Coronavirus Disease 2019 cases in Wuhan, China. [1]

Contents

List of Figures

1 Introduction

In 2019 coronavirus has given rise to global pandemic that lasts to this day. It started in Wuhan, China, most likely originated from the Huanan Seafood Wholesale Market. As of 17 February 2020, there were a cumulative total of 72,436 confirmed cases in mainland China.

Now the disease is a worldwide problem, so governments and scientists are looking for solutions to stop the pandemic. One tool for understanding the problem are deterministic models that use differential equations and those are will be the subject of the report.

2 SEIR Model

The model employed to model and predict the pandemic outbreak is an infectious disease dynamics model (SEIR model). It is based on the idea that only people are transmiting the virus. In this model, individuals are classified into four types:

1. S – susceptible (at risk of contracting the disease),
2. E – exposed (infected but not yet infectious),
3. I – infectious (capable of transmitting the disease),
4. R – removed (those who recover or die from the disease).

2.1 Assumptions

There are several assumptions for a proper use of SEIR model:

- no new transmissions from animals,
- no differences in individual immunity,
- the time-scale of the epidemic is much faster than characteristic times for demographic processes (natural birth and death),
- no differences in natural births and deaths.

2.2 Parameters

Besides the variables there are 3 parameters used in model, which one can tune to obtain a special case of epidemy (these are properties of virus). The most important of them is the reproduction number R which measures the transmissibility of a virus, representing the average number of new infections generated by each infected person. If $R > 1$ then the outbreak is self-sustaining and if $R < 1$ then the outbreak will eventually stop. The parameters needed for a model are:

1. R_0 – basic reproduction number (at the start of an epidemy outbreak),
2. γ – is the recovery rate calculated by the inverse of infectious period,
3. σ – is the infection rate calculated by the inverse of the mean latent period,

Additionally there is parameter $\beta = R_0\gamma$, which is interpreted as the transmission rate.

2.3 Differential equations

The SEIR model is represented by differential equations:

$$\begin{cases} \frac{\mathrm{d}S}{\mathrm{d}t} = -\beta SI/N \\ \frac{\mathrm{d}E}{\mathrm{d}t} = \beta SI/N - \sigma E \\ \frac{\mathrm{d}I}{\mathrm{d}t} = \sigma E - \gamma I \\ \frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I \end{cases}$$

Where $N = S + E + I + R$. Starting conditions will be discussed in later section (2.4 and 4.1).

2.4 Interpretation

One can think of this model as transfering people between 4 sets: S, E, I, R. It is starting with susceptible individuals (S) equal to number of people living in a cluster and some amount of exposed (E) and infectious (I) inviduals that will start to infect other. Recovered or dead (R) individuals of course start at 0.

Factor $-\beta SI/N$ is always negative, because $S, I > 0$ are natural numbers, representing number of people. It implicates, that S is always decreasing and as time passes by, there are less people that can be infected. S depends on how many people are infected by others (βI).

The number of exposed (E) is increasing when more people are infected (βI) than turning into infectious (σE). It's decreasing when opposite happens. It depends on how many transmissions can happen and how long virus is latent.

The number of infectious (I) is increasing when more people are turning from exposed to infectious than the infected are recovering. It's decreasing when opposite happens. It depends on how long virus is latent and how fast people are recovering.

Analogous argument to the first one implicates that number of recovered people is increasing ($\gamma I > 0$). Faster recovery rate means more people recovered at each step.

3 Analysis

3.1 Equilibrium points

The first step of differential equation analysis is to find its equilibrium points.

$$\frac{\mathrm{d}S}{\mathrm{d}t} = 0 \Leftrightarrow S = 0 \vee I = 0.$$

The starting conditions assumed $S, I > 0$, but later if possible, S or I can be equal to 0 – whole population was infected or infectious individuals recovered completely. In both cases:

$$\frac{\mathrm{d}E}{\mathrm{d}t} = -\sigma E, \quad \frac{\mathrm{d}E}{\mathrm{d}t} = 0 \Leftrightarrow E = 0.$$

And then:

$$\frac{\mathrm{d}I}{\mathrm{d}t} = -\gamma I, \quad \frac{\mathrm{d}I}{\mathrm{d}t} = 0 \Leftrightarrow I = 0.$$

