ISEN 609 - PROBABILITY FOR ENGINEERING DECISIONS

Vishwam Patel
Naveen Veluchamy

December 8, 2020

Introduction

The susceptible-infected-susceptible(SIS) stochastic model is used to analyze the probability and distribution of transmission of the epidemic by a population without considering births and deaths. As $I(t) \in S = [0, 1, 2, ...N]$ denotes the number of infected individuals at time $t > 0$ in a population of $N > 1$, this process is modeled as a continuous time Markov Chain.

SIS model is a function of $\rho = \frac{\lambda}{\mu}$, where λ is the probability of infection and μ is the recovery probability. When $\rho = \frac{\lambda}{\mu} > 1$ the disease will be in the epidemic state. We are going to analyze the behavior of distribution for different values of λ and μ, which impacts the analysis of Transmission of the disease.

Infection Rate is given by $\lambda i(1 - \frac{i}{N})$ and Recovery Rate is given by μi

The Quasi steady state distribution q_j is defined as $q_j = lim_{t\to\infty} P_r(I(t) = j|I(t) > 0)$. That follows $q_j(t) = \frac{P_j(t)}{1-P_0(t)}$

Part 1 Infinitesimal Generator

We have to calculate the rate of Probability, to get the Infinitesimal Generator (A). We have $\frac{dP_0}{dt} = 0$ and $\frac{dP_1}{dt} = \mu$.

$$A = \begin{pmatrix} 0 & & & \cdots & \\ \mu & -(\mu + \lambda(1 - \frac{1}{N})) & \lambda(1 - \frac{1}{N}) & \cdots & \\ 0 & 2\mu & -2(\mu + \lambda(1 - \frac{2}{N})) & \cdots & \\ \vdots & & \ddots & \cdots & \\ & \cdots & 0 & N\mu & -N\mu \end{pmatrix}$$

Part 2 - Proof

$$Prove,$$

$$\frac{dq(t)}{dt} = q(t)\overline{A} + \mu q_1(t)q(t)$$

$$Proof,$$

$$q_j(t) = \frac{P_j(t)}{1 - P_0(t)}$$

$$\frac{dP_0(t)}{dt} = \mu P_1(t)$$

$$\frac{dP(t)}{dt} = P(t).A$$

$$\frac{dq_1(t)}{dt} = \frac{d}{dt}(\frac{P_1(t)}{1 - P_0(t)})$$

$$= \frac{\frac{dP_1(t)}{dt}}{1 - P_0(t)} + P_1(t)\frac{1}{(1 - P_0(t))^2}\frac{dP_0(t)}{dt}$$

$$= \frac{dP_1(t)}{dt}\frac{1}{1 - P_0(t)} + P_1(t)\frac{1}{(1 - P_0(t))^2}\mu P_1(t)$$

$$\frac{dq(t)}{dt} = \frac{d}{dt}(\frac{P_j(t)}{1 - P_0(t)})$$

$$= \frac{dP_j(t)}{dt}\frac{1}{1 - P_0(t)} + \frac{dP_0(t)}{dt}\frac{P_j(t)}{(1 - P_0(t))^2}$$

$$= \frac{dP_j}{dt}\frac{1}{1 - P_0(t)} + \mu\frac{P_j(t)}{1 - P_0(t)}\frac{P_1}{1 - P_0(t))}$$

$$= \frac{dP_j(t)}{dt}\frac{1}{1 - P_0(t))} + \mu q_1(t)q(t)$$

$$\frac{dP_j(t)}{dt} = P_j(t) \times \overline{A}$$

$\overline{A}$ is the $N \times N$ matrix formed from A by deleting the first row and first column

Therefore,

$$\frac{dq(t)}{dt} = \overline{A}\frac{P_j(t)}{1 - P_0(t)} + \mu q_1(t)q(t)$$

$$\frac{dq(t)}{dt} = q(t)\overline{A} + \mu q_1(t)q(t)$$

Hence Proved.

Part 3 - Eigenvalue and q

As we want to get a Steady-State solution of the equation from part 2. We know,

$$\frac{dq(t)}{dt} = 0$$

We have,

$$\frac{dq(t)}{dt} = q(t)\overline{A} + \mu q_1(t)q(t)$$

$$0 = q\overline{A} + \mu q_1 q$$

$$q\overline{A} = -\mu q_1 q$$

Here $-\mu q_1$ is the largest eigenvalue of A. With a given value of q_1, this system is reduced from quadratic to linear equations. Largest eigenvalue of $\overline{A}$ for different values of lambdas are as follows:

