BITS PILANI GOA

MATHEMATICAL MODELLING OF BIOLOGICAL DATA

DESIGN PROJECT (INSTR F376)

Anurag Malik -2012B2A8515G

4/28/2016

A report on ‘Mathematical modeling’ for the “Design Project” to be submitted to Dr. Anita Agrawal for the partial fulfillment of the course.

ABSTRACT

In this report, SIR (Susceptible-Infected-Recovered) model for the mathematical modelling of diseases is described. The mathematics behind the model and various tools for judging effectiveness of policies and control methods has been studied. To compute the amount of susceptible, infected and recovered people in a population, the SIR model is used in epidemiology. This model works with some of the diseases and for those diseases for which this doesn’t work, other SIR-like models do. Certain assumptions are also made while studying the SIR models which are also described in this report. Also the dynamics of the fully-mixed SIR-model, the cornerstone of epidemiological modelling, was explored. Simulations were built for the deterministic versions of the SIR model in order to explore the onset of large outbreaks at a critical reproductive number, the size of those outbreaks as a function of model parameters, and their distribution.

TABLE OF CONTENTS

Introduction

Assumptions:

The SIR model is used to compute the amount of susceptible, infected and recovered people in a population during the spread of a disease. This model is an appropriate one to use under the following assumptions.

1) The population is fixed.
2) The only way a person can leave the susceptible group is to become infected. The only way a person can leave the infected group is to recover from the disease. Once a person has recovered, the person received immunity.
3) Age, sex, social status, and race do not affect the probability of being infected.
4) There is no inherited immunity.
5) The member of the population mix homogeneously (have the same interactions with one another to the same degree).

SIR Formulae:

The model starts with some basic notation:

S(t) is the number of susceptible individuals at time t

I(t) is the number of infected individuals at time t

R(t) is the number of recovered individuals at time t

N is the total population size

These assumptions lead to a set of differential equations.

$$\frac{dS}{dt} = -\beta S(t)I(t) \qquad (1)$$

$$\frac{dI}{dt} = (\beta S(t) - k)I(t) \qquad (2)$$

$$\frac{dR}{dt} = kI(t) \qquad (3)$$

where k is the recovery rate (with k greater or equal to zero), β is the average number of transmissions from an infected person in a time period (with β greater or equal to zero), and

$$S(t) + I(t) + R(t) = N$$

SIR-LIKE MODELS

- **The SIS model with births and deaths**:

  The SIS model can be easily derived from the SIR model by simply considering that the individuals recover with no immunity to the disease, that is, individuals are immediately susceptible once they have recovered.

  S→I→S

  Removing the equation representing the recovered population from the SIR model and adding those removed from the infected population into the susceptible population gives the following differential equations:

$$\frac{dS}{dt} = -\frac{\beta SI}{N} + \mu(N - S) + \gamma I$$
$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I - \mu I$$

  where

  μ= Average death rate

  β= Contact rate/rate of infection

  1/ϒ= Average infectious period

- **The SIRS model:**

  This model is simply an extension of the SIR model as can be seen from its construction.

  S→I→R→S

  The only difference is that it allows members of the recovered class to be free of infection and rejoin the susceptible class.

$$\frac{dS}{dt} = -\frac{\beta SI}{N} + \mu(N - S) + fR$$
$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I - \mu I$$
$$\frac{dR}{dt} = \gamma I - \mu R - fR$$

  f=Average loss of immunity rate of recovered individuals

Using the SIR model for Ebola outbreaks

The Ebola virus disease was first discovered in 1976 in the present Democratic Republic of Congo. Since then, there have been many outbreaks, with the greatest being the current 2014 outbreak which has spread through many countries. The model uses two parameters which can be used calibrate it, β and γ with $\beta, \gamma > 0$. Given these parameters, the model uses 3 differential equations. These will be different numbers for any given disease and situation, and will depend on things like method of transmission, and the contact rate.

