Illinois Institute of Technology

Final project

Parameter optimization using Synthetic and COVID-19 data

Ivan Gvozdanovic

supervised by
Dr. Charles Tier

September 3, 2023

Abstract

This project focuses on building an optimization algorithm with the goal of extrapolating unknown parameters from given data. The algorithm is first tested on a synthetic, random generated data to establish that the algorithm is valid. Next we use COVID-19 data[2] to expose the algorithm to a real world problem and tests its robustness. The parameters are the rate of infection, the rate of recovery and the total population. Knowing these parameters is crucial in estimating dimensionless quantity R_0 which can tell us if the virus and in turn the pandemic is dying out, or if the virus remains in the population. The algorithm is coded in Matlab, however it was translated from R programming language from a paper written by Madeline Dorr[1].

INTRODUCTION

To apply the desired algorithm, first we have to talk about the mathematical model. Namely, we are using a well known SIR model for modeling disease spread. The model consists of 3 compartments, S (susceptible), I (infected) and R (recovered). The rate of infection is represented by β and the rate of recovery is represented by γ. Figure 1 shows the visual aspect of the 3 compartment model where $b = \beta$ and $g = \gamma$.

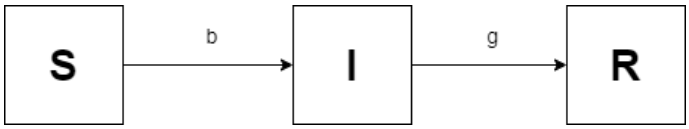


Figure 1: SIR model

Although only 2 parameters are shown in Figure 1 there is actually a 3rd parameter N which represents the total population. We need to include this parameter because the number of tested people in the COVID-19 data[2] represents a sample of the total population. Therefore, it would be misleading to think that the number of susceptible people is equal to the total population.
In order to find the desired parameters we need to establish a mathematical problem to work on. Thus, the following 3 differential equations mathematically describe the SIR model:

$$\frac{dS}{dt} = -\beta\frac{SI}{N} \tag{1}$$

$$\frac{dI}{dt} = \beta\frac{SI}{N} - \gamma I \tag{2}$$

$$\frac{dR}{dt} = \gamma I \tag{3}$$

Although we can solve this problem explicitly for S, I, R solutions, the parameters β and γ (and N) remain hidden in the solution as arbitrary constants. Therefore, we need to use the given data to retrieve them. To do this, we will use a Matlab optimization function **fminsearch** in order to minimize the residual

$$\sum (y(t_i) - y(\hat{t}_i))^2$$

where $y(t_i)$ is the actual data point for time t_i and $y(\hat{t}_i)$ is the approximation of the solution of the system of ODEs using some initial parameters β, γ, N. The smaller the output of the residual function, the better the approximation of the solution we have, and in turn we get more accurate parameters.

METHOD

In this section, we present the details of the algorithm and emphasize important sections of the code.
For our algorithm to work, we need to define the starting parameters (β_0, γ_0, N_0) and the initial starting point of the ODE system (S_0, I_0, R_0). Then we calculate values that the ODE system outputs using **DefEqn**, we solve the ODE system using **solutionFunction**, then compute the residual with **residual** function and finally the optimization function produces a new set of parameters for which the process is repeated. Figure 2 presents a sketch of the algorithm at hand. The reason why the parameter input branches off into the initial starting

