23	0.07	-0.06	-0.08	0.28	0.27	0.23
x16	0.0	0.05	-0.14	-0.11	0.11	0.12	0.15
x17	0.13	0.14	0.38	0.12	-0.01	0.03	0.03
x18	0.03	0.09	0.06	-0.04	0.0	0.02	0.05
x19	0.18	0.13	0.0	-0.08	0.2	0.19	0.16
x20	0.59	0.57	0.4	-0.44	-0.09	-0.08	-0.12
x21	1.0	0.92	0.7	-0.2	0.21	0.24	0.2
x22	0.92	1.0	0.71	-0.23	0.15	0.18	0.18
x23	0.7	0.71	1.0	-0.09	-0.04	0.0	0.0
x24	-0.2	-0.23	-0.09	1.0	0.24	0.28	0.29
x25	0.21	0.15	-0.04	0.24	1.0	0.96	0.95
x26	0.24	0.18	0.0	0.28	0.96	1.0	0.97
x27	0.2	0.18	0.0	0.29	0.95	0.97	1.0

In this table we highlighted the correlation values higher than 0.8 or lower than -0.8. This means that we have marked the more correlated pairs of variables; this will turn out to be worth later, when we will analyze some regression models and we will want to compare their variables' set.

In Appendix A, four of the most significant dispersion graphs are presented. The choice is based on the correlation values and on the analysis that we will make in paragraph 3.9.2 (Variables' subset comparison).

3.4 - Chromosome's encoding

The encoding of the chromosomes for this problem is very simple. We will use chromosomes 27 bits long, each bit representing one performance statistic. A 0-valued bit encodes the absence of that statistic from the candidate model, while a 1-valued bit encodes the presence of that statistic in the candidate model. For example, this chromosome:

100001010010010100100000000

represents the candidate model whose regressors are average, triples, rbis, sbs, arbitration, hitperso, walkperso.

3.5 - Fitness function

Like we said before, the fitness function is useful to compare the chromosomes' goodness. In this problem, the "goodness" of any chromosome is the goodness of the corresponding candidate model. So, we have to use a criterion for comparing models; we already discussed this issue and we decided to exploit AIC. In our computer program we will make use of R routines to adapt linear models to our data and to calculate the AIC values. In particular, the R functions used are *lm* and *AIC*. So, the pseudo-code of this fitness function looks like this:

Fitness function(chromosome c)
adapt the linear model to the formulation that the chromosome c represents
calculate the AIC value of the adapted model
return $f(AIC)$

It has to be noted that the lower the AIC value, the better the model. So, in this problem we would have to minimize (instead of maximize) with respect to that value (AIC). To avoid this problem, we operate a transformation on the set of chromosomes' AIC values; this is why in the pseudo-code we return $f(AIC)$ instead of simply returning AIC .

Here is how this transformation works: first of all, we change the sign of AIC values, to return to a maximization problem; then we scale them in the range $(0, 1)$.

So, suppose we have this simple set of AIC values for our population:

$$\{5350, 5370, 5380, 5400, 5430, 5450, 5500\}$$

Then, the passages of the transformation are summarized in this table:

Initial AIC value	AIC with inverted sign	AIC scaled
5350	−5350	1
5370	−5370	0.8667
5380	−5380	0.8
5400	−5400	0.6667
5430	−5430	0.4667
5450	−5450	0.3333
5500	−5500	0

Table 10: Example of the scaling of AIC values

We can see now that the “best” individual (the one with the higher transformed

fitness) is the first, that is, the chromosome with an AIC value of 5350, the lower one. We obtained what we wanted.

3.6 - A comparison term

In the example of Chapter 2 (the maximization of a univariate function), we knew what we had to find: the maximum of the function was easily observable, so we could judge the results on our own. In this case, though, we have no idea of what to expect or what to hope from our SGA: we grope in the dark.

So, we need a comparison term: in R there is a function (*step*) which “resolves” a problem of variable selection using the stepwise regression technique that we mentioned before. Therefore, we can apply that function to our data and compute the AIC of the resulting model. Then, we will use that AIC value as touchstone.

The R commands used are these:

```
> baseball = read.table( file.choose(), header=T, sep=" ", dec="." )
> attach( baseball )
> modelloConSolaIntercetta = lm( y ~ 1 )
> step( modelloConSolaIntercetta, ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
+ x10 + x11 + x12 + x13 + x14 + x15 + x16 + x17 + x18 + x19 + x20 + x21 + x22
+ x23 + x24 + x25 + x26 + x27, trace=F )
```

Call:

```
lm(formula = y ~ x8 + x13 + x14 + x26 + x10 + x7 + x18 + x24 + x3)
> miglioreModello = lm( y ~ x8 + x13 + x14 + x26 + x10 + x7 + x18 + x24 + x3 )
> AIC( miglioreModello )
[1]5377.877
```

Hence, we will keep the value **5377.877** in mind and we will compare it to our results.

N.B.: When using the *step* function, R sets automatically the direction to backward,

so these results are obtained with a backward procedure. It has to be noted that using the forward procedure leads us to the same model (that is, the model with variables: $x_8 + x_{13} + x_{14} + x_{26} + x_{10} + x_7 + x_{18} + x_{24} + x_3$).

3.7 - Description of a generation

The beauty of GAs is their providing the user with a complete framework which, excluding the choice of parameters' values, needs only the definition of:

- the chromosomes' structure;
- the fitness function.

Apart from these two issues, the rest is always the same. This is the reason why we do not have to describe anything different from what we saw in paragraph 2.2. We have already discussed about the chromosomes' structure (in paragraph 3.4) and about the fitness function (in paragraph 3.5).

3.8 - Results

It is time to see some results. Like in the example of Chapter 2, for each set of chosen parameters we will execute more than one GA's run (to avoid biased results caused by randomness). Besides, since this problem is far more complicated than the previous one, we will use a larger number of runs and we will do a more thorough analysis. In fact, in addition to the simple genetic algorithm (SGA), we will use a modification of the SGA, that is the genetic algorithm with *elitism*. We will then compare the performances of the two algorithms; also, in the elitist GA (let us call it EGA), we will try different crossover's types (two point and uniform, in addition to single point).

We shall begin with the SGA (which means roulette wheel selection and single point crossover). Before executing our SGA we have, as usual, to specify the values of the algorithm's parameters. We will use these values:

Parameter	Population size	Generations	Crossover rate	Mutation rate	Runs
Value	40	200	0.75	0.001	50

Table 11: Parameters' values

In Figure 7 we report the mean fitness for each generation in the 50 runs. This means that, during the execution of the algorithm, we store the mean fitness at each generation of each run; we obtain a matrix whose rows represent the generations and whose columns represent the runs:

	Run 1	...	Run j	...	Run 50
Generation 1	$fitness_{1,1}$...	$fitness_{1,j}$...	$fitness_{1,50}$
...	
Generation i	$fitness_{i,1}$...	$fitness_{i,j}$...	$fitness_{i,50}$
...
Generation 200	$fitness_{200,1}$...	$fitness_{200,j}$...	$fitness_{200,50}$

Table 12: Matrix of fitnesses

At the end, for each generation we compute the mean of the mean fitnesses in the 50 runs (speaking in matrix terms, we compute each mean by rows). By doing this we “smooth” the trend of the mean fitness, removing oscillations due to random mutations.

The result of these computations is shown in Figure 7:

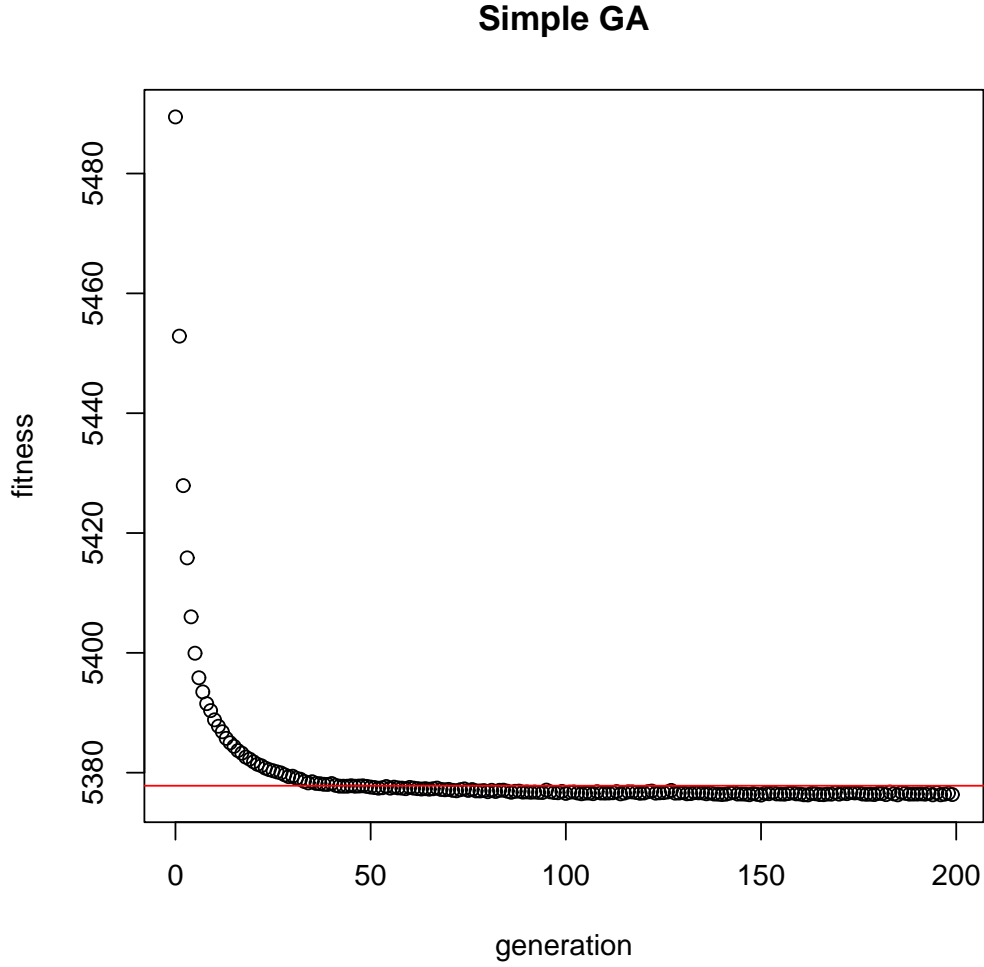


Figure 7: Results of Simple Genetic Algorithm

The red line of Figure 7 represents our comparison term, which is the AIC of the best model obtained by R. The black dots represent the mean of the mean fitnesses of each generation. We can see that the convergence towards the final value (the value at which the algorithm becomes steady) is pretty quick. In fact, the “final value” stays around 5376 and at the 27th generation we are already under 5380. Looking at our comparison term (the red line), we have a good news: our SGA slightly outperforms R. In fact, our algorithm reaches the comparison term after about 50 generations; after the 75th it remains steadily below it.

Now we can see what the elitist genetic algorithm (EGA) accomplishes. We remember from Chapter 1 that elitism is an algorithmic variant that copies some of the better elements of the current population in the next population, without risk of losing their genetic code. In this specific case, we will keep the two best elements. This is the result of the EGA's execution (the parameters' values are the same of the SGA):

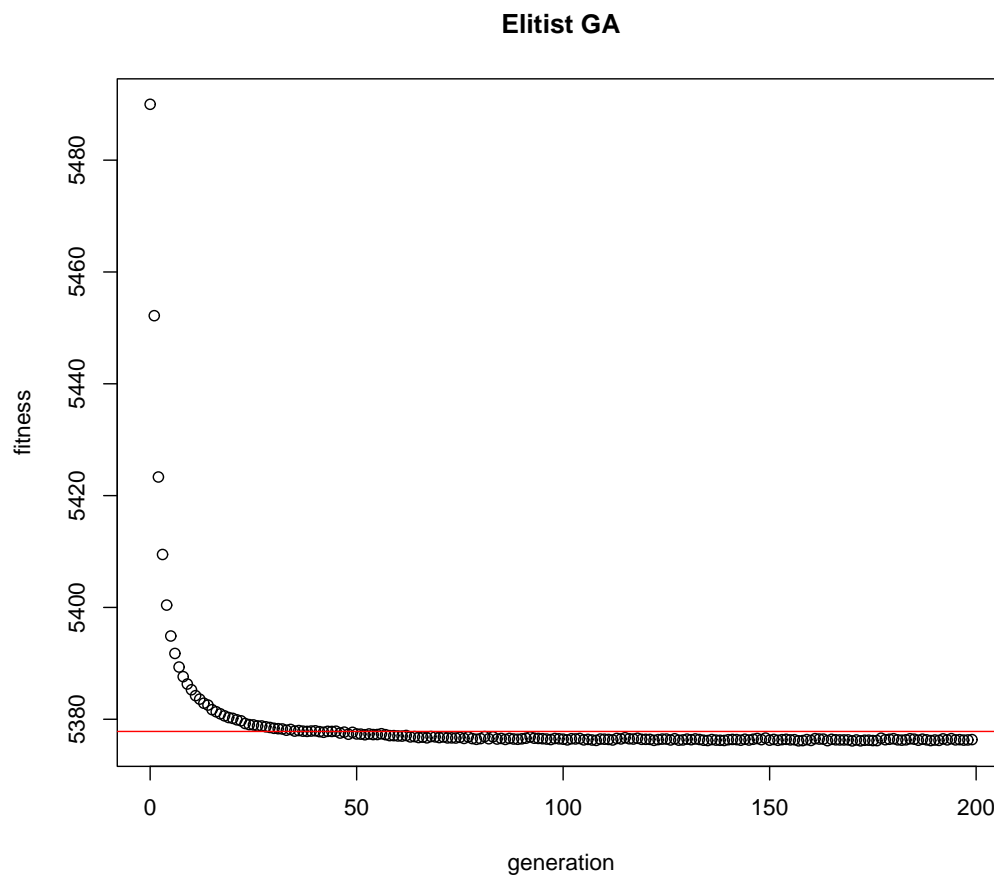


Figure 8: Results of Elitist Genetic Algorithm

The two figures that we just saw are highly detailed, but hardly comparable as regards to the convergence speed. So, let us concentrate on the first 20 generations and put both the results of SGA and EGA on one graph, as shown in Figure 9:

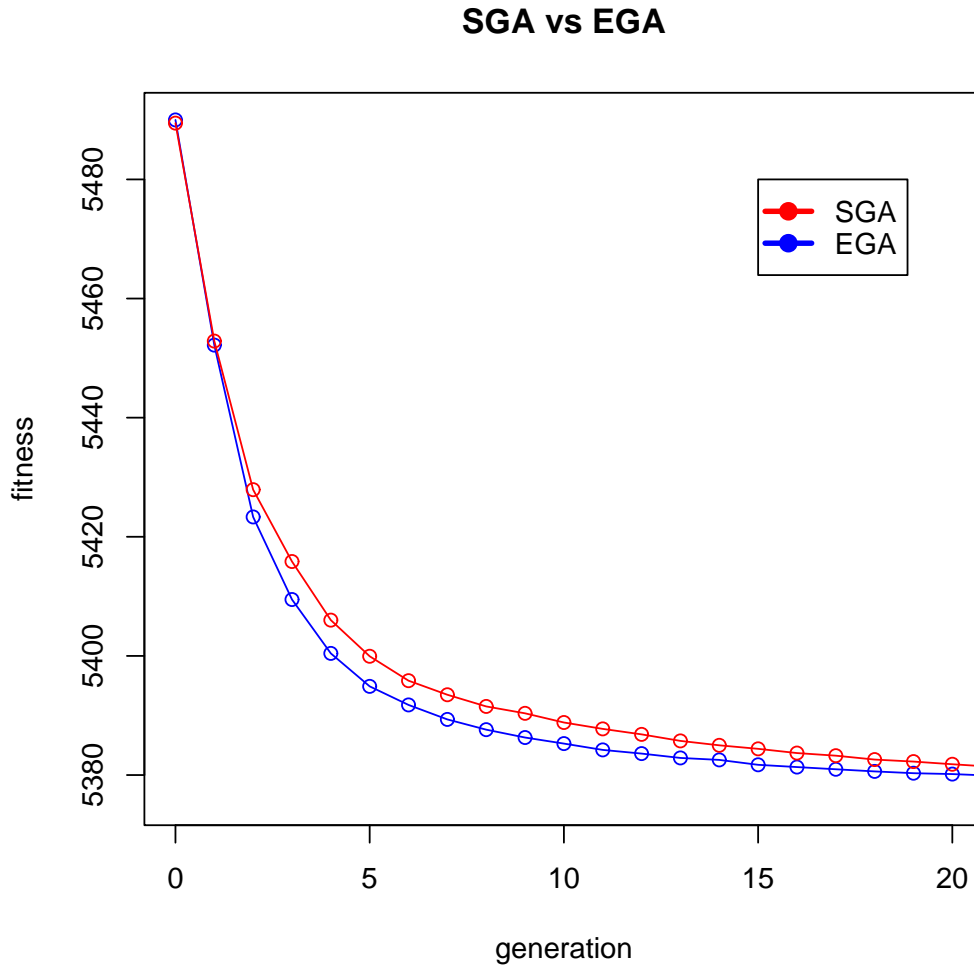


Figure 9: Comparison between SGA and EGA

From Figure 9 we can see that the elitist GA is a little faster in convergence than the simple GA. This happens because the EGA preserves the best individuals and at the beginning of the run (when there are very dissimilar individuals and many of them do not have a good genetic code) it makes a difference. In the long run, instead, the SGA too can reach good results, because it has the time to narrow the genetic pool around the best individuals' genetic codes.

After seeing this comparison, we could wonder if the crossover type makes any difference in the results of the EGA. Remember that the most common types of crossover

available are the following ones:

- *single point crossover*: one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, the rest is copied from the second parent;
- *two point crossover*: two crossover points are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent;
- *uniform crossover*: bits are randomly copied from the first or from the second parent.

As before, we concentrate on the first 20 generations and corresponding mean fitnesses.

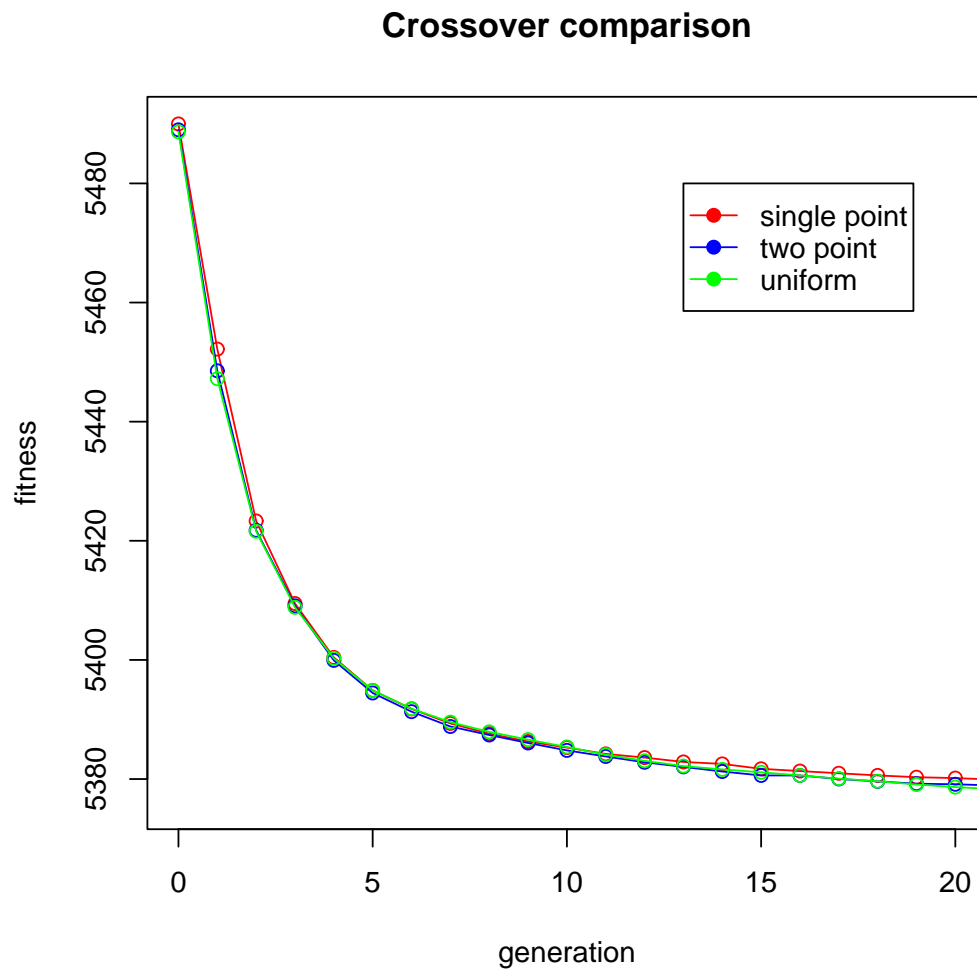


Figure 10: Comparison of the three crossover methods

From Figure 10 we cannot see a significant change in the results of the three types of crossover. If we necessarily wanted to find some difference, we could argue that the worst crossover is the single point: in fact the red line is a slightly higher than the other twos.

3.8.1 - Changing population size

We have explored some algorithmics' variations, keeping the parameters' set unchanged. Now we can try some parameters' variations (like we did in the univariate function example): we will change population size, crossover rate and mutation rate.

N.B.: For all the next tests, we will reduce *num runs* to 20 and *num generations* to 100.

We shall begin with the **population size** parameter. We will execute each of the 4 algorithms with these population sizes: 10 (small), 40 (medium), 70 (large). Then:

- for each population size, we will compare in a graph the results of the 4 algorithms;
- for each algorithm, we will compare in a graph the results of all population size values.

We begin by considering the performance of the four algorithms with population size = 10; as a first general consideration, we can say that this population is very small, so it carries little genetic variety; probably this will take us to worse results than those obtained with larger population sizes. The results are shown in Figure 11.

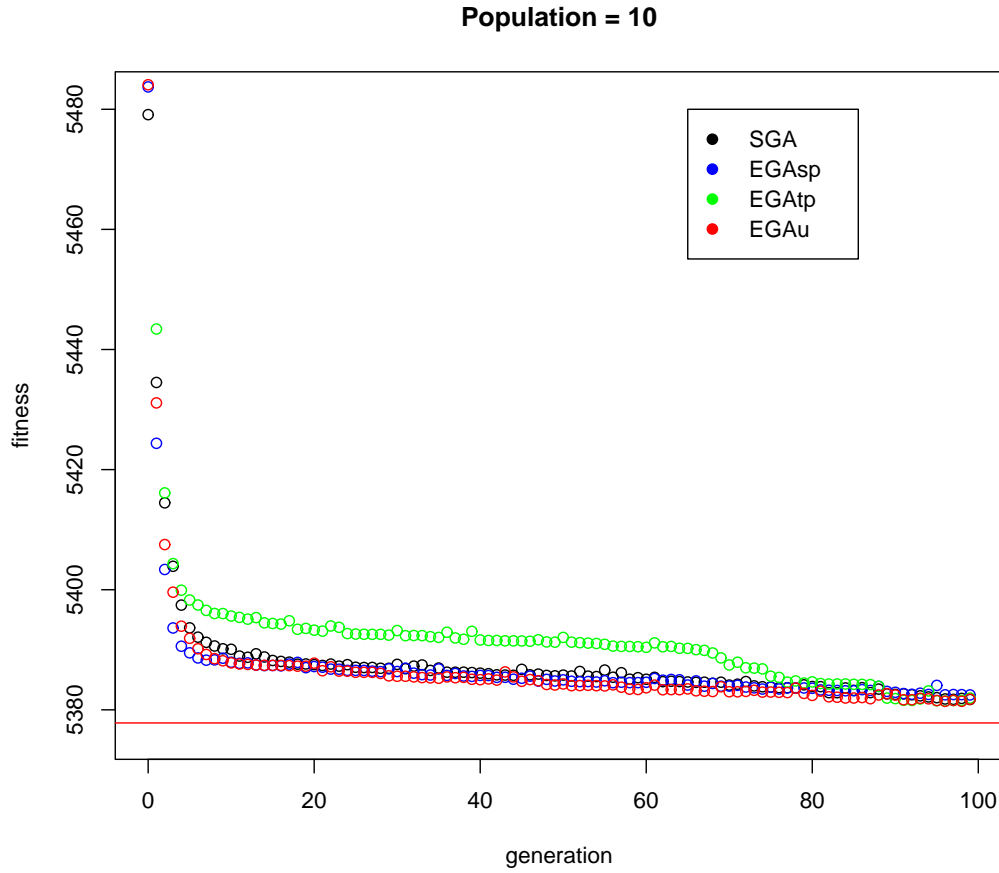


Figure 11: Population size = 10

From Figure 11, we can see what we were expecting: the convergence towards the comparison term (5377.88) is much slower than with population size = 40. In fact, after 100 generations, the mean fitness of all four algorithms is far from that value. Another thing that we can observe from this figure is the different behaviour of the elitist algorithm with two-point crossover in comparison to the other three algorithms: its convergence is noticeably the slowest. This could be due to the interaction of two facts:

- the algorithm is elitist, so the two best chromosomes are preserved at each generation: in a population of 10 elements, 2 is one fifth (a lot), so there is less shuffling of genetic codes;

- two-point crossover preserves the genetic codes more than uniform and single point crossover.

