THE MIELKE BETA-KAPPA DISTRIBUTION AS AN APPROXIMATION OF MUTATING EPIDEMICS

JONATHON D'ARCY

Abstract. In this report we looks at two classical representations of epidemics, the Birth-death model and the SIR model. We begin with the Birth-death model, and look to understand the distribution that governs it and its properties. We then pivot to gather an understand about how mutations effect this and extend this idea to the SIR Model. Finally, we look to model the effect that a vaccine has on the SIR Model.

1. Introduction

During the course of an epidemic it is of premier importance to ascertain key information about how a disease will progress. This sometimes in classical models can be made difficult as many infected individuals will act differently when infected with the virus; some may self-isolate, others may suffer from anosognosia and go to work. To better understand the effects these different responses can have we in this paper look to splitting our infections into mutants which can represent a wide category of things but most applicably, reactions to becoming infected.

2. Super-Critical Birth-Death Process

It is important to ask in viral infection modelling how many people will be infected with the virus at a certain time. We represent this value as I_t where t is the indexing time. To determine this we first have to choose a method of modelling the epidemic. Here, we will model the epidemic using the super-critical birth-death process. In general, the process works by an infected body possessing the abilities to become cured at a rate β and to infect a new body at rate α. The super-critical modifier simply means that the rate of infection for a given virus is greater than the rate of recovery. To simulate the time take between actions of the model we use a Poisson Process, giving generally that $\mathbb{E}(I_t) = e^{(\alpha-\beta)t}$.

We look specifically at the case of a virus infecting individuals at three times the rate at which it is cured, $\alpha = 0.75$ and $\beta = 0.25$. In Figure 9, we model the expected trend of I_t against 25 simulations of the virus. It can be seen that these plots are exponential in nature, thus in Figure 10, we provide a semi-log plot of the same simulations. Here, we can clearly see that while early in time these can differ greatly from the mean, as time progresses they more similarly follow the shape of the curve of the expectation. The reason for this is that as we introduce more infected individuals, the lower the rate of the individual Poisson distribution affecting each time step.

To study the distribution of the time taken to reach a given number of infections we can simply rearrange the earlier equation of $\mathbb{E}(I_t) = e^{(\alpha-\beta)t} \implies \mathbb{E}(t_I) = \frac{ln(I)}{(\alpha-\beta)}$, here t_I denotes the time taken to reach I infected. We can simulate this value by executing 5000 runs of our $\alpha = 0.75$ $\beta = 0.25$ model with focus on the total time passed until we have 1000 infected individuals, the resulting distribution is seen in Figure 1. This histogram looks at a cursory glance to be well approximated by the Gamma distribution, however we find that this does not fit the kurtosis of the data. We instead use the Weibull distribution which very accurately fits the distribution as seen in Figure

2. The true basic reproductive ratio is $\frac{0.75}{0.25} = 3$, under simulation we can approximate this ratio by dividing number infected against number cured, $\frac{60063}{20013} = 3.0011992$.

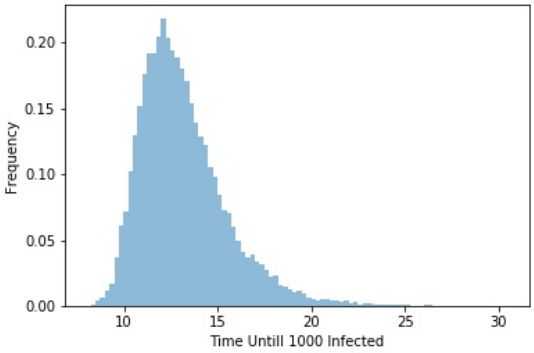

FIGURE 1. Normalized Histogram of 20000 runs of $\beta = 0.75$ $\alpha = 0.25$ Birth-death with Poisson Point Process denoting frequency of time taken to reach 1000 infected.

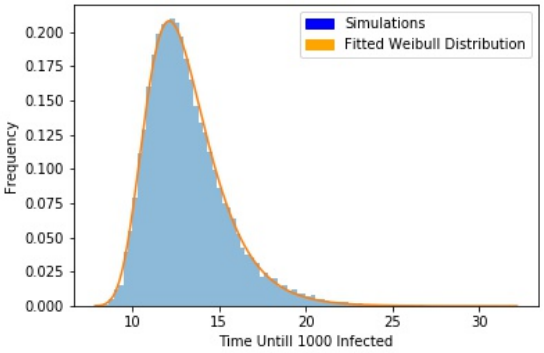

FIGURE 2. Normalized Histogram of 20000 runs of $\beta = 0.75$ $\alpha = 0.25$ Birth-death with Poisson Point Process denoting frequency of time taken to reach 1000 infected, fitted to a Weibull Distribution.

3. Super-Critical and Sub-Critical

We now look at the case where the virus can mutate from strain I to strain J during transmission. That is to say in branching notation that,

$$I \xrightarrow{\alpha} II, \quad I \xrightarrow{\beta} \varnothing, \quad I \xrightarrow{\mu} IJ \tag{1}$$

We begin by studying the first appearance of strain J, against three different measures: Current Infections, Total Infections, and Time. To analyse the appearance against these measures we begin by simulating a million epidemics for three types characteristics:

$$\text{Type 1: } \alpha = 3/4 - 1/10, \quad \beta = 1/4, \quad \mu = 1/10 \tag{2}$$

$$\text{Type 2: } \alpha = 3/4 - 1/20, \quad \beta = 1/4, \quad \mu = 1/20 \tag{3}$$

$$\text{Type 3: } \alpha = 3/4 - 1/50, \quad \beta = 1/4, \quad \mu = 1/50 \tag{4}$$

The distribution of these simulations against the three different metrics can be seen in Figures 3, 4, 11, and 12 where they are fitted to a variety of distributions. In Figure 3 and 4, we show the distribution of the first mutation against time for each of the runs. We have from [1] that

$$\mathbb{P}(T > t) = \frac{1}{1 + \frac{u\alpha}{\lambda^2} e^{\lambda t}} \tag{5}$$

From this we can derive the pdf of the distribution,

$$f_T(t; \mu, \alpha, \beta) = \frac{d}{dt}\mathbb{P}(T > t) = \frac{-\alpha\lambda^3\mu e^{\lambda t}}{(\alpha\mu e^{\lambda t} + \lambda^2)^2} = \frac{\alpha\mu e^{\lambda t}}{\lambda(1 + \frac{\alpha\mu}{\lambda^2} e^{\lambda t})^2}. \tag{6}$$

We see this fits the distribution in Figure 3. One may notice the structural similarity of this pdf to the Mielke's beta-kappa Distribution, a class of Dagum Distribution, whos pdf is given by,

$$f_T(t;k,\theta) = \frac{kt^{k-1}}{(1+t^\theta)^{1+\frac{k}{\theta}}}. \tag{7}$$

When fitted to this distribution we get a residual sum of squares (RSS) value of 0.00819, 0.006452, and 0.00079832 each respective initial condition, this fitting is shown in Figure 4. In Figure 11, we study this value of the number of infected individuals when the mutation occurs, and in Figure 12 we study the total number of individuals which had been infected when mutation occurs. These are both clearly discrete distributions and governed by the Poisson distribution.