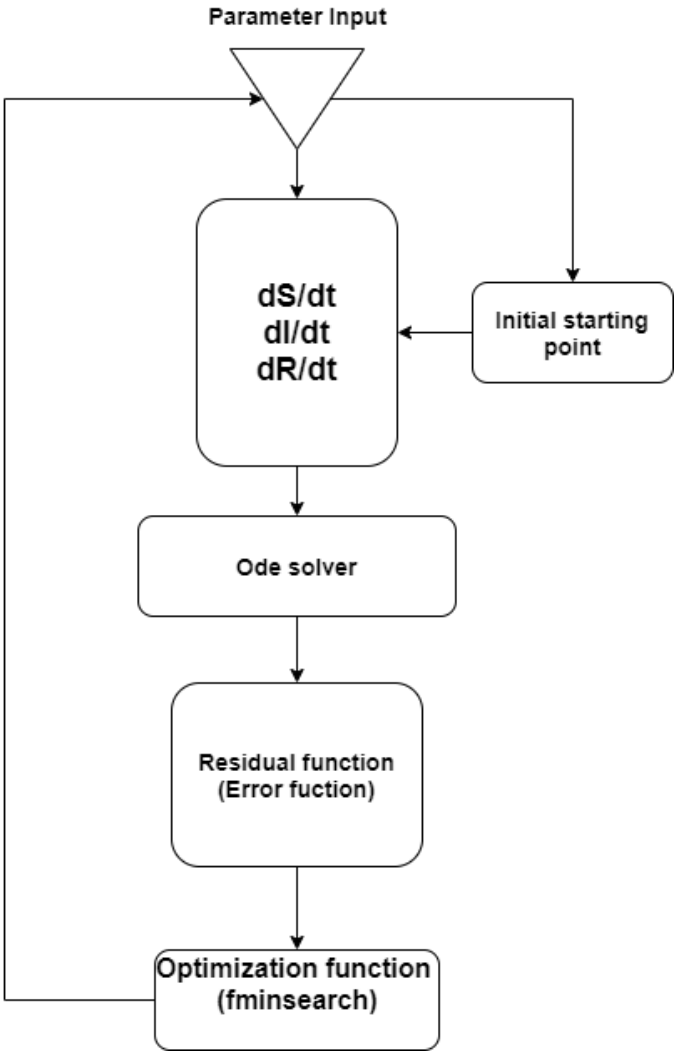


Figure 2: Optimization algorithm

point box above is that the parameter N is actually serving as S_0 for the ODE system. For every iteration, parameters β and γ are directly used as parameters in the ODE system, however, $N = S_0$ for each iteration. To better understand the reasoning behind this, assume we fix the starting point for each iteration i.e. $f_0 = (S_0, I_0, R_0)$. Although the algorithm will still work and the optimization will still happen, the residuals are going to be in the millions.

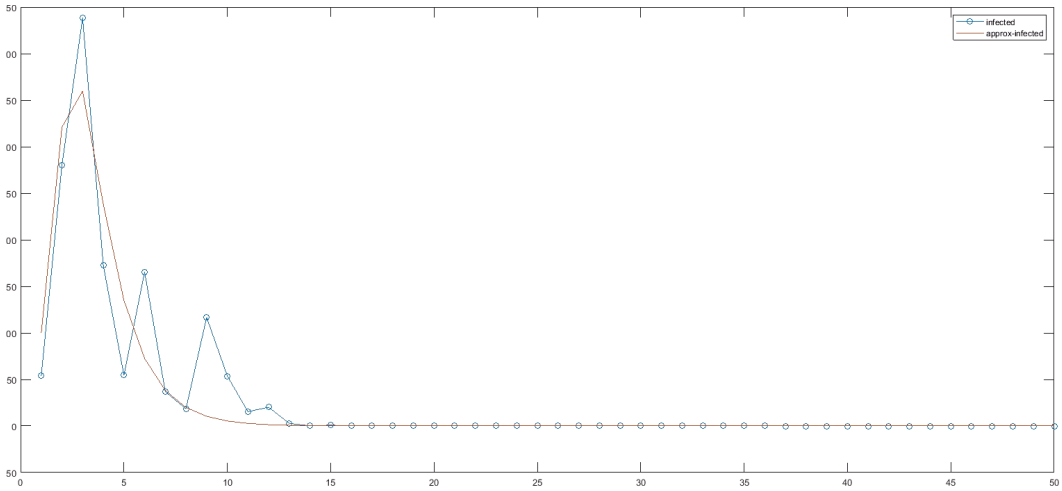
The reason is that theoretically S_0 represents the population we start with. Thus, if each iterative cycle we start of with $S_0 = 1000$ then the maximum number of infected people can only be 1000. However, the data tells us that the number of infected is much greater. Therefore, in order to fit the curve for data I, each iteration the initial coordinate S_0 needs to grow in order for the approximation I^* to grow as well. That is why we use parameter N as S_0 for each iteration and that is why the input splits of into the initial point box in Figure 2.

It is important to note that inside the **solutionFunction** we needed to relax the Relative Error of the **ode45** to e^{-5} tolerance. We do this for two reasons. Firstly we want to speed up the computation. Secondly, and more importantly, for a very smaller relative tolerance, the ode45 solver tends to crash due to division by numbers very close to zero for different times t. In fact, while testing, I noticed that the algorithm is very sensitive for the choices of initial parameters, relative tolerance and number of iterations in the **fminsearch** function.

RESULTS

To ensure that the algorithm is working properly, I tested the method using a synthetic data set. This data set is generated by solving the ODE system for some arbitrary initial parameters and initial starting point. Once we get the solution we add or multiply it by a noise generated using Matlab's **randn** function. After that the data we obtain is used as targets and the goal is to retrieve the parameters we picked using the algorithm. If we get the desired parameters, we can be reassured that the code and the logic is correct.
In testing, different range of noise was used to try and push the algorithm to its limits. Thus we obtain the following results for the synthetic data.