It means, that when there are no infectious people there are no exposed people too. So, first equation equal to 0 gives no exposed and then no infectious people. These are obvious cases: either population is not susceptible to the disease or there is noone infected and then no outbreak happens. Now we get the lines of equilibrium points:

$$(S, 0, 0, R), (0, 0, I, R).$$

Also, when $R = N$ then from the condition $N = S + E + I + R$, we know that $S, E, I = 0$ and then we get another equilibirum point:

$$(0, 0, 0, N).$$

4 Numerical analysis

All solutions to models with different parameters are calculated with Runge-Kutta method with weights as descibed in its wiki article [2] using Python3 (and package NumPy) and double precision.

4.1 Starting conditions

For starting conditions of SEIR model authors used $S = 11 \cdot 10^6$ which is the number of population in Wuhan, city where pandemic started, I is estimated to be 40 and $E = 20I$. R starts always as zero.

In paper [1] authors suggest parameters for covid virus to be as follows (based on other scientific papers): $R_0 = 2.6$, $\gamma = 1/18$, $\sigma = 1/5.2$. It means, that the virus at the begining was spreading from one person to average of 2.6 other people, average hospitalization period is 18 days and the mean incubation period of virus is 5.2 days.

4.2 Numerical solutions

Numerical solutions obtained via Runge-Kutta method are presented (fig. 4.1) with different starting conditions. The blue line is representing the original starting conditions of the real pandemic (described above).

Starting conditions differ only by the number of infectious people, that's because we estimate exposed based on that number, R is always 0 and we are interested in one population $S = 11 \cdot 10^6$.

The graphs of $S(t)$ and $R(t)$ are looking just as expected – they are both converging to 0 and full population respectively, which means that everyone in population went through the disease. The graph of $I(t)$ suggests, that there is a peak of infectious people. The same goes for exposed group, the difference is, that at the begining the numbers of exposed people are decreasing – they are turning faster to infectious than there are new exposed individuals.

The change in starting conditions mostly cause the shift in graphs. In cases of infected people, that also affects the height of the peak – more infected people, heigher the peak.

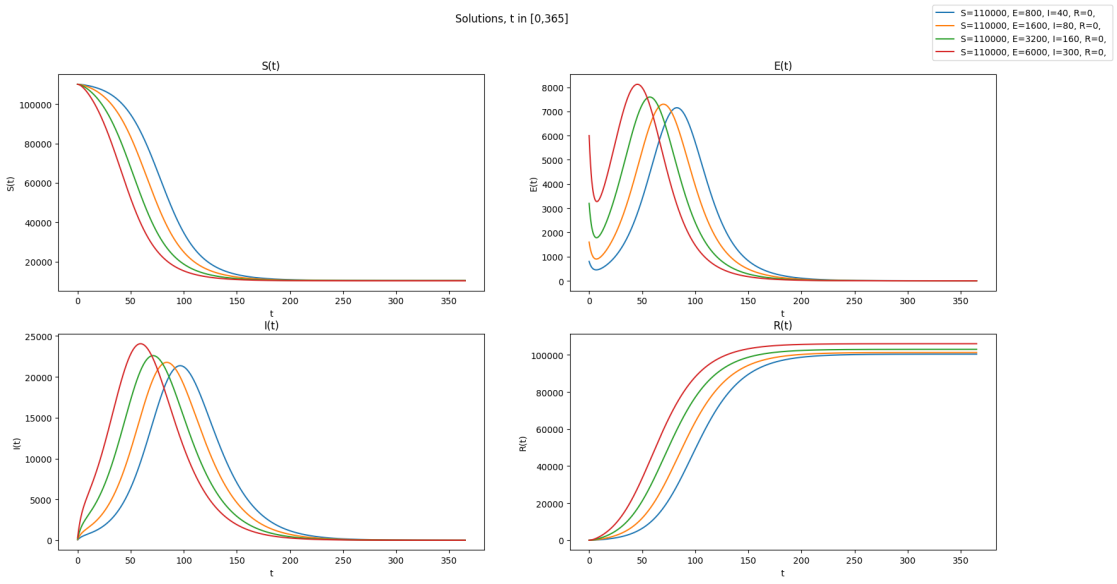

Figure 4.1: Plots of numerical solutions in $t \in [0, 365]$.