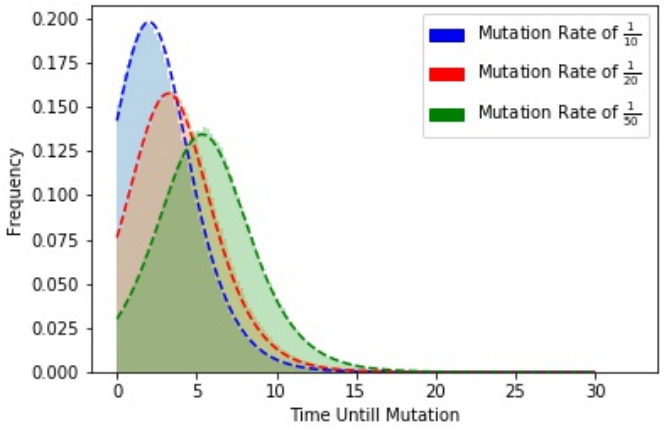

FIGURE 3. 1,000,000 Runs Bounded by True PDF

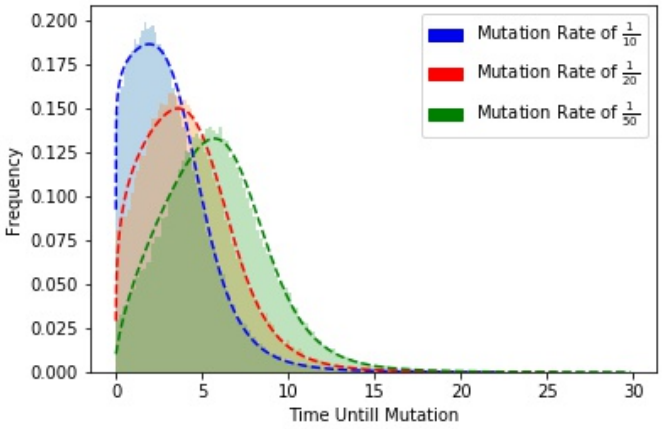

FIGURE 4. 1,000,000 Runs Bounded by Best Fit Miekle Distribution

Let us now look past the time of first mutation and give characteristics to the strain J. Let us say that an individual infected with strain J cannot spread the epidemic but still recovers at the same rate, β. This is to say in branching notation,

$$I \xrightarrow{\alpha} II, \quad I \xrightarrow{\beta} \varnothing, \quad I \xrightarrow{\mu} IJ, \quad J \xrightarrow{\beta} \varnothing \tag{8}$$

We can represent this instead as,

$$I \xrightarrow{p\alpha} II, \quad I \xrightarrow{\beta} \varnothing, \quad I \xrightarrow{(1-p)\alpha} IJ, \quad J \xrightarrow{\beta} \varnothing \tag{9}$$

Where p denotes the probability that on infection the newly infected individual will be strain I. It is known that in the standard birth death model the virus will only be expected to survive indefinitely if the rate of infection supersedes the rate of death. Thus we can calculate that for strain I to survive we require

$$p\alpha > \beta \implies RR > p, \tag{10}$$

where RR denotes the reproductive ratio of the virus. If we take this further an say that the following,

$$I \xrightarrow{p\alpha} II, \quad I \xrightarrow{\beta} \varnothing, \quad I \xrightarrow{(1-p)q\alpha} IJ, \quad I \xrightarrow{(1-p)(1-q)\alpha} II, \quad J \xrightarrow{\beta} \varnothing \tag{11}$$

We can show that for strain I long term expected survival is predicated by,

$$p\alpha + (1-p)(1-q)\alpha > \beta \implies RR > \frac{1}{q(p-1)+1}. \tag{12}$$

4. SIR with Mutation

We now study virus mutation in a finite population, specifically under the SIR model. We recall that in the standard SIR model we begin with M individuals who are susceptible to catching a virus, I, that one member of the population has. We say that this infected individual, and any subsequently infected individuals, will infect other susceptible individuals at a rate α/M and recover at rate β. Once an individual has recovered from the virus they become immune from contracting the virus again. To implement mutations into this model we denote a strain J, which at the start of the model no individuals have. This strain however can arise during the transmission of the original strain I, to a susceptible individual with probability μ. Strain J we note cannot mutate to strain I, but behaves identically to I in all other ways. Each virus strain acts independently and a recovery from one virus strain still keeps a member of the population at risk of the other virus, further an individual could be infected by both viruses at once. In Figures 5 and 6 we see possible realization of this model with parameters $\alpha = 3/4$, $\beta = 1/4$, $\mu = 1/100$, and respective population sizes of 100,000 and 100.

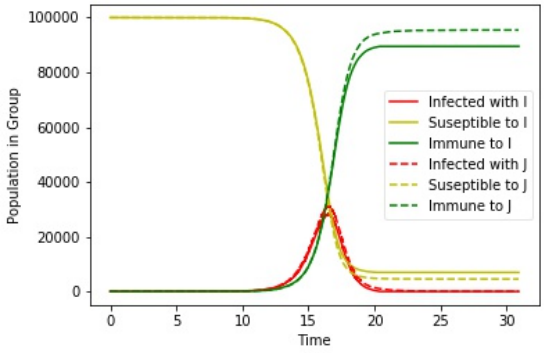

Figure 5. Mutable SIR Model (M, α, β, μ)=(100000, 3/4, 1/4, 1/100)

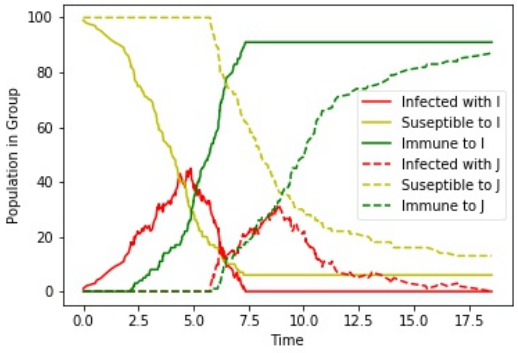

Figure 6. Mutable SIR Model (M, α, β, μ)=(100, 3/4, 1/4, 1/100)

To better understand this model we will now look at the distribution of when the number susceptible individuals is equal to that of infected individuals, conditioned on the event happening. We simulate this distribution in Figures 13-15, and find that it fits to the χ^2-Distribution. In Figure 16 we compare the provability of a virus emerging against its mutation rate for varying infection rates. In Figure 17, we simulate for 500,000 runs the average number of infected individuals when mutation first occurs against mutation rate for three different infection rates. In Figure 18, we simulate the average number of infected individuals at a given time in a population sized 1000 with an infection rate of 75/100 and a mutation rate of 1/100.

5. Vaccine

Here we aim to show the effects of the a promptly developed vaccine can have on an epidemic. To show this we will continue to use the previously created SIR with Mutation model, however we will have three new parameters, $(V_{eff}, V_{per}, V_{devtime})$. $V_{eff} \in \mathbb{R}_{(0,1)}$, denotes the effectiveness to which the vaccine is distributed when created, the larger V_{eff} the large proportion of the susceptible

population can access the vaccine in a given time step. V_{per} denotes the pertinence to which the vaccination began development, this is measured by the size of the infected populous. $V_{devtime}$ denotes the time required to create the vaccine once development begins. As we have a mutation in this model we will say that they require two fully independent vaccines that must be developed. In Figures 7 and 8, we see the implementation of a vaccine of type $(0.05, M/100, 2)$ on the mutation models observed in the previous section. With these in place we see a drastic reduction in average number of infections with a specifically large effect on strain J, as it takes longer to build up allowing dissemination into the population faster. What we find here from studying these models is that the development of a vaccine is on worth the effort if the time from the start of creation to the final development does not exceed the width of the infection curve at that time.

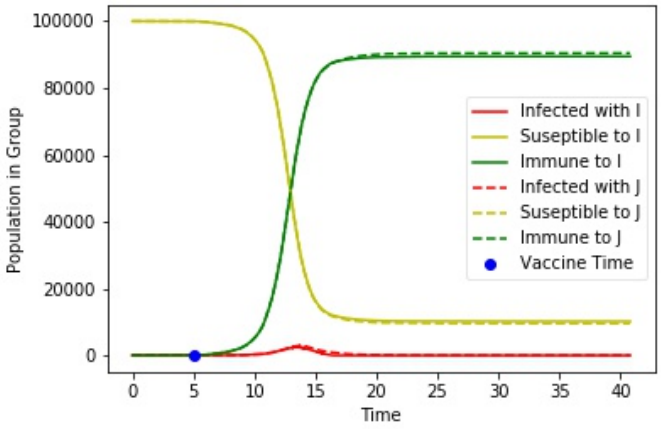

FIGURE 7. SIR with Vaccine (M, α, β, μ)=$(100000, 3/4, 1/4, 1/100)$ Vaccine=$(0.05, 1000, 2)$

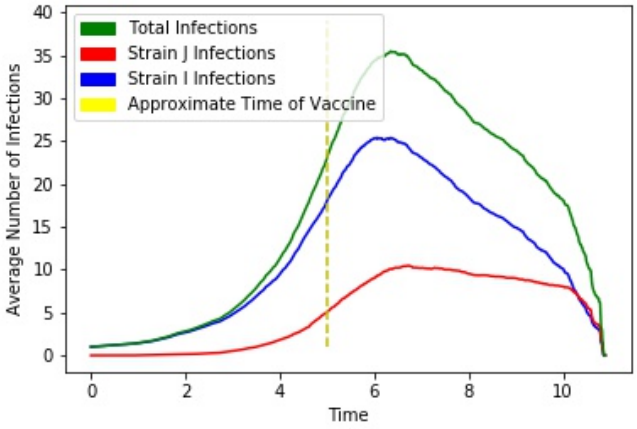

FIGURE 8. SIR with Vaccine (M, α, β, μ)=$(1000, 3/4, 1/4, 1/100)$ Vaccine=$(0.05, 5, 2)$

6. Conclusion

Throughout this report we have identified many key attributes of the key mathematical epidemiology models. There however is still much work that can be done, we find particular interest in finding if there is a relation between the parameters of the fitted Miekel Distribution and the pdf found, or if some good approximation exists. Further, there are many routes for further expansion both via simulation and analysis on the effects of vaccines in these models.