<u>Equation 1:</u> $\frac{dS}{dt} = -\beta IS$

In Equation 1, $\frac{dS}{dt}$ means the rate of change of the number of people susceptible to the disease over time. $\frac{dS}{dt}$ decreases proportionally to I because in order to become infected, you are no longer susceptible to the diseases any more. Since the only way to leave the set of susceptible people is through becoming infected with the disease itself, therefore the number of people who are susceptible to the disease is determined by the number of people who are already susceptible, the number of individuals who are already infected and the amount of contact between the susceptible and infected. An assumption is made that every individual has the same probability of becoming infected with the disease. In real life, this is highly improbable and it is a limitation. The equation also decreases proportionally to S because individuals are repeatedly being removed from the susceptible section and being transferred into the infectious section.

<u>Equation 2:</u> $\frac{dR}{dt} = \gamma I$

In equation 2, $\frac{dR}{dt}$ means the rate of change of the number of people recovered over time. This illustrates that the rate of the number of people recovering is dependent upon the number of people infected as in order to become recovered. This is because, in order to become recovered from a disease, one must have been infected at some point over a certain period of time and if the duration of time is shorter, then the rate of becoming infected increases. Therefore, this increases proportionally with the rate of the disease being infected.

Equation 3: $\frac{dI}{dt} = \beta IS - \gamma I$

In equation 3, $\frac{dI}{dt}$ means the rate of change of the number of people infected. This is dependent on the number of people susceptible and the number of people infected as well as the infection rate of the disease between the two compartments. As the population of I increases, the population of S decreases, therefore the rate at which $\frac{dI}{dt}$ increases is inversely proportional to the S because in order for there to be more infected people, there must be a decrease in the number of susceptible people. Thus, this equation is a consequence of the fact that: $\frac{dI}{dt} = -\frac{dS}{dt} - \frac{dR}{dt}$ into which we can substitute equation 1 and 2.

PARAMETERISATION OF THE MODEL

In order to calculate β (the rate of infection) and γ (the rate of recovery), it helps to define two more parameters.

D = Duration of disease for those recovered

$M =$ Mortality rate for those who die per day (*0.7 for Ebola*)

This leads to two further equations.

Equation 4: $\gamma = \frac{1}{D}$

In equation 4, the rate at which the disease is spread can be found by dividing 1 by the duration of the disease. This is because; a certain individual can only experience one recovery in a given period of time. For example if the duration of the infective period is 10 days, then the rate at which those who are infected become recovered is:

$$\frac{1}{10} = 0.1 = 10\%$$

Equation 5: $\beta = \frac{M}{S}$

Equation 5 illustrates that the infection rate of the disease is dependent upon the mortality rate and the number of people susceptible to the disease. It demonstrates the rate at which the disease passes from a susceptible individual to an infected individual. The value for β always lies between 0 and 1, because a value of 1 suggests 100% infection rate and a value of 0 suggests 0% infection rate. For example, if the mortality rate of the population is 50% and the number of people susceptible is 100, then the rate in infection will be calculated as follows:

$$\beta = \frac{0.5}{100} = 0.005$$

RUNNING THE SIR MODEL ON INITIAL FIGURES OF THE EBOLA OUTBREAK IN LIBERIA IN 2014

Taking the example of the Ebola outbreak in Liberia 2014, the parameters can be assigned with the following values. The total population of Liberia, N = 4294000, and according to data from WHO, the number of people infected, I = 846 and the number of people dead were 481. Seeing as R includes the number of people who have received permanent immunity, this includes those who have died as they have permanent immunity, in addition to those who have recovered with permanent immunity.

Therefore, number of people recovered $R = 481 + (0.3 \times 846) = 735$

$$N = 4294000$$

$$I = 846$$

$$R = 735$$

Therefore, $S = N - (I + R) = 4294000 - (735 + 846) = 4292419$

The duration of the disease ranges from 2 to 18 days, therefore a rough estimate of the duration of the disease is at the midpoint, i.e. 10 days.

$$D = 10$$

$$\gamma = \frac{1}{10} = 0.1$$

According to WHO, the mortality rate of Ebola is 0.7 and the number of people susceptible were 4292419.