4.3 Phase space

Phase space plots are trajectories of one variable vs. the other in some specific time interval. In case of models with more than 2 variables those plots are only a cross-section(?) of a higher dimensional curves. Numerical trajectories obtained via Runge-Kutta method are presented (fig. 4.2) with different starting conditions (analogous to the previous section 4.2).

Trajectories are not closed, which means that solutions are not periodical. The graph E vs I suggests that peaks of E and I are happening in short time interval.

4.4 Different parameters

Numerical solutions obtained via Runge-Kutta method are presented (fig. 4.3) with different parameters. They will be compared to the model with covid parameters (blue line). There are 3 properties of a virus to change:

1. $1/\gamma$ – infectious period (2 times shorter ≈ 9 days, yellow line),
2. $1/\sigma$ – latent period (2 times shorter ≈ 2.5 days, green line),
3. R_0 – reproduction number (a bit smaller $= 2.0$, red line).

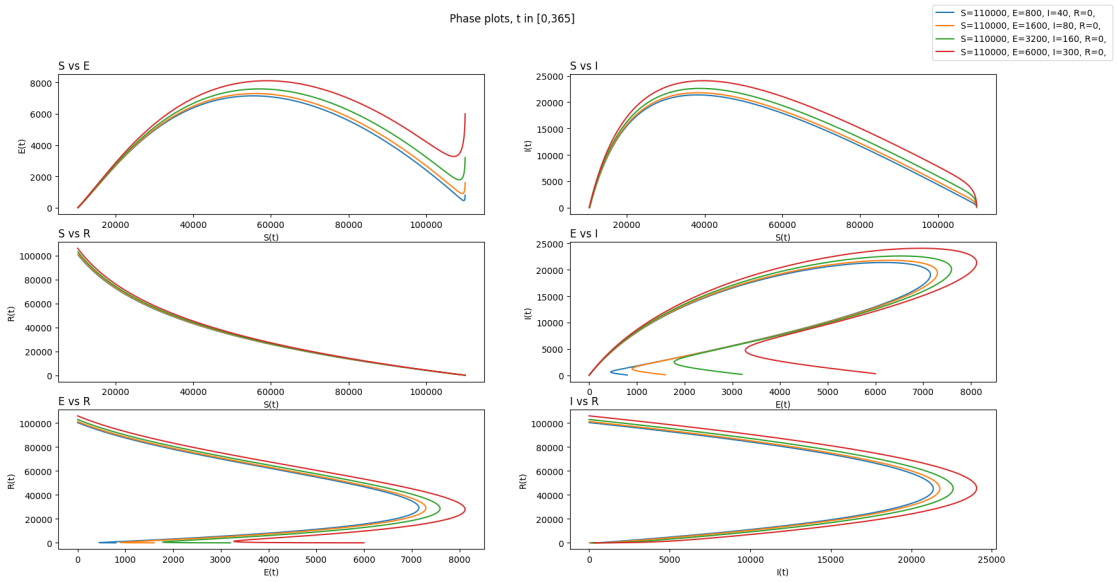

Figure 4.2: Phase space plots of pairs of variables in $t \in [0, 365]$.

This choice of parameters intended to mitigate the results of pandemic. The changes provided shifted solutions with different peaks of ill people and different equilibrium points.

Lower R_0 caused more spread out graph that does not converge to the number of cluster population – less people have suffered the disease. The reduction of latent period caused higher peak of infectious people and reduction of infectious period caused higher peak for exposed people. The changes of all parameters (except of R_0) proved not to be enough to stop the pandemic (whole population has suffered the disease).