References

[1] Lecture notes
[2] *NIST Digital Library of Mathematical Functions*: F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds.

7. Figures

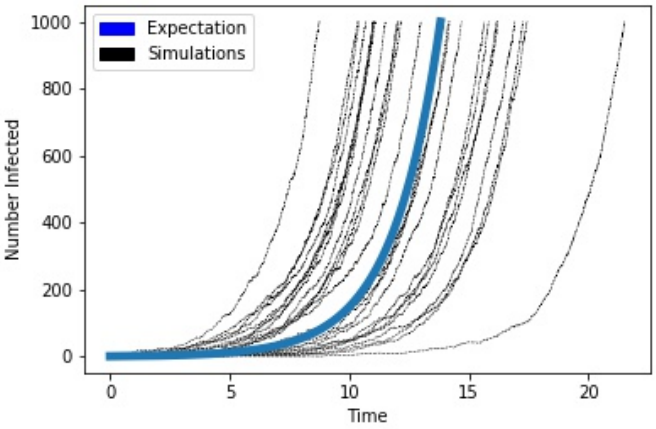

Figure 9. 25 simulations and expectation of outbreak size against time

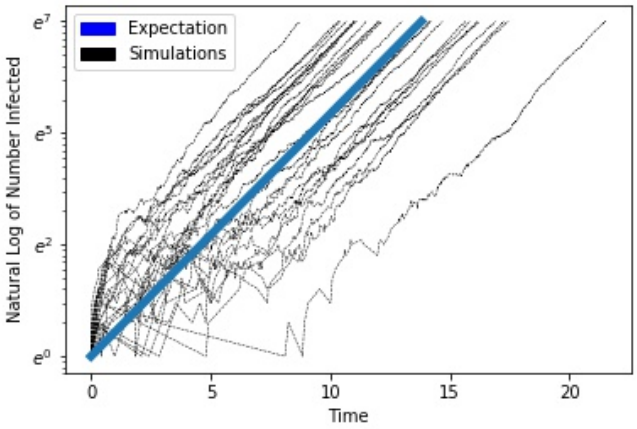

Figure 10. Semilog of Figure 1

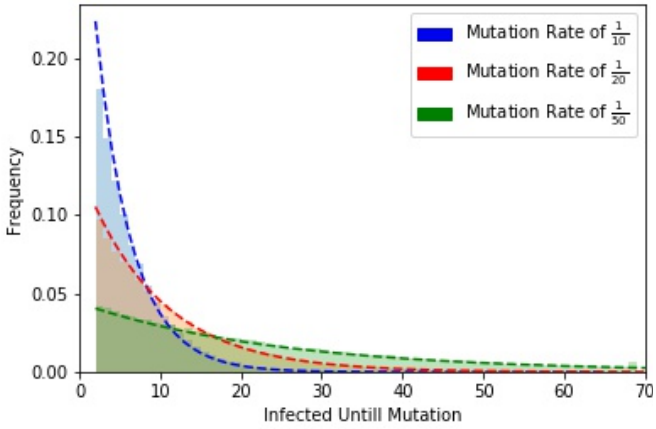

Figure 11. 1,000,000 Runs Histogram of Infected at Mutation Bounded by fitted pdf

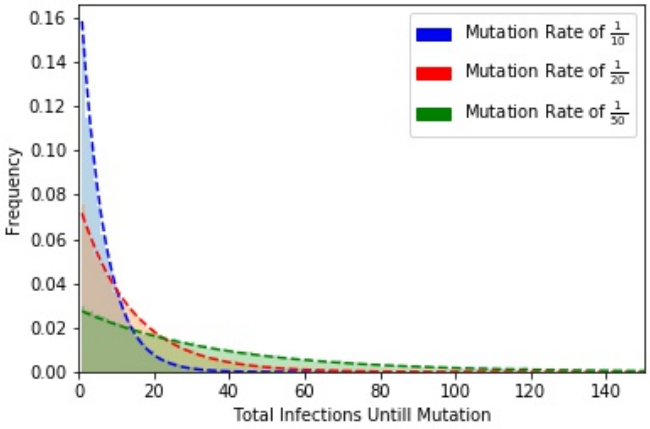

Figure 12. 1,000,000 Runs Histogram of Infected at Mutation Bounded by fitted pdf

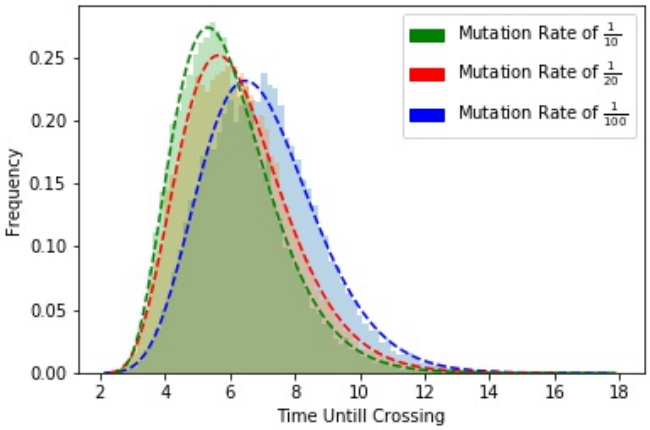

FIGURE 13. 25000 Run Histogram of Crossing Times with population 500 and $\alpha = 95/100$

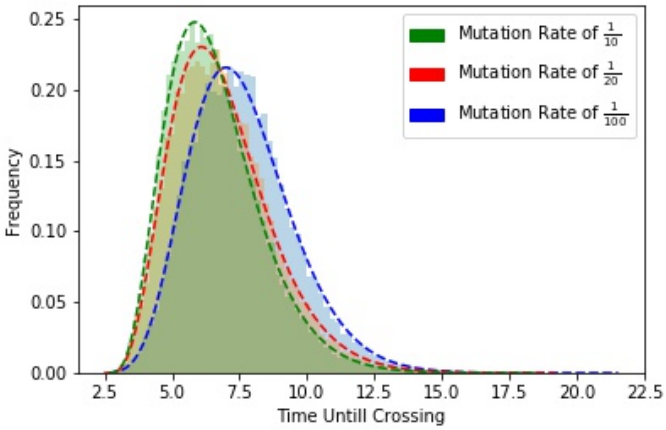

FIGURE 14. 25000 Run Histogram of Crossing Times with population 500 and $\alpha = 90/100$

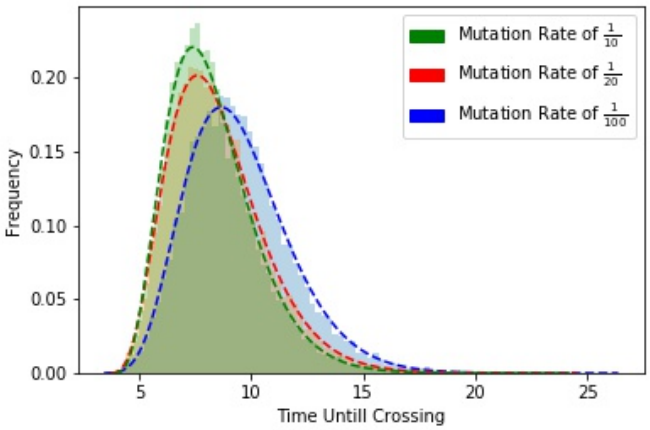

FIGURE 15. 25000 Run Histogram of Crossing Times with population 500 and $\alpha = 80/100$

