SEIR Model
For Assesment of
COVID-19

A Report by
Divyansh & Himanshu

CSE | First Year

Under the guidance of Prof. S. Balakrishnan

Indian Institute of Technology Goa

SEIR Model
For Assessment Of
COVID-19

Divyansh | Himanshu

June 20, 2020

Preface

This report has been prepared by Divyansh and Himanshu for college project under school of Biology IIT Goa.

In addition to the project requirements [1] we have build here a generalised SEIR model. A lot of effort has been invested to make this report more advanced, informative, useful and easy to understand. All codes has been inserted into the report for better referencing while understanding the model

0.1 About the report

This report has been generated in LaTeX. All the original files of the functions has been included into the LaTeX file with the help of inbuilt packages provided by LaTeX and MATLAB. This LaTeX file has been created and edited in an online editor OverLeaf. LaTeX is an useful skill and can be learnt for free from official `https://www.latex-project.org/` or you can take this free course created by IIT Bombay on edex.
All the LaTeX files and helping files for this report is available at my GitHub Repository after 25^{th} June 2020.

0.2 MATLAB the language of choice

We have chosen MATLAB for this project because it is one of the easiest language to use for solving equations and fitting curves. MATLAB is also one of the best choices to perform linear regression which was the centre idea of this project MATLAB also provides a very useful features of Live Scripts. It helps to run code in parts and we can also write documentations ans text in addition to the code.
MATLAB also lets us export live scripts as well as functions as pdf files, html files or even as LaTeX files [2].

Several functions has been used to code the model to make codes more redable and reusable. A copy of all functions has been imported in chapter 4 and section 6.1 and the Live Scripts are being used to use that function for desired use.

MATLAB is a very useful programming language for mathematical analysis and can be learnt for free at Introduction to Programming with MATLAB course on Coursera Solving discrete differential equations, curve fitting, linear regression and all the mathematics as well as coding part can be learnt at ML through MATLAB on Coursera

0.3 Running the code

All the codes both functions [3] and live scripts [4] are available at my GitHub repository after 25^th June 2020.

[1] the requirements for the project was to model and fit a SIR model though liberty was provided to choose advanced models as well
[2] The codes in chapter 5 and section 6.2 are MATLAB live scripts directly imported
[3] functions are .m files in MATLAB
[4] live scripts are .mlx files in MATLAB

The code can be run in both offline and online version of MATLAB or possibly on Octave [5] To run the code download all the files or clone the GitHub repository in a a folder and open that folder in MATLAB then run any of the two main scripts. Run Example_Country.mlx for Countries [6] or Example_US_cities.mlx for States in US [7]

0.4 Links to important files in a place

- Whole Project files ie Report, LaTeX files and MATLAB files
- Codes (functions and scripts)
- Report LaTeX files
- Data Source John Hopkins University

[5] Octave is an open source software similar to MATLAB

[6] The country is preset to India change the name to desired country and chsnge the N_{pop} for its respective population

[7] The State is preset to Nebraska change it to desired State

Contents

Chapter 1

Introduction and Theory

1.1 About the pandemic

A novel coronavirus, formerly called 2019-nCoV, or SARS-CoV-2 by ICTV (severe acute respiratory syndrome coronavirus 2, by the International Committee on Taxonomy of Viruses) caused an outbreak of atypical pneumonia, now officially called COVID-19 by WHO (coronavirus disease 2019, by World Health Organization) first in Wuhan, Hubei province in Dec., 2019 and then rapidly spread out in the whole World

During this anti-epidemic battle, besides medical and biological research, theoretical studies based on either statistics or mathematical modeling may also play a non-negligible role in understanding the epidemic characteristics of the outbreak, in forecasting the inflection point and ending time, and in deciding the measures to curb the spreading.

For this purpose, in the early stage many efforts have been devoted to estimate key epidemic parameters, such as the basic reproduction number, doubling time and serial interval, in which the statistics models are mainly used. Due to the limitation of detection methods and restricted diagnostic criteria, asymptomatic or mild patients are possibly excluded from the confirmed cases. To this end, some methods have been proposed to estimate untraced contacts, undetected international cases, or the actual infected cases in Wuhan and Hubei province based on statistics models, or the epidemic outside Hubei province and overseas, With the improvement of clinic treatment of patients as well as more strict methods stepped up for containing the spread, many researchers investigate the effect of such changes by statistical reasoning and stochastic simulation.

Compared with statistics methods mathematical modeling based on dynamical equations receive relatively less attention, though they can provide more detailed mechanism for the epidemic dynamics. Among them, the classical susceptible exposed infectious recovered model (SEIR) is the most widely adopted one for characterizing the epidemic of COVID-19 outbreak in the world. Based on SEIR model, one can also assess the effectiveness of various measures since the outbreak which seems to be a difficult task for general statistics methods. As the dynamical model can reach interpretable conclusions on the outbreak, a cascade of SEIR models are developed to simulate the processes of transmission from infection source, hosts, reservoir to human. There are also notable gen eralizations of SEIR model for evaluation of the transmission risk and prediction of patient number, in which model, each group is divided into two subpopulations, the quarantined and unquarantined. The extension of classical SEIR model with delays is another routine to simulate the incubation period and the period before recovery. However, due to the lack of official data and the change of diagnostic caliber in the early stage of the outbreak, most early published models were either too complicated to avoid the overfitting problem, or the parameters were estimated based on limited and less accurate data, resulting in questionable predictions.

1.2 Model and Methods

To characterize the epidemic of COVID19 which outbroke in Wuhan at the end of 2019, we generalize the classical SEIR model by introducing seven different states, i.e. S(t),P(t),E(t),I(t),Q(t),R(t),D(t) denoting at time t the respective number of the susceptible

cases, insusceptible cases, exposed cases (infected but not yet be infectious, in a latent period), infectious cases (with infectious capacity and not yet be quarantined), quarantined cases (confirmed and infected), recovered cases and closed cases (or death). The adding of a new quarantined sate is driven by data, which together with the recovery state takes replace of the original R state in the classical SEIR model. Their relations are given in Fig. 1 and characterized by a group of ordinary differential equations (or difference equations if we consider discrete time, see SI). Constant $N = S+P+E+I+Q+R+D$ is the total population in a certain region. The coefficients $\{\alpha, \beta, \gamma^{-1}, \delta^{-1}, \lambda(t), \kappa(t)\}$ represent the protection rate, infection rate, average latent time, average quarantine time, cure rate, and mortality rate, separately. Especially, to take the improvement of public health into account, such as promoting wearing face masks, more effective contact tracing and more strict locking-down of communities, we assume that the susceptible population is stably decreasing and thus introduce a positive protection rate α into the model. In this case, the basic reproduction number becomes $BRN = \beta \times \delta^{-1} \times (1-\alpha)T$, T is the number of days. It is noted that here we assume the cure rate λ and the mortality rate κ are both time dependent. As confirmed in Fig. 2ad, the cure rate $\lambda(t)$ is gradually increasing with the time, while the mortality rate $\kappa(t)$ quickly decreases to less than 1% and becomes stabilized after Jan. 30th. This phenomenon is likely raised by the assistance of other emergency medical teams, the application of new drugs, etc. Furthermore, the average contact number of an infectious person is calculated in Fig and could provide some clue on the infection rate. It is clearly seen that the average contact number is basically stable over time, but shows a remarkable difference among various regions, which could be attributed to different quarantine policies and implements inside and outside Hubei (or Wuhan), since a less severe region is more likely to inquiry the close contacts of a confirmed case. A similar regional difference is observed for the severe condition rate too. In Fig. 2gh, Hubei and Wuhan overall show a much higher severe condition rate than Shanghai. Although it is generally expected that the patients need a period of time to become infectious, to be quarantined, or to be recovered from illness, but we do not find a strong evidence for the necessity of including time delay (see SI for more details). As a result, the time-delayed equations are not considered in the current work for simplicity.

1.2.1 Equations for Generalised SEIR model (Highlighted is for Classical SEIR model)

The seven equations [1]

$$\frac{dS(t)}{dt} = -\beta \frac{S(t)I(t)}{N} - \alpha S(t) \tag{1.1}$$

$$\frac{dE(t)}{dt} = \beta \frac{S(t)I(t)}{N} - \gamma E(t) \tag{1.2}$$

$$\frac{dI(t)}{dt} = \gamma S(t)I(t) - \delta I(t) \tag{1.3}$$

$$\frac{dQ(t)}{dt} = \delta I(t) - \lambda(t)Q(t) - \kappa(t)Q(t) \tag{1.4}$$

$$\frac{dR(t)}{dt} = \kappa(t)Q(t) \tag{1.5}$$

$$\frac{dD(t)}{dt} = \lambda(t)Q(t) \tag{1.6}$$

$$\frac{dP(t)}{dt} = \alpha S(t) \tag{1.7}$$

[1] https://arxiv.org/pdf/2002.06563.pdf

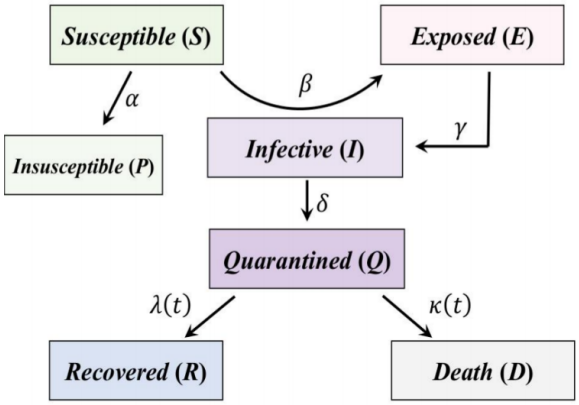


Figure 1.1

1.3 Parameter estimation

In most countries the cumulative numbers of quarantined cases, recovered cases and closed cases are available in public. However, since the latter two are directly related to the first one through the time dependent recovery rate and mortality rate, the numbers of quarantined cases Q(t) plays a key role in our modeling. A similar argument applies to the number of insusceptible cases too. Furthermore, as the accurate numbers of exposed cases and infectious cases are very hard to determine, they will be treated as hidden variables during the study. Leaving alone the time dependent parameters $\lambda(t)$ and $\kappa(t)$, there are four unknown coefficients$\{\alpha, \beta, \gamma^{-1}, \delta^{-1}\}$ and two initial conditions E0,I0 about the hidden variables (other initial conditions are known from the data) have to be extracted from the time series data Q(t). Such an optimization problem could be solved automatically by using the simulating annealing algorithm . A major difficulty is how to overcome the overfitting problem. To this end, we firstly prefix the latent time λ^{-1}, which is generally estimated within several days. And then for each fixed γ^{-1}, we explore its influence on other parameters ($\beta = 1$ nearly unchanged), initial values, as well as the population dynamics of quarantined cases and infected cases during best fitting. To produce the same outcome, the protection rate α and the reciprocal of the quarantine time δ^{-1} are both decreasing with the latent time γ^{-1}, which is consistent with the fact that longer latent time requires longer quarantine time. Meanwhile, the initial values of exposed cases and infectious cases are increasing with the latent time. Since E0 and I0 include asymptomatic patients, they both should be larger than the number of quarantined cases. Furthermore, as the time period between the starting date of our simulation and the initial outbreak of COVID-19 (generally believed to be earlier than Jan. 1st) is much longer than the latent time (3-6 days), E0 and I0 have to be close to each other, which makes only their sum E0+I0 matters during the fitting. An additional important finding is that in all cases β is always very close to 1, which agrees with the observation that COVID-19 has an extremely strong infectious ability. Nearly every unprotected person will be infected after a direct contact with the COVID-19 patients. As a summary, we conclude that once the latent time γ^{-1} is fixed, the fitting accuracy on the time series data $Q(t)$ basically depends on the values of α,δ^{-1} and $E0 + I0$. And based on a reasonable estimation on the total number of infected cases , the latent time is finally determined.

