Multi-variable function maximization using genetic algorithms

Nelson Brochado

October 31, 2020

1 Introduction

1.1 Problem

We are given the following function $f : \mathbb{R} \to \mathbb{R}$

$$f(x, y, z) = 2xze^{-x} - 2y^3 + y^2 - 3z^3$$

And we want to solve the following problem

$$\begin{aligned} \max_{x,y,z} \quad & f(x, y, z) \\ \text{s.t.} \quad & x, y, z \in [0, 100] \end{aligned} \tag{1}$$

using a genetic algorithm (GA).

1.2 Genetic algorithms

To do that, we first need to define several basic elements of GAs for our specific problem, such as

1. the *individual* (aka chromosomes or genotypes), i.e. how a solution is represented in the GA,
2. the *fitness function*, i.e. how we evaluate the individuals,
3. the *population*, i.e. the collection of individuals that need to be evolved from one generation (iteration) to the other
4. the *genetic operators*, such as
   (a) *selection*, i.e. how we select from the current population the individuals that will be combined to produce new individuals (aka the children)
   (b) *crossover*, i.e. how exactly we combine individuals to produce new individuals
   (c) *mutation*, i.e. how we (randomly) mutate an individual

There are possibly other concepts and parameters that can be defined and discussed, but these are the basic elements.

In the next subsections, I will describe more in detail the choices that I made to implement these basic elements to solve our problem.

1.3 Individual

We want to find a point of the domain of f, i.e. $x, y, z \in [0, 100]$, such that $f(x, y, z)$ is a (local) maximum. The natural way of representing solutions to this problem in a GA is then to use an array of three floating-point numbers, one for each of the coordinates of a point of the domain of f[1]. The elements of this array (i.e. the *genes*) need to be constrained to be in the range $[0, 100]$. This can easily be achieved.

1. During the generation of the population, the elements of the array can be generated with a pseudo-random number generator that returns values in this range.

2. Then, during the genetic operations, we only allow operations that maintain this constraint (I will explain this more in detail in the next corresponding subsections).

1.4 Fitness function

In GAs, the way of measuring the fitness of an individual of a population is (typically) very *specific* to the problem that needs to be solved. In our case, an individual $i = (x_i, y_i, z_i)$ is fitter than an individual $j = (x_j, y_j, z_j)$ if $f(x_i, y_i, z_i) > f(x_j, y_j, z_j)$. This is clearly a *relative* measure of fitness given that it only allows us to determine the fitness of an individual *relative to* another.

1.5 Population

Initially, we uniformly generate at random the population of individuals, so that their components lie in the range $[0, 100]$[2]. Clearly, at the beginning, the fitness of the individuals will be quite low. However, we expect that some of these individuals converge to an optimal solution (in our case, a local maximum of f) with more generations (iterations) of the GA.

The *size* of the population n can affect the running time and performance of the GA. So, one must choose this parameter in order to trade-off these. In this case, I set $n = 100$, after some trial and error. There are probably some heuristics or more principled ways to choose this parameter, but I have not investigated them.

There is also the issue of deciding *when to terminate* the GA. There are several heuristics [1]. For simplicity, in my case, I terminate the GA after a fixed number of iterations $m = 30000$.

1.6 Genetic operators

There are several *genetic operators* (or *operations*) that need to be defined and implemented to solve our problem with a genetic algorithm. The most common ones are the *selection*, the *mutation* and the *crossover*. These operations are responsible for the introduction of randomness and maintenance of the fittest individuals in the population, which can lead to the discovery of good solutions. I will describe below the specific genetic operators that I implemented and the parameters that I used.

[1]In GAs, people sometimes distinguish between *genotype* (aka *chromosome* or *individual*), which is the encoding of a solution, and *phenotype*, which is the actual solution. In our case, the genotype is equal to the phenotype.

[2]If we knew something about the solution(s), we could generate the initial individuals so that they lie closer to the solution(s).