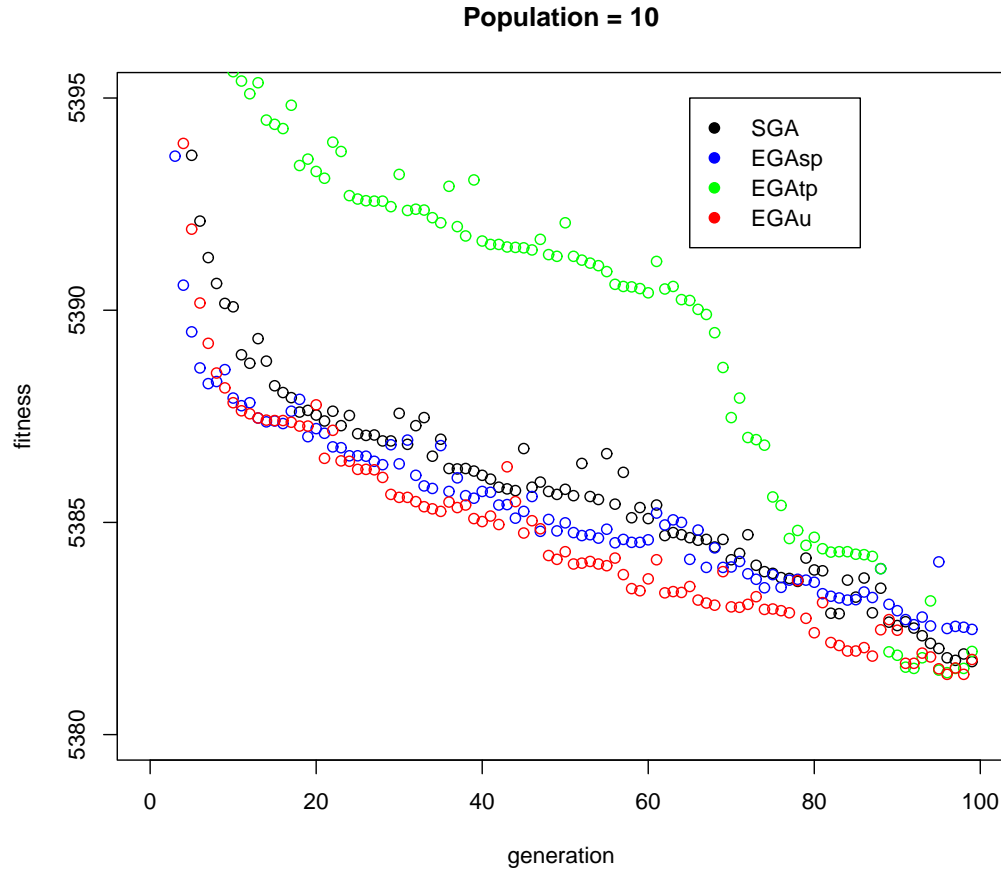


Figure 12: Zoom on population size = 10

If we look more closely in the range (5380, 5395), as shown in Figure 12, we can see that the best algorithm of the four is the elitist with uniform crossover (*EGAu*); this happens thanks to the algorithm's characteristics. In fact, the uniform crossover shuffles more randomly the genetic codes, which in a small population is fundamental to explore a larger number of candidate solutions; on top of that, the elitism assures that the best elements are not lost in this process.

We have roughly already seen what happens with population size = 40; now we have to create a graph comparable with the ones we are analysing now.

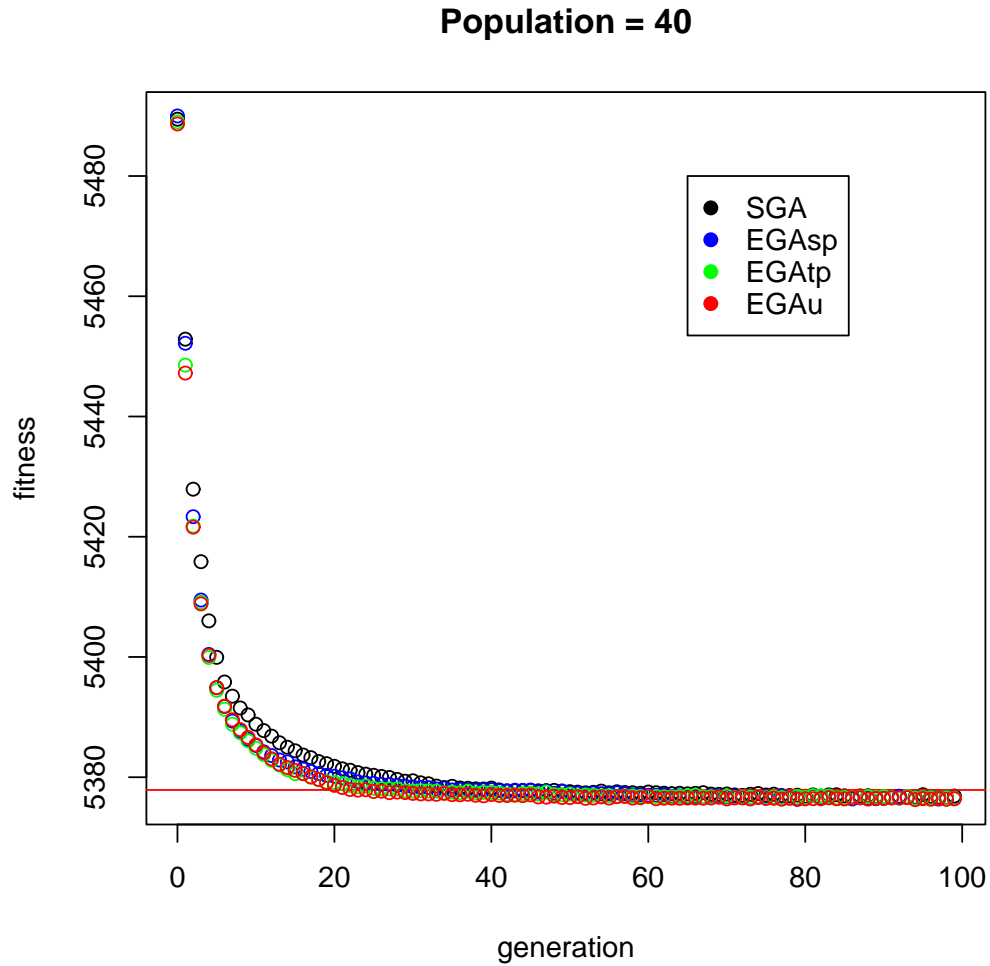


Figure 13: Population size = 40

Figure 13 shows us that all the four algorithms have a good performance, since they all reach a “final value” that is lower than the comparison term. To explore more deeply the differences between the algorithms, we can look the zoomed Figure 14:

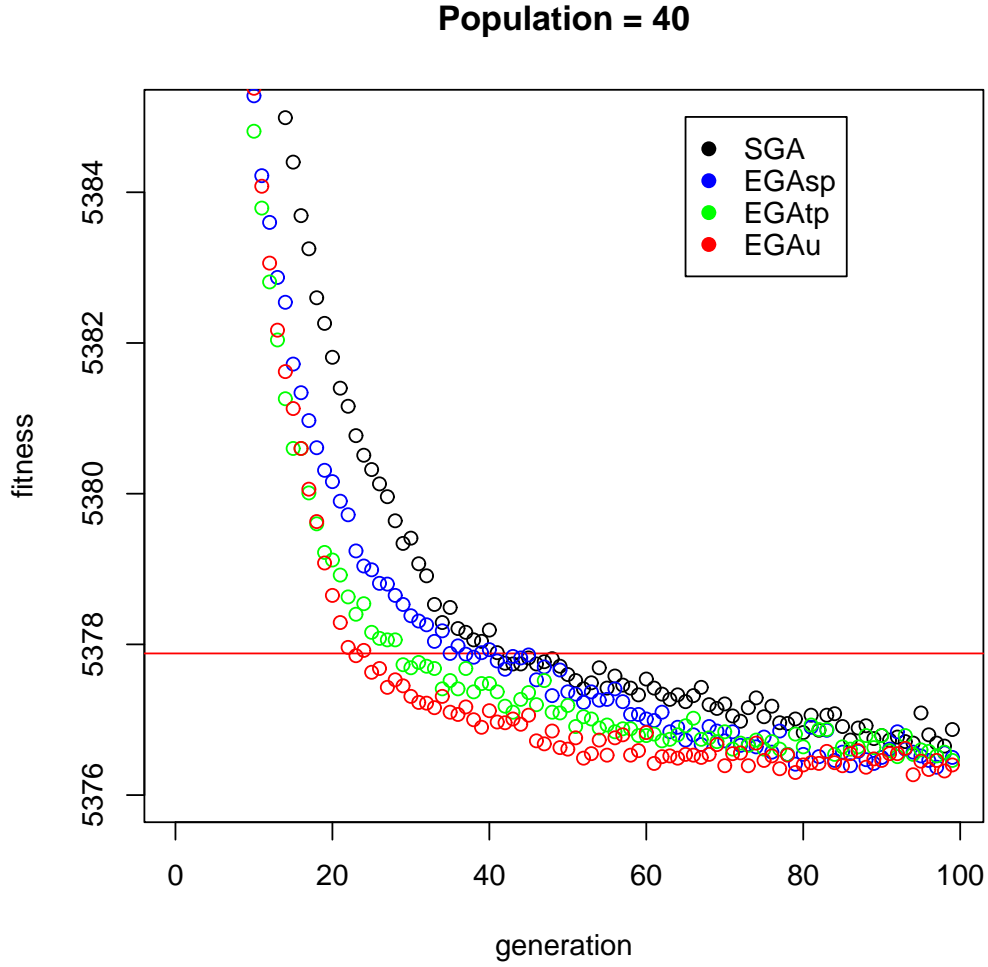


Figure 14: Zoom on population size = 40

From Figure 14 we see clearly that there is a virtual ranking in the algorithms' performances: from the best to the worse, we have EGA with uniform crossover (red line), EGA with two point crossover (green line), EGA with single point crossover (blue line) and, at last, SGA. Also in this case the EGAu turns out to be the best of the four algorithms. Anyway, this concerns mostly the convergence speed, because they all reach good results at the end of the 100 generations.

We can get on to population size = 70. We can expect a similar performance from the 4 algorithms, since with a population size so large, the algorithm's variations should

not matter so much. Moreover, the performance should be very good, having a genetic pool so vast. The results are shown in Figure 15.

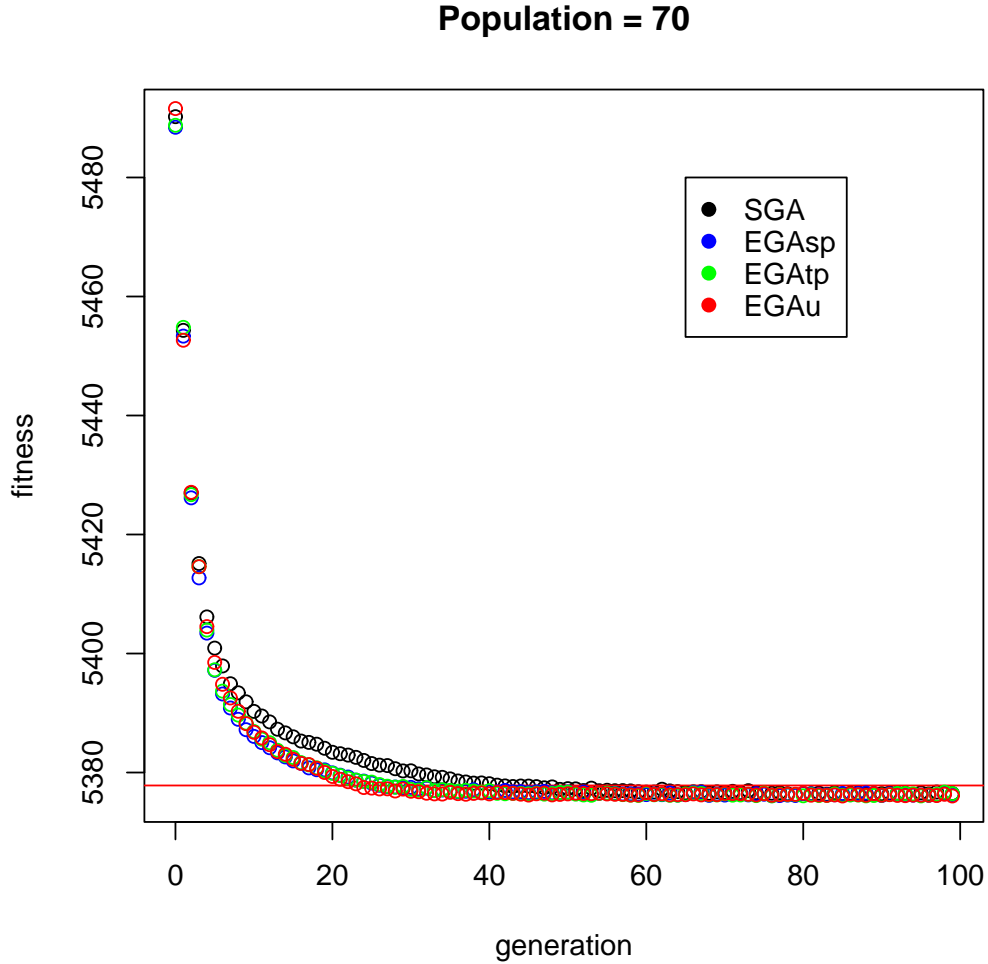


Figure 15: Population size = 70

Figure 15 confirms what we thought and at the same time introduces a new cue. In fact, the performance of the four algorithms is good, like we prefigured; the convergence speed, otherwise, is very good, but *only* for three of the four algorithms.

The elitist-algorithms (*EGAsp*, *EGAtp*, *EGAu*) convergence is quicker than the one with population size = 40: at the 20th generation their mean fitness is already below 5380 (with population size = 40, this happened after 27 generations).

The convergence of the non-elitist algorithm (SGA), instead, is slower. We can look more closely at it, narrowing the graph to the range (5376, 5385).

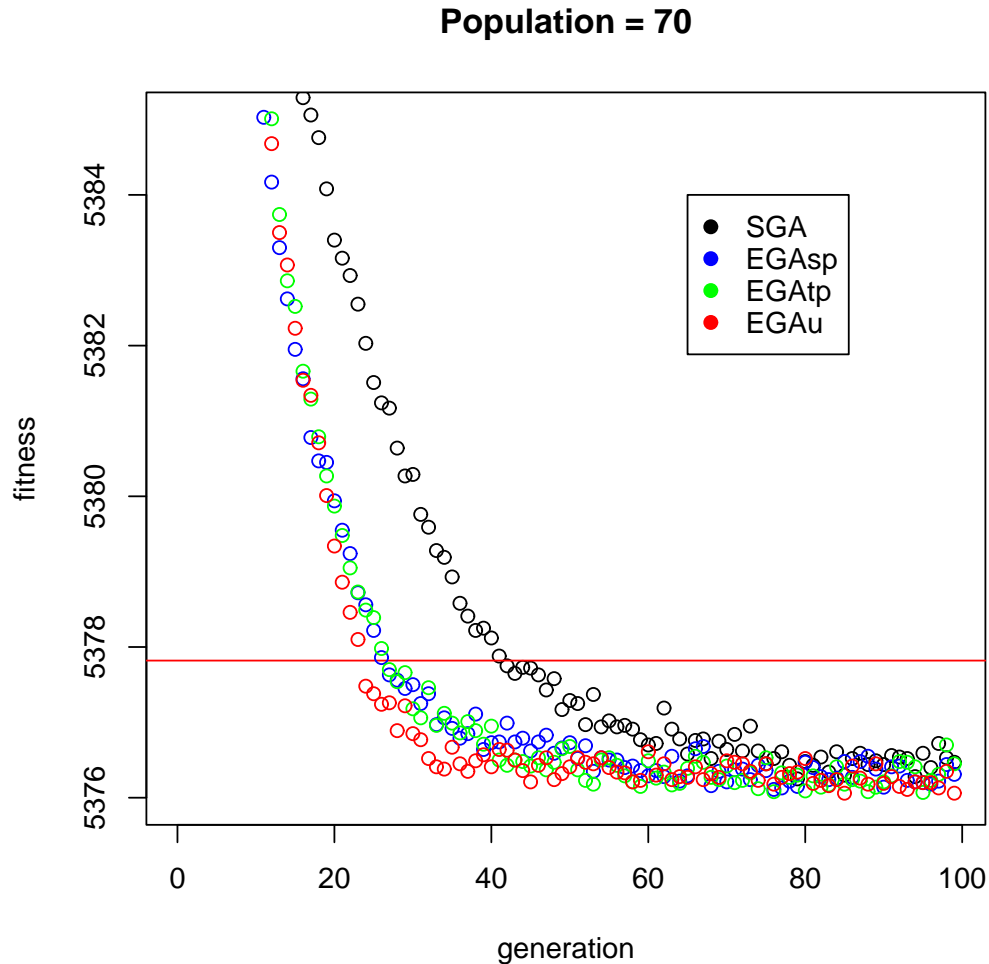


Figure 16: Zoom on population size = 70

We see more clearly from Figure 16 the difference of SGA with respect to the other algorithms. The SGA's convergence speed is firmly worse than the other three genetic algorithms' ones, especially if we compare this figure to Figure 14, in which the algorithms' convergences were fairly close. Why in this case the SGA moves itself away such markedly? A possible explanation is that the SGA could be less capable (when having such a large genetic pool available) of focusing on the very best chromosomes,

whereas the elitists versions have precisely this distinctive ability, so they let their light shine in circumstances like this (vast populations).

Now we can see, for each of the four algorithm, a comparison based on the three population sizes we tried (small, medium and large). We shall begin with the simplest of the four algorithms, that is SGA:

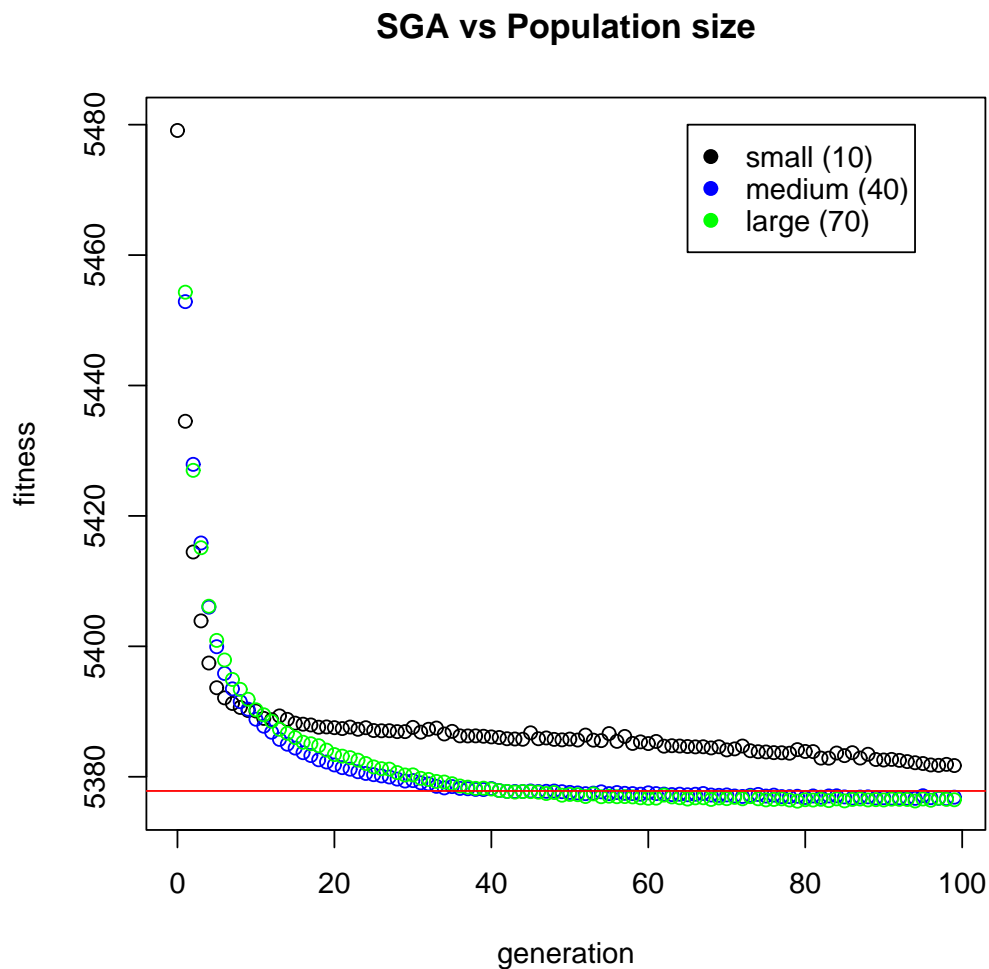


Figure 17: SGA versus population size

It is evident from Figure 17 that the small population size takes the algorithm to a seriously slower convergence than with the other two sizes. Moreover (and more

importantly), it prevents the algorithm from reaching a good final value. As for the relationship between the medium and the large population sizes, we can shrink the vertical range of the graph:

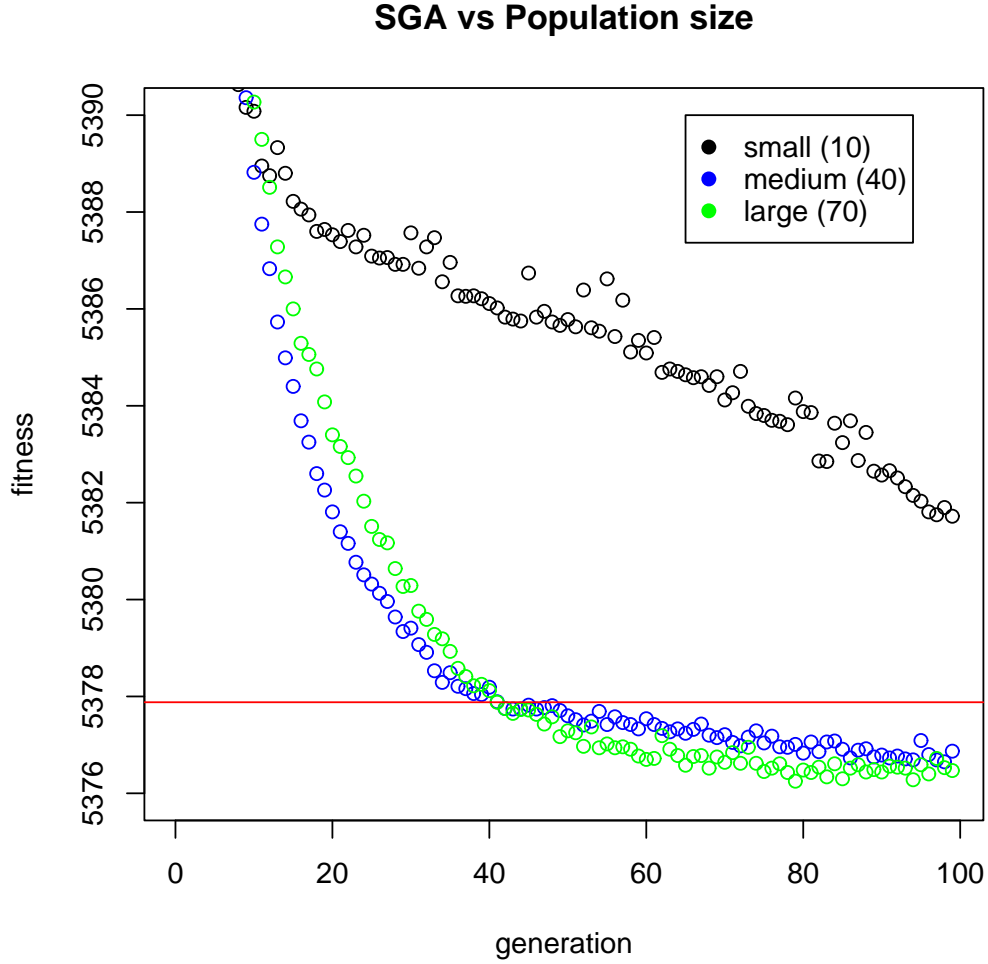


Figure 18: Zoom on SGA versus population size

From Figure 18 we see a strange behaviour: until about the 40th generation the medium-sized algorithm is better than the large-sized one; after that, the relationship changes and the large-sized algorithm does better than the medium-sized one. Considering that what we want is a good final result, we prefer the large-sized algorithm, since it does better on the last stages, reaching a lower average AIC value.

Let us pass on to the elitist algorithm with single point crossover (the one called EGAsp in the previous graphs).

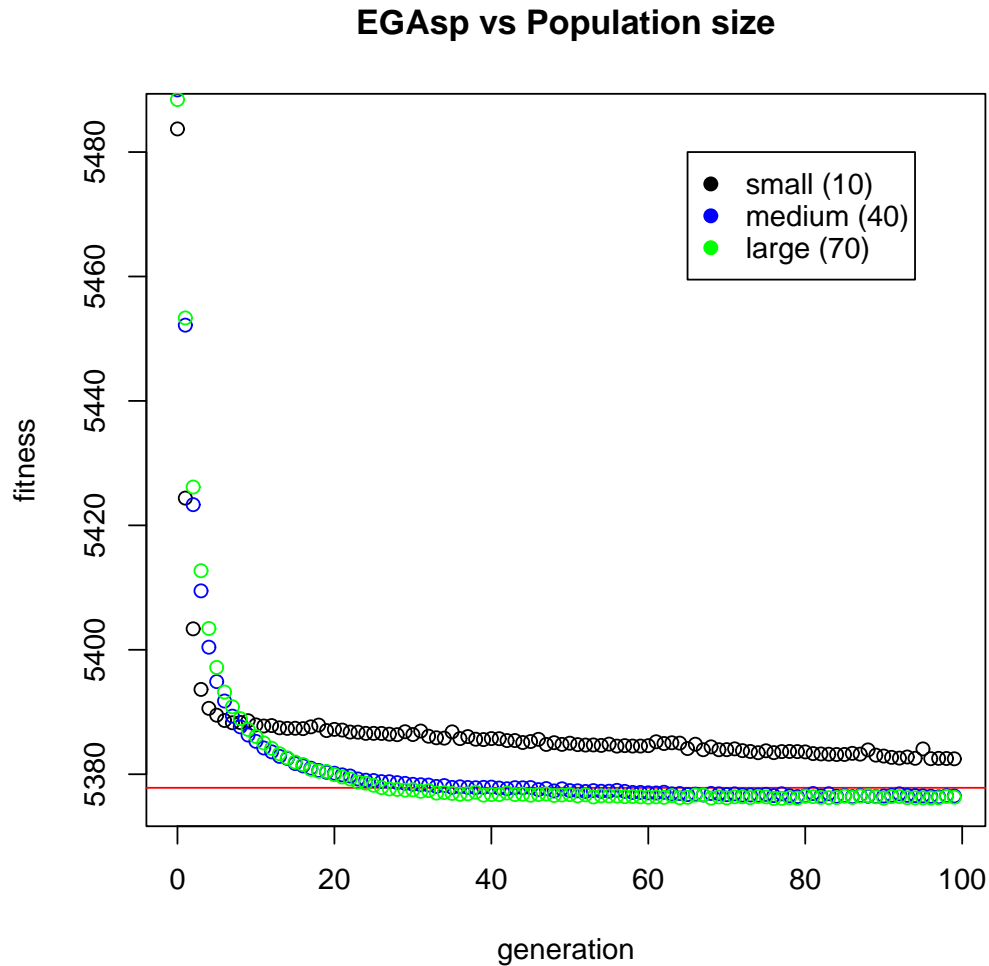


Figure 19: EGA with single point crossover versus population size

Figure 19 resembles very much Figure 17: the performance of the small-sized algorithm is far worse than the other two, which appear very similar. In order to see the difference between them, we have, as usual, to focus only on a small part of the y -axis.