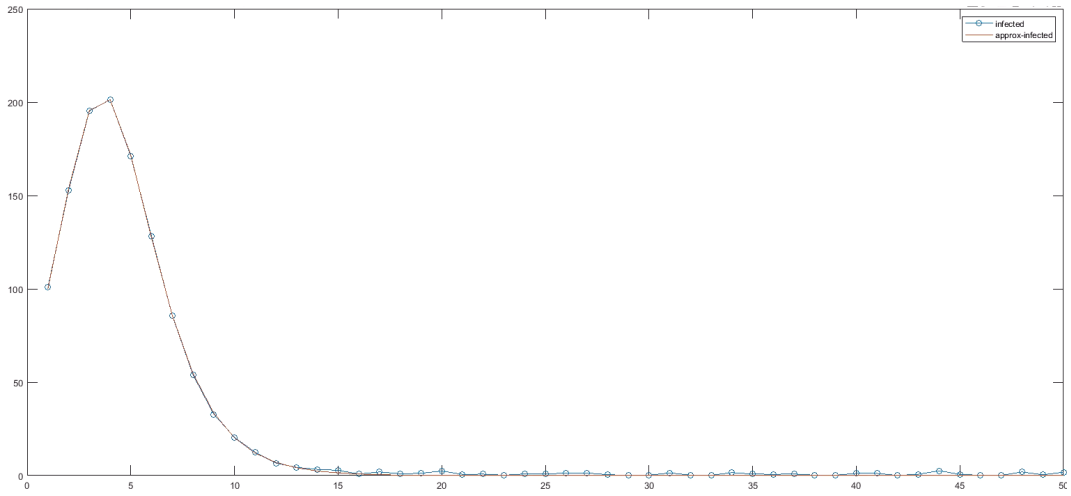
The desired target parameters are $\beta = 2, \gamma = 1.5, N = 3000$. For introducing randomness to the target data, we first coordinate-multiplied the actual solution with the noise using $(.*)$ operator. With these settings we obtained the following approximation. The approximated



parameters were $\beta = 2.3, \gamma = 0.7, N = 849$. I used this example to see how much oscillation can the algorithm take. Of course this data is not as smooth and the results above are to be

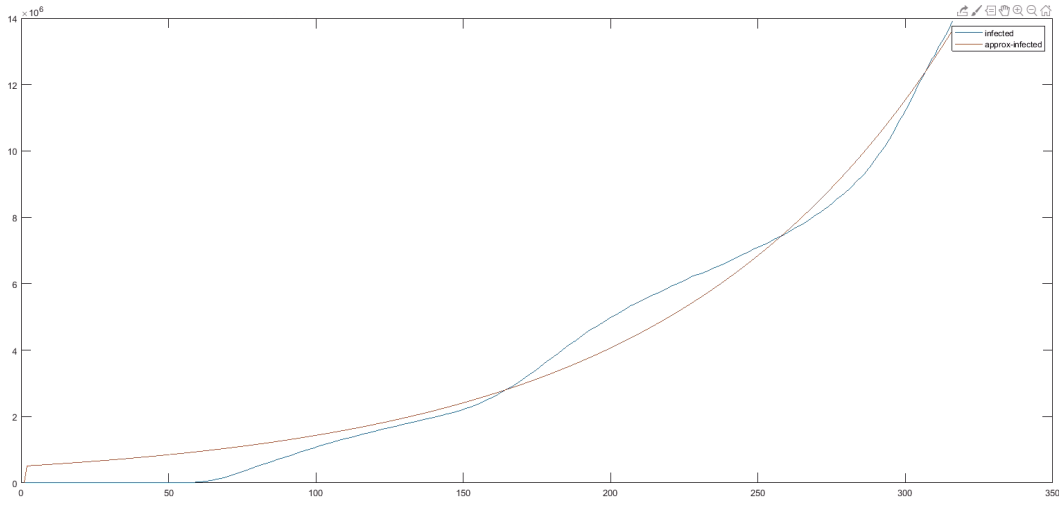
expected.

If we use the same target parameters and the same initial starting point but make the noise less oscillatory, then we get the following approximation. This time around, we get the ap-



proximations $\beta = 1.8, \gamma = 1.3, N = 2378$. Although not perfect, it is important to note that we only used 50 points to generate this data as well as only 100 iterations of the algorithm. With more points and more iterations, it is to be expected that the algorithm would produce even more precise results. Based on the graphs and output parameters, we can conclude that the algortihm is valid.

Next, we apply the algorithm to the COVID-19 data[2] set obtained thought the John Hopkins hospital. The data set is focused on the U.S. cases. The following graph shows the results. Although the approximation looks right, the issue is that the γ parameter is approximated



to a negative quantity. With multiple tries, I was not able to find good initial parameters, relative tolerances and number of iterations to get a better approximation of the parameters. As we can see, the algorithm is not perfect and there are cases, like the one above, where we cannot get a good approximation. Moreover, for bigger data sets, this approach takes a

long time to compute.