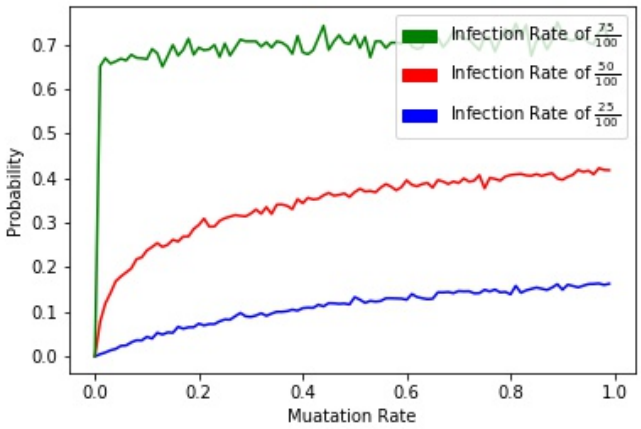

FIGURE 16. Probability of epidemic mutating for various μ and α

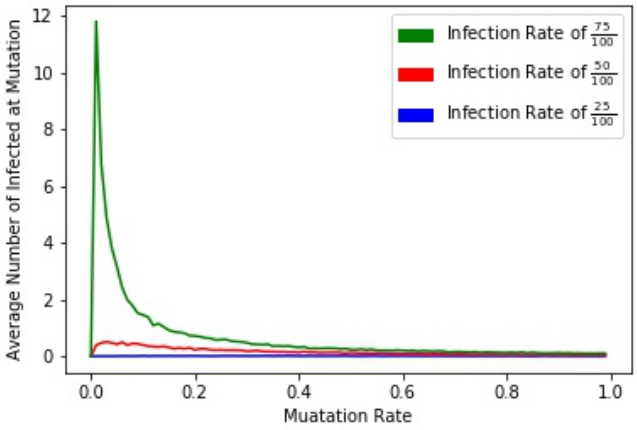

FIGURE 17. Average number of Infected individuals at mutation for various μ and α

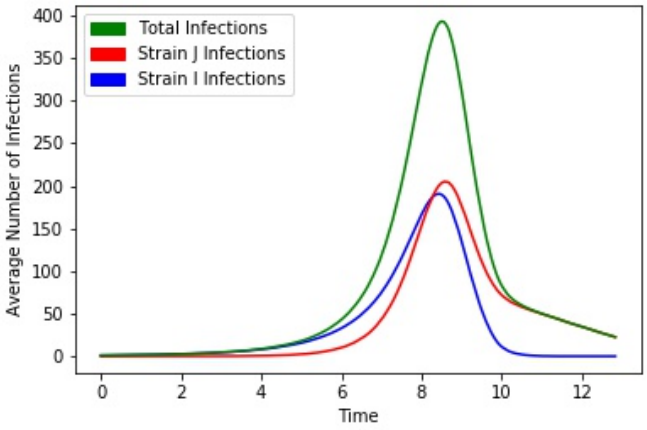

FIGURE 18. Average number of total infections against time

8. Python Code

MiA

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 26 14:22:51 2020

@author: jonny
"""

import math, random as rnd, numpy as np, matplotlib.pyplot as plt, ...
    matplotlib.patches as mpatches, matplotlib.ticker as mtick
import statistics, scipy.stats
import sympy.stats as sym
import warnings
import pandas as pd
import scipy.stats as st
##Q1
def ticks(y, pos):
    return r'$e^{{{:.0f}}}$'.format(np.log(y))
```

MiA

```
def best_fit_distribution(data, bins=200, ax=None,typ=0):
    """Model data by finding best fit distribution to data"""
    # Get histogram of original data
    y, x = np.histogram(data, bins=bins, density=True)
    x = (x + np.roll(x, -1))[:-1] / 2.0

    # Distributions to check
    if typ == 1:
        DISTRIBUTIONS = [st.mielke]
    else:
        DISTRIBUTIONS = [
                st.alpha,st.anglit,st.arcsine,st.beta,st.betaprime,
                st.bradford,st.burr,st.cauchy,st.chi,st.chi2,st.cosine,
                st.dgamma,st.dweibull,st.erlang,st.expon,st.exponnorm,
                st.exponweib,st.exponpow,st.f,st.fatiguelife,st.fisk,
                st.foldcauchy,st.foldnorm,st.frechet_r,st.frechet_l,
                st.genlogistic,st.genpareto,st.gennorm,st.genexpon,
                st.genextreme,st.gausshyper,st.gamma,st.gengamma,
                st.genhalflogistic,st.gilbrat,st.gompertz,st.gumbel_r,
                st.gumbel_l,st.halfcauchy,st.halflogistic,st.halfnorm,
                st.halfgennorm,st.hypsecant,st.invgamma,st.invgauss,
                st.invweibull,st.johnsonsb,st.johnsonsu,st.ksone,
                st.kstwobign,st.laplace,st.levy,st.levy_l,st.levy_stable,
                st.logistic,st.loggamma,st.loglaplace,st.lognorm,
                st.lomax,st.maxwell,st.mielke,st.nakagami,st.ncx2,st.ncf,
                st.nct,st.norm,st.pareto,st.pearson3,st.powerlaw,
                st.powerlognorm,st.powernorm,st.rdist,st.reciprocal,
                st.rayleigh,st.rice,st.recipinvgauss,st.semicircular,
                st.t,st.triang,st.truncexpon,st.truncnorm,st.tukeylambda,
                st.uniform,st.vonmises,st.vonmises_line,st.wald,
                st.weibull_min,st.weibull_max,st.wrapcauchy
                ]

    # Best holders
    best_distribution = st.norm
    best_params = (0.0, 1.0)
    best_sse = np.inf
```

MiA

```
    # Estimate distribution parameters from data
    for distribution in DISTRIBUTIONS:
        print(distribution)
        # Try to fit the distribution
        try:
            # Ignore warnings from data that can't be fit
            with warnings.catch_warnings():
                warnings.filterwarnings('ignore')

                # fit dist to data
                params = distribution.fit(data)

                # Separate parts of parameters
                arg = params[:-2]
                loc = params[-2]
                scale = params[-1]

                # Calculate fitted PDF and error with fit in distribution
                pdf = distribution.pdf(x, loc=loc, scale=scale, *arg)
                sse = np.sum(np.power(y - pdf, 2.0))

                # if axis pass in add to plot
                try:
                    if ax:
                        pd.Series(pdf, x).plot(ax=ax)
                    end
                except Exception:
                    pass

                # identify if this distribution is better
                if best_sse > sse > 0:
                    best_distribution = distribution
                    best_params = params
                    best_sse = sse

        except Exception:
            pass
        print(best_sse)
        print(sse)
    return (best_distribution.name, best_params)
```

MiA

```
def pandemic_model(probinf,dropout,startsize,runs):
    infwholemaster=[]
    timewholemaster=[]
    i=0
    infectedtheory=list(range(1,dropout+1))
    timetheory = [math.log(j)/(2*probinf-1) for j in infectedtheory]
    while i < runs:
        infwhole=[1]
        timewhole=[0]
        numinf = startsize
        t=0
        while numinf>0 and numinf≤dropout:
            change=np.random.choice([1,0],p=[probinf,1-probinf])
            if change == 1:
                numinf += 1
            else:
                numinf += -1
            if numinf==0:
                break
            t=t+np.random.exponential(1/(numinf))
            timewhole.append(t)
            infwhole.append(numinf)
        if numinf-1 != dropout:
            continue
        infwholemaster.append(infwhole)
        timewholemaster.append(timewhole)
        i+=1
    return infwholemaster, timewholemaster,timetheory,infectedtheory
```

MiA