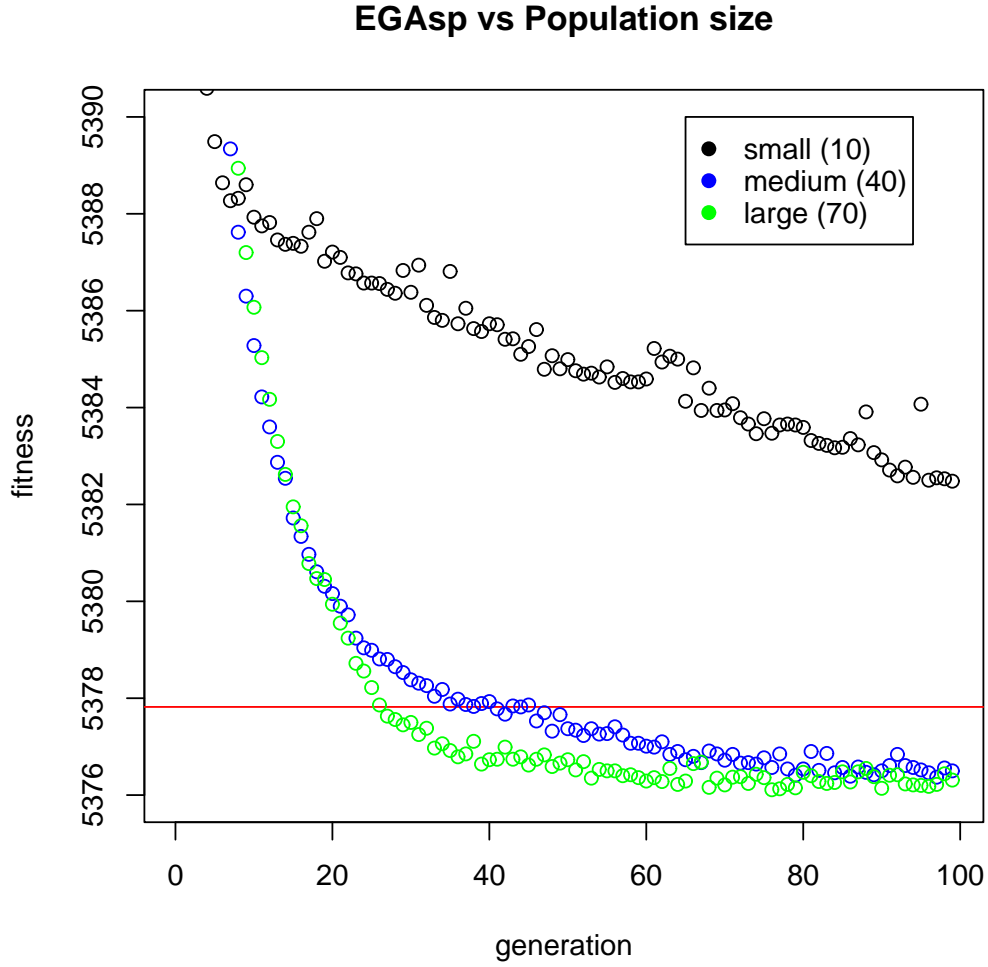


Figure 20: Zoom on EGA with single point crossover versus population size

In Figure 20 we can see that till the 20th generation, the two algorithms have almost the same mean fitness; after that, though, the large-sized algorithm outperforms (in convergence, not in “final value”) the medium-sized one.

Now, we will see the results of the elitist algorithm with two-point crossover (the one called EGAtp in the previous graphs).

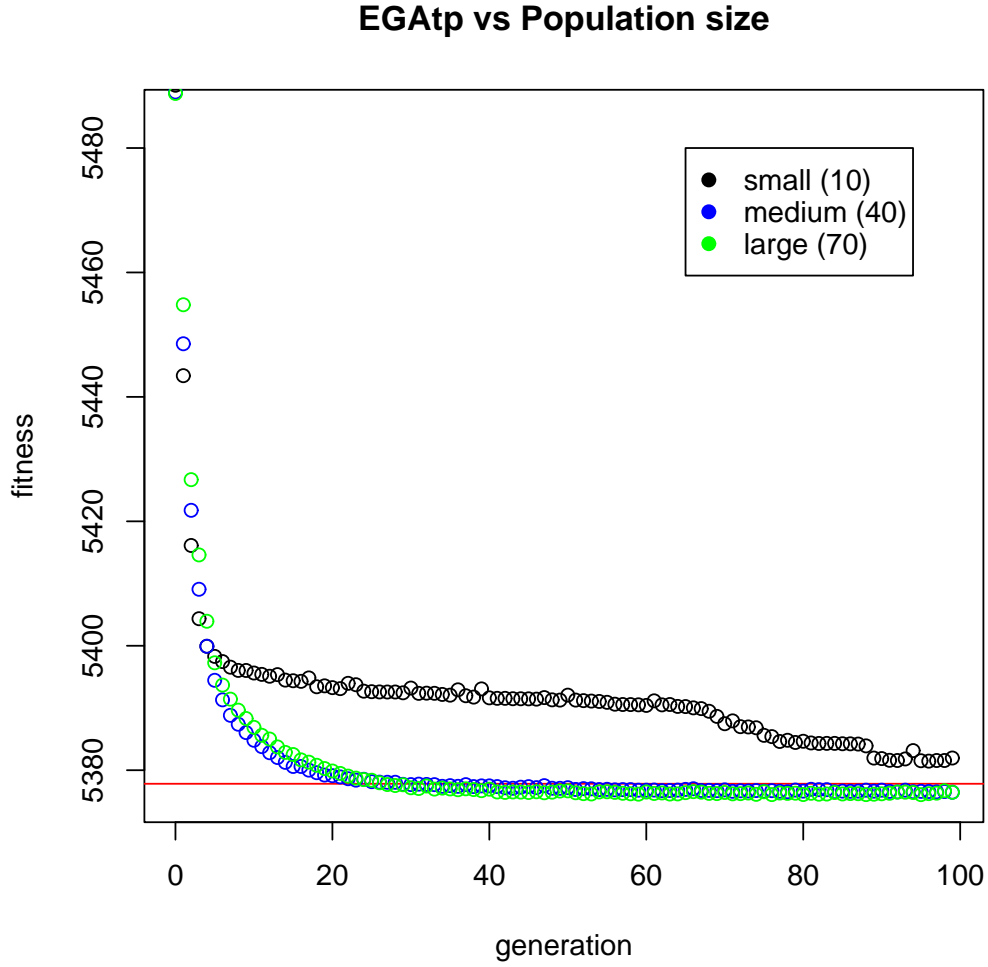


Figure 21: EGA with two point crossover versus population size

In Figure 21 we can see the same pattern of the previous figures (17 and 19), but with a difference:

- in Figure 17 and 19, the black line stayed always at a almost fixed small distance from the blue and the green lines;
- in Figure 21, instead, for the first 60 generations the black line stays at a very bigger distance, but after that it begins a descent that takes it close to the other lines (circa at the same level of Figure 17 and 19).

This could take us to the conclusion that the elitist GA with two-point crossover is highly unfit with small populations. As regards to the comparison between medium-sized and large-sized algorithm, let us develop the following comparison.

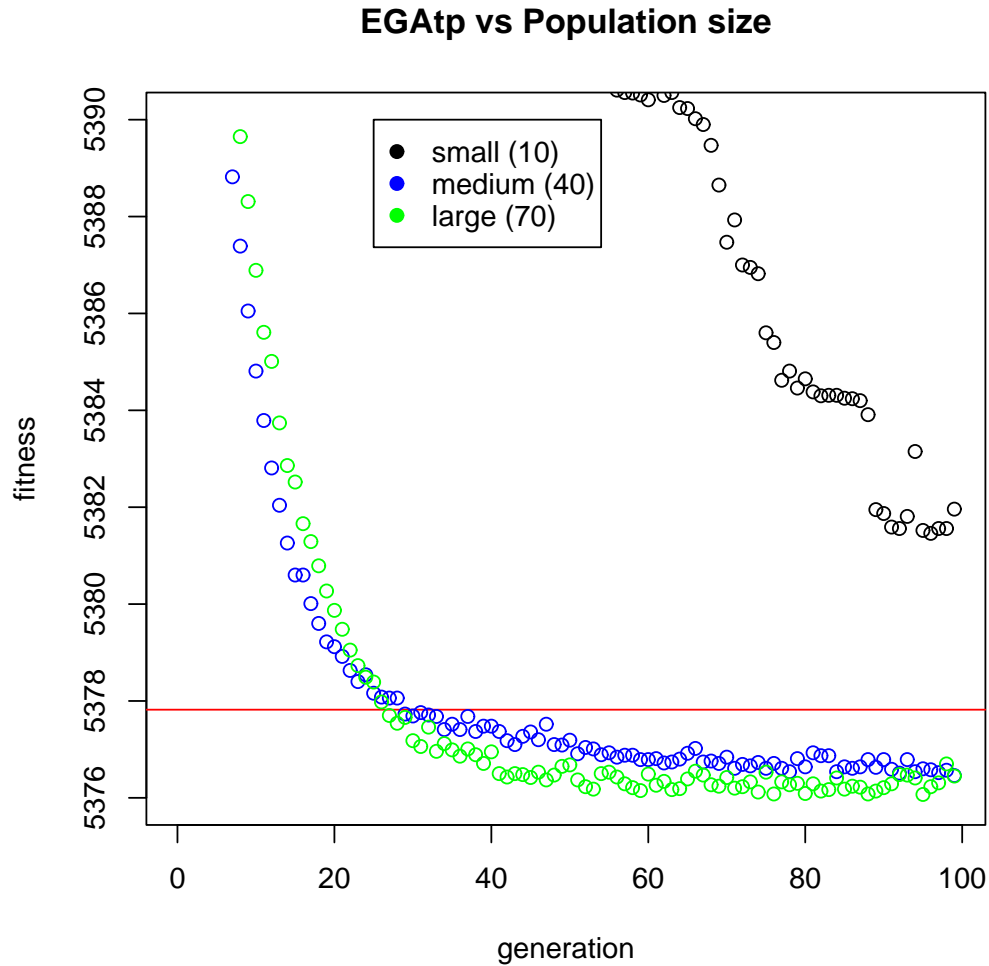


Figure 22: Zoom on EGA with two point crossover versus population size

In Figure 22 we can see a relationship (between the two algorithm) very similar to the one observed in Figure 18.

Finally, we shall see what the elitist algorithm with uniform crossover (called EGAu) has to offer us.

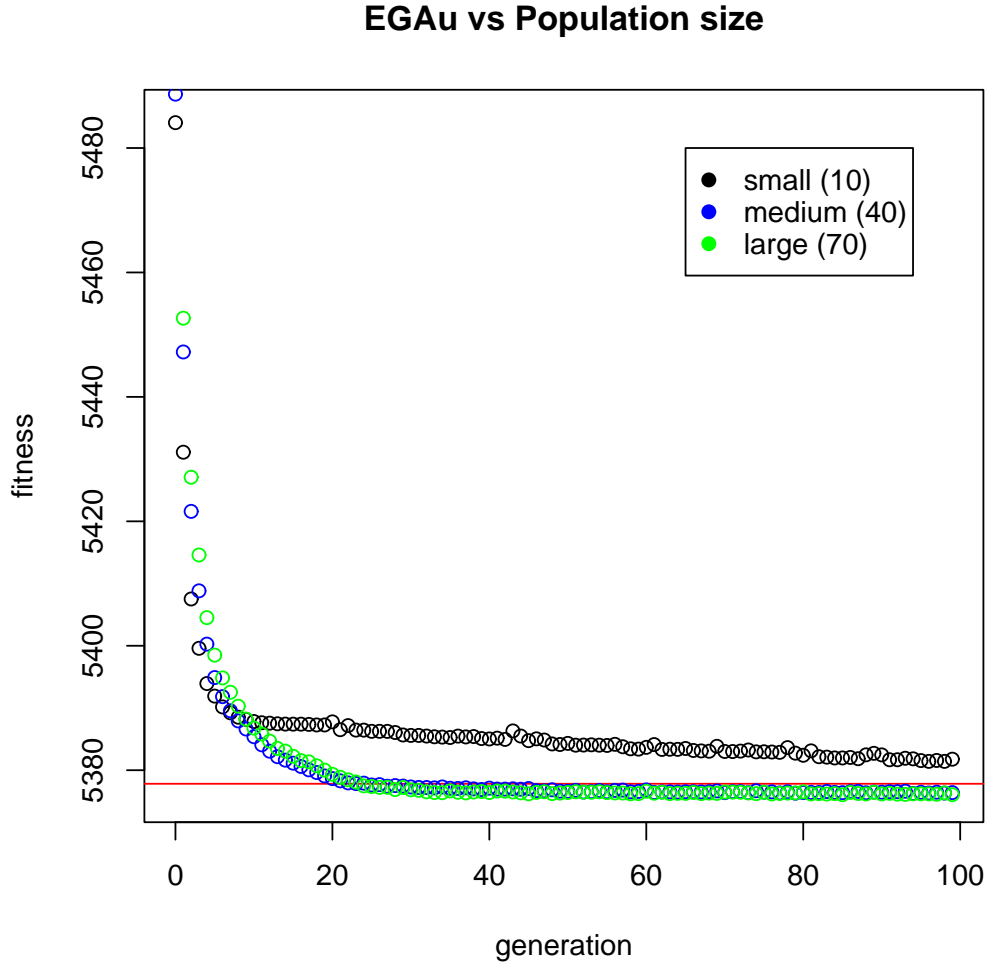


Figure 23: EGA with uniform crossover versus population size

Figure 23 does not present us with anything new than we already observed: small-sized algorithm's performance can be called "bad" compared to the other two algorithms (medium-sized and large-sized). To complete this analysis about population sizes' and genetic algorithm's variations, we shall see the last plot (an y -axis reduced version of the previous one).

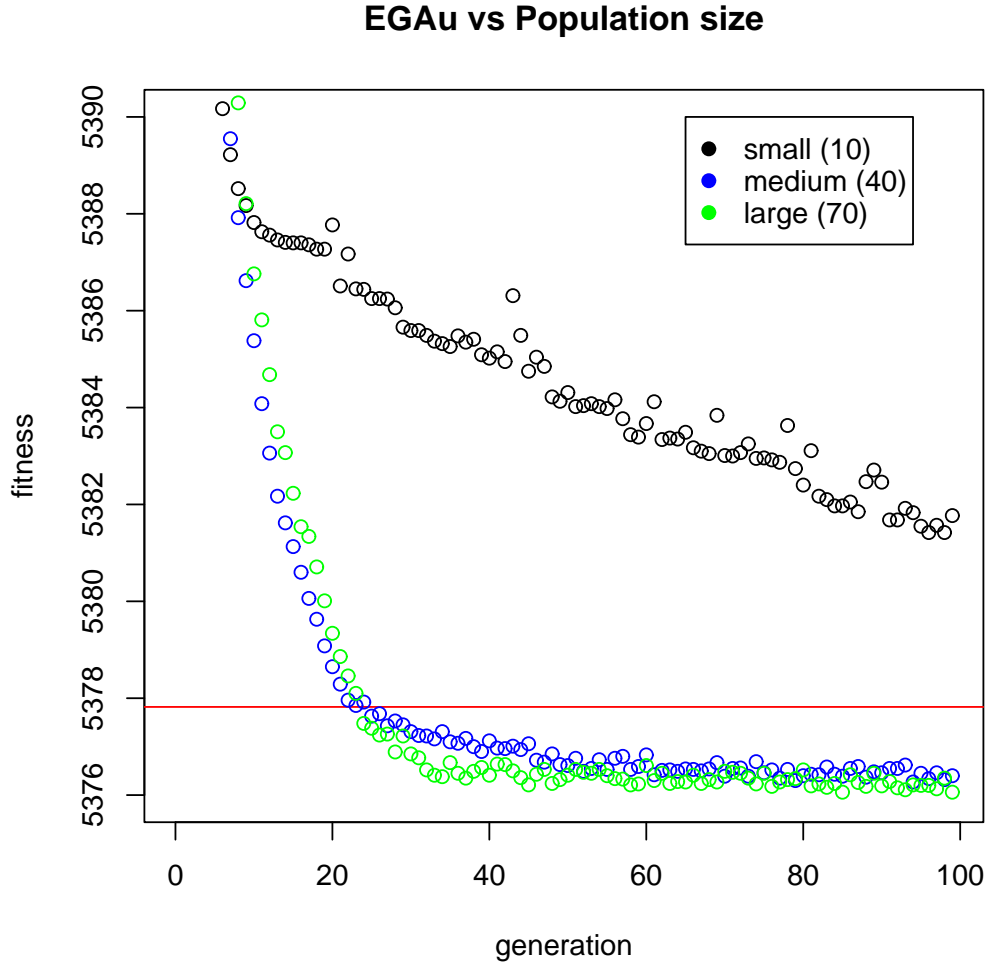


Figure 24: Zoom on EGA with uniform crossover versus population size

There is not much to comment about Figure 24, too, being it alike the other ones.

Which conclusions can we derive from this analysis? A first, evident remark is this: the larger the population, the better. In fact, with all the four genetic algorithm's variants, the best results were obtained with the large-sized version. Although, we can go deeper into this issue: we can say that with very small populations the results are not satisfactory (we remain always above the comparison term taken from the step-wise regression of R), and this happens especially with EGAtp. Instead, the results of

medium populations are almost as good as the large populations ones, while requiring less computations to be produced. In fact, we have to take into account also the executional length of the algorithms. Considering this, a medium population size is probably the best trade-off between goodness of results and speed of execution.

And what about the algorithm variants? Which of them could be the best catch? Excluding from these thoughts the results obtained with the small population size (which are not significant as they were not satisfactory), we could say that the worst algorithmic variant is the SGA (which convergence is the slowest); the elitist algorithm are instead all very good, in particular with large population sizes; when dealing with medium population sizes the best of the three is the EGA with uniform crossover.

3.8.2 - Changing crossover rate

We have analyzed all about population sizes and their relationship with some algorithmic variants. Now, we will do a similar analysis in which the subject is, instead of population size, **crossover rate**. Remember that the crossover rate is the probability with which two individuals' genetic codes are mixed to form the offspring, instead of copying their genetic codes straight into the offspring; in other words it represents the probability of effectively doing the crossover. As we did in the analysis of population size, we will consider three possible values for crossover rate: 0.25 (low), 0.75 (medium, the one we always used till now), 1.00 (high). The population size that we will use in these tests, given our final considerations on that matter, is 40 (medium).

Let us begin with the small crossover rate:

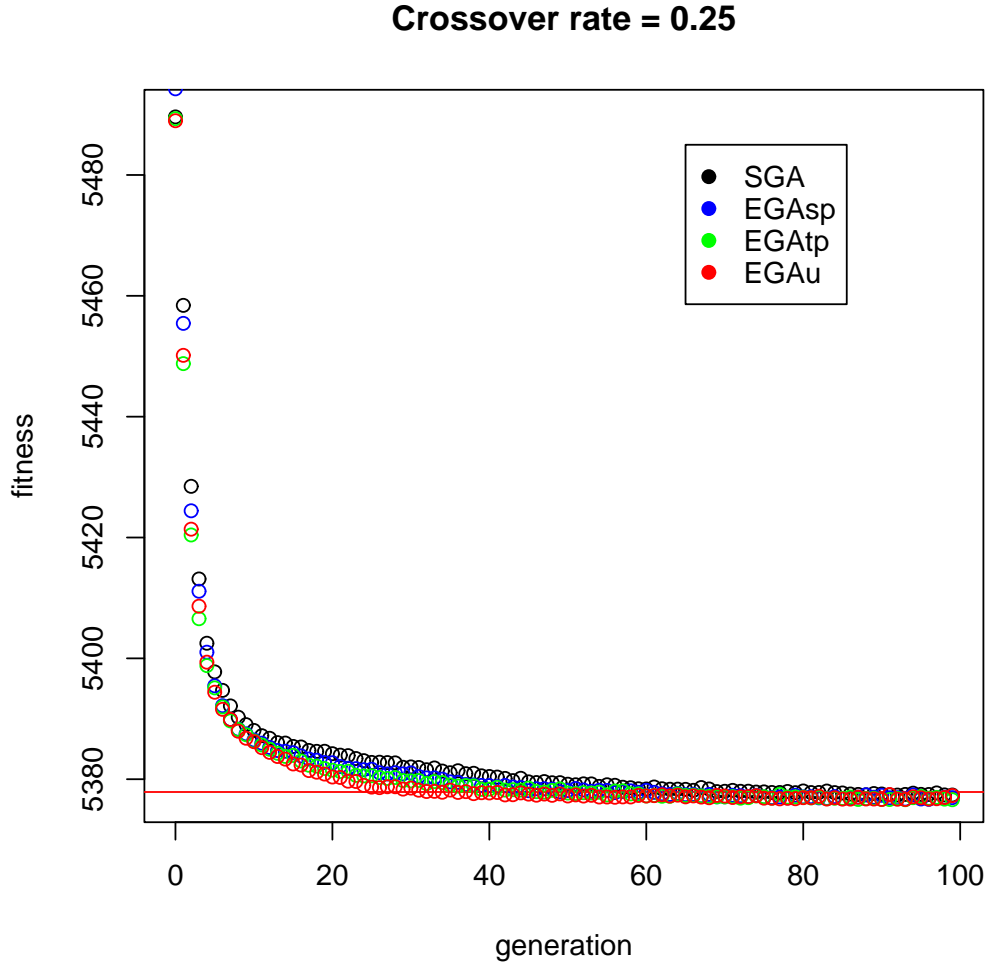


Figure 25: Crossover rate = 0.25

From Figure 25 we are compelled to do the same considerations that we did before: the performance ranking between the algorithms' variants, with a given set of parameters, is EGAu, EGAtp, EGAsp, SGA (in descending order). As usual, we can see this more clearly in the following figure i.e. Figure 26:

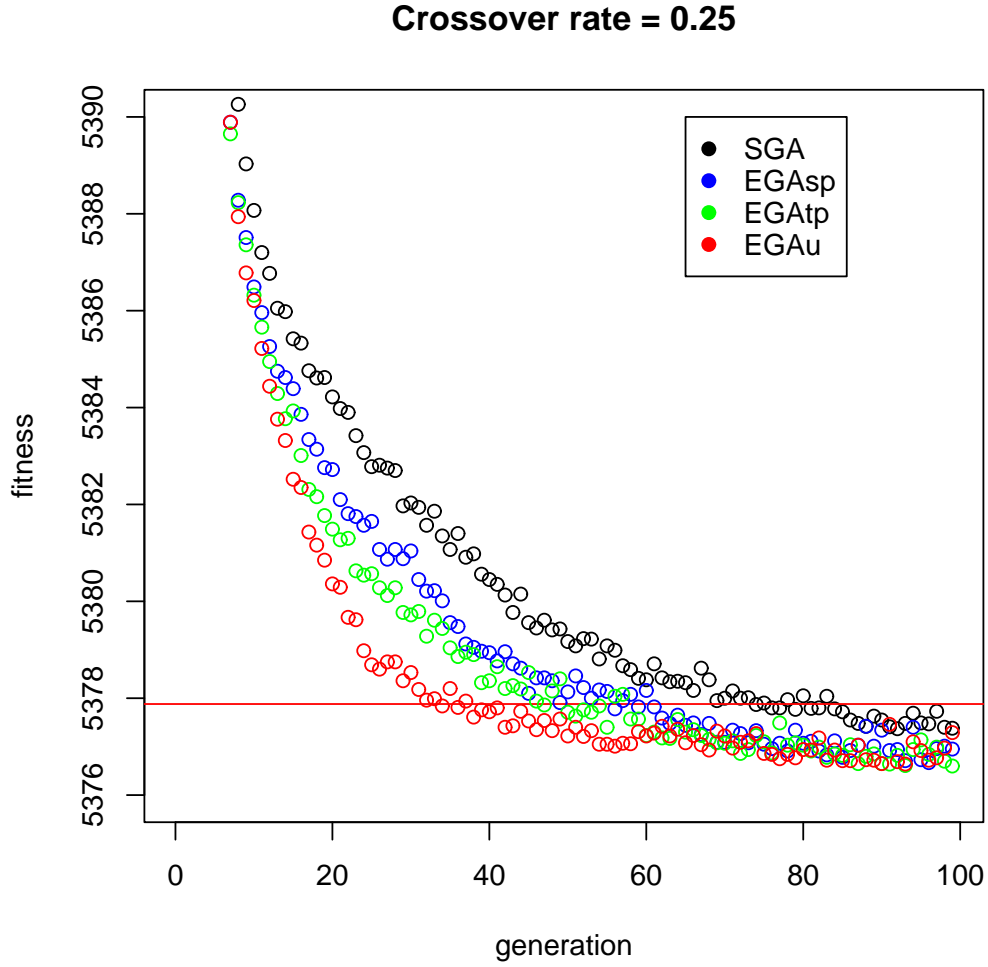


Figure 26: Zoom on crossover rate = 0.25

We already analyzed and commented the results obtained with the medium value of crossover rate (0.75). Precisely, we did it in Figure 13 and 14 (where the population size was 40 and the crossover rate 0.75). Therefore, we can get on to the high value of crossover rate (1.00): this means that the algorithm always carries out the crossover.

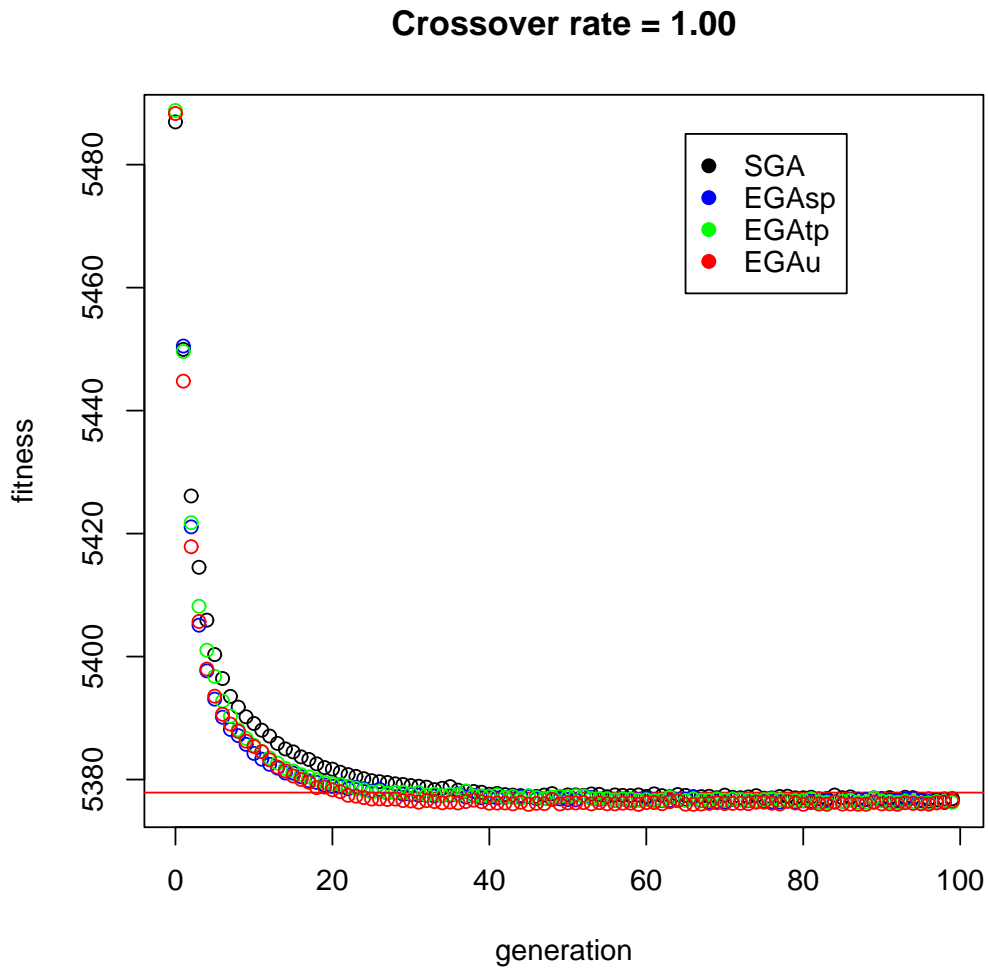


Figure 27: Crossover rate = 1.00

Apart from the usual “SGA’s convergence is slower” observation, we cannot say much from Figure 27 (because the lines are too intertwined); reducing the vertical range of the graph takes us to the following.

