# 14. ON THE IMPLEMENTATION AND USE OF A GENETIC ALGORITHM WITH GENETIC ACQUISITIONS

**George Daniel MATEESCU***

## Abstract

*A genetic algorithm is convergent when genetic mutations occur on the objective function gradient direction. These genetic mutations are called genetic acquisitions (Mateescu, 2005). We improved the algorithm and its implementation by using the characteristics of parents in order to generate new individuals. Finally, we applied the genetic algorithm in order to find the parameters of a Cobb-Douglas function.*

**Keywords**: evolutionary algorithms, optimization

**JEL Classification**: C61, C63

## Introduction

In this paper, we present the implementation of a genetic algorithm that includes genetic acquisitions. In a previous paper we proved the convergence of such an algorithm, and now we will make an improvement, by changing the method of new individuals generating. Also, taking into account the algorithm, we will present how to create C++ programs.

We plan to analyze the implementation and application of a genetic algorithm described in the form of blocks, as follows:

**Step 1**. Generate initial population

**Step 2**. Apply genetic mutations of a number of individuals

**Step 3**. Generate offspring

**Step 4**. Remove excess population

**Step 5**. Continue with Step 2, until the stop conditions are fulfilled.

Consider an optimization problem as:

$\inf f$, where: $f$ is a function $f : \mathrm{R}^m \to \mathrm{R}$

*Institute of Economic Forecasting, Romanian Academy, e-mail: daniel@mateescu.ro .

Suppose that the function *f* possesses *Frechet* derivative and it fulfills a *Lipschitz* type condition:

$$\|f'(u)-f'(v)\| \le L\|u-v\|, L>0$$

where: $\|u-v\|$ represent the *Euclidian* norm in $\mathrm{R}^m$ and $\|f'(u)-f'(v)\|$ is the corresponding linear application norm.

As shown in Mateescu (2005), the sequence $(u_n) \subset \mathrm{R}^m$ which consists of the individuals with minimum value of the function *f*, at the *n*-th generation, is convergent. If the function *f* possesses partial derivatives of second order, and the *Hessian* matrix is positively defined, i.e. satisfies a relationship of the form:

$$\langle f''(x)y, y\rangle \ge a\|y\|^2, a>0 \text{ (where } \langle,\rangle \text{ is the inner product in } \mathrm{R}^m) \quad (1)$$

then $u_n \to u_{\min}$ and the speed of convergence is measured by an inequality of the form:

$$\|u_{\min}-u_n\| \le const. \cdot \frac{1}{a}$$

## Implementation of Genetic Algorithms

The functional framework described is implemented by using the C++ language, observing the steps of a genetic algorithm.

**Population**

This is a subset of possible solutions, i.e. a finite subset of points in $\mathrm{R}^m$ space. Each point is an individual, with *m* attributes corresponding to its coordinates. Furthermore, we associate a measure of its performance, by associating the appropriate value of objective function *f*. In summary, we associate to an individual, one vector of the form:

$$(x_1, x_2, ..., x_m, f(x_1, x_2, ..., x_m))$$

In C++ implementation, we consider that the population is a matrix:

**float** pers[max] [m+1];

where: max represents the number of individuals.

On each matrix row, the first *m* components are corresponding to coordinates and the *m*+1-th is the objective function value. In implementation, the objective function has the form:

**float** goal (x1, x2, ..., xm)

and

xm+1 = goal (x1, x2, ..., xm)

**Genetic mutations**

We used genetic acquisitions that correspond to genetic modification of coordinates, with respect to the objective function gradient. As it is demonstrated in Mateescu (2005), such genetic mutations lead to the convergence algorithm.

In implementation, we will use a function that determines the sign of partial derivatives corresponding to a coordinate. Thus, the genetic mutation of a component will be implemented as an assignment of the form:

pers [n] [i] = pers [n] [i]-h *sign (n,i)

where: *h* is a value equal to the desired accuracy of calculations.

In applications we used *h*=0.001 and an approximation of the partial derivative, respectively:

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x_1, x_2, ..., x_i + h, x_{i+1}, ..., x_m) - f(x_1, x_2, ..., x_i, x_{i+1}, ..., x_m)}{h}$$

**Offspring**

I brought an improvement to the method of generation of descendants, in conformity with the optimization problem. Thus, two individuals

$$x = (x_1, x_2, ..., x_m) \text{ and}$$

$$y = (y_1, y_2, ..., y_m)$$

will generate a new one:

$$z = \frac{x+y}{2} = \left(\frac{x_1+y_1}{2}, \frac{x_2+y_2}{2}, ..., \frac{x_m+y_m}{2}\right)$$