Chapter 2

The Model

2.1 Brief Background

A generalized SEIR model is used to simulate an epidemic breakout. Seven different states are considered in the following in a similar fashion as in chapter 1 , which is derived from SEIQR models image and has several similarities with mathematical models for SARS transmission,

1. Susceptible cases S(t)
2. Insusceptible cases P(t)
3. Exposed cases E(t)
4. Infectious cases I(t)
5. Quarantined cases Q(t)
6. Recovered cases R(t)
7. Dead cases D(t)

2.1.1 The Parameters are as follows:

α**:** protection rate

β**:** infection rate

γ**:** inverse of the average latent time

δ**:** rate at which infectious people enter in quarantine

λ**:** time-dependant recovery rate

κ**:** time-dependant mortality rate

The population is assumed constant, i.e. the births and natural death are not modelled. The cure rate and mortality rate are here time-dependent but they need some empirical coefficients to tune the time-dependency of these parameters.

The Generalised SEIR model is

Note: The colored part of the equations belong to the classical SEIR model

$$\frac{dS(t)}{dt} = -\beta \frac{S(t)I(t)}{N} - \alpha S(t)$$

$$\frac{dE(t)}{dt} = \beta \frac{S(t)I(t)}{N} - \gamma E(t)$$

$$\frac{dI(t)}{dt} = \gamma S(t)I(t) - \delta I(t)$$

$$\frac{dQ(t)}{dt} = \delta I(t) - \lambda(t)Q(t) - \kappa(t)Q(t)$$

$$\frac{dR(t)}{dt} = \kappa(t)Q(t)$$

$$\frac{dD(t)}{dt} = \lambda(t)Q(t)$$

$$\frac{dP(t)}{dt} = \alpha S(t)$$

2.1.2 Recovery and mortality rates

The mortaity rate $\kappa(t)$ is modeled as

$$\kappa(t) = \frac{\kappa_0}{e^{\kappa_1(t-\tau_\kappa)} + e^{\kappa_1(t-\tau_\kappa)}}$$

or as

$$\kappa(t) = \kappa_0 e^{-(\kappa_1(t-\tau_\kappa))^2}$$

or as

$$\kappa(t) = \kappa_0 + e^{-\kappa_1(t+\tau_\kappa)}$$

where κ_0, κ_1 and $\tau_\kappa$are parameters to be empirically determined. The parameters κ_0 and κ_1 have the dimension of the inverse of a time and τ_k has the dimension of a time.

The Recovery rate $\lambda(t)$ is either modelled as

$$\lambda(t) = \frac{\lambda_0}{1 + e^{-\lambda_1(t-\tau_\lambda)}}$$

or as

$$\lambda(t) = \lambda_0 + e^{-\lambda_1(t+\tau_\lambda}$$

where λ_0 , λ_1 and τ_λ are parameters to be empirically determined. The parameters λ_0 and λ_1 have the dimension of the inverse of a time and τ_λ has the dimension of a time.

The choice of best approximation for λ_t and κ_t is done automatically inside the function fit_SEIRQDP based on a preliminary assessment of the recovery rate and the mortality rate. The idea behind these functions is that the mortality rate should become close to zero (or a constant value κ_0) at time increases while the recovery rate converges toward a constant value λ_0 .

2.1.3 Numerical solutions

Re-Writing the system of ODEs in a matrix form for the sake of clarity.

$$\frac{dY}{dt} = A \times Y + F$$

where

$$Y = [S, E, I, Q, R, D, P]^T$$

$$A = \begin{bmatrix} -\alpha & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\gamma & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma & -\delta & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\kappa(t) - \lambda(t) & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda(t) & 0 & 0 & 0 \\ 0 & 0 & 0 & \kappa(t) & 0 & 0 & 0 \\ \alpha & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$F = S(t) \cdot I(t) \cdot \begin{bmatrix} -\frac{\beta}{N_p op} \\ \frac{\beta}{N_p op} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The equation $\frac{dY}{dt} = A \times Y + F$ is then solved using the classical 4th order Runge-Kutta method

Chapter 3

Coding the model (an overview through examples)

All the folowing code and every code in the report has been written in MATLAB(R2020a

Initialisation

Case of an imaginary epidemy outbreak that took place on 2010-01-01. The simulation time is set to 6 months.

```
clearvars;close all;clc;

% Time definition
dt = 0.1; % time step
time1 = datetime(2010,01,01,0,0,0):dt:datetime(2010,06,01,0,0,0);
N = numel(time1);
t = [0:N-1].*dt;
```

Generate the data

```
Npop= 60e6; % population (60 millions)
Q0 = 200; % Initial number of infectious that have bee quanrantined
I0 = Q0; % Initial number of infectious cases non-quarantined
E0 = 0; % Initial number of exposed
R0 = 10; % Initial number of recovereds
D0 = 10; % Initial number of deads
alpha = 0.08; %  protection rate
beta = 0.9; %  infection rate
gamma= 0.2; % inverse of average latent time
delta= 0.5; % rate at which infectious people enter in quarantine
Lambda = [0.01  0.1  100]; % cure rate (time dependant)
Kappa =  [0.001  0.01,60]; % mortality rate (time dependant)

% Choice of a particular form for lambda(t)
lambdaFun0 = @(a,t) a(1)./(1+exp(-a(2)*(t-a(3))));
kappaFun0 = @(a,t) a(1).*exp(-(a(2)*(t-a(3)))).^2);

[S,E,I,Q,R,D,P] = SEIQRDP(alpha,beta,gamma,delta,Lambda,Kappa,Npop,E0
    ,I0,Q0,R0,D0,t,lambdaFun0,kappaFun0);

figure
plot(t,lambdaFun0(Lambda,t),'b',t,kappaFun0(Kappa,t),'k')
legend('Recovery rate','mortality rate','location','best');
xlabel('Time (days)')
set(gcf,'color','w')
```

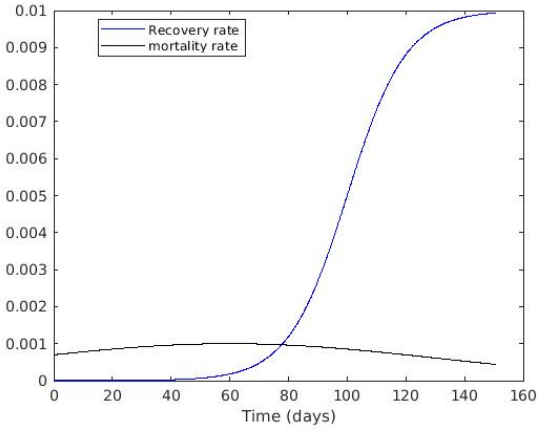


Figure 3.1: generating the data

Fit the data

The fitting is done using the time histories of the number of quarantined Q(t) , recovered R(t) and deads D(t) only. The number of exposed, susceptible, insusceptible and infectious is computed in the model but not used as target.

```
lambdaGuess = [0.01  0.5  0.1];
kappaGuess = [0.01  0.1,10];
alphaGuess = 0.05;
betaGuess = 0.7;
deltaGuess = 0.2;
gammaGuess = 0.3;
guess = [alphaGuess, betaGuess, deltaGuess, gammaGuess, lambdaGuess,
    kappaGuess]; % initial guess

[alpha1, beta1, gamma1, delta1, Lambda1, Kappa1, lambdaFun, kappaFun0] =
    fit_SEIQRDP(Q,R,D,Npop,E0,I0,time1,guess,'Display','off');
[S1,E1,I1,Q1,R1,D1,P1] =...
     SEIQRDP(alpha1,beta1,gamma1,delta1,Lambda1,Kappa1,Npop,E0,I0,Q0,
         R0,D0,t,lambdaFun,kappaFun0);
```

Comparison between fitted and generated time histories

```
figure
clf;close all;
plot(time1,Q,'r',time1,R,'c',time1,D,'g','linewidth',2);
hold on
plot(time1,Q1,'k-.',time1,R1,'k:',time1,D1,'k--','linewidth',2);
% ylim([0,1.1*Npop])
ylabel('Number of cases')
xlabel('Time (days)')
leg = {'Quarantined','Recovered','Dead','Fitted quarantined','Fitted
    recovered','Fitted Dead'};
legend(leg{:},'location','eastoutside')
set(gcf,'color','w')
axis tight
```

Case when recovered(R) and quarantined (Q) data are not available separately

The number of qarantined Q(t) and recovered cases R(t) is unknwon, However, $Q(t)+R(t)$ is known.