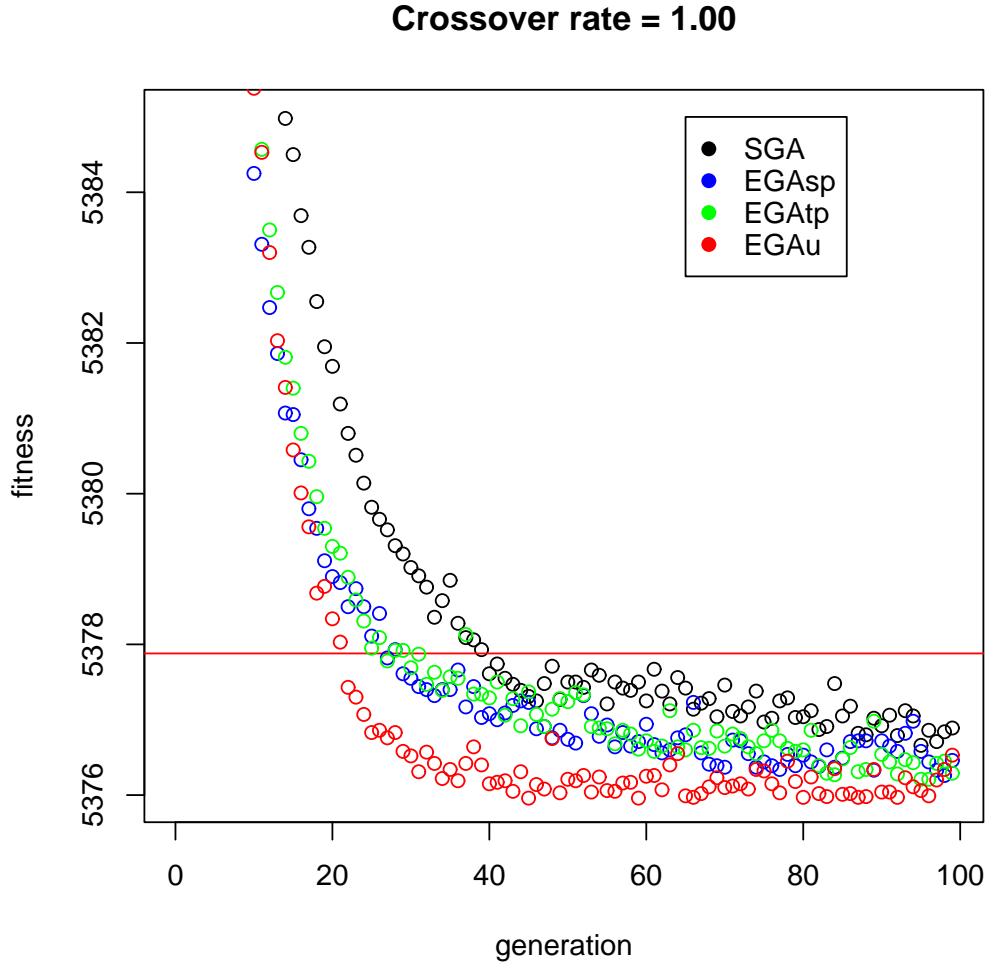


Figure 28: Zoom on crossover rate = 1.00

Figure 28 tells us no more than what we already know: the best among the four algorithms is the elitist one with uniform crossover, followed by the single and the two point algorithms (which behave quite similarly); last of all comes the SGA.

Now we can compare the three crossover rate values performances on each of the four algorithmic variants, starting with SGA.

N.B.: Since the “wider” graph doesn’t reveal much, we can skip it and get on to the “reduced” graph:

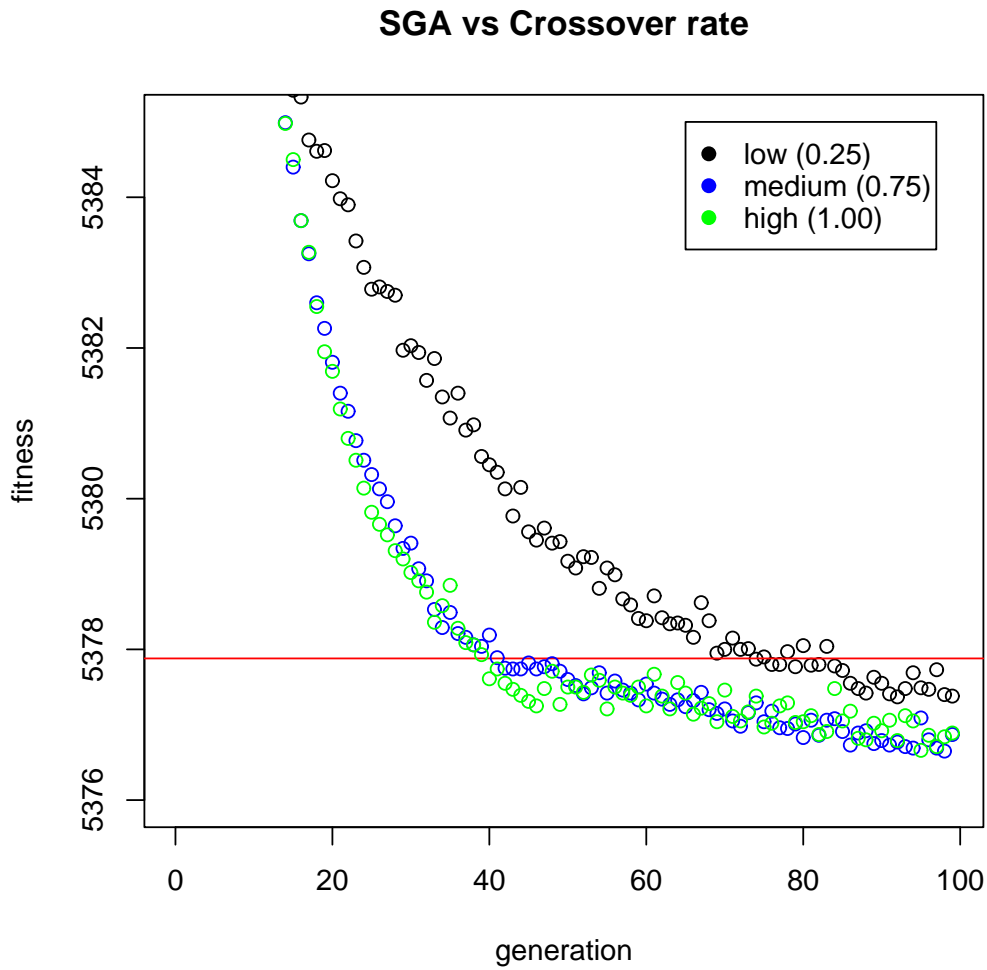


Figure 29: Zoom on SGA versus crossover rate

In Figure 29 we can see that the black line is always above the other two lines (blue and green): the algorithm with a low crossover rate performs worse than the medium and high rates' variants. In fact, it reaches the comparison term after about 70 generations, while the other two reach that after only 40 generations (near a half!). Moreover, the low rate algorithm does not go much below the comparison term, while the medium and high rates ones do. From this first graph, we could infer that the crossover is an important part of the genetic algorithm; we shall see if we can confirm this sensation with the other three algorithmics' variants.

As before, we do only the “reduced graph”, but this time with the elitist GA with single point crossover.

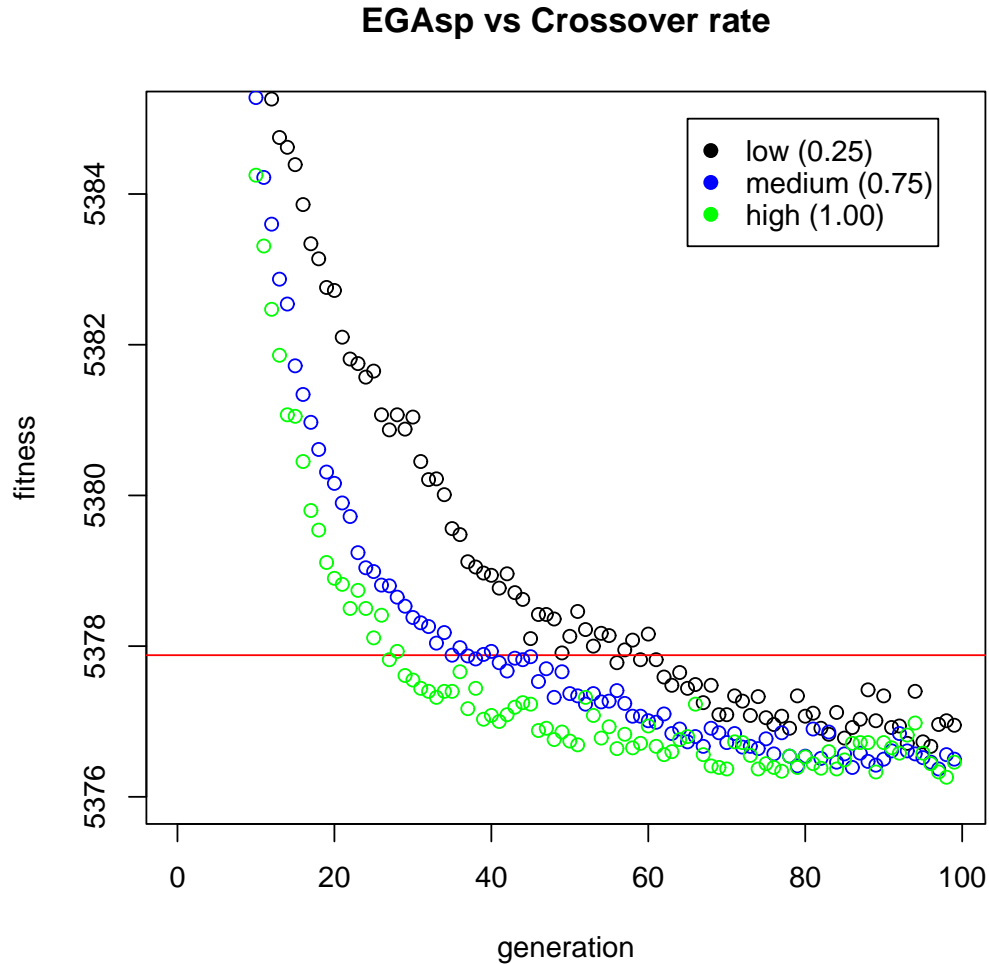


Figure 30: Zoom on EGAsp versus crossover rate

From Figure 30 we can derive conclusion similar to the previous ones. In this case, though, there is quite a difference between the medium-rated algorithm and the high-rated one (while in the previous figure their lines were almost overlapped). The conclusion is always that crossover is fundamental for the GA’s performance. In fact, in this case, the best of the three variants is definitely the one with crossover rate = 1.00 (it reaches the comparison term after circa 25/30 generations). Let us pass on to

the elitist GA with two point crossover.

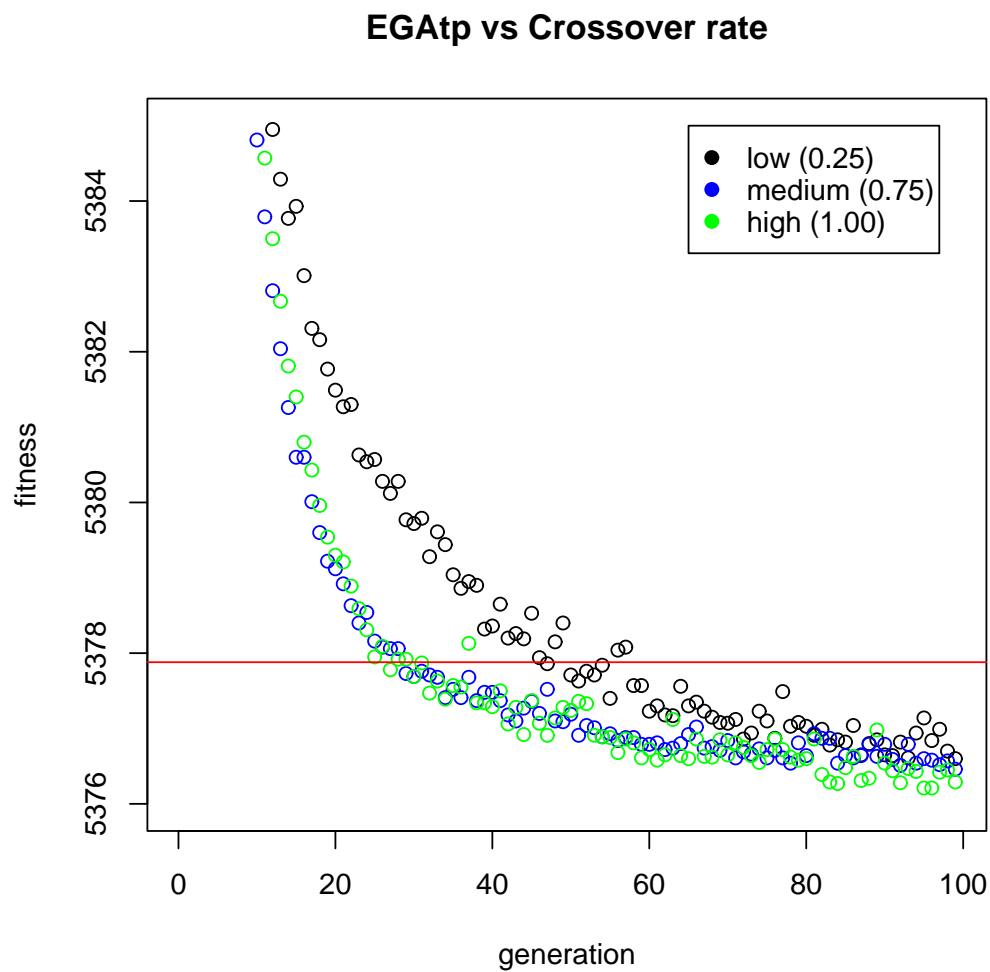


Figure 31: Zoom on EGAtp versus crossover rate

Figure 31 is very alike Figure 29, with the difference that here the black line (low crossover rate) is closer to the other two lines (especially in the last twenty generations, where they almost intertwine themselves).

The last algorithmic variant to analyze is elitist with uniform crossover.

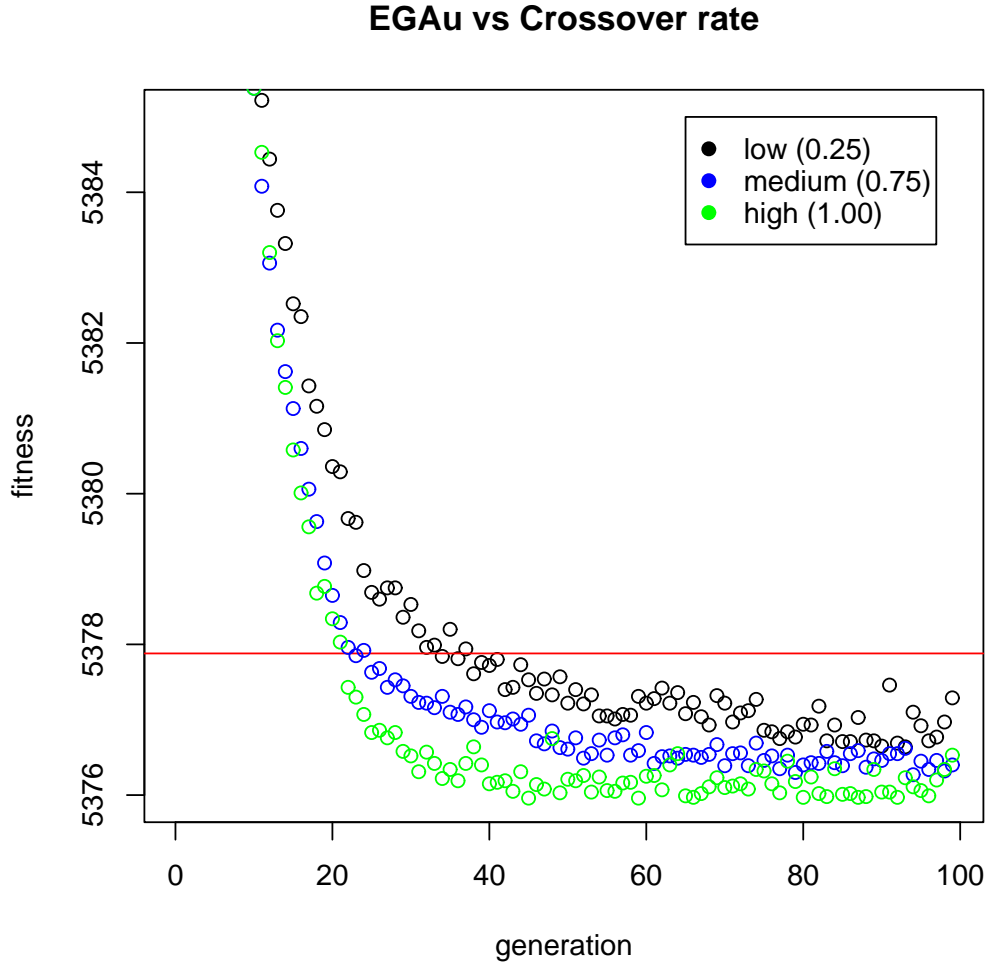


Figure 32: Zoom on EGAu versus crossover rate

This figure (Figure 32) is similar to Figure 30 and does not present any novelty.

What have we learned with this analysis? Firstly, we confirmed one conclusion derived from the analysis on population sizes (that is, the elitist GA with uniform crossover is the best algorithmic variant). Besides, we recognised the key role played by crossover in genetic algorithms, since, with all algorithmic variants, a low crossover rate made the results unsatisfactory.

3.8.3 - Changing mutation rate

Now, we should make inquiry about **mutation rate**. The role of it, though, seems very clear and transparent: the higher the rate, the worse the performance (we saw this in the univariate function example). So there is not the need to make all the test that we did in the previous analysis'. We can choose a single algorithmic variant and explore mutation rate only on that algorithm. Given our former conclusions, we pick the elitist GA with uniform crossover (EGAu), which proved to be the best algorithmic variant. As regards to the parameters, we can opt for the ones which took to best results in our tests with EGAu; that is 1.00 for the crossover rate and 70 for the population size (we keep constant the number of generations to 100 and the number of runs to 20). As mutation rates we can compare 0.1 (very high), 0.01 (high), 0.001 (medium, the one used till now), 0.0001 (low). The results of this experiment is represented in Figure 33:

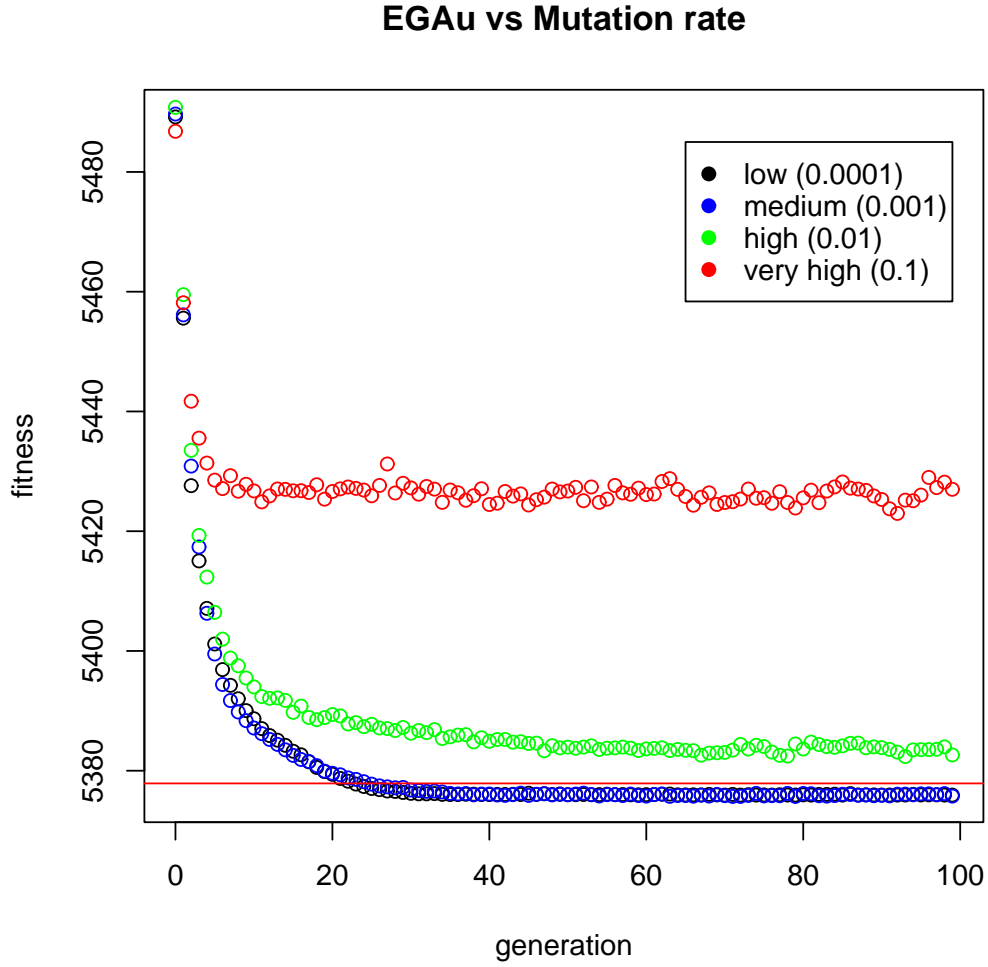


Figure 33: EGAu versus mutation rate

With a mutation rate of 0.1 (very high) the algorithm's performance is, as we expected, very poor; in fact, the randomness introduced in the algorithm is definitely too much. Instead, with a mutation rate of 0.01 (high) the algorithm's performance is not good but at the same time it is not so bad; the green line is, indeed, not so far from the blue and the black lines.

However, the best performances came (quite obviously) from the algorithms with medium and low mutation rates, that is the blue and the black lines. Those two lines are quite lower than the R's comparison term and they are practically intertwined.

Since they are too close to derive some conclusion, we can focus our attention on them and enlarge the figure.

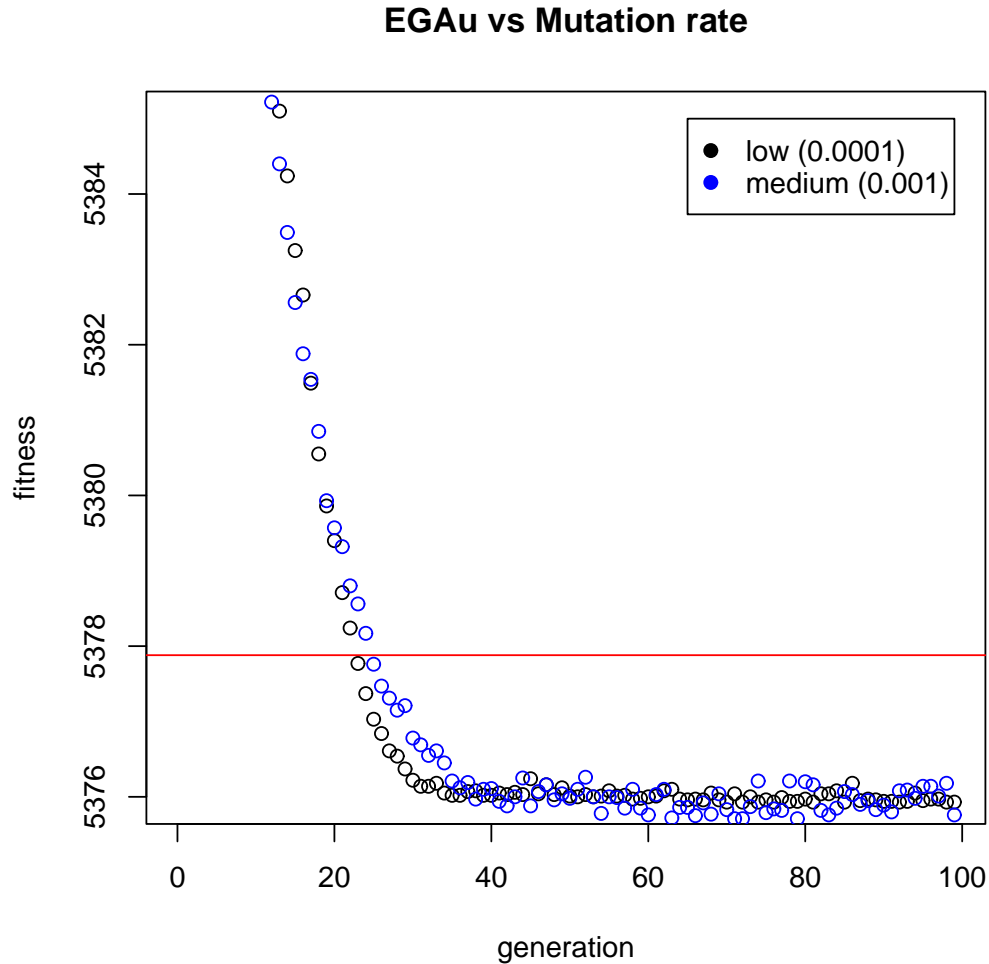


Figure 34: Zoom on EGAu versus mutation rate

From Figure 34 we can see that there is not a lot of difference between the two performances; they converge at almost the same speed and they halt at almost the same value (5376). The only clear distinction is that the black line is much “smoother” than the blue one, which, instead, continues to go up and down (even if a little) to the end of the generations. Evidently, this happens because of the mutation rates values: the lower that value, the fewer the fluctuations.

