Optimization: Genetic algorithms

Adrià Colás Gallardo (1419956)
Xavier Olivé Fernández (1394407)

February 16, 2021

---

1 Introduction

For a year now, COVID-19 disease has not ceased to affect humans until it became a pandemic. This disease has affected many people around us. That is why it is crucial to be able to understand the dynamics of virus infection and patient recovery. This simulation provides data that will be used to try to predict the evolution of the virus on humans.

In *Genetic algorithms: A possible model for COVID-19* [1] we can see the information of the SEIR model like the variables implied, the system of Differential Equations in dimension 8 to be solved for each step and the 11 parameters related with infectivity evolution. With all that characteristics, Genetic algorithms are presented as a possible solution to find the solution to a problem like this in a relatively simple way.

The objective of this assignment is to implement a Genetic Algorithm to find the best parameters that fit the model with the current data in order to predict future statements.

2 Implementation

2.1 Genetic algorithm

Genetic algorithms (GA) are metaheuristic algorithms inspired by the process of natural selection that belongs to the larger class of evolutionary[2]. In its implementation we first have to define our individuals:

INDIVIDUALS

Each individual, as we see, have two chromosomes: the first one is a vector formed by a possible choice of the 3 unknown initial conditions, and the second one is a vector formed by a possible choice of the 11 parameters[1]:

$$\texttt{CHROMOSOME 1:} \quad (E(0), I_1(0), A(0))$$

$$\texttt{CHROMOSOME 2:} \quad (\beta, \phi, \varepsilon_I, \varepsilon_Y, \sigma, \gamma_1, \gamma_2, \kappa, p, \alpha, \delta)$$

The parameters are stored as unsigned long, which have a size of 32 bits, as every parameter have a different maximum that requires some adjustments in the computation of the random parameters and in the genetic functions. In fact, that differentiation of precision for each parameter makes things very difficult for us when

at the time to introducing our mutations of genetic algorithms.

The total number of individuals we propose to solve the exercise is 200 that is the popsize number for each generations (a vector of 200 individual structures). It is important to notice that this population size is not the same that the total population implied in the data problem (1.000.000) (see [1] to see that). In order to initialize each parameter into IC an Pars arrays according the maximum intervals defined we use the function `random_long` and `set_individuals`.

First individuals are not generated randomly but pseudo-randomly as it has been prioritized to understand well the functioning of a genetic algorithm and by simply using the `rand()` function we can obtain the same results in order to be able to quantify well the changes introduced.

ODE SOLUTION

With the parameters and initial conditions established for each individual, we can solve the proposed ODEs to obtain the resulting data series. This process is done by each individual during all the generations to be formed. The solution of all these ODEs is done through a numerical method of calculation (*Runge-Kutta-Fehlberg-Simo*) and is not focusing on the optimization itself. For this reason, this function is already given with `RKF78Sys`, which we have used without making any modifications. In this sense, the `CoreModelVersusDataQuadraticError` and `GeneratePredictionFromIndividual` functions have been used to adapt the information that the Runge-Kutta function needs in the correct format.

FITNESS FUNCTION

Once the candidate data has all been calculated, a fitness function is used to see how much they resemble the real data. The fitness function that we selected for this problem has been the next one:

$$F_f = \sum_{t=1}^{100} \left(|A_d(t) - \bar{A}_d(t)| + |I_2(t) - \bar{I}_2(t)| + |Y(t) - \bar{Y}(t)| + |R(t) - \bar{R}(t)| + |D(t) - \bar{D}(t)| \right)$$

where the prediction data has been compared with the stored as double `Data[101][5]` given. We have used a summation of absolute values instead of a power of 2 as we are have very big errors (differences bigger that 1) so that we don't menage very big numbers.

GENETIC FUNCTIONS

Until now, the method used can be considered computational and it is when the functions of the genetic algorithm are implemented that we consider it optimization. In reality, there is not only one genetic algorithm, but we have different options when it comes to applying it, depending on the type of problem we have and the strategy we want to follow[3]. In this case we present a possible solution to implement it. We formulate 3 parts:

1. For the **Selection With Replacement** step we have considered tournament selection due to its simplicity which randomly select two individuals randomly and selects the fittest one. This processes is done twice in order to select two parents to generate two children. `TournamentSelection` is used for this propose.