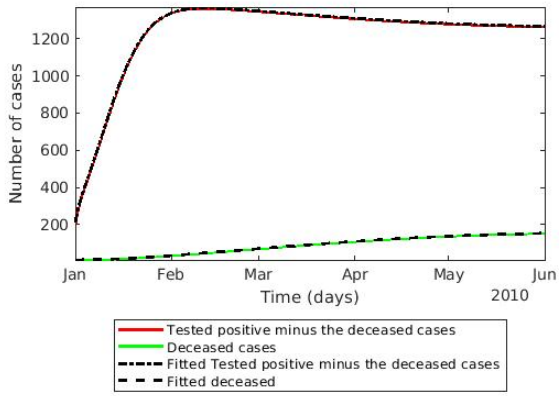


Figure 3.2: comparison between fitted and generated data

```
guess = [alphaGuess, betaGuess, deltaGuess, gammaGuess, lambdaGuess,
    kappaGuess]; % initial guess

[alpha1, beta1, gamma1, delta1, Lambda1, Kappa1, lambdaFun, kappaFun] = ...
    fit_SEIQRDP(Q+R,[],D,Npop,E0,I0,time1,guess,'Display','off');

[S1,E1,I1,Q1,R1,D1,P1] =...
    SEIQRDP(alpha1, beta1, gamma1, delta1, Lambda1, Kappa1, ...
    Npop,E0,I0,Q0,R0,D0,t,lambdaFun,kappaFun);

figure
clf;close all;
plot(time1,Q+R,'r',time1,D,'g','linewidth',2);
hold on
plot(time1,Q1+R1,'k-.',time1,D1,'k--','linewidth',2);
% ylim([0,1.1*Npop])
ylabel('Number of cases')
xlabel('Time (days)')
leg = {'Tested positive minus the deceased cases','Deceased cases','
    Fitted Tested positive minus the deceased cases','Fitted deceased'
    };
legend(leg{:},'location','southoutside')
set(gcf,'color','w')
axis tight
```

Chapter 4

Functions used

4.1 getDataCOVID function

This function just helps to get data from Johns Hopkins University and makes a temporary file containing all the data

```
function [tableConfirmed,tableDeaths,tableRecovered,time] =
    getDataCOVID()
% The function [tableConfirmed,tableDeaths,tableRecovered,time] =
    getDataCOVID
% collect the updated data from the COVID-19 epidemy from the
% John Hopkins university
%
%
%
% Author: Divyansh
%
% see also fit_SEIQRDP.m SEIQRDP.m
%% Options and names
Ndays = floor(now)-datenum(2020,01,22)-1; % minus one day because the
    data are updated with a delay of 24 h
opts = delimitedTextImportOptions("NumVariables", Ndays+5);
opts.VariableNames = ["ProvinceState", "CountryRegion", "Lat", "Long
    ", repmat("data",1,Ndays+1)];
opts.VariableTypes = ["string", "string", repmat("double",1,Ndays+3)
    ];
% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";
% Specify variable properties
%% Import the data
status = {'confirmed','deaths','recovered'};
address = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/
    master/csse_covid_19_data/csse_covid_19_time_series/';
ext = '.csv';
for ii=1:numel(status)

    filename = ['time_series_covid19_',status{ii},'_global'];
    fullName = [address,filename,ext];
%     disp(fullName)
    urlwrite(fullName,'dummy.csv');

    if strcmpi(status{ii},'Confirmed')
        tableConfirmed =readtable('dummy.csv', opts);

    elseif strcmpi(status{ii},'Deaths')
        tableDeaths =readtable('dummy.csv', opts);

```

```
37      elseif strcmpi(status{ii},'Recovered')
38          tableRecovered =readtable('dummy.csv', opts);
39      else
40          error('Unknown status')
41      end
42  end
43
44  time = datetime(2020,01,22):days(1):datetime(datestr(floor(now)))-
        datenum(1);
45
46  delete('dummy.csv')
47
48  end
```