Therefore from equation 5, β (the rate of infection) $= \frac{0.7}{4292419} = 1.63 \times 10^{-7}$

Numerical approach is used as follows. For each day, the values of $\frac{dS}{dt}, \frac{dI}{dt}$ and $\frac{dR}{dt}$ are calculated using equations 1, 2 and 3. Then it is assumed that the S value for the following day is the previous S value $+ \frac{dS}{dt}$ for that point in time.

This is done explicitly for the transition from t = 0 to t = 1. Using equations 1, 2 and 3 from earlier, the following values for the three rates of change of S, I and R can be calculated.

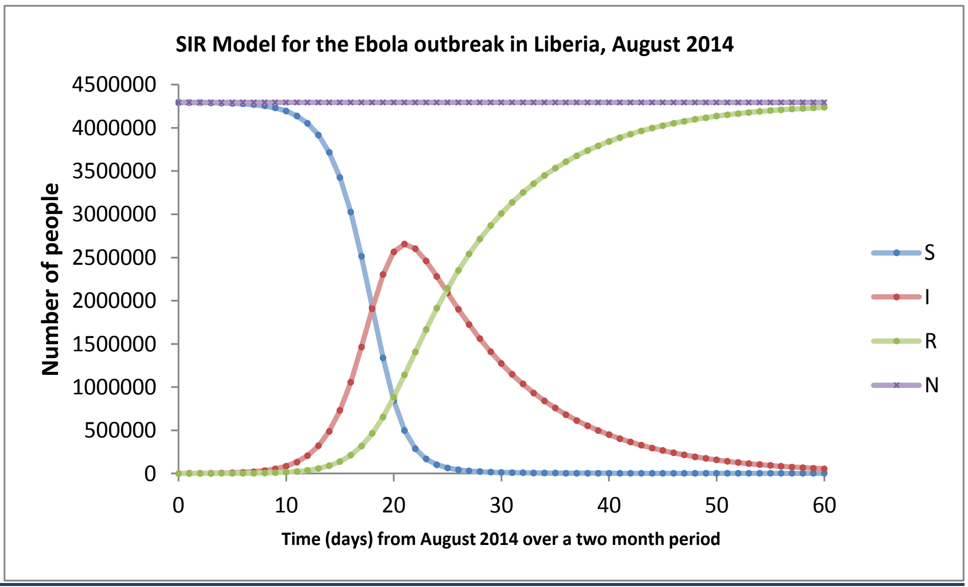
$$\left.\frac{dS}{dt}\right|_{t=0} = (-1.63 \times 10^{-7}) \times 846 \times 4292419 = -581$$

$$\left.\frac{dI}{dt}\right|_{t=0} = (1.6 \times 10^{-7}) - (0.1 \times 846) = 496$$

$$\left.\frac{dR}{dt}\right|_{t=0} = 0.1 \times 846 = 85$$

Therefore, at t=1, S = $4292419 - 581 = 4291838$

Following output was obtained when this numerical approach was carried out for Ebola over a period of 2-months.



PYTHON LIBRARIES AND CODE

For this project, we will be using mostly three libraries in Python namely: Matplotlib, Numpy and Scipy. The key features and functions which these libraries offer and which are important for us in this project are listed as follows.