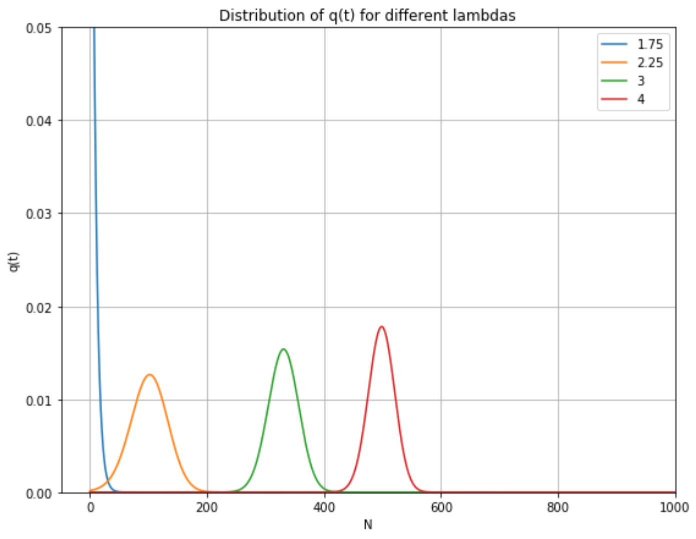
λ	Largest Eigenvalue
1.75	0.2723
2.25	0.00036
3	2.3e-12
4	2.05e-12

q is calculated from the system of linear equations using the python software, which is attached as appendix.

Part 4 - Distribution of q

In this analysis we created a plot of distribution q for $N = 1000$, $\mu = 2, \lambda = 4, 3, 2.25, 1.75$ assuming $P_{10}(0) = 1$. As λ and μ are following exponential distribution, whenever $\frac{\lambda}{\mu} > 1$, the disease will be in epidemic state.

The curve of Probability Distribution will increase exponentially. As we can see from the graph, when $\lambda = 4, 3, 2.25$ are used, distribution q increases exponentially. When $\lambda = 1.75$ is taken, $\frac{\lambda}{\mu} < 1$, the slope of q is decreasing, which means there is a decrease in transmission of the disease.Whenever $\frac{\lambda}{\mu} > 1$, the probability distribution increases exponentially, which is shown by the graph. Also a little change in the value of λ, shows a huge amount of change in Probability Distribution.

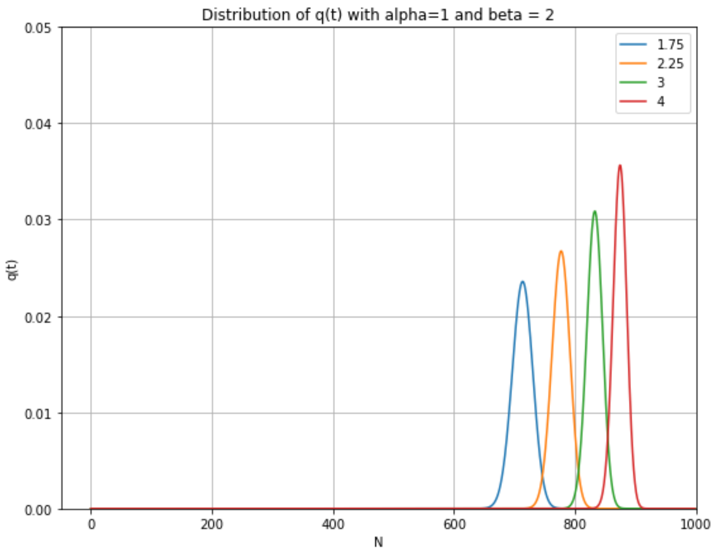


Part 5 Non-Memoryless Recovery

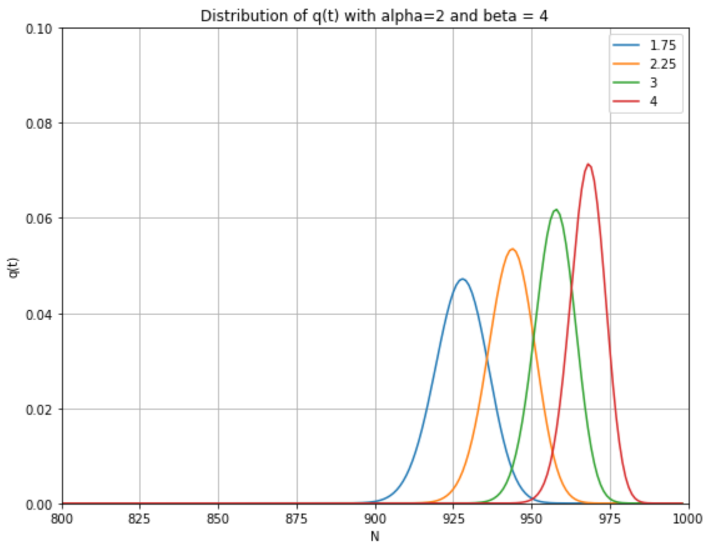
Non-Memoryless Recovery SIS epidemic model is a queuing process with state dependent arrival rate, which means infected individuals are treated with a rate μ and the arrival rate is $\lambda(i) = \lambda.i.\frac{N-i}{N}$, where $i \in (1, 2, ..., N)$ and $\lambda(0) = \lambda$. This means the state 0 is reflected back to state 1 at rate λ