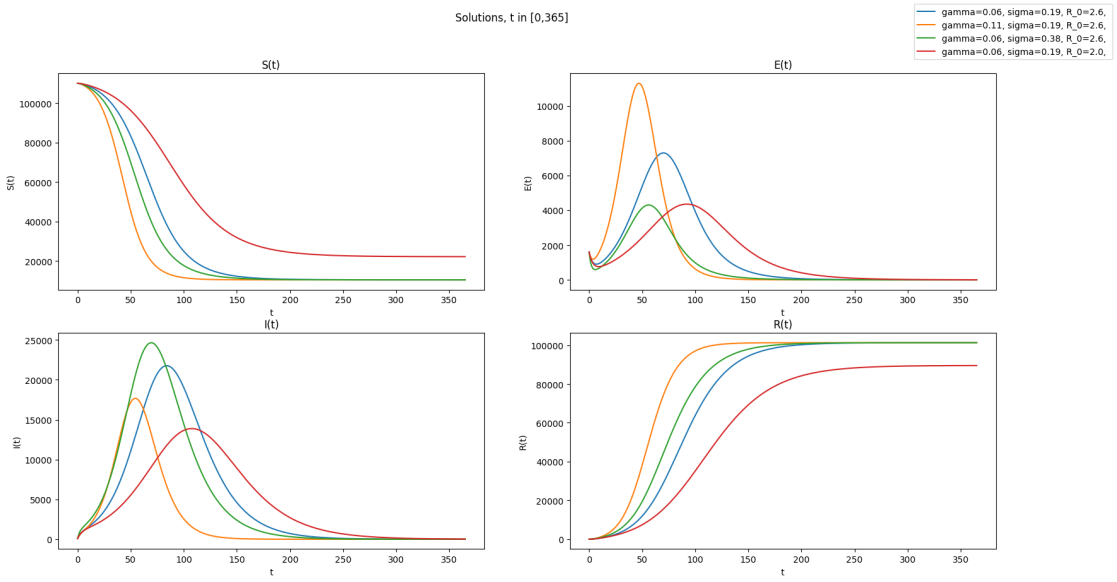

Figure 4.3: Plots of numerical solutions in $t \in [0, 365]$ for different parameters.

5 Conclusions

The SEIR model is working well with one's intuitions. Given not so rigorous assumptions it could be reliably used for predicting the end of epidemic and the highest number of ill people, which is useful for

governments and hospitals.

Trying out different parameters could be used to learn how to deal with epidemic. The previous section (sec. 4.4) revealed how important R_0 factor is. It could be reduced by wearing medical masks. Other parameters can also be affected: infectious and latent period could be reduced by quarantine.

The model could be improved by using functions of time instead of averaged constants for parameters.

References

[1] H. Wang, Z. Wang, Y. Dong, R. Chang, C. Xu, X. Yu, S. Zhang, L. Tsamlag, M. Shang, J. Huang, et al. Phase-adjusted estimation of the number of coronavirus disease 2019 cases in wuhan, china. *Cell discovery*, 6(1):1–8, 2020.

[2] Wikipedia contributors. Runge–kutta methods — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Runge%E2%80%93Kutta_methods&oldid=1055669759`, 2021. [Online; accessed 26-November-2021].

Appendices

Appendix 1 Python code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

def runge_kutta_method(y_0 = np.array([0.5,0.5]), x_min=0, x_max=50, h=0.1,
    f = lambda x: x, sol = np.exp):
    x = np.arange(x_min, x_max+h, step=h)
    y = np.zeros((x.size, len(y_0)))
    y[0,:] = y_0

    for k in range(x.size-1):
        k_1 = y[k,:] + h*f(y[k,:])
        k_2 = y[k, :] + h * f((y[k, :] + k_1)/2)
        k_3 = y[k, :] + h * f((y[k, :] + k_2)/2)
        k_4 = y[k, :] + h * f(k_3)
        y[k + 1, :] = (k_1 + 2*k_2 + 2*k_3 + k_4)/6

    return x,y

def quiver_plot(f, x1_range, x2_range):
    X1, X2 = np.meshgrid(np.linspace(x1_range[0], x1_range[1], num=30), np.
        linspace(x2_range[0], x2_range[1], num=30))
    dY = f([X1, X2])
    length = np.sqrt(dY[0]**2 + dY[1]**2)
    dY[0] /= length
    dY[1] /= length
    plt.quiver(X1, X2, dY[0], dY[1])

def phase_space_plots(Y_0, f, h=0.01, x_max=365, names=['S', 'E', 'I', 'R'
    ]): #function for models containing more than 2 variables, Y_0 -- rows
    are starting conditions
    RK = [None] * Y_0.shape[0]
    for i in range(Y_0.shape[0]):
        t, RK[i] = runge_kutta_method(f=f, h=h, y_0=Y_0[i, :], x_max=x_max)
```