4.2 fit_SEIQRDP function

This function uses the data obtained from getDataCOVId function and fits a general SEIR (SEIQRDP) model.

```
1  function [alpha1,beta1,gamma1,delta1,Lambda1,Kappa1,lambdaFun,
       kappaFun] = fit_SEIQRDP(Q,R,D,Npop,E0,I0,time,guess,varargin)
2  % [alpha1,beta1,gamma1,delta1,Lambda1,Kappa1,lambdaFun,varargout] =
3  % fit_SEIQRDP(Q,R,D,Npop,E0,I0,time,guess,varargin) estimates the
4  % parameters used in the SEIQRDP function, used to model the time-
       evolution
5  % of an epidemic outbreak.
6  %
7  % Input
8  %
9  %   Q: vector [1xN] of the target time-histories of the quarantined
       cases
10 %   R: vector [1xN] of the target time-histories of the recovered
       cases
11 %   D: vector [1xN] of the target time-histories of the dead cases
12 %   Npop: scalar: Total population of the sample
13 %   E0: scalar [1x1]: Initial number of exposed cases
14 %   I0: scalar [1x1]: Initial number of infectious cases
15 %   time: vector [1xN] of time (datetime)
16 %   guess: first vector [1x6] guess for the fit
17 %   optionals
18 %       -tolFun: tolerance  option for optimset
19 %       -tolX: tolerance  option for optimset
20 %       -Display: Display option for optimset
21 %       -dt: time step for the fitting function
22 %
23 % Output
24 %
25 %   alpha: scalar [1x1]: fitted protection rate
26 %   beta: scalar [1x1]: fitted  infection rate
27 %   gamma: scalar [1x1]: fitted  Inverse of the average latent time
28 %   delta: scalar [1x1]: fitted  rate at which people enter in
       quarantine
29 %   lambda: scalar [1x1]: fitted  cure rate
30 %   kappa: scalar [1x1]: fitted  mortality rate
31 %   lambdaFun: anonymous function giving the time-dependant recovery
       rate
32 %   kappaFun: anonymous function giving the time-dependant death rate
33 %
34 % Author: E. Cheynet - UiB - last modified 23-05-2020
35 %
36 % see also SEIQRDP.m
```

```
37
38 %% Inputparseer
39 p = inputParser();
40 p.CaseSensitive = false;
41 p.addOptional('tolX',1e-5);  %  option for optimset
42 p.addOptional('tolFun',1e-5);  %  option for optimset
43 p.addOptional('Display','iter'); % Display option for optimset
44 p.addOptional('dt',0.1); % time step for the fitting
45 p.parse(varargin{:});
46 %%%%%%%%%%%%%%%%%%%%%%%%%%
47 tolX = p.Results.tolX ;
48 tolFun = p.Results.tolFun ;
49 Display  = p.Results.Display ;
50 dt  = p.Results.dt ;
51
52 %% Options for lsqcurvfit
53 options=optimset('TolX',tolX,'TolFun',tolFun,...
54     'MaxFunEvals',1200,'Display',Display);
55 %% Initial conditions and basic checks
56
57 % Write the target input into a matrix
58 Q(Q<0)=0; % negative values are not possible
59 R(R<0)=0; % negative values are not possible
60 D(D<0)=0; % negative values are not possible
61
62 if isempty(R)
63     warning(' No data available for "Recovered" ')
64     input = [Q;D]; % In this aprticular case, Q is actually the
           number of active + recovered cases
65 else
66     input = [Q;R;D];
67 end
68
69 if size(time,1)>size(time,2) && size(time,2)==1,     time = time';end
70 if size(time,1)>1 && size(time,2)>1,  error('Time should be a vector'
    );end
71
72 %% Definition of the new, refined, time vector for the numerical
    solution
73 fs = 1./dt;
74 tTarget = round(datenum(time-time(1))*fs)/fs; % Number of days with
    one decimal
75 t = tTarget(1):dt:tTarget(end); % oversample to ensure that the
    algorithm converges
76
77 %% Preliminary fitting
78 % Decide which function to use for lambda and get first estimate of
    lambda
79 %  Preliminary fitting for lambda to find the best approximation
80 %  The final fitting is done considering simulatneously the different
81 %  parameters since the equations are coupled
82
83 if ~isempty(R) % If there exists information on the recovered cases
84     [guess,lambdaFun] = getLambdaFun(tTarget,Q,R,guess);
85 else
86     lambdaFun =  @(a,t) a(1)./(1+exp(-a(2)*(t-a(3)))); % default
           function
87 end
88
89 % Get a first estimate of kappa
90 try
```

```
91  [guess,kappaFun] = getKappaFun(tTarget,Q,D,guess);
92  catch exception
93      warning('Failure to fit the death rate. A poor fit is expected!')
            ;
94  end
95
96  %% Main fitting
97
98  modelFun1 = @SEIQRDP_for_fitting; % transform a nested function into
        anonymous function
99
100 if isempty(R)
101     % Significantly constraint the death rate
102     kappaMax = guess(8:10)*1.05; % Constrain the guess if no R
            available
103     kappaMin = guess(8:10)*0.95; % Constrain the guess if no R
            available
104     lambdaMax = [1 1 100]; % bound the guess around the initial fit
105     lambdaMin = [0 0 0]; % bound the guess around the initial fit
106 else
107     kappaMax = guess(8:10)*3; % bound the guess around the initial
            fit
108     kappaMin = guess(8:10)/3; % bound the guess around the initial
            fit
109     lambdaMax = guess(5:7)*3; % bound the guess around the initial
            fit
110     lambdaMin = guess(5:7)/3; % bound the guess around the initial
            fit
111
112     if lambdaMax(3)<1e-2
113         lambdaMax(3) = 100;
114         lambdaMin(3) = 0;
115     end
116 end
117 ub = [1, 5, 1, 1, lambdaMax, kappaMax]; % upper bound of the
        parameters
118 lb = [0, 0, 0, 0, lambdaMin, kappaMin]; % lower bound of the
        parameters
119 % call Lsqcurvefit
120 [Coeff] = lsqcurvefit(@(para,t) modelFun1(para,t),...
121     guess,tTarget(:)',input,lb,ub,options);
122
123
124 %% Write the fitted coeff in the outputs
125 alpha1 = abs(Coeff(1));
126 beta1 = abs(Coeff(2));
127 gamma1 = abs(Coeff(3));
128 delta1 = abs(Coeff(4));
129 Lambda1 = abs(Coeff(5:7));
130 Kappa1 = abs(Coeff(8:10));
131
132 %% nested functions
133
134     function [output] = SEIQRDP_for_fitting(para,t0)
135
136         % I simply rename the inputs
137         alpha = abs(para(1));
138         beta = abs(para(2));
139         gamma = abs(para(3));
140         delta = abs(para(4));
141         lambda0 = abs(para(5:7));
```

```
142         kappa0 = abs(para(8:10));
143
144
145         %% Initial conditions
146         N = numel(t);
147         Y = zeros(7,N); %  There are seven different states
148         Y(2,1) = E0;
149         Y(3,1) = I0;
150         Y(4,1) = Q(1);
151         if ~isempty(R)
152             Y(5,1) = R(1);
153             Y(1,1) = Npop-Q(1)-R(1)-D(1)-E0-I0;
154         else
155             Y(1,1) = Npop-Q(1)-D(1)-E0-I0;
156         end
157         Y(6,1) = D(1);
158
159         if round(sum(Y(:,1))-Npop)~=0
160             error(['the sum must be zero because the total population
                    ',...
161                 ' (including the deads) is assumed constant']);
162         end
163         %%
164         modelFun = @(Y,A,F) A*Y + F;
165         lambda = lambdaFun(lambda0,t);
166         kappa = kappaFun(kappa0,t);
167
168         % Very large recovery rate should not occur but can lead to
169         % numerical errors.
170         if lambda>10, warning('lambda is abnormally high'); end
171
172         % ODE resolution
173         for ii=1:N-1
174             A = getA(alpha,gamma,delta,lambda(ii),kappa(ii));
175             SI = Y(1,ii)*Y(3,ii);
176             F = zeros(7,1);
177             F(1:2,1) = [-beta/Npop;beta/Npop].*SI;
178             Y(:,ii+1) = RK4(modelFun,Y(:,ii),A,F,dt);
179         end
180
181         Q1 = Y(4,1:N);
182         R1 = Y(5,1:N);
183         D1 = Y(6,1:N);
184
185         Q1 = interp1(t,Q1,t0);
186         R1 = interp1(t,R1,t0);
187         D1 = interp1(t,D1,t0);
188         if ~isempty(R)
189             output = ([Q1;R1;D1]);
190         else
191             output = ([Q1+R1;D1]);
192         end
193
194     end
195     function [A] = getA(alpha,gamma,delta,lambda,kappa)
196         %  [A] = getA(alpha,gamma,delta,lambda,kappa) computes the
                matrix A
197         %  that is found in: dY/dt = A*Y + F
198         %
199         %   Inputs:
200         %   alpha: scalar [1x1]: protection rate
```

```
201         %   beta: scalar [1x1]: infection rate
202         %   gamma: scalar [1x1]: Inverse of the average latent time
203         %   delta: scalar [1x1]: rate of people entering in
                quarantine
204         %   lambda: scalar [1x1]: cure rate
205         %   kappa: scalar [1x1]: mortality rate
206         %   Output:
207         %   A: matrix: [7x7]
208
209         A = zeros(7);
210         % S
211         A(1,1) = -alpha;
212         % E
213         A(2,2) = -gamma;
214         % I
215         A(3,2:3) = [gamma,-delta];
216         % Q
217         A(4,3:4) = [delta,-kappa-lambda];
218         % R
219         A(5,4) = lambda;
220         % D
221         A(6,4) = kappa;
222         % P
223         A(7,1) = alpha;
224
225     end
226     function [Y] = RK4(Fun,Y,A,F,dt)
227         % NUmerical trick: the parameters are assumed constant
                between
228         % two time steps.
229
230         % Runge-Kutta of order 4
231         k_1 = Fun(Y,A,F);
232         k_2 = Fun(Y+0.5*dt*k_1,A,F);
233         k_3 = Fun(Y+0.5*dt*k_2,A,F);
234         k_4 = Fun(Y+k_3*dt,A,F);
235         % output
236         Y = Y + (1/6)*(k_1+2*k_2+2*k_3+k_4)*dt;
237     end
238
239     function [guess,kappaFun] = getKappaFun(tTarget,Q,D,guess)
240         % [guess,kappaFun] = getKappaFun(tTarget,Q,D,guess) provides
                a first
241         % estimate of the  death rate, to faciliate convergence of
                the main
242         % algorithm.
243         %
244         % Input:
245         %
246         % tTarget: vector [1xN]: time as double
247         % Q: vector [1xN] of the target time-histories of the
                quarantined cases
248         % D: vector [1xN] of the target time-histories of the dead
                cases
249         % guess: vector [1x9] Initial guess for kappa
250         %
251         % Output
252         % guess: vector [1x9]  Updated intial guess
253         % kappaFun: Empirical fucntion for the death rate
254
255         % If less than 20 reported deceased, the death rate won't be
```

```
256         % reliable. Therefore, no preliminary fitting is done.
257         if max(D)<10
258             kappaFun = @(a,t) a(1)./(exp(a(2)*(t-a(3))) + exp(-a(2)*(
                    t-a(3))));
259         else
260
261             try
262                 opt=optimset('TolX',1e-6,'TolFun',1e-6,'Display','off
                        ');
263
264 %                   myFun1 = @(a,t) a(1).*exp(-a(2)*(t+(a(3))));
265
266                 myFun1 = @(a,t) a(1)./(exp(a(2)*(t-a(3))) + exp(-a(2)
                        *(t-a(3))));
267                 myFun2 = @(a,t) a(1).*exp(-(a(2)*(t-a(3))).^2);
268                 myFun3 = @(a,t) a(1) + exp(-a(2)*(t+a(3)));
269
270                 rate = (diff(D)./median(diff(tTarget(:)))./Q(2:end);
271                 x = tTarget(2:end);
272
273                 % A death rate larger than 3 is abnormally high. It
                        is not
274                 % used for the fitting.
275                 rate(abs(rate)>3)=nan;
276                 [coeff1,r1] = lsqcurvefit(@(para,t) myFun1(para,t)
                        ,...
277                     guess(8:10),x(~isnan(rate)),rate(~isnan(rate)),[0
                            0 0],[1 1 100],opt);
278                 [coeff2,r2] = lsqcurvefit(@(para,t) myFun2(para,t)
                        ,...
279                     guess(8:10),x(~isnan(rate)),rate(~isnan(rate)),[0
                            0 0],[1 1 100],opt);
280                 [coeff3,r3] = lsqcurvefit(@(para,t) myFun3(para,t)
                        ,...
281                     guess(8:10),x(~isnan(rate)),rate(~isnan(rate)),[0
                            0 0],[1 1 100],opt);
282 %
283 %                   figure;plot(x,rate,x,myFun1(coeff1,x),'r',x,myFun2
    (coeff2,x),'g',x,myFun3(coeff3,x),'b')
284
285                 minR = min([r1,r2,r3]);
286                 if r1==minR
287                     kappaGuess = coeff1;
288                     kappaFun = myFun1;
289                 elseif r2==minR
290                     kappaGuess = coeff2;
291                     kappaFun = myFun2;
292                 elseif r3==minR
293                     kappaFun = myFun3;
294                     kappaGuess = coeff3;
295                 end
296
297                 guess(8:10) = kappaGuess; % update guess
298
299             catch exceptionK
300                 disp(exceptionK)
301                 kappaFun = @(a,t) a(1)./(exp(a(2)*(t-a(3))) + exp(-a
                        (2)*(t-a(3))));
302             end
303
304         end
```

```
305     end
306
307     function [guess,lambdaFun] = getLambdaFun(tTarget,Q,R,guess)
308         %  [guess,lambdaFun] = getLambdaFun(tTarget,Q,R,guess)
                provides a first
309         % estimate of the  death rate, to faciliate convergence of
                the main
310         % algorithm.
311         %
312         % Input:
313         %
314         % tTarget: vector [1xN]: time as double
315         % Q: vector [1xN] of the target time-histories of the
                quarantined cases
316         % R: vector [1xN] of the target time-histories of the
                recovered cases
317         % guess: vector [1x9] Initial guess for kappa
318         %
319         % Output
320         % guess: vector [1x9]  Updated intial guess
321         % lambdaFun: Empirical fucntion for the recovery rate
322
323         % If less than 20 reported deceased, the death rate won't be
324         % reliable. Therefore, no preliminary fitting is done.
325
326         % If less than 20 reported recovered, the death rate won't be
327         % reliable. Therefore, no preliminary fitting is done.
328         if max(R)<20
329             lambdaFun =  @(a,t) a(1)./(1+exp(-a(2)*(t-a(3))));
330         else
331
332             try
333
334                 opt=optimset('TolX',1e-6,'TolFun',1e-6,'Display','off
                        ');
335
336                 % Two empirical functions are evaluated
337                 myFun1 = @(a,t) a(1)./(1+exp(-a(2)*(t-a(3))));
338                 myFun2 = @(a,t) a(1) + exp(-a(2)*(t+a(3)));
339
340                 % Compute the recovery rate from the data (noisy data
                        )
341                 rate = diff(R)./median(diff(tTarget(:)))./Q(2:end);
342                 x = tTarget(2:end);
343
344                 % A daily rate larger than one is abnormally high. It
                         is not
345                 % used for the fitting. A daily recovered rate of
                        zero is
346                 % either abnormally low or reflects an insufficient
                        number
347                 % of recovered cases. It is not used either for the
                        fitting
348                 rate(abs(rate)>1|abs(rate)==0)=nan;
349
350                 [coeff1,r1] = lsqcurvefit(@(para,t) myFun1(para,t)
                        ,...
351                     guess(5:7),x(~isnan(rate)),rate(~isnan(rate)),[0
                            0 0],[1 1 100],opt);
352                 [coeff2,r2] = lsqcurvefit(@(para,t) myFun2(para,t)
                        ,...
```

```
353                         guess(5:7),x(~isnan(rate)),rate(~isnan(rate)),[0
                                0  0],[1  1  100],opt);
354
355
356 %                       figure;plot(x,rate,x,myFun1(coeff1,x),'r',x,myFun2
        (coeff2,x),'g--')
357
358                     % myFun1 is more stable on a long term persepective
359 %                   % If coeff2 have reached the upper boundaries, myFUn1
         is
360 %                   chosen
361                     if r1<r2 || coeff2(1)>0.99 || coeff2(2)>4.9
362                         lambdaGuess = coeff1;
363                         lambdaFun = myFun1;
364                     else
365                         lambdaGuess = coeff2;
366                         lambdaFun = myFun2;
367                     end
368                     guess(5:7) = lambdaGuess; % update guess
369
370
371                 catch exceptionL
372                     disp(exceptionL)
373                     lambdaFun =  @(a,t) a(1)./(1+exp(-a(2)*(t-a(3))));
374                 end
375             end
376         end
377
378
379 end
```