From the latter experiment we corroborated the hypothesis that we made: the higher the mutation rate, the worse the performance. However, we have to remember that, as a general rule, the mutation rate cannot be 0 (or something like that), because a little of randomness in shuffling the genetic codes can, *sometimes*, take the algorithm to a better solution. In our case, though, the fitness function appears to be convex: finding a local maximum should be equivalent to finding the global one (which should be located in the vicinity of 5375, as regards the AIC value). Given this assumptions, removing the mutation from the algorithm should not worsen the results: instead, they could actually improve! This happens because what the algorithm is doing (under the assumption of convexity) is simply ascending the gradient, and introducing randomness in this process (when there is only the global maximum) can slow it down. We can try this interesting experiment (executing the algorithm with a mutation rate of 0) to prove these claims.

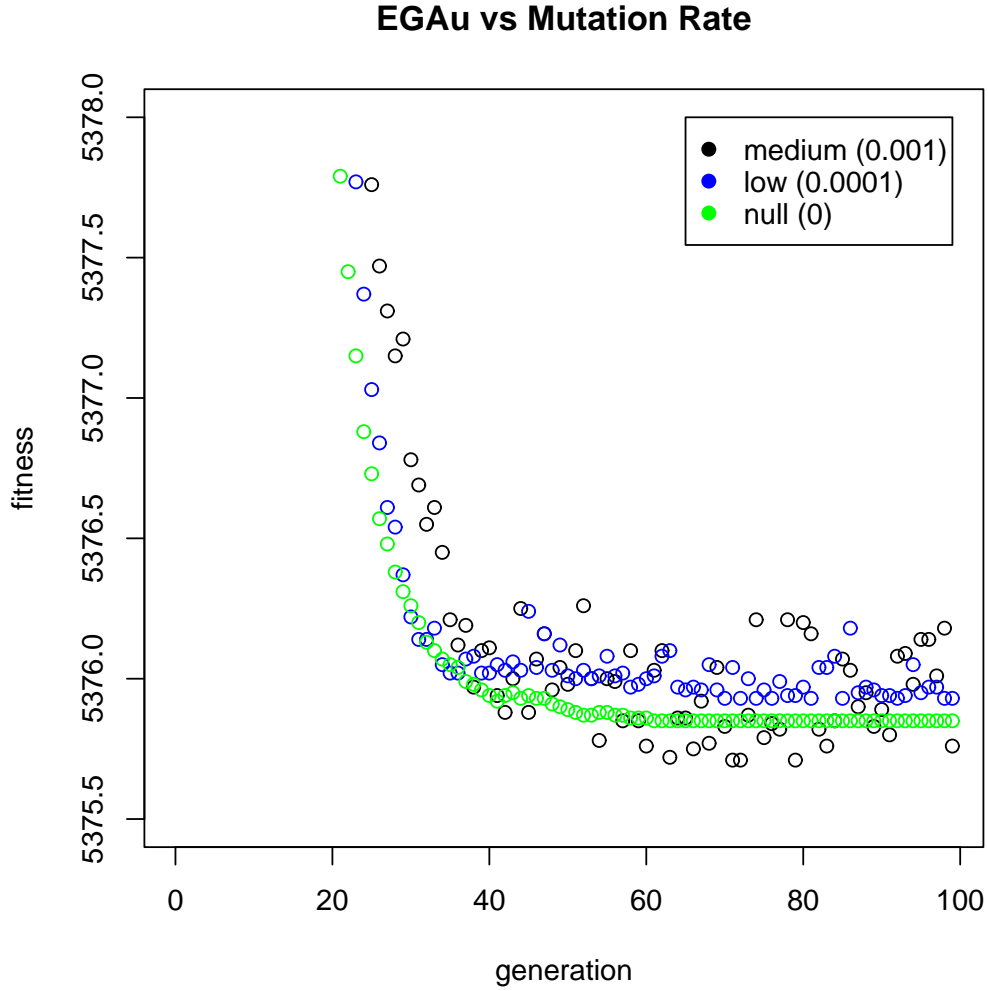


Figure 35: Zoom on EGAu versus mutation rate = 0

What does Figure 35 tell us? Firstly, we can see that the algorithm without mutation (the green line) has a strong and fast convergence on its “final value”, which then maintains steadily. Then, we see that the low-mutation algorithm (the blue line) is always worse than the no-mutation one. The medium-mutation algorithm, instead, does sometimes better than the no-mutation one; this happens, though, because its mean fitness values are quite fluctuating (due to mutations); so, in this case, the no-mutation algorithm offers us more guaranties than the others, in terms of convergence

and of stability.

3.9 - Solution's analysis

We have thoroughly explored almost all the possible facets of GAs in variable selection using AIC criterion: we have studied four different algorithmic variants, we have changed the population size, the crossover rate and the mutation rate. These analysis took us to reckon:

- the Elitist GA with uniform crossover as the best algorithmic variant;
- 70 as the best population size;
- 1.00 as the best crossover rate;
- 0.00 as the best mutation rate.

With these combination of parameters we can now perform an execution (a run) of the EGAu algorithm and store the final population: we will finally see which is the best solution (that is, the best model) to our original problem of variable selection. Since we need the results of only one run (we cannot do the mean of statistical models!), we will perform different runs and take the one with the best “final value”.

The best “final value” that we have seen in all our analysis is 5375.36; so we will take the final population of a run that terminates with that value.

The final population that we obtained with this criterion was formed by only one type of chromosome, which is:

000000111100110000100000100

This chromosome encodes the presence (in the regression model) of the following 8 variables:

- homeruns
- runs batted in
- walks
- strike outs
- freeagent
- arbitration
- walks / strike outs
- stolen bases * on base percentage

corresponding to the statistical model

$$Y_i = \beta_0 + \beta_7 x_{i7} + \beta_8 x_{i8} + \beta_9 x_{i9} + \beta_{10} x_{i10} + \beta_{13} x_{i13} + \beta_{14} x_{i14} + \beta_{19} x_{i19} + \beta_{25} x_{i25} + \epsilon_i$$

where ϵ_i ($i = 1, \dots, 337$) are independent, identically distributed $N(0, \sigma^2)$ random variables.

Now we can study in more detail this regression model via R:

```

> fit = lm( y ~ x7 + x8 + x9 + x10 + x13 + x14 + x19 + x25 )
> summary( fit )

Call:
lm(formula = y ~ x7 + x8 + x9 + x10 + x13 + x14 + x19 + x25)

Residuals:
    Min       1Q   Median       3Q      Max
-2035.0  -460.6    41.5    357.0   2944.3

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   117.733     134.449   0.876  0.38185
x7             27.302       9.378   2.911  0.00385 **
x8             17.691       3.167   5.587 4.87e-08 ***
x9             10.287       3.844   2.676  0.00782 **
x10            -14.197       2.582  -5.498 7.72e-08 ***
x13           1294.005      94.040  13.760 < 2e-16 ***
x14            823.201     110.444   7.454 8.15e-13 ***
x19           -393.221     173.936  -2.261  0.02443 *
x25             47.392      10.399   4.557 7.33e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 692.5 on 328 degrees of freedom
Multiple R-Squared:  0.6956,    Adjusted R-squared:  0.6882
F-statistic: 93.69 on 8 and 328 DF,  p-value: < 2.2e-16

```

Figure 36: Summary of linear regression model

This model's summary shows us that all the present coefficients in the regression model are statistically significant, except for the intercept; in fact their p-values (in the tests where $H_0 : \beta_i = 0$ vs $H_1 : \beta_i \neq 0$) are all inferior to the threshold of 0.05. This means that each variable (taken individually) has a corresponding coefficient statistically different from 0. The test where $H_0 : \beta_7 = \beta_8 = \beta_9 = \dots = \beta_{25} = 0$ also has a p-value highly inferior to 0.05, indicating strong evidence against null hypothesis (H_0). The adjusted R-squared of this model is 0.6882, not so bad.

To check the homoscedasticity of the standardized residuals we can produce the plot shown in Figure 37:

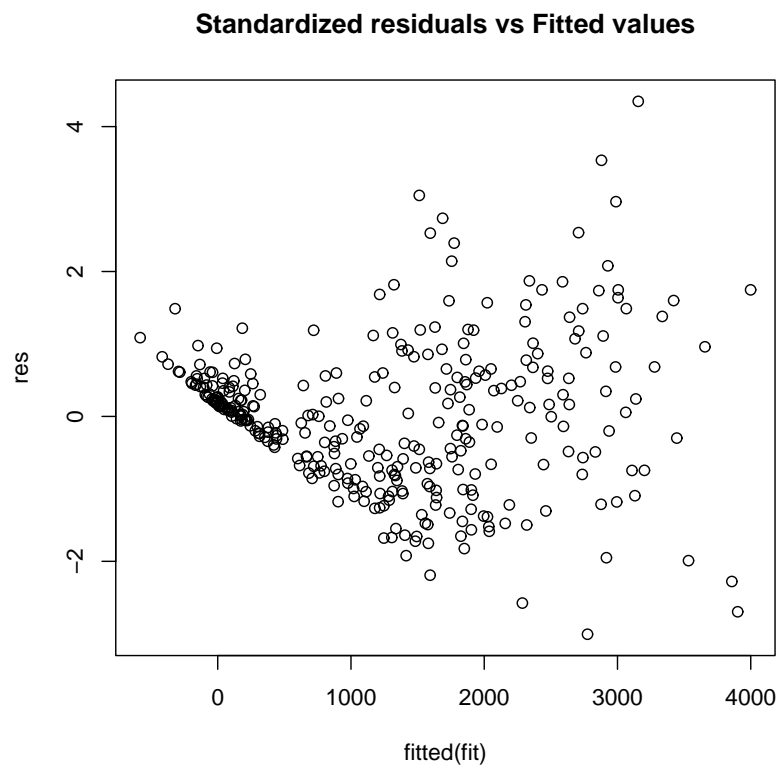


Figure 37: Standardized residuals vs Fitted values

Figure 37 shows a quite bigger variability for high fitted values: this implies that the residual's variance is not constant, that is the residuals are not homoschedastics. As regard to the residuals normality, we can perform some graphical checks.

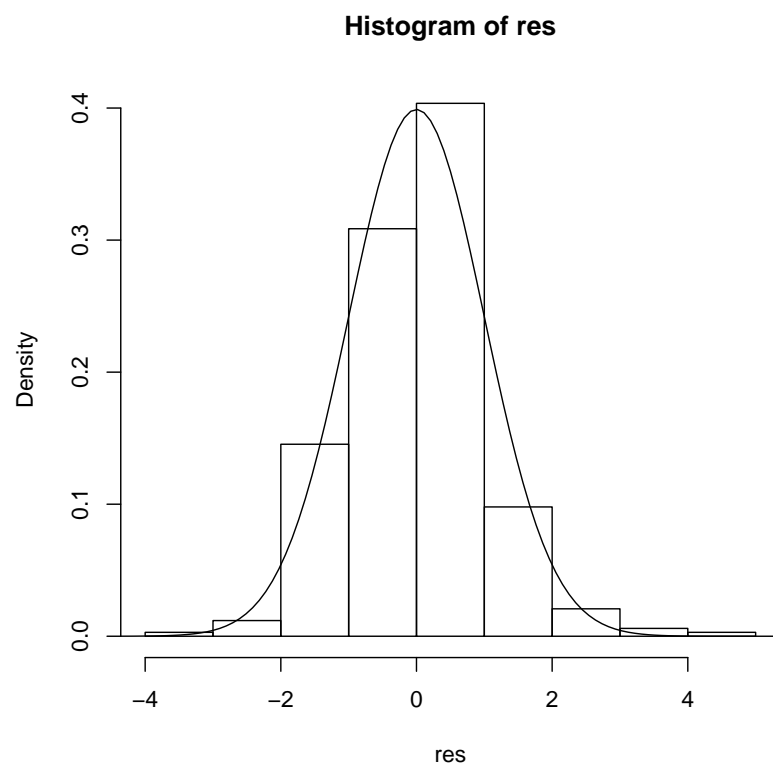


Figure 38: Histogram of standardized residuals with normal density

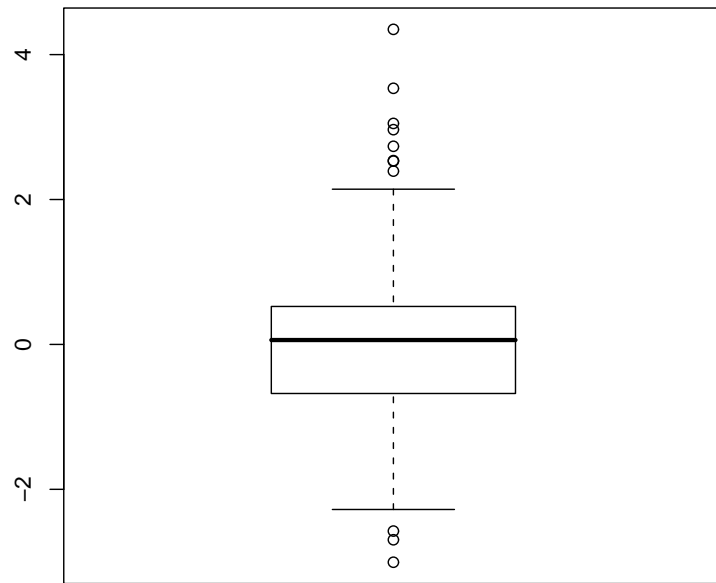


Figure 39: Boxplot of standardized residuals

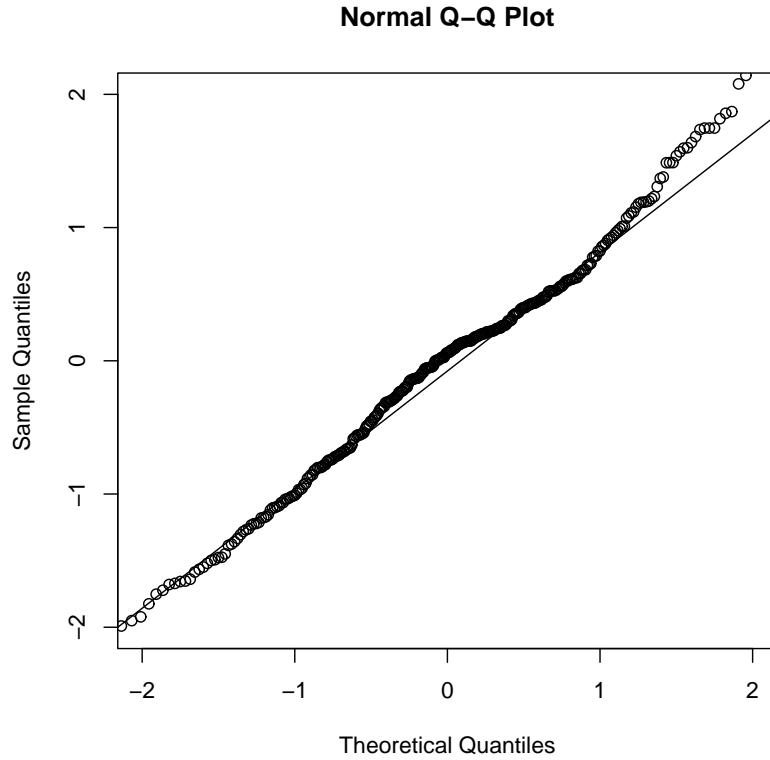


Figure 40: Normal QQ-plot

Figure 38 shows that the standardized residuals' estimated density is not quite coherent with that of the normal density. In Figure 39 we see that, although the mean is very close to zero, the boxplot has many outliers. The QQ-plot in Figure 40 is not so good, because of the marked drift in the right tail and in the center of the distribution compared to the expected behaviour.

On the whole, the standardized residuals' normality does not seem much satisfactory; moreover the residuals are clearly deficient as regard to their homoscedasticity. In conclusion, the solution we found to our original problem of variable selection is a model which do not fully satisfy the usual assumptions. To understand if the genetic algorithms failed or if the problem is the automation of the procedure, we can dig deeper (and we will do it in the next paragraph).

3.9.1 - Stepwise regression and GAs in comparison

We can continue our analysis studying the model obtained with stepwise regression in R. Let us see all the previous paragraph's figures with data from both models (the one obtained with GAs and the one obtained with stepwise regression in R):

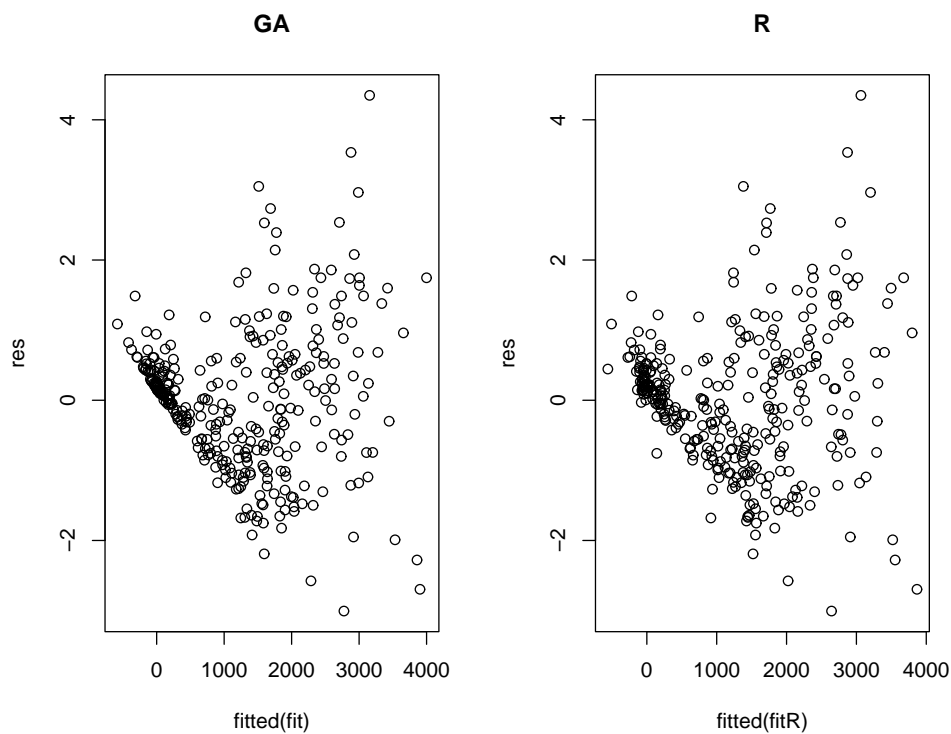


Figure 41: Standardized residuals vs Fitted values

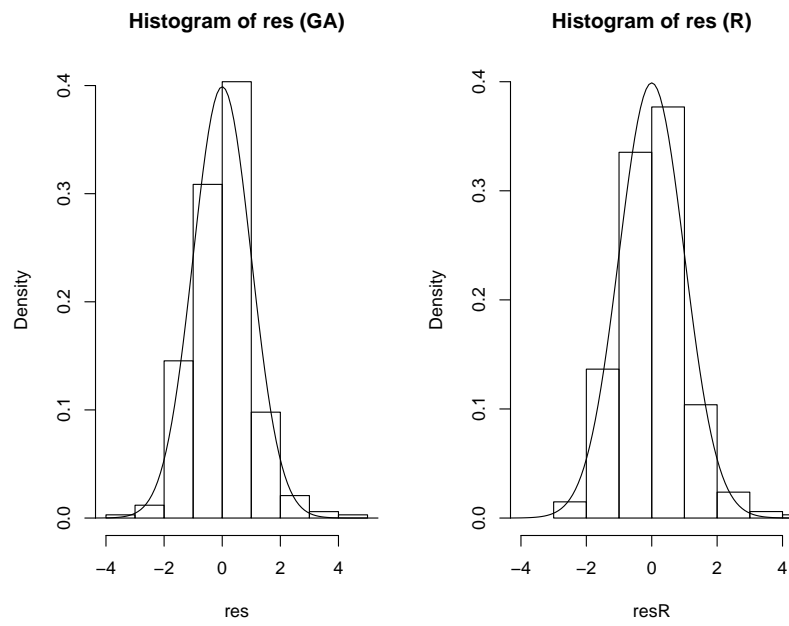


Figure 42: Histogram of standardized residuals with normal density

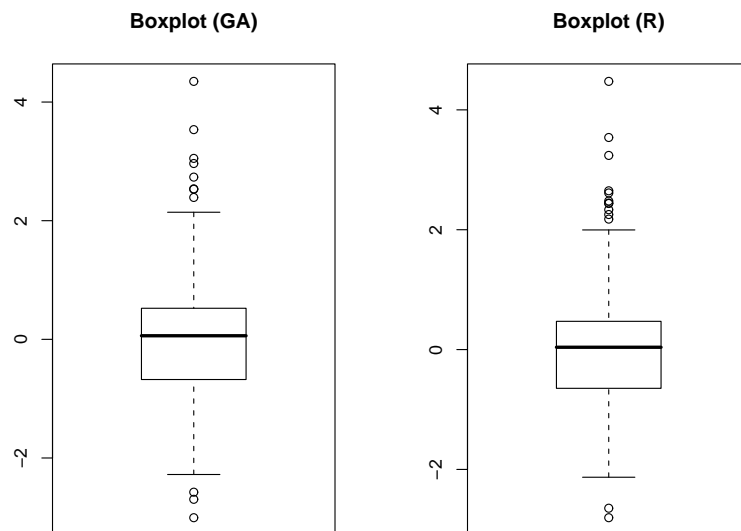


Figure 43: Boxplot of standardized residuals

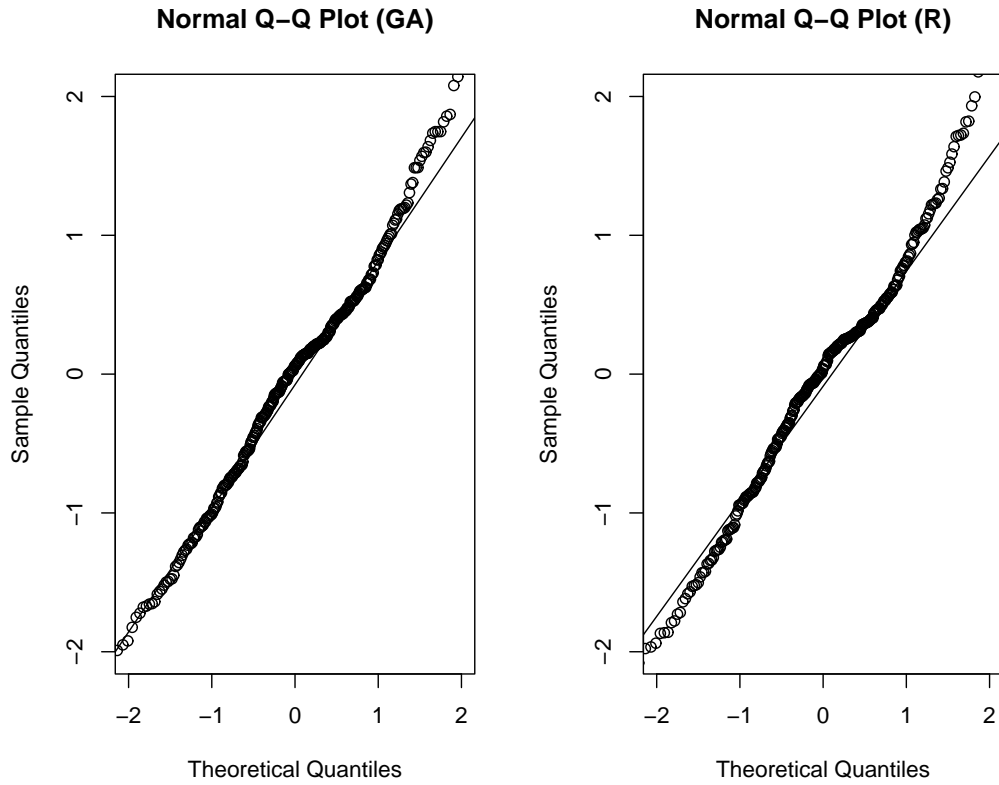


Figure 44: Normal QQ-plot

We can see in Figure 41 that the GA's standardized residuals and the R's one exhibit the same behaviour, that is, a marked heteroschedasticity. The histograms and the boxplots, too, show great resemblance. The difference lies in the Q-Q plot: while the GA's qqplot presents a drift only in the right tail, the R's qqplot presents drifts in both tail and deviates a little in the middle of the graph too. In the end, let us see also the summary of this regression model, shown in Figure 45.

```

> summary( fitR )

Call:
lm(formula = y ~ x8 + x13 + x14 + x26 + x10 + x7 + x18 + x24 +
    x3)

Residuals:
    Min       1Q   Median       3Q      Max
-1904.13  -443.35    26.77   324.10  3035.58

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   26.84531   131.14043    0.205  0.83793
x8             19.51799     4.60477    4.239 2.93e-05 ***
x13           1276.97086    93.83128   13.609 < 2e-16 ***
x14            814.64451   111.52486    7.305 2.14e-12 ***
x26              0.15612     0.05341    2.923  0.00371 **
x10            -10.80154     2.69939   -4.001 7.79e-05 ***
x7              24.51855     9.51986    2.576  0.01045 *
x18            -208.48832   130.16519   -1.602  0.11018
x24             -0.13431     0.08765   -1.532  0.12640
x3              5.04389     3.29855    1.529  0.12720
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 694 on 327 degrees of freedom
Multiple R-Squared: 0.6951,    Adjusted R-squared: 0.6867
F-statistic: 82.84 on 9 and 327 DF,  p-value: < 2.2e-16

```

Figure 45: Summary of linear regression model

The model's summary shows us that not all the regression coefficients are statistically significant. In fact, three variables (plus the intercept) have a p-value higher than the threshold of 0.05. The adjusted R-squared of this model is 0.6867, quite close to the GA's one.

What can we say from this comparison? That neither the linear model found by stepwise regression was good; in fact it was almost worse than the GA's one, as seen in the Q-Q plot.

This happens because we are using (in both genetic algorithms and stepwise regression) automatic procedures, which do not check at each step if the regression model that they are examining has good statistical properties (like residuals' homoscedasticity and

normality of the residuals' distribution). So we have to take their results as important suggestions (we could not possibly examine all the subsets from a set of 27 variables!), but we have to do it very carefully; we have to check with caution every result that is presented to us.

Now, we could wonder: if we had used another algorithmic variant (instead of “EGAu”) and/or another set of parameters, how would the model have been? In fact, at the end of the 100 generations the difference of fitness amongst the variants that we tried was not so large (except for the ones with population = 10 or with great mutation rates). Maybe one of those variants produces a model with a little bit lower AIC value but with better statistical properties. We can try one variant at random and see what happens. The choices for this experiment are:

- Algorithmic variant: SGA (simple genetic algorithm);
- Population size: 40;
- Crossover rate: 0.25;
- Mutation rate: 0.001.