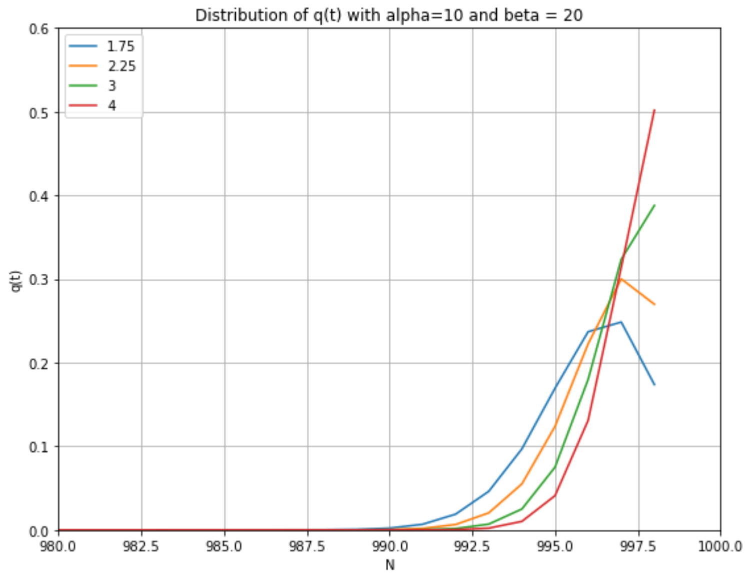
The distribution q is given by $q_i = \frac{\pi_i}{1-\pi_0}$, where π_0 is the condition of not being in state 0.

For analysis of Distribution q, we used scale parameters as $(\alpha, \beta) = ((1,2),(2,4),(10,20))$ and $\lambda = 4, 3, 2.25, 1.75$, assuming $P_{10}(0) = 1$.



As it can be seen from the graphs with different scale parameters α and β the curve of distribution q is changed. The distribution q depends on γ and scale parameters (α, β). For $(\alpha, \beta) = (1, 2)$ when $\lambda = 1.75$ the curve does not increase as much as compared to other λ values, which means infection rate is smaller in comparison with other λ. When α and β value increase, the distribution q values also increase





suggesting an increase in infection rate. The highest q can be seen in the graph of $(\alpha, \beta) = (10, 20)$. The infection rate is much higher than the recovery rate in $(\alpha, \beta) = (10, 20)$, detecting a much slower decrease in distribution q compared to $(\alpha, \beta) = ((1, 2), (2, 4))$.

Comparison between Part 4 and Part 5

Part 4 involves a Memoryless Recovery stochastic model and Part 5 uses a Non-Memoryless Recovery mode. As it can be seen from the graphs of Part 5 that the distribution q curves for different λ values are generated on higher N values compared to Part 4 graph. The infection rate is much higher than the recovery rate in this type of model, so we observe a higher amount of infected individuals N. This means recovery in Part 5 model takes longer time and the value of N is higher. When $(\alpha, \beta) = (10, 20)$ was taken, the curve of distribution q did not decrease, as seen from Part 4 graphs.