4.3 SEIQRDP function

This function generates a generalised SEIR (SEIQRDP) model using the parameters generated by function fit_SEIQRDP after fitting.

```
1  function [S,E,I,Q,R,D,P] = SEIQRDP(alpha,beta,gamma,delta,lambda0,
       kappa0,Npop,E0,I0,Q0,R0,D0,t,lambdaFun,kappaFun)
2  % [S,E,I,Q,R,D,P] = SEIQRDP(alpha,beta,gamma,delta,lambda,kappa,Npop,
       E0,I0,R0,D0,t,lambdaFun)
3  % simulate the time-histories of an epidemic outbreak using a
       generalized
4  % SEIR model.
5  %
6  % Input
7  %
8  %   alpha: scalar [1x1]: fitted protection rate
9  %   beta: scalar [1x1]: fitted  infection rate
10 %   gamma: scalar [1x1]: fitted  Inverse of the average latent time
11 %   delta: scalar [1x1]: fitted  rate at which people enter in
       quarantine
12 %   lambda: scalar [1x1]: fitted  cure rate
13 %   kappa: scalar [1x1]: fitted  mortality rate
14 %   Npop: scalar: Total population of the sample
15 %   E0: scalar [1x1]: Initial number of exposed cases
16 %   I0: scalar [1x1]: Initial number of infectious cases
17 %   Q0: scalar [1x1]: Initial number of quarantined cases
18 %   R0: scalar [1x1]: Initial number of recovered cases
19 %   D0: scalar [1x1]: Initial number of dead cases
20 %   t: vector [1xN] of time (double; it cannot be a datetime)
```

```
21 %   lambdaFun: anonymous function giving the time-dependant recovery
       rate
22 %   kappaFun: anonymous function giving the time-dependant death rate
23 %
24 % Output
25 %   S: vector [1xN] of the target time-histories of the susceptible
       cases
26 %   E: vector [1xN] of the target time-histories of the exposed cases
27 %   I: vector [1xN] of the target time-histories of the infectious
       cases
28 %   Q: vector [1xN] of the target time-histories of the
       quarantinedcases
29 %   R: vector [1xN] of the target time-histories of the recovered
       cases
30 %   D: vector [1xN] of the target time-histories of the dead cases
31 %   P: vector [1xN] of the target time-histories of the insusceptible
        cases
32 %
33 % Author:Divyansh
34 %
35 % see also fit_SEIQRDP.m
36
37 %% Initial conditions
38 N = numel(t);
39 Y = zeros(7,N);
40 Y(1,1) = Npop-Q0-E0-R0-D0-I0;
41 Y(2,1) = E0;
42 Y(3,1) = I0;
43 Y(4,1) = Q0;
44 Y(5,1) = R0;
45 Y(6,1) = D0;
46
47 if round(sum(Y(:,1))-Npop)~=0
48     error(['the sum must be zero because the total population',...
49         ' (including the deads) is assumed constant']);
50 end
51 %% Computes the seven states
52 modelFun = @(Y,A,F) A*Y + F;
53 dt = median(diff(t));
54
55 lambda = lambdaFun(lambda0,t);
56 kappa = kappaFun(kappa0,t);
57
58 % ODE resolution
59
60 for ii=1:N-1
61     A = getA(alpha,gamma,delta,lambda(ii),kappa(ii));
62     SI = Y(1,ii)*Y(3,ii);
63     F = zeros(7,1);
64     F(1:2,1) = [-beta/Npop;beta/Npop].*SI;
65     Y(:,ii+1) = RK4(modelFun,Y(:,ii),A,F,dt);
66 end
67
68 % Y = round(Y);
69 %% Write the outputs
70 S = Y(1,1:N);
71 E = Y(2,1:N);
72 I = Y(3,1:N);
73 Q = Y(4,1:N);
74 R = Y(5,1:N);
75 D = Y(6,1:N);
```

```
76  P = Y(7,1:N);
77
78
79  %% Nested functions
80      function [A] = getA(alpha,gamma,delta,lambda,kappa)
81          %  [A] = getA(alpha,gamma,delta,lambda,kappa) computes the
                matrix A
82          %  that is found in: dY/dt = A*Y + F
83          %
84          %   Inputs:
85          %   alpha: scalar [1x1]: protection rate
86          %   beta: scalar [1x1]: infection rate
87          %   gamma: scalar [1x1]: Inverse of the average latent time
88          %   delta: scalar [1x1]: rate of people entering in
                quarantine
89          %   lambda: scalar [1x1]: cure rate
90          %   kappa: scalar [1x1]: mortality rate
91          %   Output:
92          %   A: matrix: [7x7]
93          A = zeros(7);
94          % S
95          A(1,1) = -alpha;
96          % E
97          A(2,2) = -gamma;
98          % I
99          A(3,2:3) = [gamma,-delta];
100         % Q
101         A(4,3:4) = [delta,-kappa-lambda];
102         % R
103         A(5,4) = lambda;
104         % D
105         A(6,4) = kappa;
106         % P
107         A(7,1) = alpha;
108     end
109     function [Y] = RK4(Fun,Y,A,F,dt)
110         % Runge-Kutta of order 4
111         k_1 = Fun(Y,A,F);
112         k_2 = Fun(Y+0.5*dt*k_1,A,F);
113         k_3 = Fun(Y+0.5*dt*k_2,A,F);
114         k_4 = Fun(Y+k_3*dt,A,F);
115         % output
116         Y = Y + (1/6)*(k_1+2*k_2+2*k_3+k_4)*dt;
117     end
118 end
```

4.4 checkRates function

This function re-verifies the obtained data from fitting and from simulation.

```
1  function checkRates(time,Q,R,D,kappaFun,lambdaFun,kappa,lambda)
2  % function checkRates(time,Q,D,R,kappaFun,lambdaFun) compares the
       fitted
3  % and calcualted death and recovered ratios. The idea is to check
       whether
4  % the approximation of these ratios is appropriate
5  %
6  % Inputs
7  %   time: datetime: [1xN]: time array
8  %   Q: double [1xN]: Time histories of the quarantined/active cases :
9  %   D: double [1xN]: Time histories of the deceased cases :
```

```
%   R: double [1xN]: Time histories of the recovered cases :
%   kappaFun: anonymous function approximating the death rate
%   lambdaFun: anonymous function approximating the recovery rate
%
% Outputs:
% None
%
% Author: Divyansh
%
% see also SEIQRDP.m fit_SEIQRDP.m

%% Compute the rate of deceased and recovered cases

Q = Q(:);
R = R(:);
D = D(:);
time = time(:);

rateD = (diff(D)./diff(datenum(time-time(1))))./Q(2:end);
rateD(abs(rateD)>3) = nan; % remove bovious outliers

if ~isempty(R)
    rateR = (diff(R)./diff(datenum(time-time(1))))./Q(2:end);
    rateR(abs(rateR)>3) = nan;
end
%% Define the time
x = datenum(time(2:end)-time(1));
x1 = x(1):1/24:x(end);

%% Compare the fitted and effective rates

if ~isempty(R)
    figure;
    subplot(121)
    title('Death rate')
    plot(x,rateD,'k*',x1,kappaFun(kappa,x1),'r')
    xlabel('Time (days)')
    ylabel('Death rate (day^{-1})')
    axis tight
    legend('Measured','Fitted')


    subplot(122)
    title('Recovery rate')

    plot(x,rateR,'b*',x1,lambdaFun(lambda,x1),'r')
    axis tight
    set(gcf,'color','w')
    xlabel('Time (days)')
    ylabel('Recovery rate (day^{-1})')
    legend('Measured','Fitted')
else
    figure;
    plot(x,rateD,'k*',x1,kappaFun(kappa,x1),'r')
    xlabel('Time (days)')
    ylabel('Pseudo-death rate (day^{-1})')
    axis tight
    legend('Measured','Fitted')
end
end
```

Chapter 5

Running the model for India

Now for running the model foe India, a MATLAB live script has been used.
A pdf copy of these script has been attached on the following pages.

Read the script carefully and go through how each functions mentioned in previous chapter has been used.

The script has been divided into sections so each section can be executed indivisually to see the outputs for each step.

NOTES

This script is Run for India but can be changed for other countries/State/Province whos data is available in the mentioned source

Changing the Location Field will make the script run for desired Location also do not forget to change the (Population) N_pop field also as the code will generate wrong outputs.

The default projection timespan is for 20 days from the current date, this field can be changed in SEIQRDP function to desired number of days.

MATLAB Files of all the functions

Example: COVID-2019 data for India

I am taking some data from John Hopkins university [1]

https://github.com/CSSEGISandData/COVID-19

Initialisation

The parameters are here taken as constant except the death rate and the cure rate.

```
clearvars;close all;clc;
% Download the data from ref [1] and read them with the function getDataCOVID
[tableConfirmed,tableDeaths,tableRecovered,time] = getDataCOVID();
% time = time(1:end-1);
fprintf(['Most recent update: ',datestr(time(end)),'\n'])
```

```
Most recent update: 19-Jun-2020
```

```
Location = 'India';
% Version 4.8 and above have an additional table conditions (thank to Aleks Czernicki)
% For more details, see: https://github.com/ECheynet/SEIR/pull/12
try
    indR = find(contains(tableRecovered.CountryRegion,Location)==1 & (tableRecovered.ProvinceSt
    indC = find(contains(tableConfirmed.CountryRegion,Location)==1 & (tableConfirmed.ProvinceSt
    indD = find(contains(tableDeaths.CountryRegion,Location)==1 & (tableDeaths.ProvinceState.is
catch exception
    searchLoc = strfind(tableRecovered.CountryRegion,Location);
    indR = find(~cellfun(@isempty,searchLoc)) ;

    searchLoc = strfind(tableConfirmed.CountryRegion,Location);
    indC = find(~cellfun(@isempty,searchLoc)) ;

    searchLoc = strfind(tableDeaths.CountryRegion,Location);
    indD = find(~cellfun(@isempty,searchLoc));
end
```

```
indR = 127
indC = 133
indD = 133
```

```
disp(tableRecovered(indR,1:2))
```

```
    ProvinceState    CountryRegion
    -------------    -------------

      <missing>          "India"
```

```
disp(tableConfirmed(indC,1:2))
```

```
    ProvinceState    CountryRegion
    -------------    -------------

      <missing>          "India"
```

```
disp(tableDeaths(indD,1:2))
```

```
    ProvinceState    CountryRegion
    -------------    -------------

      <missing>          "India"
```

```
indR = indR(1);
indD = indD(1);
indC = indC(1);

Recovered = table2array(tableRecovered(indR,5:end));
Deaths = table2array(tableDeaths(indD,5:end));
Confirmed = table2array(tableConfirmed(indC,5:end));
% If the number of confirmed cases is small, it is difficult to know whether
% the quarantine has been rigorously applied or not. In addition, this
% suggests that the number of infectious is much larger than the number of
% confirmed cases
minNum= round(0.25*max(Confirmed));
indRemoved = unique([find(Confirmed<=minNum),find(isnan(Confirmed))]);
Recovered(indRemoved)=[];
```

```
Deaths(indRemoved)=[];
time(indRemoved)= [];
Confirmed(indRemoved)=[];

if isempty(Confirmed)
    warning('"Confirmed" is an empty array. Check the value of "minNum". Computation aborted.')
    return
end


Npop= 135.26e8; % population
disp(Npop)
```

```
   1.3526e+10
```

Fitting of the generalized SEIR model to the real data

```
% Definition of the first estimates for the parameters
alpha_guess = 0.06; % protection rate
beta_guess = 1.0; % Infection rate
LT_guess = 5; % latent time in days
Q_guess = 0.1; % rate at which infectious people enter in quarantine
lambda_guess = [0.01,0.001,0]; % recovery rate
kappa_guess = [0.001,0.001,10]; % death rate

guess = [alpha_guess,...
    beta_guess,...
    1/LT_guess,...
    Q_guess,...
    lambda_guess,...
    kappa_guess];


% Initial conditions
Q0 = Confirmed(1)-Recovered(1)-Deaths(1);
I0 = 0.1*Q0; % Initial number of infectious cases. Unknown but unlikely to be zero.
E0 = 0.5*Q0; % Initial number of exposed cases. Unknown but unlikely to be zero.
R0 = Recovered(1);
D0 = Deaths(1);

Active = Confirmed-Recovered-Deaths;
Active(Active<0) = 0; % No negative number possible
[alpha1,beta1,gamma1,delta1,Lambda1,Kappa1,lambdaFun,kappaFun] = ...
    fit_SEIQRDP(Active,Recovered,Deaths,Npop,E0,I0,time,guess);
```

```
                                         Norm of      First-order
 Iteration  Func-count     f(x)            step          optimality
     0         11      4.21742e+12                       1.57e+14
     1         22      6.17716e+11       0.186068        2.23e+13
     2         33      7.50093e+10      0.0249541         3.3e+12
     3         44      6.85794e+09      0.0965811        3.06e+11
     4         55      3.85327e+09       0.208576        4.17e+10
     5         66      3.63865e+09       0.091576        3.56e+10
     6         77       3.3415e+09       0.173143        2.87e+10
     7         88      2.93606e+09       0.234573        2.01e+10
     8         99      2.36511e+09        0.29401        1.01e+10
     9        110      2.36511e+09       0.688208        1.01e+10
    10        121      1.98216e+09       0.172052        4.49e+10
    11        132      1.58388e+09       0.281907        2.29e+11
    12        143      1.13551e+09      0.0391258        8.46e+10
```