With this parameters' set, we execute one run made of 100 generations. The resulting final population of this execution was composed by these chromosomes:

Chromosome	Absolute frequency	Relative frequency	AIC value
001000110110110001001001100	2	0.05	5377.129
001000110110110001000001100	1	0.025	5376.048
000000110110110001001001100	1	0.025	5377.986
000000110110110001000001100	35	0.875	5376.128
0000001101101100000000001100	1	0.025	5376.346

Table 13: Final population obtained with SGA

We have to pick only one regression model from this population, in order to analyze it and compare it to the others (the one obtained from stepwise regression and the one from EGAu). We should choose the best model, which is the one with lower AIC (5376.048). We want to determine if the standardized residuals are heteroschedastics and if they are normal. To do this we will have to look at the previous graphs, but with data coming from the model that we just chose.

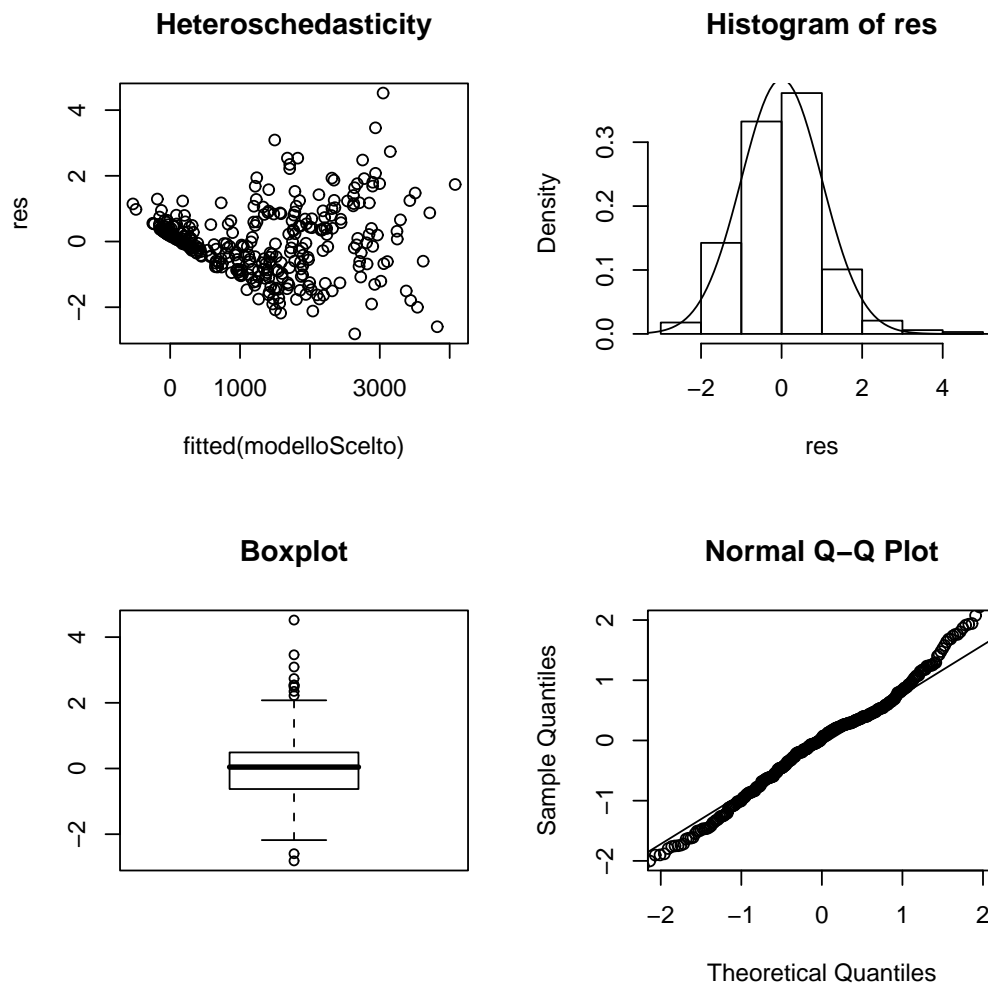


Figure 46: Analysis of regression model obtained from SGA

From Figure 46, we see nothing different in comparison to what we already analysed. The residuals are clearly heteroschedastics and their distribution is not probably normal (because of the drift in QQ-plot).

```

> summary( modelloScelto )

Call:
lm(formula = y ~ x3 + x7 + x8 + x10 + x11 + x13 + x14 + x18 +
    x24 + x25)

Residuals:
    Min       1Q   Median       3Q      Max
-1901.57  -423.75    29.62   333.48  3052.91

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.33281   129.56884   0.003  0.997952
x3              4.59144    3.23251   1.420  0.156448
x7             23.23237    9.65233   2.407  0.016643 *
x8             19.46905    4.58122   4.250  2.80e-05 ***
x10            -9.86730    2.76273  -3.572  0.000408 ***
x11            -55.46312   31.33263  -1.770  0.077639 .
x13            1253.82024   94.06855  13.329 < 2e-16 ***
x14             812.24005  111.06806   7.313  2.04e-12 ***
x18            -193.33581  129.87501  -1.489  0.137552
x24             -0.12353    0.08735  -1.414  0.158287
x25             195.05550   89.23296   2.186  0.029533 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 691.2 on 326 degrees of freedom
Multiple R-Squared:  0.6986,    Adjusted R-squared:  0.6893
F-statistic: 75.55 on 10 and 326 DF,  p-value: < 2.2e-16

```

Figure 47: Summary of regression model obtained from SGA

From this summary, we see that there are 4 variables (plus the intercept) which are not statistically significant (their p-values are higher than 0.05). The adjusted R-squared is 0.6893, a little higher than the previous ones (which were 0.6882 and 0.6867).

In conclusion, the three models that we analysed are quite similar: for sure none of them fully satisfy the statistical assumptions. The differences are:

- the AIC values;
- the adjusted R-squared values;
- the variables' set and the variables' significance.

3.9.2 - Variables' subsets comparison

Since the problem that we are trying to solve in this chapter is **variable selection**, we should examine in detail the variables' subsets chosen by the models that we took into account. We can give a first glance to this issue with the help of the following table. In the table we have marked with an "X" the variables included by each model. In this way, it is easy to see which variables are included in all the three models (those variables are highlighted in yellow) and which are not. The bold variables are those not significant in the model to which they belong.

Variable number	Stepwise model (A)	EGAu model (B)	SGA model (C)
1			
2			
3	X		X
4			
5			
6			
7	X	X	X
8	X	X	X
9		X	
10	X	X	X
11			X
12			
13	X	X	X
14	X	X	X
15			
16			
17			
18	X		X
19		X	
20			
21			
22			
23			
24	X		X
25		X	X
26	X		
27			

Table 14: Variables subsets' comparison

First of all, how many variables are included by each model? The model from stepwise regression has 9 variables, the one from EGAu has 8 variables, and the one from SGA has 10 variables.

Then, Table 14 shows us that there are 5 variables which are present in all of the three models:

- x7 (homeruns)
- x8 (runs batted in)
- x10 (strike outs)
- x13 (eligible for free agency)
- x14 (eligible for arbitration)

Very likely, this happens because those five variables are very explicative and are necessary to model the relationship between baseball players' statistics and their salary.

As regards the other variables (the ones which are not present in all models), what can we say? Many of them (a half, to be more precise) are present in two of the three models, which could mean that their role is important, but not fundamental. Here are some considerations about those variables and their potential correlation with other variables.

- The variable x3 is present in models A and C; in B, instead, there is x9 (not present in A and C) which is highly correlated to x3 (correlation value = 0.82); therefore we can consider that all the three models have a component which is represented by x3 in models A and C and by x9 in model B.
- A similar remark can be made about x11 and x25/x26: in fact, x11 is present only in model C; though, this variable is very strongly correlated to both x25 and x26 (the correlation values are, respectively, 0.99 and 0.95). Model A includes x26, while model B includes x25; so, also in this case, we can say that all the

models have in common a component which is represented by x26 in A, by x25 in B and by x11 in C.

- The model C seems to contain a redundancy, because it includes both x11 and x25 (which correlation value is, as we said earlier, 0.99).
- The variable x18 is present in models A and C but it does not have a “substitute” in model B; in fact x18 is highly correlated only to x16, a variable which cannot be found in any of our models.
- The same goes for x19: it is included only in model B and there are not variables greatly correlated to it.
- The last variable to discuss is x24, which is in model A and C but not in B; this variable is highly correlated only to x12 (0.83), but x12 is not in model B (nor in A, nor in C).
- All the variables which can be found in models A and C but not in B (except for x26) are not statistically significant in their regression model. Maybe this is the reason for model B being the one with lower AIC: model B is the only one with variables that are all significant.

These considerations reinforce the thought that these models are very alike; in fact about a half of their variables are in common (x7, x8, x10, x13, x14), while a lot of the other variables of each model are strongly correlated with a variable of the other two model.

This resemblance amongst the models could explain why their AIC values were not markedly different: in fact there exists a rule of thumb which says that two models which AIC values differs no more than 2 are to be considered equally good.

Chapter 4 - Maximization of skew-normal's likelihood

In this chapter, we are going to see a likelihood maximization problem.

Maximum likelihood estimation (MLE) is a popular statistical method used to make inferences about parameters of the underlying probability distribution from a given data set.

Commonly, one assumes the data are independent, identically distributed (i.i.d.) drawn from a particular distribution with unknown parameters and uses the MLE technique to create estimators for the unknown parameters.

The method was pioneered by geneticist and statistician Sir R. A. Fisher between 1912 and 1922.

4.1 - The skew-normal distribution

4.1.1 - A very brief history

The skew-normal (SN, hereafter) distribution is an extension of the normal (Gaussian) probability distribution, allowing for the presence of skewness. This distribution, originally introduced by Roberts (1966), was named and formally defined by Azzalini (1985, 1986). After that, it has been generalized to the multivariate case by Azzalini and Dalla Valle (1996), and Azzalini and Capitanio (1999), who also explored its statistical properties.

4.1.2 - Formulation of the univariate form

The random variable X is said to have a scalar $SN(\Lambda)$ distribution if its density function is:

$$\phi_{\Lambda}(x) = \frac{2}{\lambda_2} \phi\left(\frac{x - \lambda_1}{\lambda_2}\right) \Phi\left(\lambda \frac{x - \lambda_1}{\lambda_2}\right)$$

where:

- $\Lambda = (\lambda, \lambda_1, \lambda_2)^T$;
- $-\infty < \lambda, \lambda_1 < \infty, 0 < \lambda_2 < \infty$;
- ϕ denotes the standard Normal (Gaussian) density function, while Φ denotes its distribution function;
- the λ parameter is called the *shape* parameter, while λ_1 and λ_2 are respectively called *location* and *scale* parameters;
- the shape parameter regulates the distribution's skewness, which is positive when $\lambda > 0$ and negative when $\lambda < 0$: in fact, if the sign of λ changes, the density is reflected on the opposite side of the vertical axis; as λ increases (in absolute value), the skewness of the distribution increases.

Note that:

- when $\lambda = 0$ the skewness vanishes, and we obtain the standard Normal density;
- when $\lambda \rightarrow \infty$ the density converges to the so-called *half-normal* (or folded normal) density function.

The skew-normal distribution is often useful to fit observed data with “normal-like” shape of the empirical distribution, but with lack of symmetry. Some applications are: estimation of stochastic frontiers, non-random sampling problems, censoring on normal variates, graphical modelling, Bayesian analysis, portfolio selection of financial assets and detection of skewness in stock returns.

4.1.3 - Examples of the univariate form

We can look at graphs of the SN density function, with some parameters' variations, to better comprehend their role.

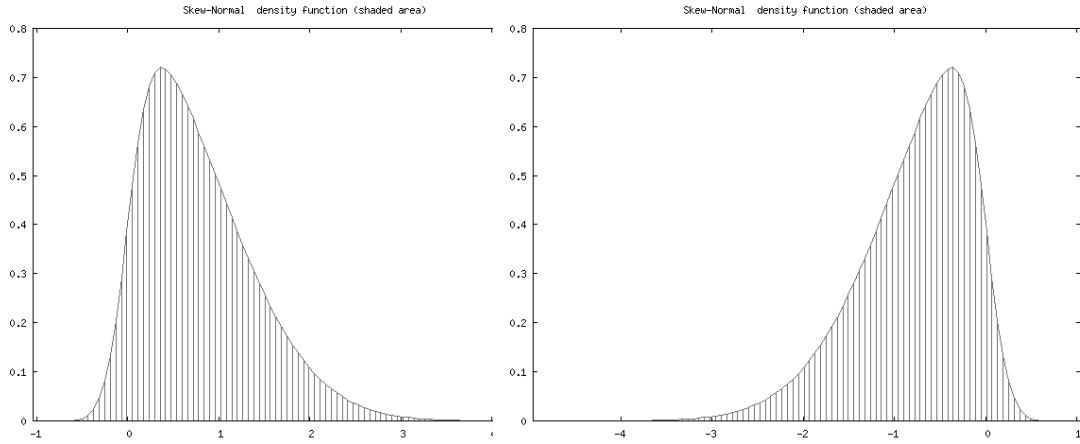


Figure 48: Density function of $SN(\Lambda)$ where $\Lambda = (5, 0, 1)^T$ Figure 49: Density function of $SN(\Lambda)$ where $\Lambda = (-5, 0, 1)^T$

From Figure 48 and 49 we can see what happens when changing the sign of λ : the density is reflected on the opposite side of the vertical axis.

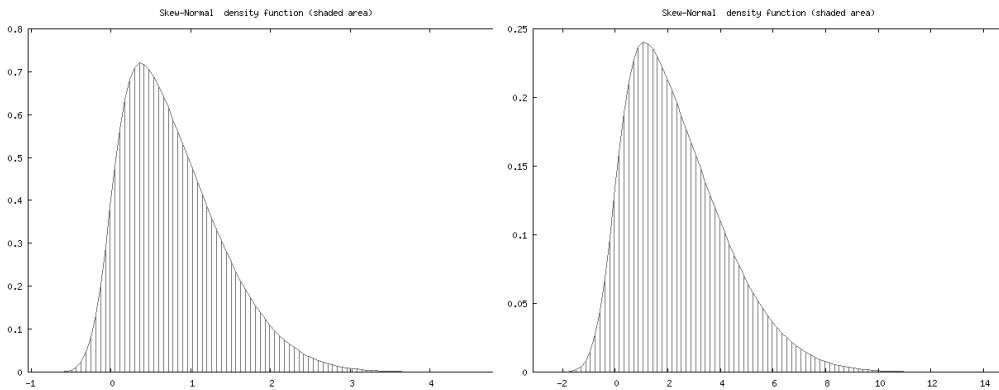


Figure 50: Density function of $SN(\Lambda)$ where $\Lambda = (5, 0, 1)^T$ Figure 51: Density function of $SN(\Lambda)$ where $\Lambda = (5, 0, 3)^T$

From Figure 50 and 51 we can see what happens when increasing the value of λ_2 : the density is spread on a larger area (it is less concentrated).

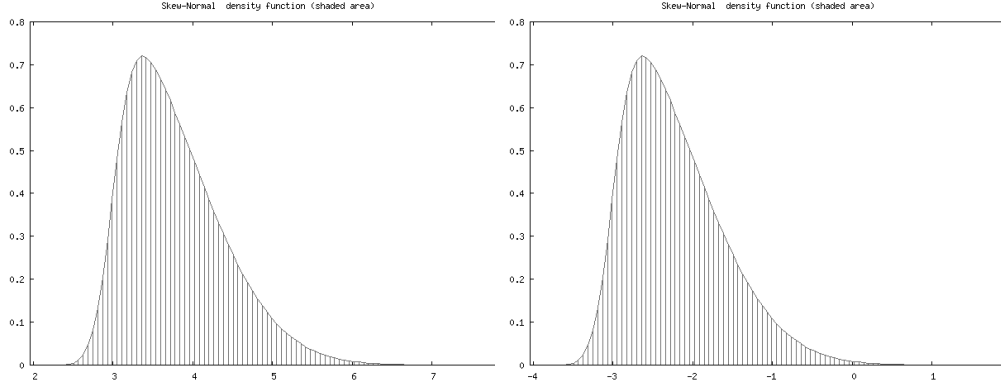


Figure 52: Density function of $SN(\Lambda)$ where $\Lambda = (5, 3, 1)^T$ Figure 53: Density function of $SN(\Lambda)$ where $\Lambda = (5, -3, 1)^T$

From Figure 52 and 53 we can see what happens when changing the value of λ_1 : the density is traslated towards the new location value.

4.1.4 - Likelihood and Log-Likelihood functions

Since our purpose is to maximize the likelihood function of the SN distribution, it is time to start looking at it. The likelihood function, L^Λ , is defined as (remember we are always talking about the univariate case):

$$\begin{aligned} L^\Lambda &= L^\Lambda(x_1, \dots, x_n; \Lambda) \\ &= 2^n \lambda_2^{-n} \prod_{i=1}^n \phi\left(\frac{x_i - \lambda_1}{\lambda_2}\right) \Phi\left(\lambda \frac{x_i - \lambda_1}{\lambda_2}\right) \end{aligned}$$

Since the above expression is not suitable for our purposes, we will work on the natural logarithm of the likelihood function, that is, the so-called **log-likelihood** function. The derivation of that function is showed here:

$$\begin{aligned}
l^\Lambda &= \log L^\Lambda(x_1, \dots, x_n; \Lambda) \\
&= \log(2^n) + \log(\lambda_2^{-n}) + \sum_{i=1}^n \log(\phi(\frac{x_i - \lambda_1}{\lambda_2}) \Phi(\lambda \frac{x_i - \lambda_1}{\lambda_2})) \\
&= n \log(2) - n \log(\lambda_2) + \sum_{i=1}^n \log(\phi(\frac{x_i - \lambda_1}{\lambda_2})) + \sum_{i=1}^n \log(\Phi(\lambda \frac{x_i - \lambda_1}{\lambda_2})) \\
&= n \log(2) - n \log(\lambda_2) + \sum_{i=1}^n \log(\frac{e^{-\frac{(\frac{x_i - \lambda_1}{\lambda_2})^2}}}{\sqrt{2\pi}}) + \sum_{i=1}^n \log(\Phi(\lambda \frac{x_i - \lambda_1}{\lambda_2})) \\
&= n \log(2) - n \log(\lambda_2) + \sum_{i=1}^n -\frac{(\frac{x_i - \lambda_1}{\lambda_2})^2}{2} - \sum_{i=1}^n \log \sqrt{2\pi} + \sum_{i=1}^n \log(\Phi(\lambda \frac{x_i - \lambda_1}{\lambda_2})) \\
&= n \log(2) - n \log(\lambda_2) - n \log \sqrt{2\pi} - \frac{1}{2} \sum_{i=1}^n (\frac{x_i - \lambda_1}{\lambda_2})^2 + \sum_{i=1}^n \log(\Phi(\lambda \frac{x_i - \lambda_1}{\lambda_2}))
\end{aligned}$$

We can exclude from the last expression the elements which do not depend on parameters. In fact, we are trying to maximize the function with respect to Λ , so the summed elements not depending on any of the three parameters (λ , λ_1 , λ_2) can be removed. In particular, we can take out $n \log(2)$ and $-n \log \sqrt{2\pi}$. Thus we obtain:

$$\begin{aligned}
l^\Lambda &= \log L^\Lambda(x_1, \dots, x_n; \Lambda) \\
&= -n \log(\lambda_2) - \frac{1}{2} \sum_{i=1}^n (\frac{x_i - \lambda_1}{\lambda_2})^2 + \sum_{i=1}^n \log(\Phi(\lambda \frac{x_i - \lambda_1}{\lambda_2})) \\
&= -n \log(\lambda_2) - \frac{1}{2} \sum_{i=1}^n (z_i)^2 + \sum_{i=1}^n \log(\Phi(\lambda z_i))
\end{aligned}$$

$$\text{where } z_i = \frac{x_i - \lambda_1}{\lambda_2}.$$

4.1.5 - Problems of the likelihood function

Since 1985 (year of the Azzalini's pioneering paper) it has been known that the estimation of the SN distribution's parameters is far from easy. In particular, there are two big problems:

- the profile likelihood function for λ has an inflection point at $\lambda = 0$, independently of the observed sample; correspondingly, at $\lambda = 0$ the expected Fisher information becomes singular.
- the likelihood function itself can be problematic: its shape can be far from quadratic, even when λ is not near 0.

The singularity of the information matrix can be resolved with a different parametrization: see Azzalini (1985), Pewsey (2001) and Chiogna (2005). However, the MLE (maximum likelihood estimator) evaluation and the likelihood shape remain largely problematic; not even the method of moments gave satisfactory results.

4.2 - An interesting dataset

We will take into consideration the challenging dataset available at:

<http://azzalini.stat.unipd.it/SN/frontier.dat>.

This dataset contains $n = 50$ simulated data-points from a $SN(5, 0, 1)$. It could seem an innocuous dataset, but it is not: the maximum likelihood estimate of λ is infinite. No other frequentist method seems to work on this dataset; the Bayesian approach conducted by Lisero and Coperfido in 2002 brought to an estimate for λ of about 2.1 (quite far from the true one, 5). As Azzalini and Capitanio state about the divergence of $\hat{\lambda}$ (1999): “...*The source of this sort of anomaly is easy to understand in the one-parameter case with λ_1 and λ_2 known; $\lambda_1 = 0$ and $\lambda_2 = 1$, say. If all sample values have the same sign, the final term of log-likelihood increases with $\pm\lambda$, depending the sign of the data but irrespective of their actual value.*” “... *When all three parameters are being estimated, the explanation of this fact is not so clear, but it is conceivable that a similar mechanism is in action.*”.

What we are asking ourselves is: how will GAs act in this very difficult context? Could they give us some help solving this problem? Could they at least clue us in when a dataset presents these anomalies? We are going to answer to these questions.

4.3 - Problem's definition

As we already said before, the genetic algorithm depends on only two things, which are the chromosomes' encoding (that is the relationships between chromosomes and candidate solutions) and the fitness function.

4.3.1 - Chromosomes' encoding

First of all, we have to distinguish between two different cases:

1. maximizing the log-likelihood function with respect to all three parameters (λ , λ_1 , λ_2);
2. maximizing the log-likelihood function with respect to only one parameter (λ), while keeping the other two at their true value ($\lambda_1 = 0$, $\lambda_2 = 1$).

The two cases will require different encoding; however they will be based on the same notion, which we already saw in Chapter 2 (while maximizing the simple univariate function).

The notion at issue is: each considered parameter will be represented by a certain number of bits. The i -th bit of the bit string will contribute to the parameter's provisional value with an addend of either 0 or 2^{-i} depending on its value (respectively 0 or 1). Thus, the bit string will encode values in the range $[0, 1)$. From this range we can obtain a value in any other desired range; suppose that the range we want the parameter to belong to is $(rangeMin, rangeMax)$. Then it is sufficient to apply this formula (the parameter's provisional value is represented by x , where $x \in [0, 1)$):

$$\text{parameter's final value} = x * (rangeMax - rangeMin) + rangeMin$$

Let us see an example of this encoding. Suppose that we want $\lambda \in (-10, +10)$ and that we represent λ with a string made of 7 bits. A possible bit string is then:

1011101

This bit string takes us to the provisional value of:

$$1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4} + 1 * 2^{-5} + 0 * 2^{-6} + 1 * 2^{-7} = 0.7265625$$

Then we apply the formula we saw before:

$$0.7265625 * (10 - (-10)) + (-10) = 0.7265625 * 20 - 10 = 4.53125$$

So, the bit string encoded the parameter value of 4.53125.

What is the precision of this kind of representation? We can compute it with a simple formula:

$$p = 2^{-k} * w$$

where:

- p = precision of the representation = distance between two contiguous values;
- k = number of bits used in the encoding;
- w = range's wideness = $rangeMax - rangeMin$.

In the example we just saw, the precision is:

$$p = 2^{-7} * [10 - (-10)] = 2^{-7} * 20 = 0.15625 \cong 0.16$$

In the case of only one parameter to optimize, the chromosome is simply the bit string which encodes its value. In the case of multiple parameters to optimize (three in our case), the chromosome is the assembly of the three bit strings in a single string. For example, if we use 7 bits for λ , 5 for λ_1 and 5 for λ_2 , we will have chromosomes made of 17 ($7 + 5 + 5$) bits.

4.3.2 - Fitness function

Once we have decoded a chromosome into the parameters' values that it represents, calculating the fitness value is very simple. In fact, the fitness function is the log-likelihood

function (in particular we will use the final formulation obtained in paragraph 4.1.4). We only have to insert the parameters's values (encoded by the chromosome at issue) into that formulation. The result of this computation will be the fitness value of that chromosome.

If we are in the case of optimizing one parameter, the procedure is very similar; the only difference is that the fitness formulation includes the fixed values of the other two parameters and that the chromosome represents only one parameter's value.

N.B.: When the parameters assume certain values, the fitness function goes to $-\infty$. If that happens, the fitness value is set to -10000.

4.4 - Results

We have defined all that we need to execute the algorithm, except for the decisions of:

- which algorithmic variant to use;
- which parameters' values to use.

The algorithmic variant we will make use of is the *elitist GA with uniform crossover* ("EGAu"). In fact, in Chapter 3, EGAu has been proven to be the best of all the variants we tried.

As regards to the parameters' values, they will be the following:

- number of generations = 200;
- population size = 100;
- crossover rate = 0.75;
- mutation rate = 0.001.

In this problem we will not make changes to the parameters' values or to the algorithmic variant. In fact, the problem at issue is very difficult and it provides enough interesting cues.

4.4.1 - The “one parameter” case (λ , 0, 1)

We shall begin with maximizing the log-likelihood function with only one parameter unknown (λ). As for λ_1 and λ_2 , we will use their “true” values (respectively, 0 and 1).

Now, we have to choose a range for the values of λ (remember our encoding process). Since we should not have any knowledge of the λ value, we have to consider a range with both positive and negative values. In particular, we can begin with the range $(-10, +10)$.

The number of bits with which we represent λ is 10. Therefore the distance between two contiguous values is $0.02 \cong 0.01953125$ ($2^{-10} * 20$, where 20 is the range wideness).

All is set: we have only to execute the algorithm. In particular, we will execute 10 runs, just to be sure to avoid taking wrong conclusions because of the randomness role.

The results are shown in the following table. The first column contains the index of the run. The second column contains the value of the highest fitness reached during that run. The third column contains the value of λ that produced that value of fitness during that run (so we can call it “the best λ ”).