```
1
2  # ============================================================================
3  #
4  # infwholemaster,timewholemaster,timetheory,infectedtheory= ...
       pandemic_model(3/4,1000,1,25)
5  # plt.figure(0) # In this example, all the plots will be in one figure.
6  # for i in range(len(infwholemaster)):
7  #     plt.plot(timewholemaster[i],infwholemaster[i],'k--',linewidth=0.5)
8  # plt.plot(timetheory,infectedtheory,linewidth=5.0)
9  # plt.xlabel('Time')
10 # plt.ylabel('Number Infected')
11 # black_patch = mpatches.Patch(color='black',label='Simulations')
12 # blue_patch = mpatches.Patch(color='blue', label='Expectation')
13 # plt.legend(handles=[blue_patch,black_patch])
14 # plt.savefig('Q1a.jpg')
15 # plt.show()
16 #
17 # plt.figure(1)
18 # fig,ax1=plt.subplots()
19 # for i in range(len(infwholemaster)):
20 #     ax1.semilogy(timewholemaster[i],infwholemaster[i],'k--', basey=np.e, ...
       linewidth=0.5)
21 # ax1.semilogy(timetheory,infectedtheory,linewidth=5.0)
22 # ax1.yaxis.set_major_formatter(mtick.FuncFormatter(ticks))
23 # ax1.set(xlabel='Time')
24 # ax1.set(ylabel='Natural Log of Number Infected')
25 # black_patch = mpatches.Patch(color='black',label='Simulations')
26 # blue_patch = mpatches.Patch(color='blue', label='Expectation')
27 # plt.legend(handles=[blue_patch,black_patch])
28 # plt.show()
29 # fig.savefig('Q1b.jpg')
30 # ============================================================================
```

MiA

```
# ================================================================================
# def pandemic_model_timetodrop(probinf,dropout,startsize,runs):
#     timetodropsimulation=[]
#     sizes=[]
#     i=0
#     timetodroptheory=[math.log(dropout)/((2*probinf)-1)]
#     while i < runs:
#         infwhole=[1]
#         timewhole=[0]
#         numinf = startsize
#         t=1
#         while numinf>0 and numinf<dropout:
#             change=np.random.choice([1,0],p=[probinf,1-probinf])
#             if change == 1:
#                 numinf += 1
#             else:
#                 numinf += -1
#             if numinf==0:
#                 break
#             t=np.random.exponential(1/(numinf+0.000001))
#             timewhole.append(timewhole[-1]+t)
#             infwhole.append(numinf)
#         if i ==1:
#             print(timewhole)
#             print(infwhole)
#         if numinf != dropout:
#             continue
# #           print('Numinf {} time {}'.format(numinf,t))
# #       print('Appened {} time {}'.format(numinf,t))
#         timetodropsimulation.append(timewhole[-1])
#         sizes.append(len(infwhole))
#         i+=1
#     return timetodropsimulation,timetodroptheory,sizes
#
# # ...
    ================================================================================
```

MiA

```
# # ... ==========================================================================
# ... timetodropsimulation,timetodroptheory,sizes=pandemic_model_timetodrop(3/4,1000,1,20000)
# timetodropsimualtionmean=statistics.mean(timetodropsimulation)
# timetodropsimualtionvar=statistics.variance(timetodropsimulation)
# plt.figure(2)
# fig1,ax1=plt.subplots()
# ax1.hist(timetodropsimulation,90,normed=True,alpha=0.5)
# ax1.set(xlabel='Time Untill 1000 Infected', ylabel='Frequency')
# fig1.savefig('Q1d.jpg')
# fig1.show()
# #
# # ... ==========================================================================
#
# ... timetodropsimulation,timetodroptheory,sizes=pandemic_model_timetodrop(3/4,1000,1,200)
# timetodropsimualtionmean=statistics.mean(timetodropsimulation)
# timetodropsimualtionvar=statistics.variance(timetodropsimulation)
# plt.figure(2)
# fig1,ax1=plt.subplots()
# ax1.hist(timetodropsimulation, 90,normed=True,alpha=0.5)
# ax1.set(xlabel='Time Untill 1000 Infected', ylabel='Frequency')
# black_patch = mpatches.Patch(color='orange',label='Fitted Weibull ... Distribution')
# blue_patch = mpatches.Patch(color='blue', label='Simulations')
# plt.legend(handles=[blue_patch,black_patch])
# #alpha=2+(timetodropsimualtionmean**2/timetodropsimualtionvar)
# ... #beta=(1+(timetodropsimualtionmean**2/timetodropsimualtionvar))*timetodropsimualtionmea
# #mean, var, skew, kurt = scipy.stats.gengamma.stats(alpha,c=-1, ... scale=beta, moments='mvsk')
# #x = np.linspace(0,25,1000)
# #rv = scipy.stats.gengamma(alpha,c=-1,scale=beta)
# #ax1.plot(x, rv.pdf(x), 'k-', lw=2, label='frozen pdf')
# timetodropsimulation.sort()
# ax1.plot(timetodropsimulation, ... scipy.stats.exponweib.pdf(timetodropsimulation, ... *scipy.stats.exponweib.fit(timetodropsimulation, 1, 1, scale=2, loc=0)))
# mean, var, skew, kurt =scipy.stats.exponweib.stats(timetodropsimulation, ... *scipy.stats.exponweib.fit(timetodropsimulation, 1, 1, scale=2, loc=0))
# #fig1.savefig('Q4a.jpg')
# fig1.show()
# ==========================================================================
```

MiA

```
1
2  # ================================================================================
3  # def pandemic_model_B(probinf,probmut,dropout,startsize,runs):
4  #     infwholemaster=[]
5  #     infmutwholemaster=[]
6  #     timewholemaster=[]
7  #     i=0
8  #     infectedtheory=list(range(1,dropout+1))
9  #     timetheory = [math.log(j)/(2*probinf-1) for j in infectedtheory]
10 #     while i < runs:
11 #         infwhole=[startsize]
12 #         infmutwhole=[0]
13 #         timewhole=[0]
14 #         numinf = startsize
15 #         nummut = 0
16 #         t=0
17 #         while numinf>0 and numinf≤dropout:
18 #             ...
       branch=np.random.choice([1,0],p=[infwhole[-1]/(infwhole[-1]+infmutwhole[-1]),infmutwhol
19 #             if branch==1:
20 #                 ...
       change=np.random.choice([2,1,0],p=[probmut,probinf,1-probinf-probmut])
21 #                 if change == 1:
22 #                     numinf += 1
23 #                     infwhole.append(numinf)
24 #                 elif change == 2:
25 #                     nummut += 1
26 #                     infmutwhole.append(nummut)
27 #                 else:
28 #                     numinf += -1
29 #                     infwhole.append(numinf)
30 #                 if numinf==0:
31 #                     break
32 #                 t=t+np.random.exponential(1/(numinf))
33 #                 timewhole.append(t)
34 #             else:
35 #                 if nummut≤0:
36 #                         continue
37 #                 ...
       change=np.random.choice([1,0],p=[probinf+(probmut/2),1-probinf-(probmut/2)])
38 #                 if change == 1:
39 #                     nummut += 1
```

MiA

```
#                 else:
#                     nummut += -1
#                 t=t+np.random.exponential(1/(nummut+0.00001))
#                 timewhole.append(t)
#                 infmutwhole.append(nummut)
#         if numinf-1 != dropout:
#             continue
#         infwholemaster.append(infwhole)
#         infmutwholemaster.append(infmutwhole)
#         timewholemaster.append(timewhole)
#         i+=1
#     return infwholemaster, ...
    infmutwholemaster,timewholemaster,timetheory,infectedtheory
#
# infwholemaster, ...
    infmutwholemaster,timewholemaster,timetheory,infectedtheory = ...
    pandemic_model_B(5/8,1/8,100,1,1)
#

# ================================================================================
def pandemic_model_B(probinf,probmut,startsize,runs):
    infwholemaster=[]
    infmutwholemaster=[]
    timewholemaster=[]
    i=0
    while i < runs:
        infwhole=[startsize]
        infmutwhole=[0]
        timewhole=[0]
        numinf = startsize
        nummut = 0
        t=0
```

MiA