Code

```
import numpy as np #importing required packages
import matplotlib.pyplot as plt
import math
from decimal import Decimal
from functools import reduce
# known parameters
N      = 1000
mu     = 2
lamda = [1.75, 2.25, 3, 4]
alpha = [1, 2, 10]
beta  = [2, 4, 20]
def findq(N, mu, lamda):
    A = np.zeros((N+1,N+1)) # infinitesimal generator
    for i in range(N+1):
        for j in range(N+1):
            if i == 0 and j == 0:
                A[i,j] = 0
                continue
            if i == N and j == N:
                A[i,j] = round(-(N) * mu,4)
                continue
            if i == N and j == N-1:
                A[i,j] = round((N) * mu,4)
                continue
            if i != N and i == j:
                A[i,j] = round(- (((i) * mu) + (((i) * lamda) * (1 - ((i)/(N))))),4)
                continue
            if i != N and j == i+1:
                A[i,j] = round((i * lamda) * (1 - (i/(N))),4)
                continue
            if i != N and j == i - 1:
                A[i,j] = round((i) * mu,4)
                continue
    # Removing the first row and column
    A_b = A[1:,1:] # N x N
    u,v = np.linalg.eig(A_b) # eigen values
    uReal = []
    for i in range(len(u)):
        if isinstance(u[i], complex): # removing complex numbers
            uReal.append(u[i].real)
        else:
            uReal.append(u[i])
    q1 = -max(uReal)/mu # largest eigenvalue
    # formulating the system of equations
    # q\overline{A} = -\mu q_1q
    A_bar = np.transpose(A_b)
    A_eqn = A_bar + (np.identity(len(A_bar))*mu*q1)
    q = []
    for i in range(len(A_eqn)):
        if i == 0:
            q.append(q1)
            q.append(-A_eqn[0,0]*q1/A_eqn[0,1])
        elif i == len(A_eqn)-1:
            pass
        else:
            q.append((-A_eqn[i,i-1]*q[i-1]-A_eqn[i,i]*q[i])/A_eqn[i,i+1])
    qsum = sum(q)
    qt = q/qsum
    return qt
def plotd(lamda):
    for i,j in enumerate(lamda):
        qt = findq(N,mu,lamda[i]) # iterating over different lambdas
        plt.rcParams['figure.figsize'] = (9,7)
        plt.plot(qt)
    plt.xlabel("N")
    plt.ylabel("q(t)")
    plt.title("Distribution of q(t) for different lambdas")
    plt.legend(lamda)
    plt.grid()
    plt.axis([-50.0, 1000, 0.0, 0.05]) # [xstart, xend, ystart, yend]
    plt.show()
plotd(lamda)
```

```
def findPi(N,lamda,mu):
    Pi_list = []
    Pi_list.append(1) # P10(0) = 1
    Pi_list.append(lamda)
    lamda_list = []
    lamda_list.append(lamda) # \lambda(0) = \lambda
    lamda_list.append(lamda*(N-1)/N)
    for i in range(2,N):
        mean_pow = Decimal(mu)**Decimal(i-1)
        ldec = list(map(Decimal,lamda_list))
        y = reduce(lambda x, y: x*y, ldec)
        numerator = mean_pow * y
        ifact = Decimal(math.factorial(i)) # factorial stores long float leading to stor
        pi_i = Decimal(numerator)/Decimal(ifact)
        Pi_list.append(pi_i)
        L = lamda*i*(1 - (i/N)) # \lambda(i) = \lambda.i((N-i))/N
        lamda_list.append(L)
    pi_sum = np.sum(Pi_list)
    pi_0 = 1/pi_sum
    Pi_sec = [j * pi_0 for j in Pi_list]
    Pi = Pi_sec[1:]
    return Pi
alpha0 = alpha[0]
beta0 = beta[0]
mul = alpha0*beta0
lamda = [1.75, 2.25, 3, 4]
def plotdg(lamda):
    for i,j in enumerate(lamda):
        Pi = findPi(N,mul,lamda[i])
        plt.rcParams['figure.figsize'] = (9,7)
        plt.plot(Pi)
    plt.xlabel("N")
    plt.ylabel("q(t)")
    plt.title("Distribution of q(t) with alpha=1 and beta=2")
    plt.legend(lamda)
    plt.axis([-50.0, 1000, 0.0, 0.05])
    plt.grid()
    plt.show()
plotdg(lamda)
alpha0 = alpha[1]
beta0 = beta[1]
mul = alpha0*beta0
```

```
def plotdg(lamda):
    for i,j in enumerate(lamda):
        Pi = findPi(N,mul,lamda[i])
        plt.rcParams['figure.figsize'] = (9,7)
        plt.plot(Pi)
    plt.xlabel("N")
    plt.ylabel("q(t)")
    plt.title("Distribution of q(t) with alpha=2 and beta=4")
    plt.legend(lamda)
    plt.axis([800.0, 1000, 0.0, 0.1])
    plt.grid()
    plt.show()
plotdg(lamda)
alpha0 = alpha[2]
beta0 = beta[2]
mul = alpha0*beta0
lamda = [1.75, 2.25, 3, 4]
def plotdg(lamda):
    for i,j in enumerate(lamda):
        Pi = findPi(N,mul,lamda[i])
        plt.rcParams['figure.figsize'] = (9,7)
        plt.plot(Pi)
    plt.xlabel("N")
    plt.ylabel("q(t)")
    plt.title("Distribution of q(t) with alpha=10 and beta=20")
    plt.legend(lamda)
    plt.axis([980.0, 1000, 0.0, 0.6])
    plt.grid()
    plt.show()
plotdg(lamda)
```

Contributions

```
Part 1      - Vishwam
Part 2      - Naveen
Part 3      - Vishwam
Part 4      - Naveen, Vishwam
Part 5      - Vishwam
Comparison  - Naveen
```