1.6.1 Selection

The *selection* is the operation of picking individuals from the current population for the purpose of *reproduction* (i.e. *crossover*) in order to produce new possibly fitter individuals, which can then be mutated. In theory, nobody prevents us from mutating (the parents) before the crossover. In my case, I arbitrarily chose to mutate only the children.

There are different selection strategies, such as *fitness proportionate selection* (a.k.a. roulette wheel selection), *stochastic universal sampling*, *tournament selection*, *truncation selection* or *Boltzmann selection* [1, 3, 2]. Some of them are more appropriate for certain problems or settings than others.

I decided to implement the *fitness proportionate selection* (FPS) because of its simplicity. In FPS, the probability of selecting an individual for reproduction is proportional to its fitness, so the higher the fitness of an individual the higher the probability of that individual being selected.

The co-domain of f is $\mathbb{R}$, so the fitness of an individual by just evaluating f can be negative. If we compute the FPS with the following formula

$$p_i = \frac{f_i}{\Sigma_{j=1}^{N} f_j},$$

where p_i is the probability of individual i being selected and f_i is the fitness of individual i, then the probabilities could be negative. To solve this issue, we need to constraint the fitness of all individuals in the population to be greater or equal to 0. This is typically done with a technique called *windowing* [1]. More precisely, let $f_{\min}$ be the fitness value of the least fit individual. Then $|f_{\min}|$ is added to all fitnesses at every generation. This shifts all fitnesses to be greater or equal to zero, but it does not change the relative fitness of the individuals, so it doesn't also change their relative likelihood of being chosen for reproduction.

1.6.2 Mutation

The *mutation* of an individual can be implemented in different ways, depending on the problem, the representation of the solution or how you want to explore the space of solutions.

In my case, with a certain probability $m = 0.5$, I randomly mutate a gene (i.e. either x, y or z) of the individual by uniformly sampling a number in the range $[-100, 100]$ and adding it to the gene. If the addition of the mutation doesn't maintain the constraint of the gene being in the range $[0, 100]$, the mutation is undone and a new mutation is attempted. This is done separately for all three genes of the chromosome. So, each gene has 50% probability of being mutated independently of the others. The choice $m = 0.5$ is a little bit arbitrary and based on some trial and error[3].

1.6.3 Crossover

Similarly to the mutation, the *crossover* (or *reproduction*) could also be implemented in different ways.

In my case, with a probability $c = 0.8$, a gene in the first chromosome is replaced with the corresponding gene in the other chromosome. This is done independently for every gene.

[3] There are probably some heuristics to choose this and other parameters of the GA, as I mentioned before, but, for simplicity, I have ignored them.

2 Implementation

You can find my implementation at https://github.com/nbro/function_max_with_ga.

2.1 Results

I created several plots that show the progress of the genetic algorithm throughout the generations. In particular, I created a plot that shows the fitness (figure 1), the value of the function f (figure 2) and the sum of the absolute value of each component of the gradient of f (figure 3) of the best individual at every generation. The plots show that the value of function evaluated at the best individual increases (as expected), while the gradient tends to zero (as expected). The fitness of the best individual oscillates a little bit. Maybe this can be avoided by tuning some of the parameters, so that the GA converges more smoothly.

The best individual that I found after only some trials is

$$\begin{bmatrix} 0.999992 \\ 0.33767089 \\ 0.28161934 \end{bmatrix}$$

which is very close to the actual local maximum[4]

$$\begin{bmatrix} 1 \\ \frac{1}{3} \\ \frac{\sqrt{\frac{2}{e}}}{3} \end{bmatrix} = \begin{bmatrix} 1 \\ 0.33333333 \\ 0.28592129 \end{bmatrix}$$

References

[1] Dezdemona Gjylapi and Vladimir Kasëmi. The genetic algorithm for finding the maxima of single-variable functions. *Research Inventy: International Journal Of Engineering And Science*, 4:46–54, 03 2014.