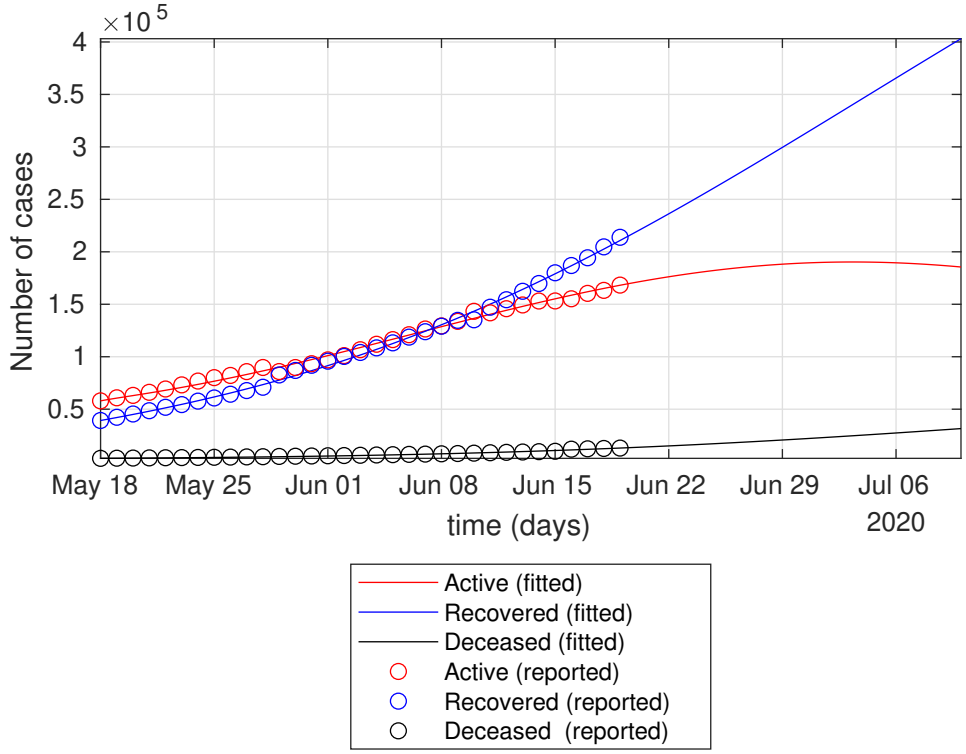
```
    13        154     9.44456e+08       0.0704767       6.18e+09
    14        165     7.60108e+08        0.138124       1.03e+11
    15        176     6.15368e+08        0.069756       7.49e+09
    16        187     5.64824e+08       0.0849426       2.64e+09
    17        198     5.40437e+08        0.140953       1.25e+09
    18        209     5.09262e+08        0.281907       1.13e+10
    19        220      4.9973e+08        0.149579       4.95e+09
    20        231     4.86983e+08       0.0851992       1.27e+09
    21        242     4.85734e+08        0.281907       4.29e+08
    22        253     4.75817e+08        0.563813       2.88e+09
    23        264     4.70991e+08        0.388364       1.23e+09
    24        275     4.70991e+08        0.583889       1.23e+09
    25        286     4.68878e+08        0.145972       1.14e+09
    26        297     4.66898e+08        0.291945        1.2e+09
    27        308     4.65424e+08        0.583889       6.37e+08
    28        319     4.65424e+08        0.421752       6.37e+08
    29        330     4.63398e+08        0.105438       2.05e+09
    30        341      4.6132e+08        0.210876       1.33e+09
    31        352     4.59497e+08        0.390548       5.55e+09
    32        363     4.57809e+08    0.000256109       3.02e+08
    33        374      4.5778e+08        0.097637       8.74e+07
    34        385     4.57567e+08        0.195274       4.19e+08
    35        396     4.57376e+08        0.390548        1.3e+08
    36        407     4.57025e+08        0.417972       3.31e+07
    37        418     4.57024e+08       0.0131443       3.24e+07

Local minimum possible.

lsqcurvefit stopped because the final change in the sum of squares relative to
its initial value is less than the value of the function tolerance.

<stopping criteria details>
```

Simulate the epidemy outbreak based on the fitted parameters

```
dt = 1/24; % time step
time1 = datetime(time(1)):dt:datetime(datestr(floor(datenum(now))+datenum(20)));
N = numel(time1);
t = [0:N-1].*dt;
[S,E,I,Q,R,D,P] = SEIQRDP(alpha1,beta1,gamma1,delta1,Lambda1,Kappa1,...
    Npop,E0,I0,Q0,R0,D0,t,lambdaFun,kappaFun);
```

Display the fitted and measured death and recovery rates

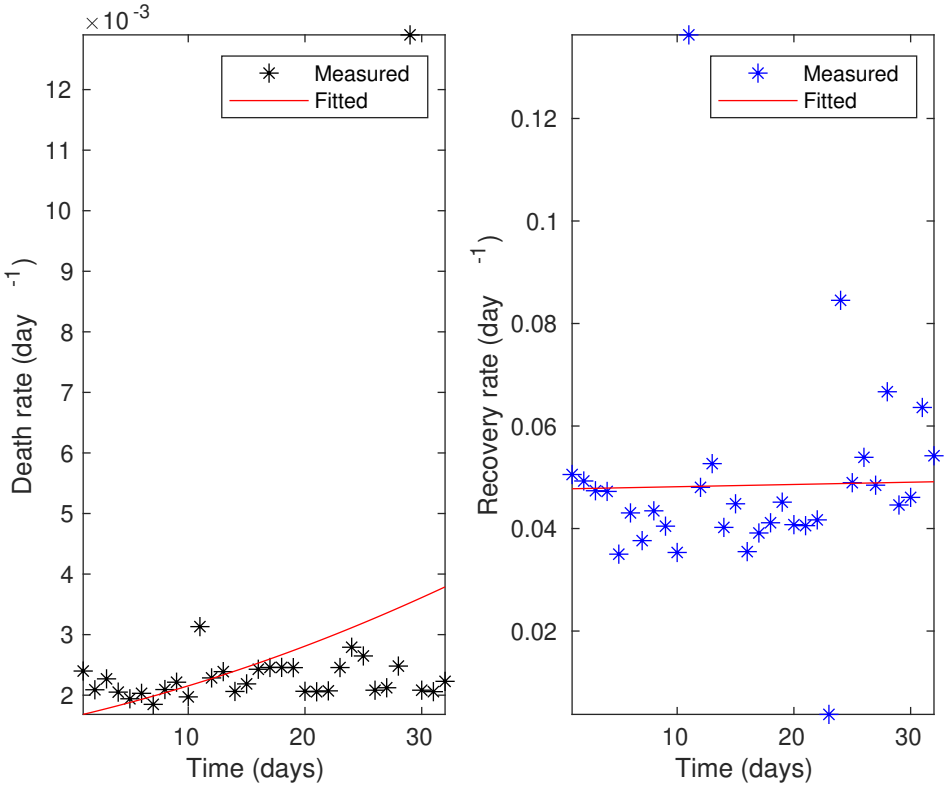
```
checkRates(time,Active,Recovered,Deaths,kappaFun,lambdaFun,Kappa1,Lambda1);
```



Comparison of the fitted and real data

Active cases = Confirmed-Deaths-Recovered (database) = Quarantined (SEIQRDP model)

```
clf;close all;
figure
semilogy(time1,Q,'r',time1,R,'b',time1,D,'k');
hold on
semilogy(time,Active,'ro',time,Recovered,'bo',time,Deaths,'ko');
% ylim([0,1.1*Npop])
ylabel('Number of cases')
xlabel('time (days)')
leg = {'Active (fitted)',...
        'Recovered (fitted)','Deceased (fitted)',...
        'Active (reported)','Recovered (reported)','Deceased  (reported)'};
legend(leg{:},'location','southoutside')
set(gcf,'color','w')
grid on
axis tight
 set(gca,'yscale','lin')
```


×10^-3
Measured
Fitted
Death rate (day^-1)
Time (days)
Measured
Fitted
Recovery rate (day^-1)
Time (days)

Chapter 6

Running the model for assigned location

Assigned Location

Location: Nebraska

Country United States

Population 19.3 lakhs (2019)

The data source which I am using here which has been provided by Johns Hopkins University has different formats for global data and US data. So to keep things easy I have made a different function for getting data for US and US provinces/cities.

6.1 getDataCOVID_US function

This Function acts same as the normal getDataCovid function but deals with only US data.
You will yourself understand the need of writing a separate function once you see and compare the data formats for global and for US

```
function [tableConfirmed,tableDeaths,tableRecovered,time] =
    getDataCOVID_US()
% The function [tableConfirmed,tableDeaths,tableRecovered,time] =
    getDataCOVID
% collect the updated data from the COVID-19 epidemy from the
% John Hopkins university
%
% References:
%  https://github.com/CSSEGISandData/COVID-19
%
% author: Himanshu
%
% see also fit_SEIQRDP.m SEIQRDP.m

%% Number of days of data
Ndays = floor(datenum(now))-datenum(2020,01,22)-1; % minus one day
    because the data are updated with a delay of 24 h
address = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/
    master/csse_covid_19_data/csse_covid_19_time_series/';
ext = '.csv';
%% Options and names for confirmed
opts = delimitedTextImportOptions("NumVariables", Ndays+11);
opts.VariableNames = ["UID",          "iso2",       "iso3",    "
    code3",    "FIPS",    "Admin2",    "Province_State",    "
```

```
    Country_Region" , "Lat" , "Long_" , "Combined_Key", repmat
    ("day",1,Ndays+1)];
opts.VariableTypes = [repmat("string",1,11), repmat("double",1,Ndays
    +1)];
% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";



filename = ['time_series_covid19_confirmed_US'];
fullName = [address,filename,ext];
urlwrite(fullName,'dummy.csv');
tableConfirmed =readtable('dummy.csv', opts);
delete('dummy.csv')
%% Options and names for deceased
%  One more row is used for the population!
%  Inconsistent format used by John Hopkins university

clear opts
opts = delimitedTextImportOptions("NumVariables", Ndays+12);
opts.VariableNames = ["UID", "iso2" , "iso3" , "
    code3" , "FIPS" , "Admin2" , "Province_State", "
    Country_Region" , "Lat" , "Long_" , "Combined_Key", "
    Population", repmat("day",1,Ndays+1)];
opts.VariableTypes = [repmat("string",1,11), repmat("double",1,Ndays
    +2)];
% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";


filename = ['time_series_covid19_deaths_US'];
fullName = [address,filename,ext];
urlwrite(fullName,'dummy.csv');
tableDeaths =readtable('dummy.csv', opts);
delete('dummy.csv')

%% Get time
time = datetime(2020,01,22):days(1):datetime(datestr(floor(datenum(
    now))))-datenum(1);

%% So far no data on recovered

tableRecovered = [];

end
```

6.2 Running live script for assigned location (Nebraska)

Example: COVID-2019 data for US cities and states (Nebraska))

I am again taking the data from John Hopkins university [1]. However, the format for the US data is different and not consistent.