2. For the **Breeding** step we have selected the one-point crossover choice. Here we select randomly a parameter of the first child. All the parameters before this one will be given by parent 1 and the subsequent ones by parent 2. The other child will have the values exchanged.

   It is important to see that each phenotype parameter has a different precision. This translates to having a genotype of different length for each parameter. For this reason, and with the intention of

simplifying the problem, crossovers are not implemented in the middle of any digit of the genotype that represents the same parameter.

3. For the **mutation** step we have implemented a incremented bit-flip mutation selected randomly. It selects a single parameter of the individual and introduces in its genotype 2 mutations. It is important to remark that we have had to consider for each parameter where to introduce the mutation according to the maximum size stored. Depending on the type of perimeters (with different factor from genotype to phenotype) the mutation will affect more or less digits.

   It is IMPORTANT to notice that, as we want to be very exploratory, we apply 2 mutations for each new individual. All program has problem at the time to be exploitatory, even when we just use one mutation that affects less digits. That gives us a very painful process to get a great solution, so the exploratory process has been prioritized.

2.2 Code

The code can be divided in some scripts. All the program must be compiled and executed the `GA_covid.c` script. In this section we are going to comment deeply each file used:

- The principal script of our program is `GA_covid.c` where all program is structured. It contains the `main` function that we use to solve generally the Genetic Algorithm problem. First we allocate memory for all the individuals. Then we initialize all the individuals assigning pseudo-random parameters and computing their fitness respectively. Secondly we call `find_fittest` function that returns a pointer to the best fittest individual storing this individual. Finally we do the GA loop where all new generations are created and the fitness for all individuals is calculated. We have performed in the order of 1000 iterations to find the fittest individual.

  In this file we also observe some constants and the functions related with the Genetic algorithms:

  - `ExitError`: Defines how the Exit errors are done in our program.
  - `uniform`: Is used to generate a the pseudo-random numbers between 0 and 1 for mutations and the crossover.
  - `random_long` Is used to generate the pseudo-random numbers for the initial numbers.
  - `set_individuals`: Function to generate the initial conditions and the parametres for an individual.
  - `fittest_ind`: It allows us to identify the best individual in the population in which is used.
  - `printIndividual`: It is used to print the individual (it prints the calculation figure and the real value it represents). We will use it at the beginning and in the end of the program to see all the evolution we have.
  - `TournamentSelection`: This function takes the 2 individuals that will be used to generate 2 children for the new generation. It is used n=popsize times as we want the same number of children than parents. It picks two candidates and selects the fittest one to be one of the parents.
  - `OnePointCrossover`: It takes 2 individuals and pseudo-randomly selects a crossover point. Then 2 children will we created with the cut in crossover perimeters from one parent and the other.
  - `Mutation`: It selects a single parameter of the individual and introduces in its genotype 2 mutations. Depending on the type of parameters (with different factor from genotype to phenotype) the mutation will affect more or less digits of the genotype.

- The `Fit_documentation.h` script is used as auxiliary script. Principally is used to calculate the data series of a proposed individual and to know the fitness value it has. Contains the libraries used for `GA_covid.c`, and the constants and structs which will be used to store the information about the updated nodes in the `main` program and in this data calculation part. The principal function of this part is

`CoreModelVersusDataQuadraticError` that is the principal to calculate the fitness value. However, in this file we found other functions that are used to obtain that value

- `CoreModel`: Function where the derivatives to be solved with Runge-Kutta method are expressed.
- `fitness_function`: It contains the Fitness function used to see how our result is adjusted. In consists in simple summatory of the differences of absolute differences.
- `GeneratePredictionFromIndividual`: Function that uses Runge-Kutta78 to calculate all the data for all day and all the EDO parametres for each individual (we use the function `RKF78Sys` which has been given by theory).
- `CoreModelVersusDataQuadraticError`: Function that stores the needed values for calculate all data in vectors. Then `GeneratePredictionFromIndividual` in introduced to do that calculation.

This file, also contains the `Data` matrix with the original data of the table given[1] that will be used to set the fitness of each proposed candidate.

- `RKF78.h` and `RKF78.c` scripts contains the function for solving the ODEs problem with a computational method for Runge-Kutta-Fehlberg-Simo implementation. They contain several function and some vectors of data. However they are given with the program and we will not explain it as they are vary calculus complex methods.