Matplotlib

1. Plotting functions, numbers, set of numbers
2. Figures with navigation (Histograms, sinusoids, piecharts, constants etc)
3. Legends, Width etc
4. Shapes (circles, squares, dashes etc)
5. Text properties (labelling X & Y-axes, title etc)
6. Mathematical expressions (producing symbols using TeX subset)
7. Annotating axes
8. Image handling
9. Graphics

Numpy

1. Generating arrays
2. Various operations of arrays (Array mathematics)
3. Sum, product, mean, variance etc (a.sum(), a.prod(), a.mean() etc)
4. Sorting, max, min
5. Vector and matrix mathematics (dot & cross pdt etc)
6. Linear algebra (determinants, eigenvalues etc)
7. Polynomials (roots, equations, integration, differentiation, value)
8. Statistics (median, covariance etc)
9. Random number generation
10. Different distributions (gamma, normal, poisson etc)

Scipy

1. Fourier transform
2. Double integral
3. Triple integral
4. N-dimensional integral
5. Ordinary differential equation integrator
6. Interpolation
7. Linear algebra

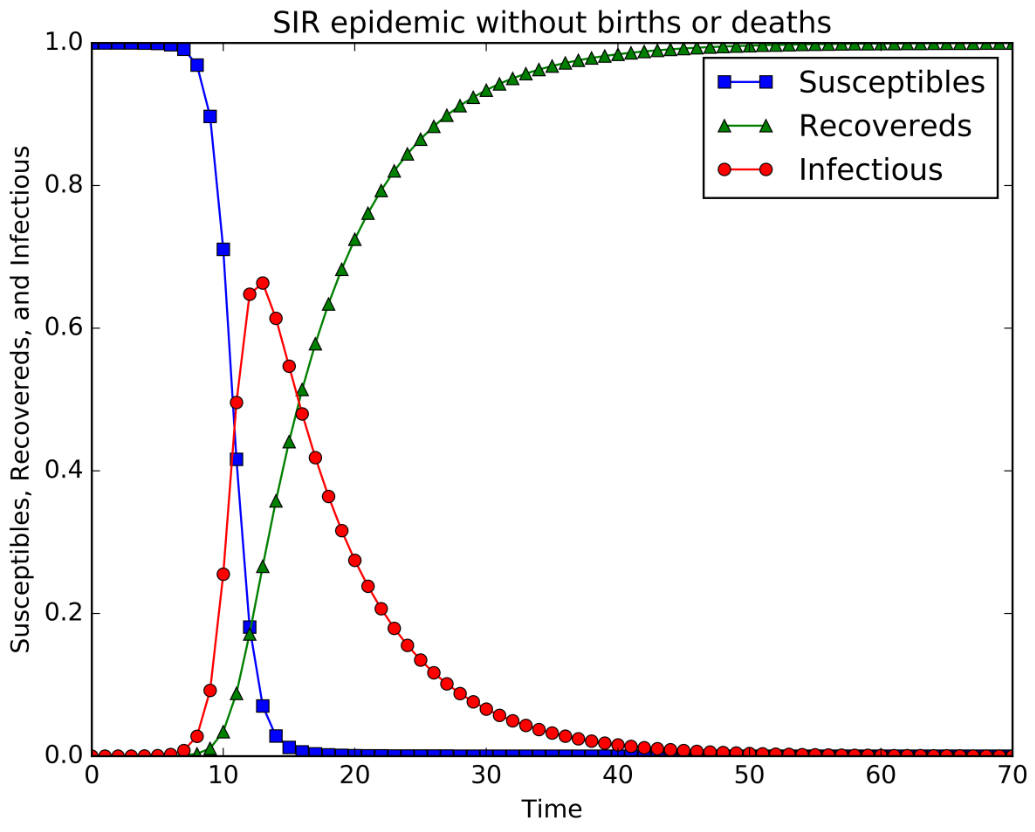
The documentation of these libraries is available online and is being studied side by side for reference in this project.