```
    labels = [''] * Y_0.shape[0] #labels for legend
    for i in range(Y_0.shape[0]):
        for j in range(Y_0.shape[1]):
            labels[i] += names[j]+'='+str(Y_0[i,j])+', '

    if(Y_0.shape[1] == 2): #if model has only 2 variables, quiver plot is
        reasonable
        quiver_plot(f=f, x1_range=(RK[0][:, 0].min(), RK[0][:, 0].max()),
            x2_range=(RK[0][:, 1].min(), RK[0][:, 1].max()))
        for i in range(Y_0.shape[0]):
            plt.plot(RK[i][:, 0], RK[i][:, 1])
            plt.title(names[0]+' vs '+names[1], loc='left')
            plt.xlabel(names[0]+'(t)')
            plt.ylabel(names[1]+'(t)')
            plt.legend(labels=labels)
    else:
        n = math.comb(Y_0.shape[1], 2) # number of plots
        fig, ax = plt.subplots(math.ceil(n/2), 2)
        d = l = 0
        for i in range(Y_0.shape[1]):
            for j in range(i+1, Y_0.shape[1]):
                for k in range(Y_0.shape[0]):
                    ax[d, l].plot(RK[k][:, i], RK[k][:, j])
                    ax[d, l].set_title(names[i]+' vs '+names[j], loc='left'
                        )
                    ax[d, l].set_xlabel(names[i]+'(t)')
                    ax[d, l].set_ylabel(names[j]+'(t)')
                d += 1
                l = (l+1)%2

        fig.suptitle('Phase plots, t in [0,'+str(x_max)+']')
        fig.legend(labels=labels)
    plt.show()

def solution_plots(Y_0, f, h=0.01, x_max=365, names=['S', 'E', 'I', 'R']):
    RK = [None] * Y_0.shape[0]
    for i in range(Y_0.shape[0]):
        t, RK[i] = runge_kutta_method(f=f, h=h, y_0=Y_0[i, :], x_max=x_max)

    if(Y_0.shape[1]==2):
      fig, ax = plt.subplots(1, 2)
      for i in range(2):
        for j in range(Y_0.shape[0]):
          ax[i].plot(t, RK[j][:, i])
        ax[i].set_title(names[i]+'(t)')
        ax[i].set_xlabel('t')
        ax[i].set_ylabel(names[i]+'(t)')
    else:
      fig, ax = plt.subplots(math.ceil(Y_0.shape[1]/2), 2) #number of plots
      d = l = 0
      for i in range(Y_0.shape[1]):
          for j in range(Y_0.shape[0]):
              ax[d, l].plot(t, RK[j][:, i])
          ax[d, l].set_title(names[i]+'(t)')
          ax[d, l].set_xlabel('t')
          ax[d, l].set_ylabel(names[i]+'(t)')
          d += 1
```

```
            l = (l+1)%2

    labels = [''] * Y_0.shape[0] #labels for legend
    for i in range(Y_0.shape[0]):
        for j in range(Y_0.shape[1]):
            labels[i] += names[j]+'='+str(Y_0[i,j])+', '

    fig.suptitle('Solutions, t in [0,'+str(x_max)+']')
    fig.legend(labels=labels)
    plt.show()

def parameters_solutions_plot(y_0, model, PARAM, h=0.01, x_max=365, p_names
    =['gamma', 'sigma', 'R_0'], v_names=['S', 'E', 'I', 'R']):
    RK = [None] * PARAM.shape[0]
    for i in range(PARAM.shape[0]):
        t, RK[i] = runge_kutta_method(f=model(*PARAM[i, :]), h=h, y_0=y_0,
            x_max=x_max)