A possible improvement for the algorithm is potentially found in neural networks. They use a similar approach for optimization using gradient descent to minimize error. However, NNs are designed to take large matrices all at once and thus speed up the computation. Moreover, algorithm that use neural networks have very similar computations to that of a GPU. Therefore, utilizing the GPU, we can get much faster computational times. Currently, I do not have an exact structure of the algorithm that uses a neural network, however, this is something that is worth looking into in the future.

REFERENCES

1. Dorr, I. Madeline. (Spring 2019). Fitting the SIR Epidemiological Model To Influenza Data

2. John Hopkins Coronavirus Resource Center. Retrieved from https://github.com/datasets/covid-19

APPENDIX

Listing 1: MAIN FUNCTION

```
function main()

    %ORGANIZE THE DATA
    filename = 'dataset1.xlsx';
    data = readtable(filename);
    data = data(2:end,5:7);
    %disp(data(2:end,1));
    data = cat(2,data(2:end,1),data(2:end,3));
    toCELL = table2cell(data);
    DATA = str2double(toCELL);


    [t,S,I] = organizeData(DATA);


    global SS; %global data S
    global II; %global data I
    global time; %global time
    global targets; %global targets (data I)
    SS = S;
    II = I;
    time = t;
    targets = I;

    %INITIAL PARAMETERS
    initial_p = [1.1, 0.1,1000];

    %START THE OPTIMIZER
    P = findOptimalParameter(initial_p);
    %disp(RES(1,size(RES,2)));
    %disp(size(RES,2));

    %CHECK THE FIT
    F = solutionFunction(P);


    %GRAPH
    close all
    figure(2);
    plot(t,I);
    hold on
    plot(t,F(:,2));
    legend('infected','approx-infected');
```

```
44      end
```

Listing 2: ORGANIZE THE DATA

```
1     function [t,S,I] = organizeData(data)
2           total_ppl = zeros(size(data,1),1);
3           t = zeros(size(data,1),1);
4           I = zeros(size(data,1),1);
5           for i = 1:size(data,1)
6               t(i,1) = i;
7               total_ppl(i,1) = double(data(i,1));
8               %I(i,1) = double(data(i,1))*(double(data(i,2))/100);
9               I(i,1) = double(data(i,2));
10          end
11
12          S = total_ppl - I;
13      end
```

Listing 3: OPTIMIZER

```
1     %OPTIMIZER FUNCTION (USE fminsearch)
2      %Optimize the error of the apporximation
3      function P = findOptimalParameter(p)
4
5           options = optimset('Display', 'off','LargeScale',...
6                   'off','MaxIter',500);
7           pp = fminsearch(@residual,p,options);
8           format long
9           P = pp;
10          disp(P);
11     end
```

Listing 4: CALCULATE THE RESIDUAL

```
1     %CALCULATE THE RESIDUAL
2      function r = residual(params)
3           global targets;
4
5           %FIND APPOXIMATE SOLUTION
6           f_approx = solutionFunction(params);
7
8           %CALCULATE THE RESIDUAL
9           %r = sum(((f_approx(:,2) - targets).^2)./targets,'all');
10          r = norm(f_approx(:,2) - targets);
11
12     end
```

Listing 5: DEFINE ODE SYSTEM

```
%COMPUTE THE ODE SYSTEM.
 function [diff_eqn] = DefEqn(t,f,parameters)
     %BETA
     b = parameters(1);
     %GAMMA
     g = parameters(2);
     %TOTAL POPULATION
     N = parameters(3);

     S = f(1);
     I = f(2);
     R = f(3);

     %DEFINE THE SYSTEM OF ODEs
     diff_eqn = zeros(3,1);
     diff_eqn(1) = -b*((S*I))/N;
     diff_eqn(2) = b*((S*I)/N) - g*I;
     diff_eqn(3) = g*I;
 end
```

Listing 6: SOLVE THE ODE SYSTEM

```
%SOLVE THE SYSTEM OF ODEs FOR THE SIR MODEL
 function [F] = solutionFunction(params)

     global time;
     global II;

     %STARTING POINT
     f0 = [params(3),II(1,1),0];

     %SOLVE THE ODE SYSTEM
     options = odeset('RelTol',1e-5,'Stats','on','OutputFcn',@odeplot);
     [t, s] = ode45(@(t,f) DefEqn(t,f,params), time, f0,options);

     %RETURN THE SOLUTION OF THE SYSTEM
     F = s;

 end
```