```
        while numinf>0:
                change=np.random.choice([2,1,0],p=[probmut,probinf,1-probinf-probmut])
                if change == 1:
                    numinf += 1
                    infwhole.append(numinf)
                elif change == 2:
                    nummut += 1
                    infmutwhole.append(nummut)
                    infwhole.append(numinf+1)
                    i+=1
                    t=t+np.random.exponential(1/(numinf))
                    timewhole.append(t)
                    break
                else:
                    numinf += -1
                    infwhole.append(numinf)
                if numinf==0:
                    break
                t=t+np.random.exponential(1/(numinf))
                timewhole.append(t)
        if numinf == 0:
            continue
        infwholemaster.append(infwhole)
        infmutwholemaster.append(infmutwhole)
        timewholemaster.append(timewhole)
        i+=1
    return infwholemaster, infmutwholemaster,timewholemaster

# ================================================================================
# infwholemaster, infmutwholemaster,timewholemaster = ...
    pandemic_model_B(3/4,1/1000,1,30)
# plt.figure(3) # In this example, all the plots will be in one figure.
# for i in range(len(infwholemaster)):
#     plt.plot(timewholemaster[i],infwholemaster[i],'k-',linewidth=0.5)
# plt.xlabel('Time')
# plt.ylabel('Number Infected')
# black_patch = mpatches.Patch(color='black',label='Simulations')
# blue_patch = mpatches.Patch(color='blue', label='Expectation')
# plt.legend(handles=[blue_patch,black_patch])
# # plt.savefig('Q1a.jpg')
# plt.show()
# ================================================================================
```

MiA

```
def pandemic_model_B_time(probinf,probmut,startsize,runs):
    infwholemaster=[]
    infmutwholemaster=[]
    inftotaltodropsimulation=[]
    timewholemaster=[]
    timetodropsimulation=[]
    inftodropsimulation=[]
    i=0
    while i < runs:
        if (i % 1000) == 0:
            print('{}% complete'.format(100*i/runs))
        infwhole=[startsize]
        inftotalwhole=[startsize]
        infmutwhole=[0]
        timewhole=[0]
        numinf = startsize
        nummut = 0
        t=0

        while numinf>0:
                t=t+np.random.exponential(1/(numinf))
                timewhole.append(t)
                change=np.random.choice([2,1,0],p=[probmut,probinf,1-probinf-probmut])
                if change == 1:
                    numinf += 1
                    infwhole.append(numinf)
                    inftotalwhole.append(inftotalwhole[-1]+1)
                elif change == 2:
                    nummut += 1
                    infmutwhole.append(nummut)
                    infwhole.append(numinf+1)
                    i+=1
                    break
                else:
                    numinf += -1
                    infwhole.append(numinf)
                if numinf==0:
                    break
        if numinf == 0:
            continue
        if nummut == 1:
            infwholemaster.append(infwhole)
            infmutwholemaster.append(infmutwhole)
            timewholemaster.append(timewhole)
            inftotaltodropsimulation.append(inftotalwhole[-1])
            timetodropsimulation.append(timewhole[-1])
            inftodropsimulation.append(infwhole[-1])
    return inftodropsimulation,timetodropsimulation,inftotaltodropsimulation
# ================================================================================
```

MiA

```
#
#inftodropsimulation_a,timetodropsimulation_a,inftotaltodropsimulation_a = ...
    pandemic_model_B_time(3/4-(1/10),1/10,1,100000)
#inftodropsimulation_b,timetodropsimulation_b,inftotaltodropsimulation_b = ...
    pandemic_model_B_time(3/4-(1/20),1/20,1,100000)
#inftodropsimulation_c,timetodropsimulation_c,inftotaltodropsimulation_c = ...
    pandemic_model_B_time(3/4-(1/50),1/50,1,100000)
#
#a,b=best_fit_distribution(timetodropsimulation_a, bins=150, ax=None, typ=1)
#c,d=best_fit_distribution(timetodropsimulation_b, bins=150, ax=None, typ=1)
#e,f=best_fit_distribution(timetodropsimulation_c, bins=150, ax=None, typ=1)

# ================================================================================
# def pdf_timetodrop(mu,alpha,beta,t,param=[0,0,0]):
#     t=(t+param[-2])/param[-1]
#     lam=alpha-beta
#     x1=alpha*(lam**3)*mu*np.exp(lam*t)
#     x2=(lam**2+(alpha*mu*np.exp(lam*t)))**2
#     x=param[-3]*(x1/(x2))
#     return x
#
# y=list(np.arange(0,30,30/10000))
# # timetodropsimulation_a_pdf=[]
# timetodropsimulation_btest_pdf=[]
# timetodropsimulation_test_pdf=[]
# for i in y:
# #     ...
    timetodropsimulation_a_pdf.append(pdf_timetodrop(1/10,(3/4)-(1/10),1/4,i,param=[1.98,-0
#       ...
    timetodropsimulation_btest_pdf.append(pdf_timetodrop(1/20,(3/4)-(1/20),1/4,i,param=[1,0
#       ...
    timetodropsimulation_test_pdf.append(pdf_timetodrop(1/50,(3/4)-(1/50),1/4,i,param=[1,0,
# #
#
# #c,d=best_fit_distribution(timetodropsimulation_test_pdf, bins=150, ...
    ax=None, typ=1)
#
# fig, ax = plt.subplots(1, 1)
# timetodropsimulation_c.sort()
# ax.plot(timetodropsimulation_c, ...
    st.mielke.pdf(timetodropsimulation_c,2.8,5.5,loc=-2.7,scale=10.7))
# ax.plot(y,timetodropsimulation_test_pdf,'g--')
# fig.show()
#
# fig, ax = plt.subplots(1, 1)
# timetodropsimulation_b.sort()
# ax.plot(timetodropsimulation_b, st.mielke.pdf(timetodropsimulation_b,2.3, ...
    5.5,loc=-4,scale=10.7))
# ax.plot(y,timetodropsimulation_btest_pdf,'g--')
# fig.show()
```

MiA

```
# ================================================================================
# plt.figure(4)
# fig1,ax1=plt.subplots()
# timetodropsimulation_a.sort()
# ax1.hist(timetodropsimulation_a,150,normed=True,alpha=0.3)
# ax1.plot(y,timetodropsimulation_a_pdf,'b--')
# #ax1.plot(timetodropsimulation_a, ...
    scipy.stats.burr.pdf(timetodropsimulation_a, ...
    *scipy.stats.burr.fit(timetodropsimulation_a,3.6,0.29, scale=3, loc=0)), ...
    'b--')
# ax1.hist(timetodropsimulation_b,150,normed=True,alpha=0.3)
# timetodropsimulation_b.sort()
# ax1.plot(y,timetodropsimulation_b_pdf,'r--')
# #ax1.plot(timetodropsimulation_b, ...
    scipy.stats.burr.pdf(timetodropsimulation_b, ...
    *scipy.stats.burr.fit(timetodropsimulation_b,4,0, scale=5.5, loc=0)), 'r--')
# ax1.hist(timetodropsimulation_c,150,normed=True,alpha=0.3)
# timetodropsimulation_c.sort()
# ax1.plot(y,timetodropsimulation_c_pdf,'g--')
# #ax1.plot(timetodropsimulation_c, ...
    scipy.stats.burr.pdf(timetodropsimulation_c, ...
    *scipy.stats.burr.fit(timetodropsimulation_c,5,0, scale=7, loc=0)), 'g--')
# ax1.set(xlabel='Time Untill Mutation', ylabel='Frequency')
# green_patch = mpatches.Patch(color='green',label=r'Mutation Rate of ...
    $\frac{1}{50}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Mutation Rate of ...
    $\frac{1}{10}$')
# red_patch = mpatches.Patch(color='red',label=r'Mutation Rate of ...
    $\frac{1}{20}$')
# plt.legend(handles=[blue_patch,red_patch,green_patch])
# #plt.savefig('Q2aii.jpg')
# fig1.show()
```

MiA