As far as I know (at last on 2020-03-04), no data for the recovered cases are (yet) available. The function fit_SEIQRDP is modified to account for this possibility.

The fitting is also slightly different than in the case where R is available: The (pseudo) death rate is first fitted and the coefficient identified this way are only allowed to change by +/-5 %. This is a pseudo death rate because the number of Confirmed-Deaths is used instead of Quarantined. The main fitting is then done using the heavily-constrained kappa values.

[1] https://github.com/CSSEGISandData/COVID-19

Database access

The parameters are here taken as constant except the death rate and recovery rate.

```
clearvars;close all;clc;
[tableConfirmed,tableDeaths,tableRecovered,time] = getDataCOVID_US();
timeRef = time;
```

Case of an entire state

Every city in one state is selected and the cases are added

```
Location = 'Nebraska'; % Find every cities in Washington state
% Location = 'New York'; % Find every cities in New York state
try
    indC = find(contains(tableConfirmed.Province_State,Location)==1);
    indD = find(contains(tableDeaths.Province_State,Location)==1);
catch exception
    searchLoc = strfind(tableConfirmed.Province_State,Location);
    indC = find(~cellfun(@isempty,searchLoc))  ;

    searchLoc = strfind(tableDeaths.Province_State,Location);
    indD = find(~cellfun(@isempty,searchLoc))   ;
end

% disp(tableConfirmed(indC,1:2))


% Initialisation
Confirmed = 0;
Deaths = 0;
Npop = 0;
for ii=1:numel(indC)
    Confirmed =  Confirmed + table2array(tableConfirmed(indC(ii),12:end));
end

for ii=1:numel(indD)
    Deaths =  Deaths + table2array(tableDeaths(indD(ii),13:end));
    Npop= Npop + table2array(tableDeaths(indD(ii),12)); % population (dummy number here)
end
```

Initial conditions for the fitting

```
% If the number of confirmed Confirmed cases is small, it is difficult to know whether
% the quarantine has been rigorously applied or not. In addition, this
% suggests that the number of infectious is much larger than the number of
% confirmed cases
time = timeRef;
```

```
minNum= round(0.25*max(Confirmed)); % 5% of the maximal number of confirmed is used for the ini
Deaths(Confirmed<=minNum)=[];
time(Confirmed<=minNum)= [];
Confirmed(Confirmed<=minNum)=[];

fprintf(['Population = ',num2str(Npop),' \n'])
```

```
Population = 1934408
```

```
% Definition of the first estimates for the parameters
alpha_guess = 0.05;
beta_guess = 0.8; % Infection rate
LT_guess = 5; % latent time in days
Q_guess = 0.5; % rate at which infectious people enter in quarantine
lambda_guess = [0.1,0.1,10]; % recovery rate
kappa_guess = [0.01,0.01,10]; % death rate

guess = [alpha_guess,...
    beta_guess,...
    1/LT_guess,...
    Q_guess,...
    lambda_guess,...
    kappa_guess];

E0 = 0.25*Confirmed(1); % Initial number of exposed cases. Unknown but unlikely to be zero.
I0 = 0.5*E0; % Initial number of infectious cases. Unknown but unlikely to be zero.
Q0 = Confirmed(1)-Deaths(1);
R0 = Deaths(1); % Unknown but unlikely to be zero. Taken as equal to the number of deaths
D0 = Deaths(1);

% Parameter estimation with the lsqcurvefit function[alpha1,beta1,gamma1,delta1,Lambda1,Kappa1]
    [alpha1,beta1,gamma1,delta1,Lambda1,Kappa1,lambdaFun,kappaFun] = ...
        fit_SEIQRDP(Confirmed-Deaths,[],Deaths,Npop,E0,I0,time,guess,'Display','off');
```

```
Warning:  No data available for "Recovered"
```

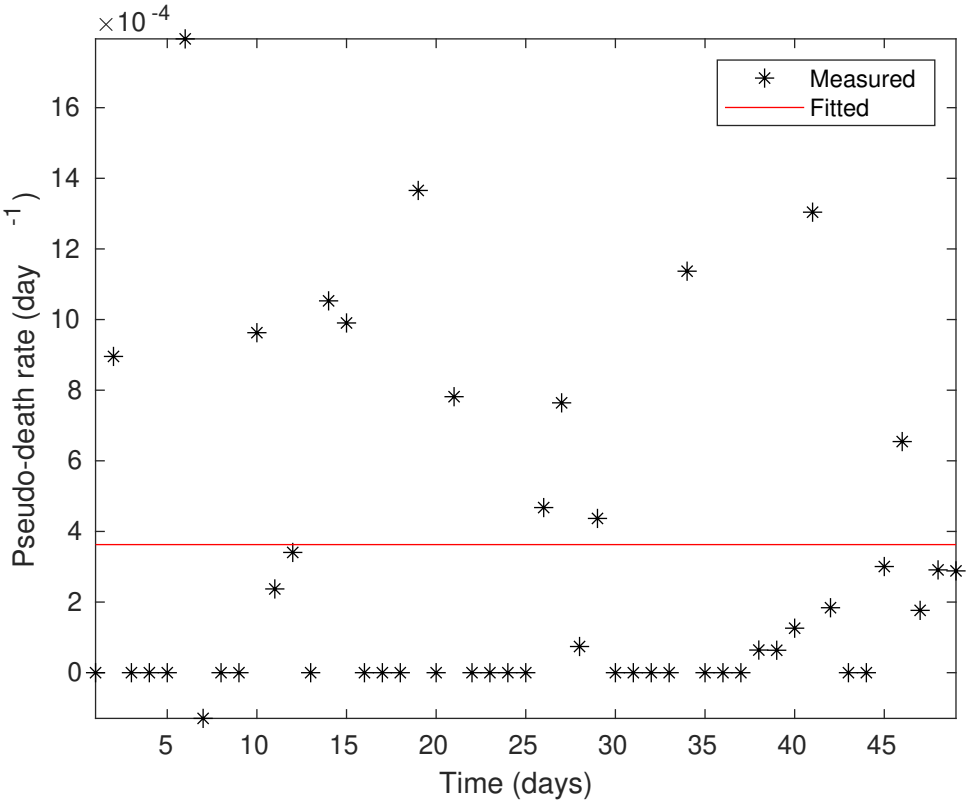
Simulate the epidemy outbreak based on the fitted parameters

```
dt = 1/24; % time step
time1 = datetime(time(1)):dt:datetime(datestr(floor(datenum(now))+datenum(10)));
N = numel(time1);
t = [0:N-1].*dt;
[S,E,I,Q,R,D,P] = SEIQRDP(alpha1,beta1,...
    gamma1,delta1,Lambda1,Kappa1,Npop,E0,I0,Q0,R0,D0,t,lambdaFun,kappaFun);
```

Display the fitted and measured (pseudo) death rates

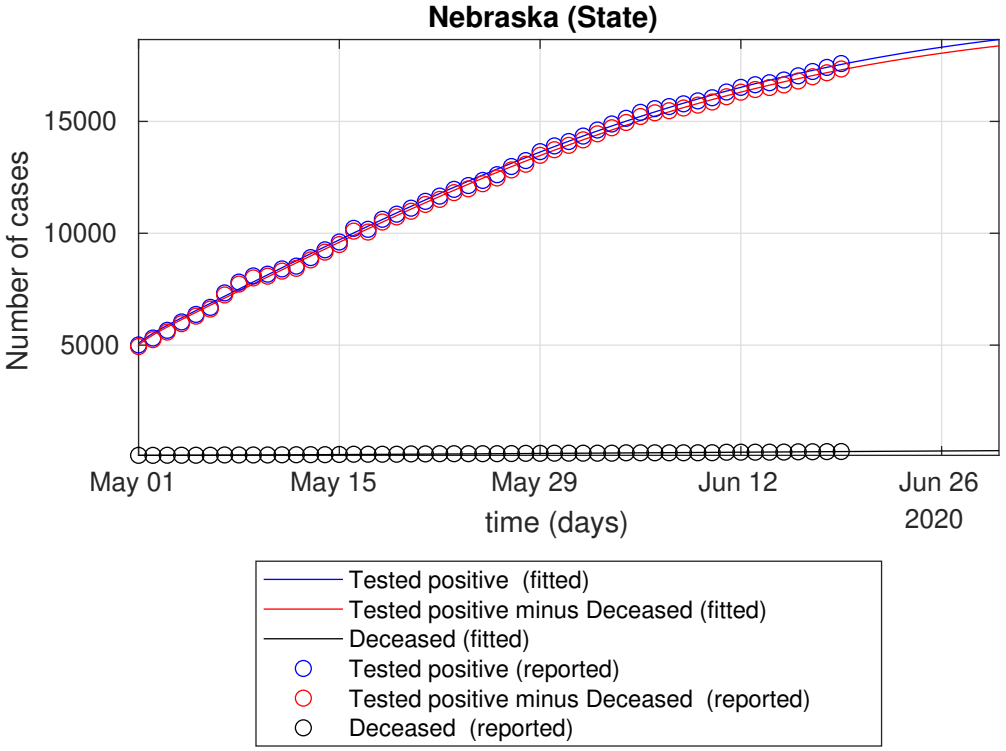
This is a "pseudo" death rate because it si calculated based on (Q(t)+R(t)) and not only from Q(t).

```
checkRates(time,Confirmed-Deaths,[],Deaths,kappaFun,lambdaFun,Kappa1,Lambda1);
```