Run	Best fitness	Best λ
1	-39.8135	6.54297
2	-39.8135	6.54297
3	-39.8135	6.54297
4	-39.8135	6.54297
5	-39.8135	6.5625
6	-39.8135	6.54297
7	-39.8135	6.54297
8	-39.8135	6.54297
9	-39.8135	6.54297
10	-39.8135	6.54297

Table 15: Results of the 10 runs

We can make some considerations about the results shown in Table 15:

- the runs were all successful: none of them gave strange results and all of them reached the exact same value of fitness (that is, of log-likelihood);
- the “best λ s” are all the same, except for the fifth one, which is only “a bit away” from it. In fact their distance is $6.5625 - 6.54297 = 0.01953 \cong 0.02$, that is the minimum possible distance (in our codification) between two values. However, they reach the same value of fitness, so they can be considered equivalent.
- the value that the GA found for λ is 6.54. This result is good. In fact, the true value was 5 and we are not so far from it.

Moreover, to see if the GA found the global maximum in the fitness function, we can look at a plot of the latter.

N.B.: We can do this only in this case, that is, when one parameter is unknown.

Obviously, in the next paragraph, when we shall talk about the “three-parameter case”, we will not be able to do the same.

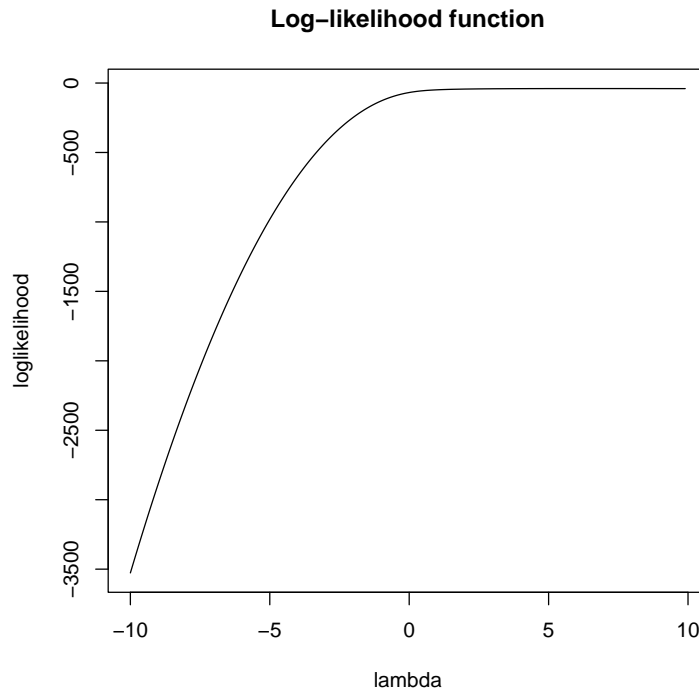


Figure 54: Log-likelihood function of $SN(\Lambda)$ where $\Lambda = (\lambda, 0, 1)^T$

Figure 54 seems to tell us that the values of λ in range $(0, 10)$ take to equivalent values of log-likelihood. The plot, though, is inaccurate, since its y -axis starts from -3500 and we know that the “interesting” values are around -39. Thus, we can make another plot, zoomed on that area.

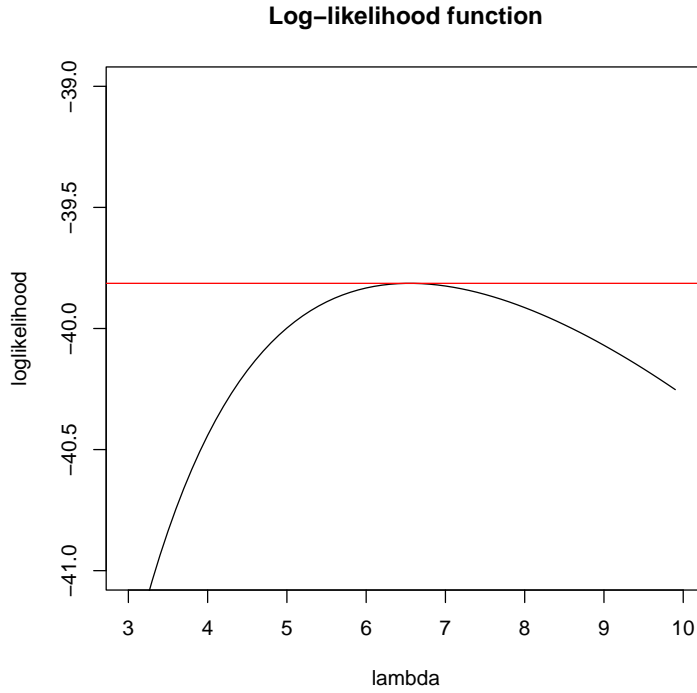


Figure 55: Zoom on log-likelihood function of $SN(\Lambda)$ where $\Lambda = (\lambda, 0, 1)^T$

In Figure 55 we see more clearly that the function has effectively one global maximum located in the proximity of 6.5. The red line represents the best fitness value found by the GA (-39.8135). This shows us the precision of the GA in finding the maximum. We have to remember, though, that this was a simple problem for the GA, since there was only one parameter to optimize.

Now, we can proceed in two directions before passing on to the “three parameters case”:

- see what happens changing the precision of the representation (that is, increasing/decreasing the number of bits of the chromosomes);
- see what happens changing the range in which to optimize the parameter λ .

4.4.1.1 - Changing precision

It could be interesting to see both what happens increasing and decreasing the precision of our chromosomes representation. We shall begin increasing the precision. Theoretically, this could improve our solution or it could have no effect at all; anyway, it should not worsen the results.

Let the chromosomes be made of 20 bits. The distance between two contiguous values is then $0.00002 \cong 0.000019073486328125$ ($2^{-20} * 20$).

The results are shown in the following table (we reduced the number of runs to 5).

Run	Best fitness	Best λ
1	-39.8135	6.54392
2	-39.8135	6.54667
3	-39.8135	6.54758
4	-39.8135	6.54646
5	-39.8135	6.55230

Table 16: Results of the 5 runs with increased precision

Table 16 shows us that the results are not changed. In fact, the λ values are slightly different from the ones of Table 15, but, despite this, the corresponding fitness values are always the same.

Let us see now what happens by decreasing precision. In particular, we will use chromosomes 5-bits long, thus reaching a distance between contiguous values of 0.625 ($2^{-5} * 20$). The results are shown in the following table (the number of runs is 5).

Run	Best fitness	Best λ
1	-39.819	6.25
2	-39.819	6.25
3	-39.819	6.25
4	-39.819	6.25
5	-39.819	6.25

Table 17: Results of the 5 runs with decreased precision

Table 17 shows us that decreasing like that the precision of the candidate solutions takes to slightly worse result. In fact, the fitness value has fallen from -39.8135 to -39.819, and the best λ found is 6.25. In this case the difference is not so relevant, but we will have to be more careful when optimizing three parameters at once.

4.4.1.2 - Changing range

Now we will see what happens on changing the range in which to optimize the parameter.

Firstly, we can reduce the wideness of the range, leaving out of it the optimal value found in the previous tests. In particular, we shall use the range $(-5, +5)$ and chromosomes 10-bits long. Thus the distance between contiguous values is $0.01 \cong 0.009765625$ ($2^{-10} * 10$).

The result are shown in the following table.

Run	Best fitness	Best λ
1	-40.0007	4.99023
2	-40.0007	4.99023
3	-40.0007	4.99023
4	-40.0007	4.99023
5	-40.0007	4.99023

Table 18: Results of the 5 runs with decreased range's wideness

Table 18 shows us that the GA finds, as the optimal value, 4.99. That value is the closest, in the employed range, to the previous optimal value (6.54). We could have been expecting a similar behaviour, since the log-likelihood function increases until it reaches the proximity of 6.5 and then starts decreasing. The GA has, as always, found the maximum in the data which we gave to it.

Let us now increase the range. We will make use of the range $(-40, +40)$ and of chromosomes 20-bits long. The distance between two contiguous values is, in this case, $0.00008 \cong 0.0000762939453125 (2^{-20} * 80)$.

Run	Best fitness	Best λ
1	-39.8135	6.54549
2	-39.8135	6.54335
3	-39.8135	6.54549
4	-39.8135	6.55365
5	-39.8135	6.54953

Table 19: Results of the 5 runs with increased range's wideness

Table 19 shows that the GA is not confused by the wideness of the range. It goes straight to the optimal value for the parameter, which is, we repeat it, in the proximity

of 6.54.

Now that we have explored those two directions, we can pass on to the “three parameters” case.

4.4.2 - The “three parameters” case $(\lambda, \lambda_1, \lambda_2)$

For “three parameters” case we mean the maximization of the log-likelihood with respect to all three parameters at issue $(\lambda, \lambda_1, \lambda_2)$. The interactions amongst the values of the three parameters make the problem exponentially more difficult.

To be more clear, hereafter we will make use of a table to show the precision of each parameter and the range in which each parameter is considered. A prototype of such table is the following.

Parameter	Range min	Range max	Bits	Distance
λ	min	max	$bits$	$2^{-bits} * (max - min)$
λ_1	min_1	max_1	$bits_1$	$2^{-bits_1} * (max_1 - min_1)$
λ_2	min_2	max_2	$bits_2$	$2^{-bits_2} * (max_2 - min_2)$

Table 20: Schema of the parameters’ table

Table 20 shows us how we will represent the data concerning the three parameters. For each experiment that we shall make with the algorithm, we will propose a similar table containing the actual values used during that execution of the GA.

We can start making use of such a table right now, for the first test.

Parameter	Range min	Range max	Bits	Distance
λ	-10	+10	11	0,009765625
λ_1	-5	+5	10	0,009765625
λ_2	+0.01	+5	9	0,009765625

Table 21: Parameters' table

N.B.: The range-min for λ_2 is 0.01 instead of 0 to avoid computing $\log(0)$.

We shall execute 10 runs of the GA. The results are shown in the following table.

Run	Best fitness	Best λ	Best λ_1	Best λ_2
1	-38.6061	7.58789	0	1.16004
2	-38.0434	9.99023	-0.0878906	1.2575
3	-38.0267	9.36523	-0.0585938	1.18928
4	-38.0356	9.99023	-0.078125	1.2575
5	-38.0356	9.99023	-0.078125	1.2575
6	-38.6061	7.58789	0	1.16004
7	-38.0369	9.98047	-0.078125	1.2575
8	-38.9200	8.26172	0	1.2575
9	-38.0356	9.99023	-0.078125	1.2575
10	-38.0434	9.99023	-0.0878906	1.2575

Table 22: Results of the 10 runs

Here comes the problem that we anticipated in paragraph 4.1.5. We can see from Table 22 that the highest value of fitness found in the 10 runs is approximately -38.03. This value, besides being quite higher than the one found in the “one parameter” case (-39.8135), corresponds to a strange triplet of parameters' values. In fact, the λ value associated to the higher fitnesses is very close to the upper bound of the considered

range. These values go from 9.36523 to 9.99023.

These results urge a question: what would happen if we moved the upper bound of the range? We shall answer right now. Let us consider the following parameters' table.

Parameter	Range min	Range max	Bits	Distance
λ	0	20	11	0,009765625
λ_1	-5	+5	10	0,009765625
λ_2	+0.01	+5	9	0,009765625

Table 23: Parameters' table

The only difference between Table 23 and 21 is the range in which to optimize λ . The results obtained by the algorithm are showed in the following table (we made 5 runs).

Run	Best fitness	Best λ	Best λ_1	Best λ_2
1	-37.3354	19.9902	-0.107422	1.2575
2	-37.3490	19.9902	-0.0976563	1.2575
3	-37.3354	19.9902	-0.107422	1.2575
4	-37.3358	19.9805	-0.107422	1.2575
5	-37.3369	19.9512	-0.107422	1.2575

Table 24: Results of the 5 runs

It happened what we were expecting: this time the optimal λ value found is 19.99, which means on the edge of the range. The values of λ_1 and λ_2 are, respectively, -0.1 and 1.26, not far from the true ones. The problem is that the λ value is going *very* far from the true one: we can move the range indefinitely (towards $+\infty$) and the optimal value would always be on the upper bound of the range.

The algorithm, though, does what it is supposed to do: it maximizes the log-likelihood value, which effectively increase while increasing the value of λ . While in the “one parameter case” the maximum log-likelihood value was -39.8135, now we have a value of about -37.3, which is better (though not substantially).

Now, it could be interesting to see intermediate cases, which means setting the value of one parameter to its true value and maximizing the log-likelihood function with respect to the other two parameters. We could call this the “two parameters case”. In particular we could consider two situations:

1. λ and λ_1 unknown, λ_2 fixed to its true value;
2. λ and λ_2 unknown, λ_1 fixed to its true value;

We shall not consider the situation with λ fixed to its true value, because that is the most interesting parameter and the one which diverges to $+\infty$.

4.4.3 - The “two parameters” case $(\lambda, \lambda_1, 1)$

We shall begin seeing the usual parameters’ table. In this case, though, λ_2 is excluded from the table, since we fixed it to its true value (that is, 1).

Parameter	Range min	Range max	Bits	Distance
λ	-10	+10	11	0,009765625
λ_1	-5	+5	10	0,009765625

Table 25: Parameters’ table

The results of the GA’s execution are shown in the following table.

Run	Best fitness	Best λ	Best λ_1
1	-39.8135	6.54297	0
2	-39.7731	9.99023	-0.0488281
3	-39.8135	6.54297	0
4	-39.8135	6.5625	0
5	-39.7731	9.99023	-0.0488281
6	-39.8135	6.54297	0
7	-39.7757	9.46289	-0.0390625
8	-39.7739	9.95117	-0.0488281
9	-39.7759	9.55078	-0.0390625
10	-39.7855	8.16406	-0.0292969

Table 26: Results of the 10 runs

From Table 26 we can see an evident pattern and we can make some considerations.

- When the optimal value found for λ_1 is 0, then the optimal value found for λ is about 6.5 (like in the “one parameter case”). We have another confirm that with $\lambda_1 = 0$ and $\lambda_2 = 1$ the log-likelihood function exhibits an appropriate behaviour.
- When the optimal value of λ_1 is less than 0 (about -0.049), then the optimal value found for λ goes to 9.99 (which is the upper bound of the range).
- The fitness function assumes a higher value when $\lambda \cong 9.99$ and $\lambda_1 \cong -0.05$ than when $\lambda \cong 6.5$ and $\lambda_1 = 0$. This means that the best solution found by the algorithm, throughout all the runs, has the λ value at the upper bound of its range.

To better comprehend why the algorithm has found multiple solutions to this problem, we can look at a plot. In this plot we will draw the log-likelihood function for three values of λ_1 , while λ_2 will be fixed to 1 and λ will vary in the range (5, 10).

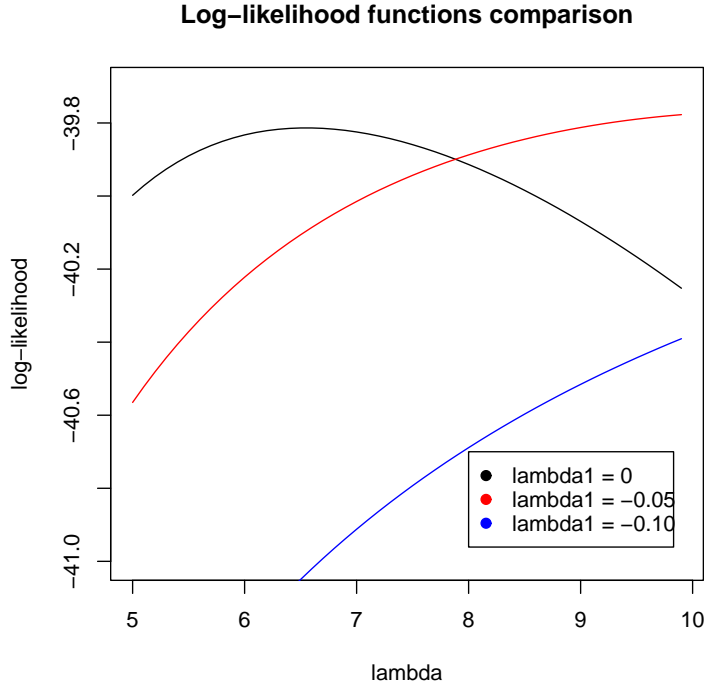


Figure 56: Log-likelihood function of $SN(\Lambda)$ where $\Lambda = (\lambda, \lambda_1, 1)^T$

In Figure 56 we can see that, when $\lambda \in (-10, +10)$, the curves with $\lambda_1 = 0$ and $\lambda_1 = -0.05$ have their maximums located at almost the same value (about -39.8). This is why in some runs the GA finds the solution $(6.5, 0)$ and in other runs it finds the solution $(9.99, -0.05)$.

Now we can move forward the range of λ to see what happens. The parameters' table for the following execution of the algorithm is shown in Table 27.

Parameter	Range min	Range max	Bits	Distance
λ	0	+20	11	0,009765625
λ_1	-5	+5	10	0,009765625

Table 27: Parameters' table

The results of the GA's execution are shown in the following table.

Run	Best fitness	Best λ	Best λ_1
1	-39.7563	19.9902	-0.0878906
2	-39.7599	16.4648	-0.0781250
3	-39.7562	13.9844	-0.0683594
4	-39.7589	12.1777	-0.0585938
5	-39.7589	12.1875	-0.0585938
6	-39.7562	13.9844	-0.0683594
7	-39.7563	19.9902	-0.0878906
8	-39.7562	13.9746	-0.0683594
9	-39.7563	19.9902	-0.0878906
10	-39.7562	13.9844	-0.0683594

Table 28: Results of the 10 runs

As Table 28 shows, in this case, the GA has found a different solution in almost every run. Despite that, the fitness values are very similar: they are all in the proximity of -39.76.

Why does this happen? Because log-likelihood functions based on different λ_1 values reach their maximum with a different λ value. Moreover, in this particular range, some of these maxima reach almost the same “height”, as proved by the fitness values. In other words, in this situation there are more global maxima, so the algorithm finds each time a different result. To assess these considerations, we can plot the fitness function for some of the optimal λ_1 values found, with $\lambda \in (10, 20)$.

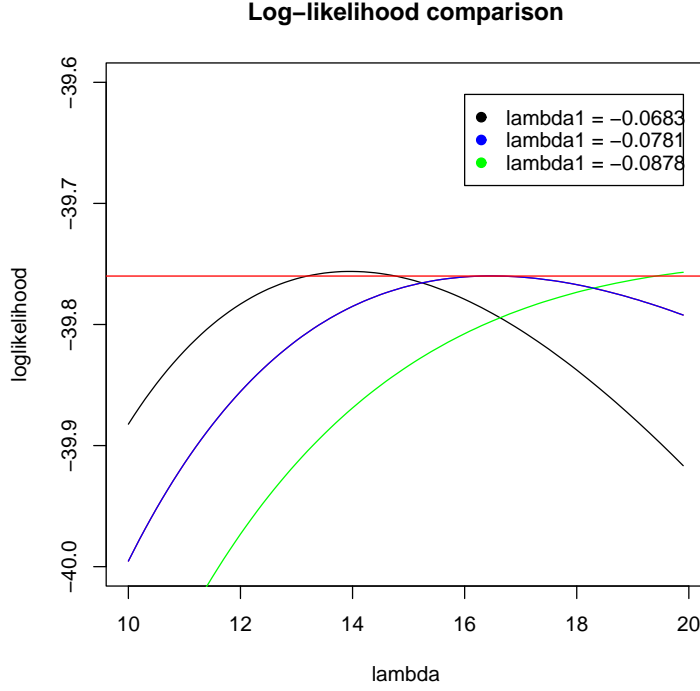


Figure 57: Log-likelihood function of $SN(\Lambda)$ where $\Lambda = (\lambda, \lambda_1, 1)^T$

As we anticipated, Figure 57 shows that the three loglikelihood functions plotted reach almost the same value, with very different λ values (respectively about 13.98, 16.46 and 19.99).

In conclusion, it *could* seem that fixing λ_2 eliminates the problem of λ 's divergence to $+\infty$. To be absolutely sure about this, let us do another test. In this test the wideness of λ 's range will be severely increased, as shown in the following parameters' table.

Parameter	Range min	Range max	Bits	Distance
λ	0	+80	13	0,009765625
λ_1	-5	+5	10	0,009765625

Table 29: Parameters' table

The results of this test are shown in the following table (only 5 runs were performed).

Run	Best fitness	Best λ	Best λ_1
1	-39.135	79.9902	-0.107422
1	-39.135	79.9902	-0.107422
1	-39.135	79.9902	-0.107422
1	-39.135	79.9902	-0.107422
1	-39.135	79.9902	-0.107422

Table 30: Results of the 5 runs

From Table 30 it is clear that fixing λ_2 's value does not prevent the divergence of λ , it only delays its appearance.

We can pass on to the last case that we have chosen to examine.

4.4.4 - The “two parameters” case $(\lambda, 0, \lambda_2)$

Again, we shall begin seeing the parameters' table. In this case λ_1 is excluded from the table, since we fix it to its true value (that is, 0).

Parameter	Range min	Range max	Bits	Distance
λ	-10	+10	11	0,009765625
λ_2	0.01	+5	9	0,009765625

Table 31: Parameters' table

The results of the 10 performed runs are shown in the following table.

Run	Best fitness	Best λ	Best λ_2
1	-38.6061	7.59766	1.16004
2	-38.6061	7.59766	1.16004
3	-38.6061	7.59766	1.16004
4	-38.6066	7.49023	1.16004
5	-38.6061	7.58789	1.16004
6	-38.6061	7.59766	1.16004
7	-38.6063	7.65625	1.16004
8	-38.6061	7.59766	1.16004
9	-38.6066	7.49023	1.16004
10	-38.6067	7.48047	1.16004

Table 32: Results of the 10 runs

Table 32 gives rise to some considerations.

- With the parameters' table shown in Table 31, the optimal value of λ_2 is clearly 1.16004. In fact, all the 10 runs found exactly the same value.
- The optimal values for λ range from 7.48 to 7.66 (values that are not too distant from the real λ value, that is, 5).

- The fitness values reached by the 10 runs are almost the same, all of them in proximity of -38.6 .

Does λ 's optimal value diverge to $+\infty$ also in this case? We shall answer right now to this question, doing a test with a quite wide range for λ , as shown in the following parameters' table.

Parameter	Range min	Range max	Bits	Distance
λ	0	+80	13	0,009765625
λ_2	0.01	+5	9	0,009765625

Table 33: Parameters' table

The results of this experiment are shown in the following table.

Run	Best fitness	Best λ	Best λ_2
1	-38.6233	8.12500	1.15029
2	-38.6066	7.49023	1.16004
3	-38.6073	7.76367	1.16004
4	-38.6061	7.58789	1.16004
5	-38.6091	7.55859	1.15029
6	-38.6061	7.59766	1.16004
7	-38.6066	7.49023	1.16004
8	-38.6066	7.49023	1.16004
9	-38.6061	7.58789	1.16004
10	-38.6061	7.58789	1.16004

Table 34: Results of the 10 runs

From Table 34 it is clearly visible that in this case (λ_1 fixed to 0) the optimal value for λ does not diverge to $+\infty$.

Thus we can say, with quite certainty, that fixing the λ_1 value to its true one (0) the MLE for λ and λ_2 is ($\cong 7.5, 1.16004$), with a loglikelihood value of -38.6061 .

4.4.5 - Summary

Let us briefly recap the results of all the tests that we made.

- $\Lambda = (\lambda, 0, 1)^T$: the MLE for λ found by the GA is 6.54.
- $\Lambda = (\lambda, \lambda_1, 1)^T$: the MLE for λ diverges to $+\infty$.
- $\Lambda = (\lambda, 0, \lambda_2)^T$: the MLEs for λ and λ_2 are, respectively, $\cong 7.5$ and 1.16004.
- $\Lambda = (\lambda, \lambda_1, \lambda_2)^T$: the MLE for λ diverges to $+\infty$.

4.5 - A method to acknowledge the problem

We have seen that in the “three parameters” case (that is, the most interesting one) there is no way to stop the divergence of λ ’s estimate. In fact, the log-likelihood continues to increase while increasing the value of λ . The GA (which should maximize the log-likelihood) does its job and, by doing so, it finds the λ ’s value which corresponds to the higher log-likelihood. That value, though, changes each time we move its range.

Thus, it seems that we cannot solve the problem at all. Is there something that we (with the help of a GA) *can* do? We could find a method to tell the user of this GA that there is a problem in the dataset. Said in other words: if we cannot solve the problem, we can, at least, warn that there *is* a problem.

How can we accomplish this task? With a “**sliding-ranges**” system. Let us explain how this method works:

1. we fix the initial ranges for λ , λ_1 , λ_2 in which the GA has to maximize the log-likelihood. In general, since we should not know the true values of the parameters, reasonable ranges are:

- (-10, +10) for λ ;
 - (-5, +5) for λ_1 ;
 - (0.01, +5) for λ_2 .
2. we store the centers of the three ranges in the variables c, c_1, c_2 (with the above choice of the ranges, these variables would, respectively, assume the values 0, 0, 2.495); we also store the wideness of the ranges (20, 10, 4.99);
 3. we execute the algorithm for r runs and we store the best values that it finds in each run (let us call them $\widehat{\lambda}^i, \widehat{\lambda}_1^i, \widehat{\lambda}_2^i$, where i is the index of the run);
 4. we compute the mean values: $\bar{\lambda} = \frac{1}{r} \sum_{i=1}^r \widehat{\lambda}^i, \bar{\lambda}_1 = \frac{1}{r} \sum_{i=1}^r \widehat{\lambda}_1^i, \bar{\lambda}_2 = \frac{1}{r} \sum_{i=1}^r \widehat{\lambda}_2^i$;
 5. we assign the new centers of the ranges: $c' = \bar{\lambda}, c'_1 = \bar{\lambda}_1, c'_2 = \bar{\lambda}_2$;
 6. if $|c' - c| < \frac{\text{widenessRange}}{m}$ and $|c'_1 - c_1| < \frac{\text{widenessRange}_1}{m}$ and $|c'_2 - c_2| < \frac{\text{widenessRange}_2}{m}$:
STOP. The parameters' estimates are steady.
 7. if the previous conditions have failed, then assign the new values for the ranges' centers ($c = c', c_1 = c'_1, c_2 = c'_2$); the ranges have been translated;
 8. goto 3.

This algorithm is repeated for no more than k times (that is, the ranges can be moved at most k times). The value for k is decided by the user of the GA, as well as the value of r .