[2] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *FOGA*, 1990.

[3] Khalid Jebari. Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3:333–344, 12 2013.

[4] I found this local maximum with Wolfram Alpha.

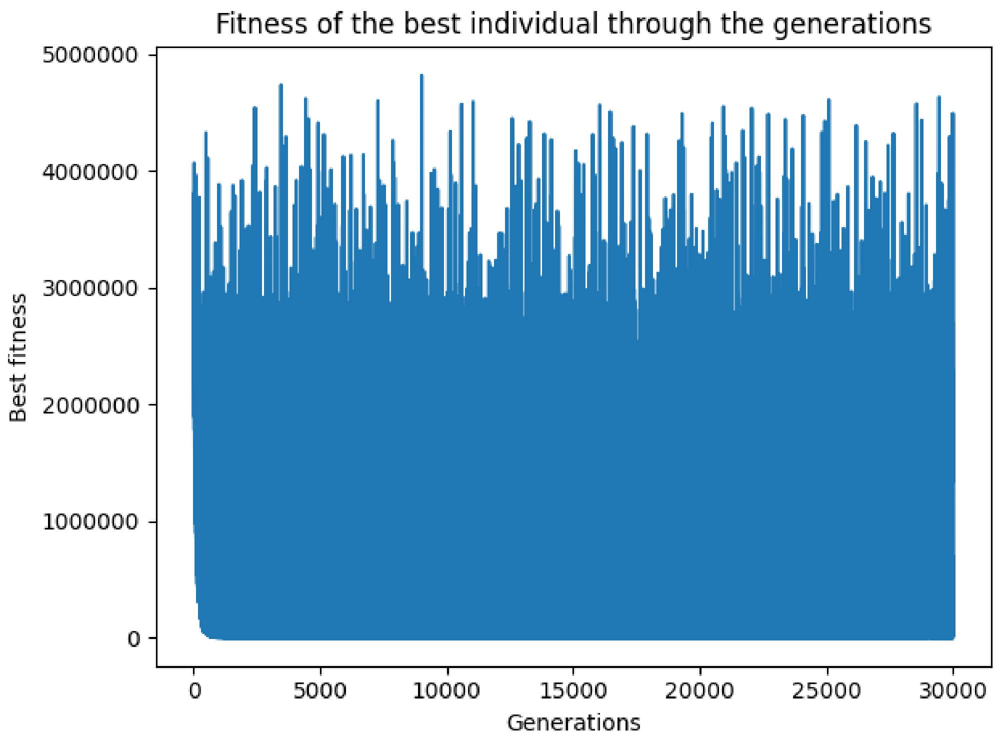

Figure 1: The fitness of the best individual throughout the generations.