Python code 1

```
---------------------------------------------------------------------------------------------------------------
#To import python libraries

import scipy.integrate as spi

import numpy as np

import pylab as pl                                  #pylab is a module in matplotlib only


#Initialising the variables (which are used in the equations) to fixed values which have
already been studied

beta=1.4247

gamma=0.14286

TS=1.0

ND=70.0

S0=1-1e-6

I0=1e-6

INPUT = (S0, I0, 0.0)


#Writing down the equations

def diff_eqs(INP,t):

   '''The main set of equations'''

   Y=np.zeros((3)) #Y=[0,0,0]

   V = INP

   Y[0] = - beta * V[0] * V[1]

   Y[1] = beta * V[0] * V[1] - gamma * V[1]

   Y[2] = gamma * V[1]

   return Y   # For odient in scipy to solve the differential equation
```

```
t_start = 0.0; t_end = ND; t_inc = TS


#to build an array using numpy

t_range = np.arange(t_start, t_end+t_inc, t_inc)


#callable function,array(initial conditions),sequence of time points for which to solve for y

RES = spi.odeint(diff_eqs,INPUT,t_range)

print RES


#Ploting and labelling

pl.plot(RES[:,0], '-bs', label='Susceptibles')

pl.plot(RES[:,2], '-g^', label='Recovereds')

pl.plot(RES[:,1], '-ro', label='Infectious')

pl.legend(loc=0)

pl.title('SIR epidemic without births or deaths')

pl.xlabel('Time')

pl.ylabel('Susceptibles, Recovereds, and Infectious')

pl.savefig('2.1-SIR-high.png', dpi=900)

pl.show()
```

This code was written by *Cheng-Jun-Wang* and while studying the SIR and SIR-like models we have studied and compiled this code and have got the same results as follows.



Advanced level programming

This advanced level programming for the same includes two classes that we have defined in our code and are described as follows.

- ***SIRsystem***

  In this class, there are three functions namely, *__init__*, *reset and get_total_infected*.

  - ***__init__***: Initializes an instance of an SIRsystem, providing values for the model parameters beta and gamma, and the initial numbers of hosts in each of the model compartments (S = Susceptible, I = Infectious, R = Recovered). It also initializes internal instance variables t (time), S, I, R, N=S+I+R and a numpy array trajectory to hold [S, I, R] numbers at every time step. This is a base class for subsequent specialization, depending on whether one wants to simulate a deterministic model of SIR dynamics or not.

  - ***reset***: Resets the system by setting S, I, R and t to specified values, and reinitializes a trajectory array with this starting configuration.

  - ***get_total_infected***: Returns the total number of hosts either currently or previously infected.

- ***DeterministicSIRsystem***

  This is a specialized subclass of the general SIRsystem for modeling SIR dynamics with a deterministic differential equation model. This includes following functions.

  - ***dydt:*** Defines the right-hand-side of the equation dy/dt = dydt(y,t) for use with odeint integrator; and because this is defined as a method on the class rather than as a free-standing function, the first argument of the function is the self instance rather than the current state vector y.

  - ***run:*** Integrates the ODE for the deterministic model from time 0 to time T, starting with the initial values stored in the S,I,R state variables and stores the result in self.trajectory.

There are a few more functions which are defined in the code, the names and the use is described as follows.

- ***SimulateDeterministicOutbreakSize***
  For a given population size N and an array of basic reproductive numbers R0, this function integrates a DeterministicSIRsystem model for long enough time to allow the outbreak to die out, and then records the final size of the outbreak as a function of R0.

- ***CalculateDeterministicOutbreakSize***
  For a given population size N and an array of basic reproductive numbers R0, it solves an implicit equation for the final outbreak size using the fsolve root-finding routine. Also, it compares the simulated results found in SimulateDeterministicOutbreakSize with those solved for here.

- ***FractionLargeOutbreaks***
  For a given array of outbreak sizes, with total population size N, it calculates the fraction of those outbreaks that are 'large'. We are intended in outbreaks whose sizes constitutes a finite fraction of the population in the limit of infinite population size, but for a finite system size N, we can implement a heuristic cut-off separate 'large' from 'small' outbreaks; determining such a cut-off can be assisted by examining the outbreak size distribution.

- ***yesno***
  This function takes the input from the user and determines if the user wants to continue or not.