Comparison of the fitted and real data

```
figure
semilogy(time1,Q+R+D,'b',time1,Q+R,'r',time1,D,'k');
hold on
semilogy(time,Confirmed,'bo',time,Confirmed-Deaths,'ro',time,Deaths,'ko');
% ylim([0,1.1*Npop])
ylabel('Number of cases')
xlabel('time (days)')
leg = {'Tested positive  (fitted)','Tested positive minus Deceased (fitted)',...
    'Deceased (fitted)',...
    'Tested positive (reported)',...
    'Tested positive minus Deceased  (reported)','Deceased  (reported)'};
legend(leg{:},'location','southoutside')
set(gcf,'color','w')
grid on
axis tight
title([Location,' (State)'])
set(gca,'yscale','lin')
```



6.3 Parameters' values (for Nebraska)

Calulating Parameters

```
fprintf("protection rate: %d\n",alpha1);
```

```
protection rate: 5.916385e-03
```

```
fprintf("infection rate: %d\n",beta1);
```

```
infection rate: 8.524935e-01
```

```
fprintf("latent time: %d\n",1/gamma1);
```

```
latent time: 4.740417e+00
```

```
fprintf("quarentine rate: %d\n",delta1);
```

```
quarentine rate: 8.283187e-01
```

```
crr = lambdaFun(Lambda1,t(end));
cmr = kappaFun(Kappa1,t(end));
fprintf("current recovery rate: %d\n",crr);
```

```
current recovery rate: 2.101048e-02
```

```
fprintf("current mortality rate: %d\n",cmr);
```

```
current mortality rate: 3.627382e-04
```

α **(protection rate):** 5.916385×10^{-3}

β **(infection rate):** 0.8524935

γ **(inverse of the average latent time):** 4.740417

δ **(rate at which infectious people enter in quarantine):** 0.8283187

λ **(current recovery rate):** 0.02101048

κ **(current mortality rate):** 3.627382×10^{-4}

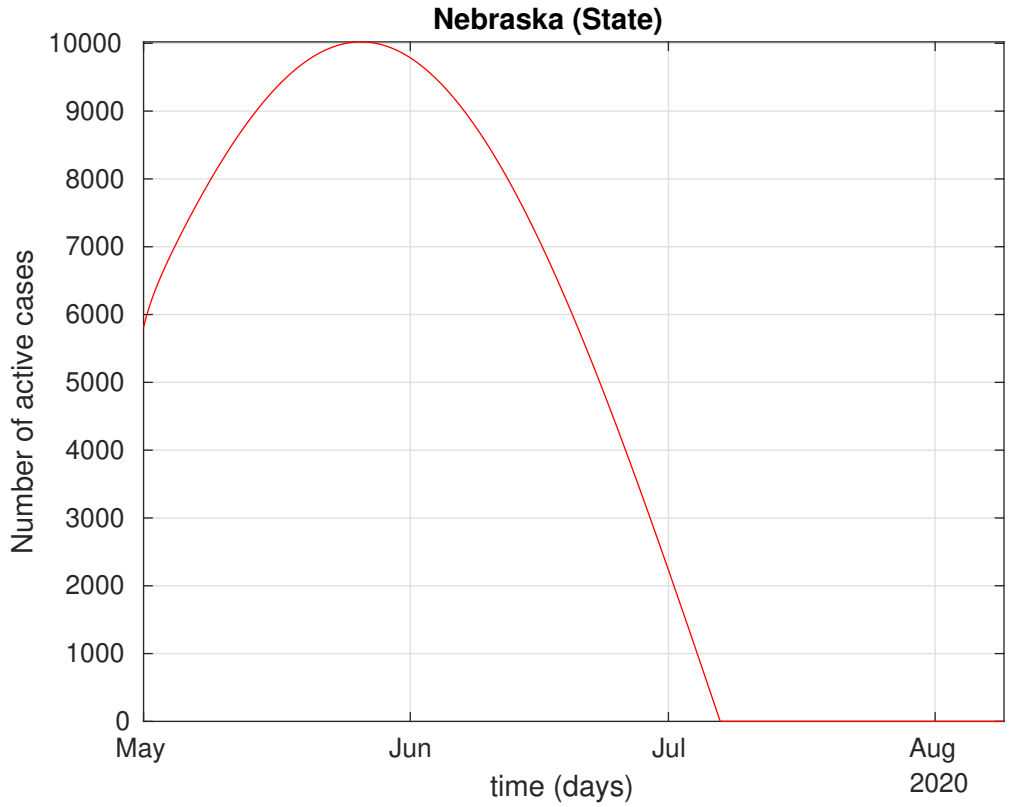
6.4 Finding time for minimum infection

It is easy to find time for maximum infection for locations for those recovered data is available as we know the total number of active cases at every point of time. Here we will have to use certain assumptions as those used int the function getDataCOVID_US and in the Live Script.

Finding point of Maximum Infection

note: this script can only be run after running script Example_US_cities.mlx

```
A = Q/delta1-R-D;
A(A<0) = 0;
figure
semilogy(time1,A,'r');

ylim([0,1.1*Npop])
ylabel('Number of active cases')
xlabel('time (days)')

set(gcf,'color','w')
grid on
axis tight
title([Location,' (State)'])
set(gca,'yscale','lin')
```




```
[maxI,maxIp] = max(A);
timemaxp = datestr(time1(maxIp));
maxI = floor(maxI);
maxistr = num2str(maxI);
Ans = ['maximum infection at a time was ',maxistr,' on ',timemaxp];
disp(Ans);
```

```
maximum infection at a time was 10021 on 26-May-2020 06:00:00
```

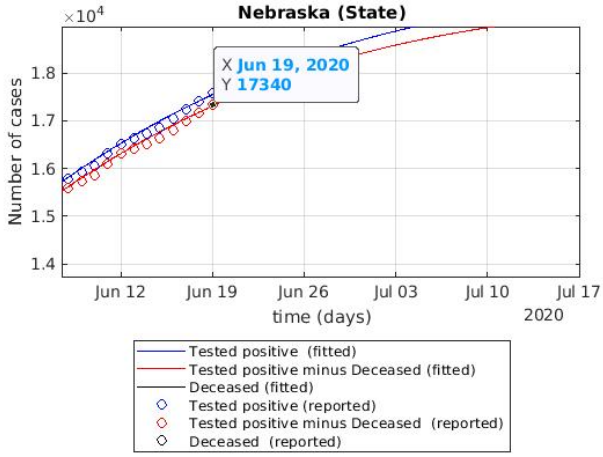
So as per the simulation maximum infection at a particular time was about 10000 around May 26^{th} 2020.

6.4.1 Duration of pandemic

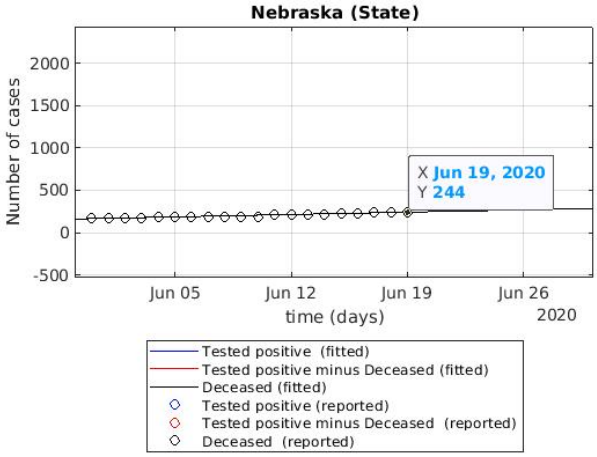
As by the above plot the active cases nears zero around 7^{th} of July, so the pandemic seems to end after first week of July.
Here we are not considering second wave.
The first case in Nebraska was discovered on 2^{nd} March 2020, so duration of the epidemic will be around 4 months 10 days.

6.4.2 Casualties and recovered

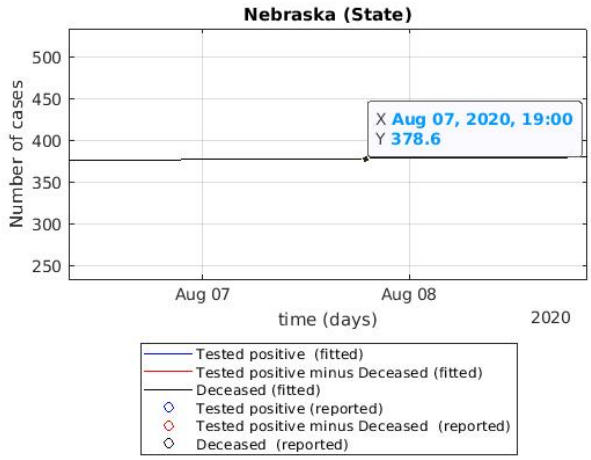
From the plot in section 6.2 we can observe the current casualties and projected casualties. Though recovered is not directly known for the assigned location we can indirectly estimate the recovered from the same plot.



(a) Recovered



(b) Total causalities by today



(c) Total projected causalities

Figure 6.1: Deaths and Recovered

So from figure 6.1 we can clearly get the causalities and recovered.

6.5 Manual analysis and discussions

There have been at least 17,591 cases of corona virus in Nebraska, according to a New York Times database. As of Saturday morning, at least 249 people had died. From actual vs simulated comparison in section 6.2 we can see that our estimation is quite close to the actual data. We have projected the same fitted curve ans assume as well as hope for the actual situation to follow the trajectory of our projection.

If nothing wrong happens in Nebraska, it is expected that Nebraska is going to be free from COVID very soon. And lets hope for no second wave to happen.

By now casualties in Nebraska is also low as compered to many states, though it is a fact that Nebraska is not as populous as other states, it is also little less developed so foreign travellers relatively visit less at Nebraska

Chapter 7

References

https://arxiv.org/pdf/2002.06563.pdf
https://en.wikipedia.org/wiki/Orders_of_magnitude_(mass)

Liangrong Peng), Wuyue Yang), Dongyan Zhang, Changjing Zhuge), Liu Hong2b) 1College of

Mathematics and Data Science, Minjiang University, Fuzhou, 350108, P.R.C. 2Zhou Pei-Yuan Center for Applied Mathematics, Tsinghua University, Beijing, 100084, P.R.C. 3Beijing Institute for Scientific and Engineering Computing, College of Applied Sciences, Beijing University of Technology, Beijing, 100124, P.R.C.
https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology#

Bio-mathematical_deterministic_treatment_of_the_SIR_model
https://github.com/pcm-dpc/COVID-19

https://github.com/cedricguadalupe/FRANCE-COVID-19

https://se.mathworks.com/videos/importing-data-from-text-files-interactively-71076.

html
https://github.com/matteosecli

https://in.mathworks.com/matlabcentral/profile/authors/642467-yair-altman

https://in.mathworks.com/matlabcentral/profile/authors/3727172-indranil-saaki

https://in.mathworks.com/matlabcentral/profile/authors/869215-john-d-errico