3 Results

Now in this section, we will comment the evolution of the results found. First, we show the best individual in the generation 0 (individual with lower fitness of all 200).

```
IC 0: 226224595  -  226224.595000
IC 1: 36996362  -  36996.362000
IC 2: 101340326  -  101340.326000
Parametre 1: 1346094045  -  0.001224265402
Parametre 2: 527127  -  0.502707
Parametre 3: 279  -  0.272
Parametre 4: 132  -  0.129
Parametre 5: 658581  -  0.628072
Parametre 6: 546297  -  0.520989
Parametre 7: 691949  -  0.659894
Parametre 8: 286  -  0.279
Parametre 9: 511273  -  0.487588
Parametre 10: 1474391173  -  0.001340950960
Parametre 11: 278641878  -  0.000253423312
fitness: 35509310.000000
```

This individu has been selected with the `fittest_ind` function. In all that generation the Runge-Kutta procedure has been applied in order to obtain the data and then the fitness of each individual is calculated. So in this first individual no Genetic procedure has been applied. In this conditions the value of our fitness function is about $3.55 \cdot 10^7$. This is a very large number, however we have to consider that is high because of the accumulation of all errors for all 5 numbers to compare in all 100 iterations.

When applying our genetic algorithm with all this 200 individuals and 1000 iterations we get this results:

```
IC 0: 132322295  -  132322.295000
IC 1: 94894  -  94.894000
IC 2: 358  -  0.358000
Parametre 1: 2143619446  -  0.001949610529
```

```
Parametre 2: 851466  -  0.812021
Parametre 3: 733  -  0.716
Parametre 4: 460  -  0.449
Parametre 5: 15855  -  0.015121
Parametre 6: 6374  -  0.006079
Parametre 7: 24747  -  0.023601
Parametre 8: 49  -  0.048
Parametre 9: 326533  -  0.311406
Parametre 10: 2521263975  -  0.002293076227
Parametre 11: 2722835417  -  0.002476404386
fitness: 1375574.000000
```

As we can see the value of the fitness still being high, however it has reduced a lot to $1.37 \cdot 10^6$ (more than a 10 factor).

Although we note that the improvement is significant, these results are not good. These are far from a good enough tolerance that it should be around $5 \cdot 10^5$ points.

To find a good enough result many more iterations are necessary. This means a huge increase in time. Throughout the tests, some attempts had been tested, yet the improvements observed are getting smaller and smaller as the iterations get bigger. In that sense, several hours would be necessary to obtain the results.

A possible improvement would also be the increase of individuals to use. When the number of individuals is small, the fitness is very high. In exchange for a greater number of individuals the results improve significantly. However, increasing the number of individuals a lot is not convenient either; since the memory required is very large and the results do not increase significantly.

The main problem of this program that does not allow us to find good results, consists of a certain incompatibility of an exploratory model with an exploitative model. In our case, this model is clearly exploratory, since a very deep model would produce changes that are too small and would not be optimal. This incompatibility causes the program to not work properly.

Even if we had prioritized a dual model (with all the complexity that this entails) the results would not have been much better either, since a large amount of time would have been required again. This allows us to affirm that for this model a very painful process is necessary to obtain good results.

4 Conclusions

In this assignment, Genetic algorithm has been applied in order to obtain an good approximation of the parameters that can predict the evolution of the COVID-19 infectivitiy. The result found however has been so far from the real values and we haven't found successfully the result we expected.

During the process, the genetic algorithms and also the Range-Kutta computational calculation method have been implemented to find the values of the proposed ODEs in each step.

As we have previously commented, a good result is very expensive and it takes a very painful process to find this solution. Some alternatives such as the use of some Stepest-Descent algortihm could have helped to find a good result. However, this was not the main purpose of this release and at least the main objective of being able to implement a genetic algorthm has been achieved.

References

[1] Lluís Alsedà. *Genetic algorithms: A possible model for COVID-19*.

[2] Wikipedia: the free encyclopedia. *Genetic algorithm*, 2021. URL: `https://en.wikipedia.org/wiki/Genetic_algorithm`

[3] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2013. ISBN: 978-1-300-54962-8. URL: `http://cs.gmu.edu/~sean/book/metaheuristics/`