By using the functional framework, it results that the objective function *f* is convex. Indeed, by using Taylor's formula and (1) we have:

$$f(x) - f(z) - f'(z)(x-z) = \frac{1}{2}\langle f''(\sigma)(x-z), (x-z)\rangle \geq 0$$

$$f(y) - f(z) - f'(z)(y-z) = \frac{1}{2}\langle f''(\tau)(y-z), (y-z)\rangle \geq 0$$

which implies, by adding the above relations:

$$f(x) + f(y) - 2f(z) - f'(z)(x+y-2z) \geq 0$$

While $x + y - 2z = 0$, it follows:

$$f(x) + f(y) - 2f(z) \geq 0 \text{ or}$$

$$f(z) \leq \frac{f(x)+f(y)}{2}$$

The above relationship shows that the new individual will have the objective function value lower than that those of almost one of his/her parents, i.e.:

$$f(x) \geq f(z) \text{ or } f(y) \geq f(z)$$

**Selection**

Next, we ordered the population according to objective function values in the form

$u_1, u_2, ..., u_i, ..., u_j, ..., u_{\max}$, where $f(u_q) \geq f(u_{q+1}), q = 1..\max-1$

Assuming that $u_k$ is generated by using $u_i$ and $u_j$, we already stated that $f(u_i) \geq f(u_k)$ or $f(u_j) \geq f(u_k)$ which implies that we may eliminate any element situated at the left side of $u_i$ and $u_j$.

Consequently, in our implementation, we used a population consisting into 100 individuals, but we used only the last 50 in order to create new individuals. This new population may replace the firsts 50 individuals, as we observed above. Finally we ordered the population, descending, according to the objective function *f*.

By using such a strategy we keep constant the number of population, but, as it is demonstrated in Mateescu (2005), the algorithm is convergent, i.e., the sequence of the best individuals, in each generation, is convergent and the limit is a local optimum point for the function *f*.

## Validation of the Genetic Algorithm

Finally, we need to verify our algorithm, by using some significant values. We used statistical data on GDP, employment and gross fixed capital formation, in order to determinate the coefficients of a *Cobb-Douglas* function:

$$Y = AL^{\alpha}K^{\beta}$$

To avoid inconsistent units of measurement, we used a dimensionless form, corresponding to two-year consecutive evolution, starting from:

$$Y_{t+1} = AL_{t+1}^{\alpha}K_{t+1}^{\beta}$$

$$Y_t = AL_t^{\alpha}K_t^{\beta}$$

$$\frac{Y_{t+1}}{Y_t} = \left(\frac{L_{t+1}}{L_t}\right)^{\alpha}\left(\frac{K_{t+1}}{K_t}\right)^{\beta}$$

Using National Institute of Statistics official web site we processed the following data:

**Table 1**

| | $Y_{t+1}/Y_t$ | $L_{t+1}/L_t$ | $K_{t+1}/K_t$ |
|---|---|---|---|
| 1997/1996 | 0.938 | 0.961 | 1.016 |
| 1998/1997 | 0.952 | 0.977 | 0.942 |
| 1999/1998 | 0.988 | 0.955 | 0.951 |
| 2000/1999 | 1.021 | 1.024 | 1.054 |
| 2001/2000 | 1.058 | 0.992 | 1.101 |
| 2002/2001 | 1.050 | 0.972 | 1.082 |
| 2003/2002 | 1.052 | 0.998 | 1.084 |
| 2004/2003 | 1.085 | 0.990 | 1.111 |
| 2005/2004 | 1.041 | 1.018 | 1.126 |
| 2006/2005 | 1.079 | 1.017 | 1.228 |
| 2007/2006 | 1.062 | 1.004 | 1.290 |

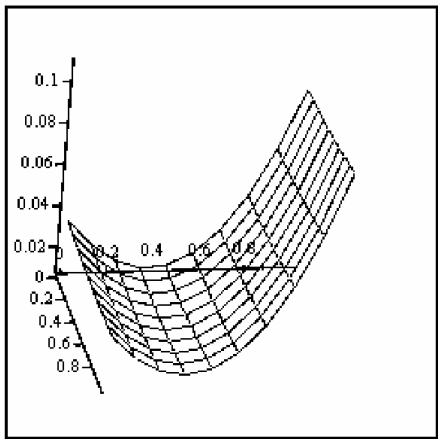
*Source:* http://www.insse.ro

Applying the genetic algorithm described, we optimized the function:

$$\varphi(\alpha,\beta) = \sum_t \left(\frac{Y_{t+1}}{Y_t} - \left(\frac{L_{t+1}}{L_t}\right)^{\alpha}\left(\frac{K_{t+1}}{K_t}\right)^{\beta}\right)^2 \quad (1)$$

**Figure 1**

**Objective function φ**




and we obtained the values $\alpha = 0.245$, $\beta = 0.359$ (we used the above data from Table 1 for *Y*, *L*, *K*).