- ***demo***
  This function is responsible for giving the outputs in the form of plots using pylab library which is a subset of matplotlib. It calls other functions and gives the results and takes care of all the labelling and marking in the output figures.

Python code 2

```
--------------------------------------------------------------------------------------------------------------------------------

# Module for simulating infectious disease dynamics with the SIR (Susceptible-Infected-Recovered)
model

import numpy as np, scipy.integrate, scipy.optimize, sys

#########################################################################################

class SIRsystem:


    def __init__(self, beta, gamma, S, I, R):


        self.beta=beta

        self.gamma=gamma

        self.S=S

        self.I=I

        self.R=R

        self.t=0

        self.N=S+I+R

        self.trajectory=np.array([[self.S,self.I,self.R]],dtype=float)/self.N

        self.times=None




    def reset(self, S, I, R, t=0.):


        self.t=t

        self.S=S

        self.I=I

        self.R=R

        self.trajectory=np.array([[self.S,self.I,self.R]],dtype=float)/self.N
        def get_total_infected(self):
          return self.I + self.R
########################################################################################

class DeterministicSIRsystem (SIRsystem):
    def dydt(self, y, t):
        s,i,r=y
        dsdt=-self.beta * self.S * self.I
        didt=(self.beta * self.S * self.I) - (self.gamma * self.I)
        drdt= self.gamma * self.I
        return np.array([dsdt,didt,drdt])

    def run(self, T, dt=None):
        y0=np.array([[self.S,self.I,self.R]],dtype=float)/self.N
        if dt is None:
          self.times=np.linspace(0.,T,int(T+1),endpoint=True)
        else:
          self.times=np.arange(0,T,dt)
        self.trajectory=scipy.integrate.odeint(self.dydt,y0,self.times)
########################################################################################

def SimulateDeterministicOutbreakSize(N, R0_range=np.arange(0.,5.,0.1)):
    gamma = 1.0
    beta = 0.0
    Nf = float(N)
    dsir = DeterministicSIRsystem(beta, gamma, (N-1), 1, 0)
```

```
    sizes = []
    for R0 in R0_range:
        beta = R0 * gamma
        dsir.beta = beta
        dsir.reset((N-1),1,0,0.)
        dsir.run(100.)
        dsir.S, dsir.I, dsir.R = list(map(int, N*dsir.trajectory[-1]))
        R_inf = dsir.get_total_infected()
        sizes.append((R0, R_inf))
    return np.array(sizes)
##################################################################################

def CalculateDeterministicOutbreakSize(N, R0_range=np.arange(0.,5.,0.1)):

    func = lambda R_inf, R0: R_inf - (1.-np.exp(-R0*R_inf))
    sizes = []
    for R0 in R0_range:
        R_inf = scipy.optimize.fsolve(func, 0.5, args=(R0,))[0]
        sizes.append((R0, R_inf))
    return np.array(sizes)
#####################################################################################

def FractionLargeOutbreaks(osd, N):

    Nthresh = 0.1*N
    return (1.*np.sum(osd<Nthresh))/len(osd)
```

```
####################################################################################

def yesno():
    response = input('    Continue? (y/n) ')
    if len(response)==0:       # [CR] returns true
        return True
    elif response[0] == 'n' or response[0] == 'N':
        return False
    else:                     # Default
        return True
######################################################################################

def demo():
    import pylab
    N = 1000
    print("SIR demo")
    print("Deterministic SIR dynamics")
    pylab.figure(1)
    pylab.clf()
    dsir = DeterministicSIRsystem(1.5, 1.0, N-1, 1, 0)
    #dsir.run(30,0.1)
    pylab.plot(dsir.times, dsir.trajectory[:,0], 'b-', label='S')
    pylab.plot(dsir.times, dsir.trajectory[:,1], 'r-', label='I')
    pylab.plot(dsir.times, dsir.trajectory[:,2], 'g-', label='R')
    pylab.legend(loc='upper right')
    if not yesno(): return
    print("Deterministic outbreak size")
    R0_range = np.arange(0.,5.,0.1)

    simulated_sizes = SimulateDeterministicOutbreakSize(N=N, R0_range=R0_range)

    theoretical_sizes = CalculateDeterministicOutbreakSize(N=N, R0_range=R0_range)

    pylab.figure(2)

    pylab.clf()

    pylab.plot(R0_range, N*theoretical_sizes[:,1], 'b-', label='theory')

    pylab.plot(R0_range, simulated_sizes[:,1], 'bo', label='simulations')

    pylab.xlabel('R0')

    pylab.ylabel('total outbreak size')

    if not yesno(): return

    print("Stochastic SIR dynamics")

    pylab.figure(3)

    pylab.clf()

    pylab.plot(dsir.times, N*dsir.trajectory[:,1], 'r-', linewidth=2)

    for n in range(20):

        tfinal = ssir.run(20)

        pylab.plot(ssir.times, ssir.trajectory[:,1], 'b-')

    if not yesno(): return

##################################################################################

------------------------------------------------------------------------------------------------------------------------
```