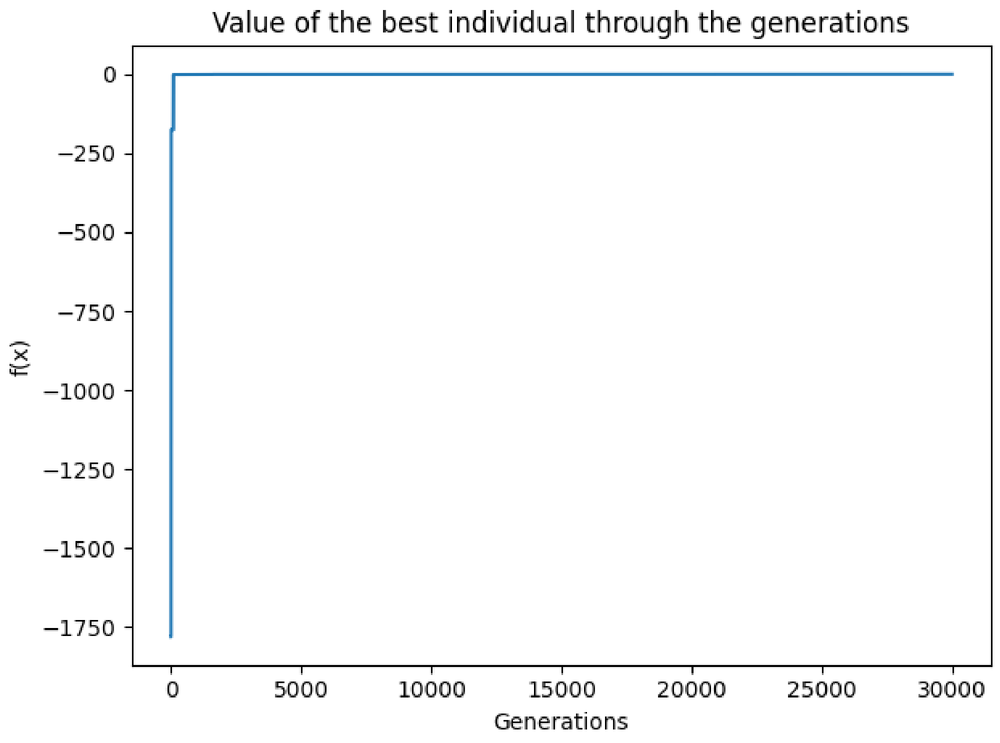

Figure 2: The value of the function f of the best individual throughout the generations.

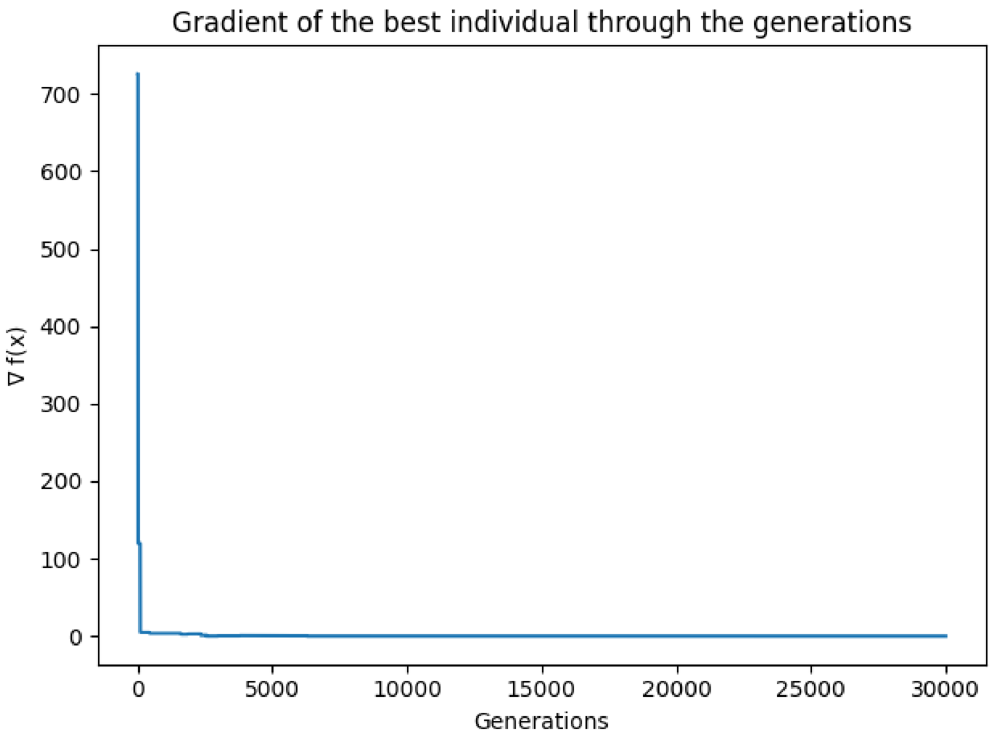

Figure 3: The sum of the absolute value of each component of the gradient evaluated at the best individual throughout the generations.