Although the algorithm is convergent, the function (1) graph shape suggests the direction of slow convergence (Figure 1).

Taking into account this observation, we questioned the mathematical behavior of a function related to a more general *Cobb-Douglas* function:

$$\frac{Y_{t+1}}{Y_t} = \gamma \left( \frac{L_{t+1}}{L_t} \right)^{\alpha} \left( \frac{K_{t+1}}{K_t} \right)^{\beta} \tag{2}$$

Such a function would correspond to a case when the parameter *A* is not constant. Minimizing the corresponding function *j* provides an appropriate example of spectacular function, having more points of local minimum. Thus, using data from the table, the simulation with genetic algorithm, we obtained various points minimum, such as:

$\alpha$=0.195, $\beta$=0.354, $\gamma$=1.001

$\alpha$=0.364, $\beta$=0.331, $\gamma$=1.004

## Conclusions

The genetic algorithm may be used in order to find the minimum points for an objective function. If such a function corresponds to a last square expression, the algorithm may be used in order to find some known function parameters.

On the other hand, by using the genetic algorithm it is possible to identify multiple minimum points. Such a situation may occur simply by executing repetitively the associated C++ program. The existence of multiple optimal points raise serious doubt on the possibility of using the *Cobb-Douglas* function with three parameters. This is

the case of the function (2) the conclusion being that the function *Cobb Douglas* having more than 2 parameters is not reliable according to the statistical data in Table 1.

In fact, taking into account the shape of the function in Figure1 it results the important conclusion that *Cobb Douglas* function becomes less reliable as the number of parameters increases.

**Annex**

## C++ PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
FILE * date;
float pers[100][3];
float h=0.001;

float goal(float alfa,float beta){
float s,y,l,k;
s=0;
date=fopen("date.art.dat","r");
for(int i=0;i<11;i++){
  fscanf(date,"%f,%f,%f",&y,&l,&k);
  s+=(y-pow(l,alfa)*pow(k,beta))*(y-pow(l,alfa)*pow(k,beta));
        }
  fclose(date);
  return s;
}
int sign(int n,int i){
  switch(i){
        case 0: {if(goal(pers[n][0]+h,pers[n][1])-goal(pers[n][0],pers[n][1])>0) return 1;else return -1;};
        case 1: {if(goal(pers[n][0],pers[n][1]+h)-goal(pers[n][0],pers[n][1])>0) return 1;else return -1;};
  }
}
void generate(){
randomize();
for(int i=0;i<100;i++){
        for(int j=0;j<2;j++) pers[i][j]=(1+random(40))/10.;
  pers[i][2]=goal(pers[i][0],pers[i][1]);
  }
}
void mutation(int n){
        pers[n][0]+=-h*sign(n,0);
        pers[n][1]+=-h*sign(n,1);
}
float * crossover(float (* a)[3],float (* b)[3]){
        float (* p)[4] = new float[3];
  int i=random(3)-1,j;
        for(j=0;j<=i;j++)(*p)[j]=(*a)[j];
  for(j=i+1;j<2;j++)(*p)[j]=(*b)[j];
  (*p)[2]=goal((*p)[0],(*p)[1]);
  return p;
}
void order(){
```

```
float temp;
for(int p=99;p>0;p--) for(int q=0;q<p;q++) if(pers[q][2]<pers[q+1][2])
            for(int j=0;j<3;j++){
            temp=pers[q][j]; pers[q][j]=pers[q+1][j]; pers[q+1][j]=temp;
            }
}
void main(){
float * r;
int i,p,q;
generate();
order();
do{
  for(i=0;i<50;i++) mutation(random(100));
  for(i=0;i<50;i++){
            p=50+random(50);q=50+random(50);
            r=crossover(pers[p],pers[q]);
            pers[i][0]=r[0]; pers[i][1]=r[1];
            pers[i][2]=goal(pers[i][0],pers[i][1]);
  }
  order();
}while(fabs(pers[0][0]-pers[99][0])+fabs(pers[0][1]-pers[99][1])>h);
printf("%f\n%f\n%f\n%i\n",pers[99][0],pers[99][1]);
getchar();
}
```

## References


Back, Th. (1966), "*Evolutionary Algorithms in Theory and Practice"*, New York: Oxford University Press.

Bakhvalov, N. (1976), *Methodes numeriques*, Editions MIR.

Mateescu, G.D. (2005), "Optimization by Using Evolutionary Algorithms With Genetic Acquisitions"*, Romanian Journal of Economic Forecasting*, 6(2): 26-30.

National Institute of Statistics, http://www.insse.ro

Pchenitchny, B., Daniline, Y. (1977), *Methodes numeriques dans les problemes d'extremum*, Editions MIR.

Stoer, J., Bulirsch, R. (1992), *Introduction to Numerical Analysis*, Springer-Verlag.