The code which has been mentioned here is not complete, and the proper output has not been obtained. This is because, to get the outputs, one should know the exact theory, conditions and the initializations behind these models which were not our aim in this project. The working of every class and function has been studied and has been discussed. This code snippet can be used further if one wants to expand this to stochastic simulations of SIR models.

Summary

The SIR Model is used in the modelling of infectious diseases by computing the amount of people in a closed population that are susceptible, infected, or recovered at a given period of time. The model is also used by researchers and health officials to explain the increase and decrease in people needing medical care for a certain disease during an epidemic. From numbers generated by the SIR model researcher's health officials can calculate different numbers that allow them to see if policies are effective and if occurrence of the disease is increasing, decreasing, or stable. We have used *Ebola* as an example of how the SIR model works. However, the SIR model has some serious disadvantageous. The population has to be fixed and the population needs to mix homogeneously. The model does not take into account any variation in the disease among people of different sexes, races, or ages. The SIR model is the basis for other similar models. The SI model, also known as the SIS model, is the model where once a person is no longer infectious, this person becomes susceptible once again. The common cold can be modelled with the SI model. There is also the SEIR model, where people are categorized as susceptible, exposed, infected, or recovered. The SIR model can be adjusted to include variation due to seasonal changes.

Using more advanced level programming, this model can also be extended to the simulation of stochastic SIR models using Gillespie algorithm. Stochastic outbreaks size distribution can also be computed that generates N runs in different stochastic outbreaks in a population of size N with beta and gamma as model parameters.

References

- http://chengjun.github.io/en/2013/08/learn-basic-epidemic-models-with-python/
- https://en.wikipedia.org/wiki/Epidemic_model
- http://people.oregonstate.edu/~medlockj/
- Rhodes, John A., and Elizabeth S. Allman. Mathematical Models in Biology : An Introduction. New York: Cambridge UP, 2003. 280-301.
- Bauch, Chris, and David Earn. "Interepidemic Intervals in Forced and Unforced SEIR models." Dynamical Systems and Their Applications in Biology. Ed. Shigui Ruan, Gail S. Wolkowicz and Jianhong Wu. New York: American Mathematical Society, 2003. 33-43.
- "2014 Ebola Outbreak in West Africa." *Centers for Disease Control and Prevention*. Centers for Disease Control and Prevention, 06 Mar. 2015. Web. 07 Nov. 2014.
- Dolgoarshinnykh, Regina, Columbia University, Steven P. Lalley, and University Of Chicag. "Epidemic Modeling: SIRS Models." *Epidemic Modeling: SIRS Models* (n.d.): n. pag. Web.
- "Ebola Virus Disease." *WHO*. N.p., n.d. Web. 17 Nov. 2014.
- "Ebola Virus Disease." *WHO*. N.p., n.d. Web. 18 Nov. 2014.