```
# plt.figure(14)
# fig1,ax1=plt.subplots()
# timetodropsimulation_a.sort()
# ax1.hist(timetodropsimulation_a,150,normed=True,alpha=0.3)
# ax1.plot(timetodropsimulation_a, ...
    scipy.stats.mielke.pdf(timetodropsimulation_a, ...
    *scipy.stats.mielke.fit(timetodropsimulation_a,1,4, scale=3, loc=0)), 'b--')
# ax1.hist(timetodropsimulation_b,150,normed=True,alpha=0.3)
# timetodropsimulation_b.sort()
# ax1.plot(timetodropsimulation_b, ...
    scipy.stats.mielke.pdf(timetodropsimulation_b, ...
    *scipy.stats.mielke.fit(timetodropsimulation_b,1,5, scale=5.5, loc=0)), ...
    'r--')
# ax1.hist(timetodropsimulation_c,150,normed=True,alpha=0.3)
# timetodropsimulation_c.sort()
# ax1.plot(timetodropsimulation_c, ...
    scipy.stats.mielke.pdf(timetodropsimulation_c, ...
    *scipy.stats.mielke.fit(timetodropsimulation_c,1,6, scale=7, loc=0)), 'g--')
# ax1.set(xlabel='Time Untill Mutation', ylabel='Frequency')
# green_patch = mpatches.Patch(color='green',label=r'Mutation Rate of ...
    $\frac{1}{50}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Mutation Rate of ...
    $\frac{1}{10}$')
# red_patch = mpatches.Patch(color='red',label=r'Mutation Rate of ...
    $\frac{1}{20}$')
# plt.legend(handles=[blue_patch,red_patch,green_patch])
# plt.savefig('Q2aiIi.jpg')
# fig1.show()
```

MiA

```
#
# plt.figure(5)
# fig1,ax1=plt.subplots()
# bins=list(range(70))
# ax1.hist(inftodropsimulation_a,bins=bins,normed=True,alpha=0.3)
# inftodropsimulation_a.sort()
# ax1.plot(inftodropsimulation_a, ...
    scipy.stats.genexpon.pdf(inftodropsimulation_a, ...
    *scipy.stats.genexpon.fit(inftodropsimulation_a, 1, 1, 1, scale=2, ...
    loc=2)), 'b--')
# ax1.hist(inftodropsimulation_b,bins=bins,normed=True,alpha=0.3)
# inftodropsimulation_b.sort()
# ax1.plot(inftodropsimulation_b, ...
    scipy.stats.genexpon.pdf(inftodropsimulation_b, ...
    *scipy.stats.genexpon.fit(inftodropsimulation_b, 1, 1, 1, scale=2, ...
    loc=2)), 'r--')
# ax1.hist(inftodropsimulation_c,bins=bins,normed=True,alpha=0.3)
# inftodropsimulation_c.sort()
# ax1.plot(inftodropsimulation_c, ...
    scipy.stats.genexpon.pdf(inftodropsimulation_c, ...
    *scipy.stats.genexpon.fit(inftodropsimulation_c, 1, 1, 1, scale=2, ...
    loc=2)), 'g--')
# ax1.set(xlabel='Infected Untill Mutation', ylabel='Frequency',xlim=[0,70])
# green_patch = mpatches.Patch(color='green',label=r'Mutation Rate of ...
    $\frac{1}{50}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Mutation Rate of ...
    $\frac{1}{10}$')
# red_patch = mpatches.Patch(color='red',label=r'Mutation Rate of ...
    $\frac{1}{20}$')
# plt.legend(handles=[blue_patch,red_patch,green_patch])
# #plt.savefig('Q2b.jpg')
# fig1.show()
#
```

MiA

```
# plt.figure(9)
# fig1,ax1=plt.subplots()
# bins=list(range(120))
# ax1.hist(inftotaltodropsimulation_a,bins=bins,normed=True,alpha=0.3)
# inftotaltodropsimulation_a.sort()
# ax1.plot(inftotaltodropsimulation_a, ...
    scipy.stats.genexpon.pdf(inftotaltodropsimulation_a, ...
    *scipy.stats.genexpon.fit(inftotaltodropsimulation_a, 1, 1, 1, scale=2, ...
    loc=2)), 'b--')
# ax1.hist(inftotaltodropsimulation_b,bins=bins,normed=True,alpha=0.3)
# inftotaltodropsimulation_b.sort()
# ax1.plot(inftotaltodropsimulation_b, ...
    scipy.stats.genexpon.pdf(inftotaltodropsimulation_b, ...
    *scipy.stats.genexpon.fit(inftotaltodropsimulation_b, 1, 1, 1, scale=2, ...
    loc=2)), 'r--')
# ax1.hist(inftotaltodropsimulation_c,bins=bins,normed=True,alpha=0.3)
# inftotaltodropsimulation_c.sort()
# ax1.plot(inftotaltodropsimulation_c, ...
    scipy.stats.genexpon.pdf(inftotaltodropsimulation_c, ...
    *scipy.stats.genexpon.fit(inftotaltodropsimulation_c, 1, 1, 1, scale=2, ...
    loc=2)), 'g--')
# ax1.set(xlabel='Total Infections Untill Mutation', ...
    ylabel='Frequency',xlim=[0,150])
# green_patch = mpatches.Patch(color='green',label=r'Mutation Rate of ...
    $\frac{1}{50}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Mutation Rate of ...
    $\frac{1}{10}$')
# red_patch = mpatches.Patch(color='red',label=r'Mutation Rate of ...
    $\frac{1}{20}$')
# plt.legend(handles=[blue_patch,red_patch,green_patch])
# #plt.savefig('Q2c.jpg')
# fig1.show()
# ================================================================================
```

MiA

```
def pandemic_model_C(probinf,probtypeI,startsize,runs,dropout):
    infectablewholemaster=[]
    cureablewholemaster=[]
    timewholemaster=[]
    surviveordie=[]
    i=0
    while i <runs:
        infectablewhole=[startsize]
        timewhole=[0]
        curablewhole=[startsize]
        numinfectable = startsize
        numcureable = startsize
        t=0
        while numcureable < dropout:
            change=np.random.choice([1,0],p=[probinf,1-probinf])
            if change==1:
                change=np.random.choice([1,0],p=[probtypeI,1-probtypeI])
                if change==1:
                    if numinfectable > 0:
                        numinfectable += 1
                        numcureable += 1
                        curablewhole.append(numcureable)
                        infectablewhole.append(numinfectable)
                        t=np.random.exponential(1/(numcureable+0.0001))
                        timewhole.append(timewhole[-1]+t)
                else:
                    if numinfectable > 0:
                        numcureable += 1
                        curablewhole.append(numcureable)
                        infectablewhole.append(numinfectable)
                        t=np.random.exponential(1/numcureable)
                        timewhole.append(timewhole[-1]+t)
            else:
                change=np.random.choice([1,0], p=[probtypeI,1-probtypeI])
                if change==1:
                    t=np.random.exponential(1/(numcureable+0.001))
                    timewhole.append(timewhole[-1]+t)
                    numcureable += -1
                    numinfectable += -1
                    infectablewhole.append(numinfectable)
                    curablewhole.append(numcureable)
                    if numcureable≤0:
                        infectablewholemaster.append(infectablewhole)
                        cureablewholemaster.append(curablewhole)
                        timewholemaster.append(timewhole)
                        surviveordie.append(0)
                        i+=1
                        break
```

MiA

```
                else:
                    t=np.random.exponential(1/(numcureable+0.001))
                    timewhole.append(timewhole[-1]+t)
                    numcureable += -1
                    curablewhole.append(numcureable)
                    infectablewhole.append(numinfectable)
                    if numcureable≤0:
                        infectablewholemaster.append(infectablewhole)
                        cureablewholemaster.append(curablewhole)
                        timewholemaster.append(timewhole)
                        surviveordie.append(0)
                        i+=1
                        break
        if numcureable == dropout:
            infectablewholemaster.append(infectablewhole)
            cureablewholemaster.append(curablewhole)
            timewholemaster.append(timewhole)
            surviveordie.append(1)
            i+=1
    return infectablewholemaster,cureablewholemaster,timewholemaster, ...
        surviveordie