    if (y_0.size == 2):
        fig, ax = plt.subplots(1, 2)
        for i in range(2):
            for j in range(PARAM.shape[0]):
                ax[i].plot(t, RK[j][:, i])
            ax[i].set_title(v_names[i] + '(t)')
            ax[i].set_xlabel('t')
            ax[i].set_ylabel(v_names[i] + '(t)')
    else:
        fig, ax = plt.subplots(math.ceil(y_0.size / 2), 2)  # number of
            plots
        d = l = 0
        for i in range(y_0.size):
            for j in range(PARAM.shape[0]):
                ax[d, l].plot(t, RK[j][:, i])
            ax[d, l].set_title(v_names[i] + '(t)')
            ax[d, l].set_xlabel('t')
            ax[d, l].set_ylabel(v_names[i] + '(t)')
            d += l
            l = (l + 1) % 2

    labels = [''] * PARAM.shape[0]  # labels for legend
    for i in range(PARAM.shape[0]):
        for j in range(PARAM.shape[1]):
            labels[i] += p_names[j] + '=' + str(round(PARAM[i, j],2)) + ',
                '


    fig.suptitle('Solutions, t in [0,' + str(x_max) + ']')
    fig.legend(labels=labels)
    plt.show()

#MODELS
def SEIR_model(gamma=1/18, sigma=1/5.2, R_0=2.6): #parameters for covid
    beta = R_0*gamma
    return lambda y: np.array([ -beta*y[0]*y[2]/ (y[0]+y[1]+y[2]+y[3]),
                                beta *y[0]*y[2]/(y[0]+y[1]+y[2]+y[3]) - sigma
                                    *y[1],
                                sigma*y[1] - gamma*y[2],
                                  gamma*y[2]])
```

```
def SIR_model(gamma=1/18, sigma=1/5.2):
    return lambda y: np.array([ -sigma * y[0] *y[1] / (y[0] + y[1] + y[2]),
                                sigma * y[0] * y[1] / (y[0] + y[1] + y[2])
                                   - gamma * y[1],
                                gamma * y[1]])

def LV_model(a=1, b=1, c=1, d=1):
    return lambda y: np.array([ (a-b*y[1])*y[0], (-c+d*y[0])*y[1] ])

def whale_model(k=6, a=1/27, d=1):
  return lambda y: np.array([ y[0]*( (k-y[0]) - y[1]/(1+y[0]) ),
                              d * y[1] * ( y[0]/(1+y[0]) - a*y[1] )])

Y_0 = np.array([[11*10**4, 40*20, 40, 0],
                [11*10**4, 80*20, 80, 0],
                [11*10**4, 160*20, 160, 0],
                [11*10**4, 300*20, 300, 0]])

Y_0_whale = np.array([[1,8],
                      [8,1],
                      [6,6]
                      ])

PARAM = np.array([ [1/18, 1/5.2, 2.6],
                   [2/18, 1/5.2, 2.6],
                  [1/18, 2/5.2, 2.6],
                     [1/18, 1/5.2, 2.0]])

#phase_space_plots(Y_0=Y_0, f=SEIR_model())
#solution_plots(Y_0=Y_0, f=SEIR_model(tuple(PARAM[0,:])))
parameters_solutions_plot(y_0=Y_0[1, :], PARAM=PARAM, model=SEIR_model)

#phase_space_plots(Y_0=Y_0[:, [0,2,3]], f=SIR_model(), names=['S', 'I', 'R
    '])
#solution_plots(Y_0=Y_0[:, [0,2,3]], f=SIR_model(), names=['S', 'I', 'R'])
#parameters_solutions_plot(y_0=Y_0[1, [0,2,3]], PARAM=PARAM[:, [0,1]],
    model=SIR_model, p_names=['gamma', 'sigma'], v_names=['S', 'I', 'R'])

#phase_space_plot(Y_0=np.array([[10,10],[5,5]]), f=LV_model(), names=['x',
    'y'])
#parameters_solutions_plot(y_0=np.array([5, 5]), PARAM=np.array
    ([[1,1,1,1],[1,2,3,4]]), model=LV_model, p_names=['a', 'b', 'c', 'd'],
    v_names=['x', 'y'], x_max=50)

#phase_space_plots(Y_0=Y_0_whale, f=whale_model(), x_max=100, names=['p','h
    '])
#solution_plots(Y_0=Y_0_whale, f=whale_model(), x_max=100, names=['p','h'])
```