The possible outcomes of such method are the following:

- the algorithm stops itself before completing k steps. This means that it has found the parameters' optimal values and that they are stable.
- the algorithm stops itself after the $k - th$ step. This means that the ranges continued to be moved and that the optimal values for the parameters have not been reached. This could happen for three reason:

1. the steps (k) were too few and the algorithm could not reach the parameters' optimal values. This situation can be easily fixed by increasing k and re-executing the algorithm.
2. there are not optimal values, probably because λ 's estimate diverges to $+\infty$. In this case, no fixing is possible.
3. the demanded precision was too much: we refer in particular to the value of m , which influences the termination condition. If we use a value too high for m , the algorithm could not stop itself, although it finds approximately the same values in each step. For simplicity, suppose we are optimizing only one parameter (λ , for example); if *widenessRange* = 20 and $m = 200$, then the termination condition is $|c' - c| < \frac{20}{200} = 0.1$. Then, if in one step the GA finds 6.5 as the optimal value for λ , and in the following step it finds 6.7, the GA will not stop, because $|c' - c| = |6.5 - 6.7| > 0.1$.

A similar situation could present itself if r is too low: then the randomness (which, remember, has a very important role in the GAs) could prevent the algorithm's termination.

4.5.1 - Sliding-ranges method applied to our problem

We have explained the method; now it is time to try it. Firstly, we will apply the method to our problem, which is represented by the *frontier* dataset. We know that maximizing the log-likelihood with respect to all three parameters takes to the divergence of λ 's value. Thus, our method should execute all the scheduled steps and terminate with the message "There is a problem in your dataset".

The parameters we are going to give to our method are:

- k = number of steps = 10;
- r = number of runs per step = 5;
- initial range for $\lambda = (-10, 10)$, that is, $c = 0$ and *widenessRange* = 20;

- initial range for $\lambda_1 = (-5, 5)$, that is, $c_1 = 0$ and $widenessRange_1 = 10$;
- initial range for $\lambda_2 = (0.01, 5.01)$, that is, $c_2 = 2.495$ and $widenessRange_2 = 5$;
- $m = 100$.

The parameters of the GA (number of generations, population size, crossover rate, mutation rate) are always the same.

The results of this experiment are shown in the following table. For simplicity, we will report here only the mean values reached in each step (that is, the mean of the best values reached in each run of the considered step).

Step	Mean λ	Mean λ_1	Mean λ_2	λ range	λ_1 range	λ_2 range
1	9.13867	-0.044922	1.21267	$(-10, 10)$	$(-5, 5)$	$(0.01, 5)$
2	19.9844	-0.095703	1.23656	$(0, 20)$	$(-5, 5)$	$(0.01, 5)$
3	28.9902	-0.119141	1.25219	$(9, 29)$	$(-5, 5)$	$(0.01, 5)$
4	37.9902	-0.117188	1.25219	$(18, 38)$	$(-5, 5)$	$(0.01, 5)$
5	46.9824	-0.119141	1.24633	$(27, 47)$	$(-5, 5)$	$(0.01, 5)$
6	55.9902	-0.119141	1.25805	$(36, 56)$	$(-5, 5)$	$(0.01, 5)$
7	64.9902	-0.119141	1.25219	$(45, 65)$	$(-5, 5)$	$(0.01, 5)$
8	73.9805	-0.126953	1.25609	$(54, 74)$	$(-5, 5)$	$(0.01, 5)$
9	82.9902	-0.119141	1.25609	$(63, 83)$	$(-5, 5)$	$(0.01, 5)$
10	91.0371	-0.136719	1.25414	$(72, 92)$	$(-5, 5)$	$(0.01, 5)$

Table 35: Results of 10 steps

At the end of the 10 steps, as we were expecting, the program gives the message “There is a problem in your dataset”. In fact, from Table 35 we clearly see that the λ value is constantly moving: its position is always on the right edge of its range.

Thus, with this problem, the sliding-ranges method does what it is supposed to do.

It warns the user of the GA that the considered dataset has a problem, since the λ 's value is not at all steady.

4.5.2 - Sliding-ranges method applied to another problem

It could be interesting to try the sliding-ranges method on another problem (that is, on values simulated from a skew-normal with another set of parameters). In particular, we can consider the following parameters' set:

- $\lambda = 22$;
- $\lambda_1 = 0$;
- $\lambda_2 = 1$.

Also, in order to have more precise results, we will consider a dataset made of 500 data points (instead of 50). This dataset can be found in Appendix B.

As regards the parameters of our method, we will use the same of the previous paragraph, except for r (which we fix to 10) and m (which we fix to 50).

The results of this experiment are shown in the following table.

Step	Mean λ	Mean λ_1	Mean λ_2	λ range	λ_1 range	λ_2 range
1	8.86230	0.1533200	1.044060	$(-10, 10)$	$(-5, 5)$	$(0.01, 5)$
2	17.3662	0.0214844	0.993399	$(-1, 19)$	$(-4, 6)$	$(0.01, 5)$
3	20.5488	0.0068359	0.990469	$(7, 27)$	$(-4, 6)$	$(0.01, 5)$
4	19.4717	0.0166016	0.973867	$(10, 30)$	$(-4, 6)$	$(0.01, 5)$
5	20.7373	0.0126953	1.004140	$(9, 29)$	$(-4, 6)$	$(0.01, 5)$
6	21.3145	0.0087891	0.986563	$(10, 30)$	$(-4, 6)$	$(0.01, 5)$
7	20.0537	0.0156250	0.974844	$(11, 31)$	$(-4, 6)$	$(0.01, 5)$
8	19.0566	0.0205078	0.979727	$(10, 30)$	$(-4, 6)$	$(0.01, 5)$
9	19.1846	0.0156250	0.972891	$(9, 29)$	$(-4, 6)$	$(0.01, 5)$
10	//	//	//	//	//	//

Table 36: Results of 10 steps

As we can see from Table 36, the algorithm stops at the 9-th steps and it gives the message “All the ranges are steady.”. Besides, from step 3 to step 9, the algorithm always finds λ ’s optimal values that belong to a narrow gap $(19, 21.3)$, so the center of the range does not move itself so much. If we had used a lower value for m (thus reducing the demanded precision), the algorithm would have stopped before reaching the 9-th step.

So, in this case, the GA does not warn about problems in the dataset. Moreover, it reaches quite good estimates for all the parameters: 19.2 instead of 22 (λ), 0.02 instead of 0 (λ_1), 0.97 instead of 1 (λ_2).

Conclusions

The purpose of this thesis was, like we said in the introduction, to apply genetic algorithms to problems typically encountered in statistics and to explore their performances. At the end of this work, we can draw some conclusions.

In Chapter 2, we saw a first application of genetic algorithms: the maximization of a univariate function. This problem was very easy to solve. In fact, SGA (the simplest of all GAs) found quite quickly an optimal solution.

We also tried changing the initial parameters of the algorithm to see if the performance would have been affected. The considerations arisen from those experiments are the following.

- The size of the chromosomes' population cannot be too small (like 20 or less individuals).
- The number of generations cannot be too small (at least 50 generations are required, even for a simple problem).
- The crossover rate did not have valuable effects, but this could have been consequence of the problem's simplicity.
- A mutation rate too high can have destructive effects. A mutation rate too low, instead, does not do particular harm, except for situations with a lot of local maxima (where the algorithm could get stuck). The problem presented in Chapter 2, though, did have only one maximum. Thus, a mutation rate very low gave

rise to even better results (an average fitness of 99.26 with $p_m = 0.001$ versus 98.5 with $p_m = 0.01$).

In Chapter 3, we focused on the problem of variable selection. The task of the GA was to choose a subset of (hopefully) meaningful variables from the original set, using the AIC to compare models.

We used four different algorithmic variants:

- simple GA = SGA,
- elitist GA with single point crossover = EGAsp,
- elitist GA with two-point crossover = EGAtp,
- elitist GA with uniform crossover = EGAu

and compared their performance. The comparisons were repeated with some possible parameters' sets. Thus, the analysis has been as thorough and complete as possible. The final purpose was to find out the best combination of parameters and algorithmic variant. By “best combination” we mean the one which gave rise to the best results. The outcomes were: EGAu, population size = 70, crossover rate = 1.00, mutation rate = 0.0.

We studied the regression model obtained with EGAu and those parameters. We also compared the model with the one achieved by R (with stepwise regression). The differences between the two models were not so marked. Both the models did not have good statistical properties (as concerns residuals' normality and homoscedasticity). However, the AIC reached by GAs was a little better than the one reached by R (5375.36 versus 5377.88).

The most important conclusion we can draw from Chapter 3 is that we have to be careful while using automatic procedures like GAs, stepwise regression and so on. Sometimes they can find very good solutions and they can take away the burden of

a lot of manual work. Nevertheless, they are *automatic* procedures and for this reason they have to be attended by a human being, which has to check what the algorithm cannot.

In the particular case of variable selection, one surely can use GAs or stepwise regression. However, when a model is found, he/she has the duty to study it and analyze it.

In Chapter 4, we tackled the likelihood maximization problem. We considered a particular distribution, the skew-normal, and tried to maximize its likelihood with a problematic dataset. In this Chapter we did not focus on the algorithmic variant, nor on the parameters' set. Instead, we focused completely on the problem, which was quite fascinating. In fact, maximizing the likelihood with respect to all three distribution's parameters $(\lambda, \lambda_1, \lambda_2)$ causes the λ 's MLE divergence to $+\infty$.

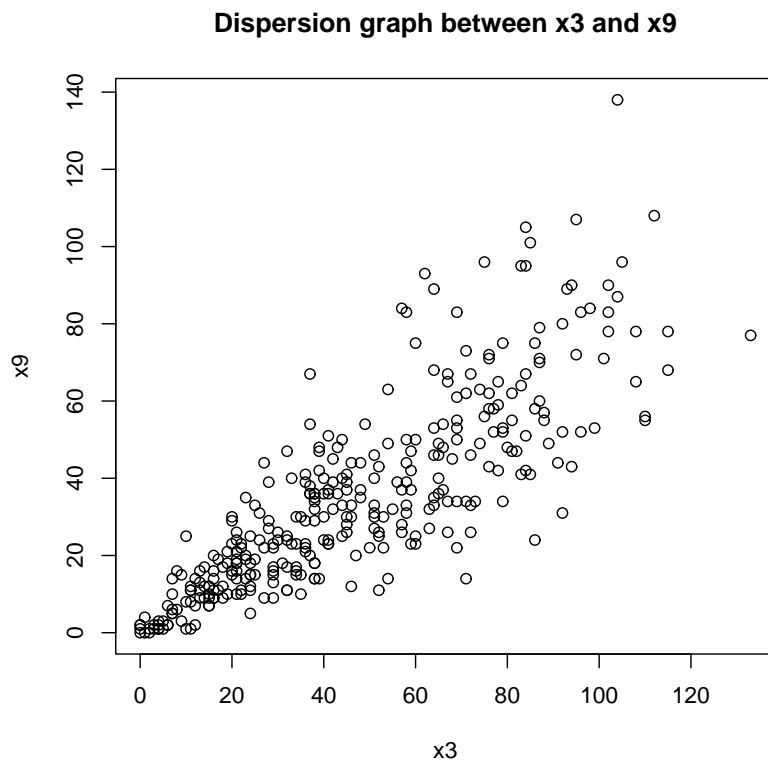
We initially studied some different situations (maximizing with respect to only one or two parameters out of three), to get acquainted with the problem. Then, we concentrated on the most troublesome case, which is the maximization of all three parameters. Since it seems that there is no way to solve a problem like this, we settled for a humbler goal: to advise the user of the GA that he/she has encountered a problematic dataset. In order to achieve this purpose we created a "sliding-ranges" system. This method moves dynamically the ranges into which optimize the parameters, until the centers of these ranges have become steady. If this steadiness is not reached after a certain number of iterations, then the method gives a message like "There is a problem in this dataset". Instead, if the steadiness is reached, the algorithm stops and gives a message like "The parameters' optimal values have stabilized themselves". We tried this method with two different datasets.

The first was the problematic dataset we just analyzed. In that case we knew that λ 's estimate diverges to $+\infty$, so the method should have given the message "There is a problem in this dataset". Actually, the program gave that message. Then, we tried with another dataset. In this case the method showed that the λ 's estimate does not

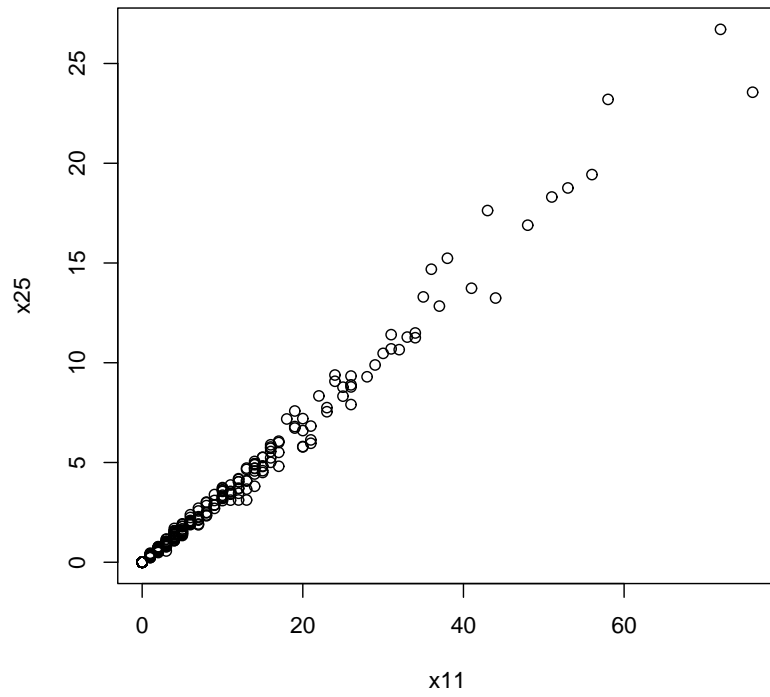
diverge to $+\infty$. In fact, after 9 steps, it stopped giving the message “The parameters’ optimal values are steady”. Also, the optimal values reached were not far from the true ones. This method seems to work.

Appendix A

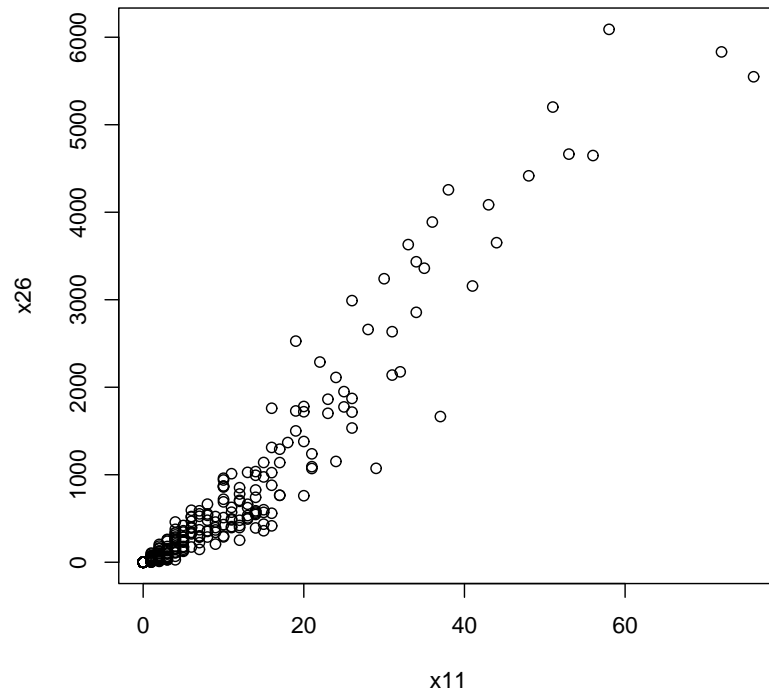
Dispersion graphs



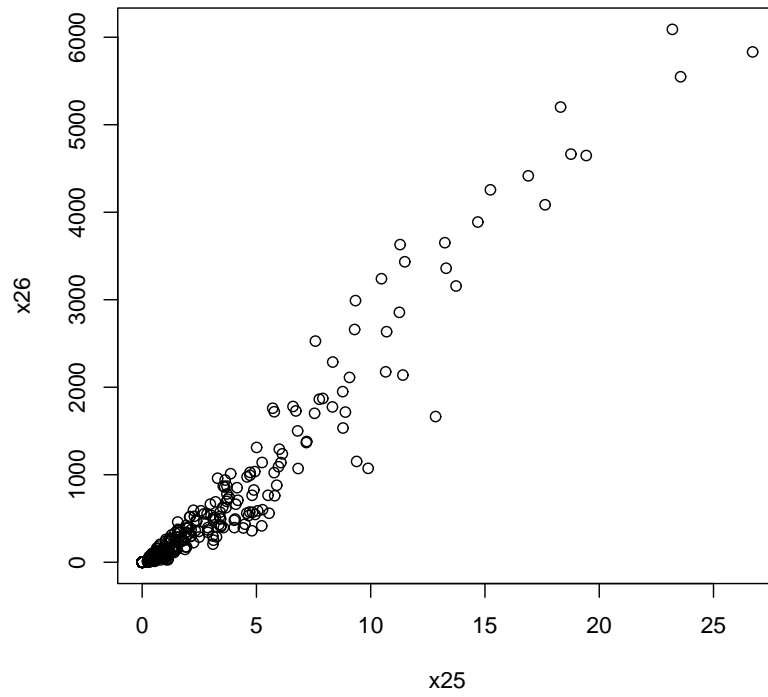
Dispersion graph between x11 and x25



Dispersion graph between x11 and x26



Dispersion graph between x25 and x26



Appendix B

Dataset used in paragraph 4.5.2

0.05827, 0.5244, 0.2155, 0.3604, 1.586, 2.108, 1.511, 0.4639, 0.1606, 0.1307, 0.4315, 2.221, 0.3525, 0.8684, 0.719, 0.8396, 0.7573, 0.7928, 0.5228, 1.678, 0.144, 0.5611, 0.1216, 0.4485, 0.1029, 1.304, 0.967, 1.360, 0.6285, 1.625, 1.513, 2.345, 1.223, 0.1653, 0.06251, 0.3519, 0.2824, 0.4093, 0.2007, 2.145, 0.6152, 0.6245, 0.1514, 0.1606, 0.5325, 0.08263, 0.2445, 0.724, 1.125, 0.07981, 0.5328, 0.2928, 0.2816, 0.8759, 0.4034, 0.7858, 0.1188, 0.9293, 1.365, 0.005259, 1.340, 0.2567, 1.032, 0.009084, 0.1544, 1.499, 0.3424, 1.209, 0.3839, 0.9549, 0.9144, 0.1299, 1.248, 2.327, 1.048, 0.2911, 0.8706, 0.1909, 0.681, 1.930, 0.1474, 3.836, 0.2509, 0.4249, 0.538, 0.5575, 0.6365, 0.4921, 0.07098, 0.7984, 0.62, 0.9887, 0.9548, 0.0375, 1.148, 0.5608, 0.5514, 0.3421, 0.9861, 0.9601, 1.475, 1.551, 0.1052, 1.635, 1.070, 0.154, 1.181, 0.1257, 0.5014, 0.1375, 1.084, 0.1634, 1.802, 1.513, 2.251, 0.333, 0.484, 0.6137, 0.4207, 1.000, 1.122, 0.3697, 0.74, 0.9435, 1.520, 1.504, 0.5774, 1.280, 0.5519, 0.1180, 1.432, 0.6461, 0.8965, 0.8662, 1.154, 1.094, 0.1122, 1.071, 0.7185, 1.350, 0.7194, 1.008, 0.7466, 0.4252, 0.05436, 0.7845, 0.1927, 0.1652, 0.4345, 1.504, 0.1499, 0.2246, 0.8957, 0.3008, 0.5776, 1.773, 1.497, 0.7973, 0.4909, 0.1304, 0.2159, 0.3037, 0.8352, 1.009, 0.4018, 0.7048, 0.7127, 1.249, 1.631, 0.4638, 0.08656, 0.1824, 1.039, 1.097, 0.4205, 0.2713, 2.122, 1.276, 0.3132, 1.425, 0.4822, 0.1031, 0.2103, 0.3249, 0.9486, 0.5953, 0.6322, 0.6958, 0.05114, 0.4762, 1.243, 0.939, 1.177, 0.09517, 0.7097, 2.398, 1.835, 0.8783, 1.92, 1.076, 1.112, 1.162, 0.3612, 0.3233, 0.3653, 1.254,

2.010, 0.2996, 0.4214, 0.4512, 0.5467, 0.4052, 0.5375, 0.3788, 0.553, 2.29, 1.675, 0.2032,
 0.2506, 1.231, 0.8458, 0.3064, 0.5359, 0.2278, 0.4817, 0.1592, 0.799, 1.701, 0.6112, -
 0.001033, 0.7979, 1.207, 0.2932, 0.9423, 1.373, 2.707, 0.5936, 0.9086, 0.6557, 1.691,
 0.4636, 1.553, 1.338, 0.1157, 0.8808, 0.989, 1.038, 0.1122, 1.214, 0.4610, 1.817, 0.8668,
 2.167, 0.7121, 0.8638, 0.7695, 0.1537, 0.992, 1.575, 0.1775, 0.3503, 0.6528, 0.5082,
 0.4526, 0.4838, 0.4447, 0.01374, 0.101, 0.8204, 0.5034, 0.1076, 0.9489, 1.701, 0.6437,
 0.5326, 0.9552, 0.421, 0.3275, 0.819, 0.4843, 2.337, 0.766, 0.2210, 0.508, 1.295, 0.4799,
 1.004, 0.8519, 0.3416, 2.009, 0.6061, 0.587, 0.00902, 1.457, 0.1104, 0.929, 0.3144, 0.6317,
 0.0724, 1.036, 1.136, 0.2372, 0.1962, 0.9185, 0.601, 2.413, 0.6851, 0.9493, 0.1103, 0.6715,
 0.3881, 0.7671, 0.3838, 0.063, 0.2364, 0.4115, 1.992, 0.1267, 0.1738, 0.6953, 1.545, 1.081,
 0.2868, 0.3608, 0.4957, 1.145, 1.208, 0.6858, 0.3088, 0.6317, 1.775, 1.041, 0.1528, 0.4159,
 0.02736, 0.5353, 0.6089, 2.913, 1.045, 0.6482, 0.4768, 2.68, 0.739, 0.8479, 0.5364, 0.4818,
 1.591, 0.6413, 1.601, 1.885, 1.187, 1.058, 0.2617, 0.5849, 0.4108, 0.2554, 2.170, 0.05038,
 0.922, 0.548, 1.118, 0.4375, 0.2867, 1.258, 1.946, 1.285, 3.529, 0.1636, 0.6345, 0.4854,
 0.7843, 0.1206, 0.01085, 1.109, 0.2941, 0.6337, 2.153, 0.259, 0.0638, 0.5984, 0.6349,
 1.994, 0.2299, 0.613, 1.368, 0.4662, 0.8512, 0.952, 0.979, 0.05576, -0.005093, -0.02776,
 0.7244, 0.4402, 2.255, 0.4005, 1.979, 1.303, 0.1755, 1.446, 1.91, 0.2385, 0.01685, 1.503,
 0.5205, 1.468, 0.000354, 0.4726, 1.542, 0.6626, 0.2530, 1.789, 0.46, 0.4155, 1.057, 0.2215,
 0.7964, 0.9316, 0.5375, 0.556, 0.5803, 0.6672, 1.044, 0.1653, 0.2469, 0.3610, 0.4914,
 0.4712, 0.3293, 0.4251, 0.1339, 0.07557, 0.6384, 0.9532, 1.499, 0.2817, 0.0725, 0.2009,
 0.1070, 0.3102, 0.2726, 0.1441, 0.7449, 0.4174, 1.006, 1.396, 0.4628, 1.693, 0.7006, 1.023,
 0.7822, 1.744, 0.2818, 1.150, 1.674, 0.2679, 0.03785, 0.9016, 0.6949, 0.2040, 0.4537,
 2.236, 0.069, 0.5856, 1.153, 0.321, 1.436, 1.260, 0.252, 0.1965, 0.01678, 1.396, 2.419,
 0.6925, -0.09542, 0.3303, 1.326, 0.5899, 0.9218, 2.091, 0.4583, 0.3659, 0.3332, 1.122,
 1.270, 0.652, 0.6947, 1.614, 0.7205, 2.409, 0.2418, 0.5793, 0.04454, 1.146, 0.1012, 2.021,
 0.1094, 0.1273, 0.6518, 1.228

Bibliography

- [1] Akaike H., (1973), Information Theory and an Extension of the Maximum Likelihood Principle, *Proc. 2nd Inter. Symposium on Information Theory*, 267-281, Budapest
- [2] Azzalini A., Capitanio A., (1999), Statistical applications of the multivariate skew-normal distribution, *J. R. Statist. Soc. B*, **61**, 579-602
- [3] Burnham K. P., Anderson D. R., (2004), Multimodel Inference: Understanding AIC and BIC in Model Selection, *Sociological Methods & Research*, **33**, 261-304
- [4] Cavanaugh J., (2005), AIC and Corrected AIC, AICc, *Advanced Biostatistics Seminar: Model Selection*
- [5] Cetin M. C., Erar A., (2002), Variable Selection with Akaike Information Criteria: A Comparative Study, *Hacettepe Journal of Mathematics and Statistics*, **31**, 89-97
- [6] Chambers L., (1995), *Practical handbook of genetic algorithms: new frontiers*, CRC Press
- [7] Chiogna M., (2005), A note on the asymptotic distribution of the maximum likelihood estimator for the scalar skew-normal distribution, *Statistical Methods & Applications*, **14**, 331-341
- [8] Everett J.E., (1995), Model Building, Model Testing and Model Fitting, *Practical Handbook of Genetic Algorithms: Applications Volume 1*, ed. Lance Hambers, New York, CRC Press Inc, 1, 5-6

- [9] George E. I., (2000), The Variable Selection Problem, *Journal of the American Statistical Association*, **95**, No. 452., 1304-1308
- [10] Goldberg D. E., (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley
- [11] Haupt R. L., Haupt S.E., (2004), *Practical Genetic Algorithms*, John Wiley & Sons, Inc., Hoboken, New Jersey, p 22
- [12] Holland J., (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press: Ann Arbor, MI
- [13] Liseo B., Loperfido N., (2006), Default Bayesian analysis of the skew-normal distribution, *Journal of Statistical Planning and Inference*, **136**, 2, pp. 373-389
- [14] Liseo B., Loperfido N., (2003), A Bayesian interpretation of the multivariate skew-normal distribution, *Statistics and Probability and Letters*, **61**, 395401
- [15] Kamepalli H.B., (2001), The Optimal Basics for GAs, *IEEE Potentials*, **20**, Issue 2, pp. 25-27
- [16] Mitchell M., (1996), *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, pp 2-3, 7-9
- [17] Rodriguez C., (2005), The ABC of Model Selection: AIC, BIC, and the new CIC, *Bayesian Inference and Maximum Entropy Methods in Science and Engineering, AIP Conf. Proc. 803*, **80**
- [18] Whitley D., (1994), A genetic algorithm tutorial, *Statistics and Computing*, **4**, 65-85