# ================================================================================
# infectablewholemaster,cureablewholemaster,timewholemaster, ...
    surviveordie=pandemic_model_C(3/4,1/4,1,500,1000)
#
# plt.figure(9)
# print('Survived={}'.format(surviveordie.count(1)))
# print('Extinct={}'.format(surviveordie.count(0)))
# fig1,ax1=plt.subplots()
# for i in range(25):
#     ax1.plot(timewholemaster[i], infectablewholemaster[i], 'b')
#     ax1.plot(timewholemaster[i], cureablewholemaster[i], 'g')
#     subtract=[a_i - b_i for a_i, b_i in zip(cureablewholemaster[i], ...
    infectablewholemaster[i])]
#     ax1.plot(timewholemaster[i], subtract , 'y')
# fig1.show()
#
# ================================================================================
# ================================================================================
```

MiA

```
# q3
#
# ============================================================================

def SIR_model(probinfbase,probmutbase,startsize,totalpop):
    numIinf = [startsize]
    numJinf = [0]
    numIcured = [0]
    numJcured = [0]
    numIsus= [totalpop-startsize]
    numJsus= [totalpop]
    time=[0]
    while (totalpop>numIinf[-1] and numIinf[-1]>0) or (totalpop>numJinf[-1] ...
        and numJinf[-1]>0):
        time.append(time[-1]+np.random.exponential(1/(numIinf[-1]+numJinf[-1])))
        alpha=(probinfbase*numIsus[-1]*numIinf[-1]/totalpop)
        beta=(1-probinfbase)*numIinf[-1]
        mut=(probmutbase*numIinf[-1])
        numIinf.append(numIinf[-1])
        numJinf.append(numJinf[-1])
        numIsus.append(numIsus[-1])
        numJsus.append(numJsus[-1])
        numIcured.append(numIcured[-1])
        numJcured.append(numJcured[-1])

        if numIinf[-1] == 0 or numIsus[-1]≤0:
            probIinf=0
        else:
            probIinf= alpha/(alpha+beta+mut)
        if numIinf[-1] == 0 or numIsus[-1]≤0:
            probImut=0
        else:
            probImut=mut/(alpha+beta+mut)
        if numIinf[-1]>0:
            change=np.random.choice([0,1,2],p=[probIinf, ...
                probImut,1-probIinf-probImut])
            if change == 0:
                numIinf[-1]=numIinf[-1]+1
                numIsus[-1]=numIsus[-1]-1
            elif change==1:
                numJinf[-1]=numJinf[-1]+1
                numIinf[-1]=numIinf[-1]-1
                numJsus[-1]=numJsus[-1]-1
            else:
                numIinf[-1]=numIinf[-1]-1
                numIcured[-1]=numIcured[-1]+1
```

MiA

```
        if numJinf[-1]>0:
            if numJinf[-1] == 0 or numJsus[-1]≤0:
                probinf=0
            else:
                probinf= ...
                    (probinfbase*numJsus[-1]*numJinf[-1]/totalpop)/((probinfbase*numJsus[-1
            changeJ=np.random.choice([0,1],p=[probinf,1-probinf])
            if changeJ == 0:
                numJinf[-1]=numJinf[-1]+1
                numJsus[-1]=numJsus[-1]-1
            else:
                numJinf[-1]=numJinf[-1]-1
                numJcured[-1]=numJcured[-1]+1
        if numIinf[-1]<0:
            numIinf[-1]=0
        if numJcured[-1]>totalpop:
            numJcured[-1]=totalpop
        if numIcured[-1]>totalpop:
            numIcured[-1]=totalpop
        if numJinf[-1]<0:
            numJinf[-1]=0
        if numIsus[-1]<0:
            numIsus[-1]=0
        if numJsus[-1]<0:
            numJsus[-1]=0
    return numIinf,numIsus,numIcured,numJinf,numJsus,numJcured,time

numIinf=[]
while len(numIinf)<50:
    numIinf,numIsus,numIcured,numJinf,numJsus,numJcured,time=SIR_model(3/4,1/100,1,100)

# ================================================================================
# plt.figure(11)
# fig1,ax1=plt.subplots()
# ax1.plot(time, numIinf,'r', label='Infected with I')
# ax1.plot(time, numIsus, 'y', label='Suseptible to I')
# ax1.plot(time, numIcured, 'g', label='Immune to I')
# ax1.plot(time, numJinf, 'r--', label='Infected with J')
# ax1.plot(time, numJsus, 'y--', label='Suseptible to J')
# ax1.plot(time, numJcured, 'g--', label='Immune to J')
# ax1.set(xlabel='Time', ylabel='Population in Group')
# plt.gcf().subplots_adjust(left=0.15)
# legend = ax1.legend()
# legend.get_frame().set_alpha(0.5)
# plt.savefig('Q3ai.jpg')
# fig1.show()
```

MiA

```
# ================================================================================
def SIR_modelruns(probinfbase,probmutbase,startsize,totalpop,runs):
    crossingtimes=[]
    k=1
    while k<runs:
        numIinf = [startsize]
        numJinf = [0]
        numIcured = [0]
        numJcured = [0]
        numIsus= [totalpop-startsize]
        numJsus= [totalpop]
        time=[0]
        crossed=0
        while (totalpop>numIinf[-1] and numIinf[-1]>0) or ...
            (totalpop>numJinf[-1] and numJinf[-1]>0):
            if numIinf == numIsus and crossed==0:
                crossed==1
                crossingtimes.append(time[-1])
                k+=1
                if (k % 100) == 0:
                    print('{}% complete'.format(100*k/runs))
                break
            time[-1]=time[-1]+np.random.exponential(1/(numIinf[-1]+numJinf[-1]))
            alpha=(probinfbase*numIsus[-1]*numIinf[-1]/totalpop)
            beta=(1-probinfbase)*numIinf[-1]
            mut=(probmutbase*numIinf[-1])
            if numIinf[-1] == 0 or numIsus[-1]≤0:
                probIinf=0
            else:
                probIinf= alpha/(alpha+beta+mut)
            if numIinf[-1] == 0 or numIsus[-1]≤0:
                probImut=0
            else:
                probImut=mut/(alpha+beta+mut)
            if numIinf[-1]>0:
                change=np.random.choice([0,1,2],p=[probIinf, ...
                    probImut,1-probIinf-probImut])
                if change == 0:
                    numIinf[-1]=numIinf[-1]+1
                    numIsus[-1]=numIsus[-1]-1
                elif change==1:
                    numJinf[-1]=numJinf[-1]+1
                    numIinf[-1]=numIinf[-1]-1
                    numJsus[-1]=numJsus[-1]-1
                else:
                    numIinf[-1]=numIinf[-1]-1
                    numIcured[-1]=numIcured[-1]+1
```

MiA

```
1
2               if numJinf[-1]>0:
3                   if numJinf[-1] == 0 or numJsus[-1]≤0:
4                       probinf=0
5                   else:
6                       probinf= ...
                            (probinfbase*numJsus[-1]*numJinf[-1]/totalpop)/((probinfbase*numJsu
7                   changeJ=np.random.choice([0,1],p=[probinf,1-probinf])
8                   if changeJ == 0:
9                       numJinf[-1]=numJinf[-1]+1
10                      numJsus[-1]=numJsus[-1]-1
11                  else:
12                      numJinf[-1]=numJinf[-1]-1
13                      numJcured[-1]=numJcured[-1]+1
14              if numIinf[-1]<0:
15                  numIinf[-1]=0
16              if numJcured[-1]>totalpop:
17                  numJcured[-1]=totalpop
18              if numIcured[-1]>totalpop:
19                  numIcured[-1]=totalpop
20              if numJinf[-1]<0:
21                  numJinf[-1]=0
22              if numIsus[-1]<0:
23                  numIsus[-1]=0
24              if numJsus[-1]<0:
25                  numJsus[-1]=0
26      return crossingtimes
27
28  # ============================================================================
29  # #crossingtimes_a=SIR_modelruns(90/100,1/100,1,500,25000)
30  # #crossingtimes_b=SIR_modelruns(90/100,5/100,1,500,25000)
31  # #crossingtimes_c=SIR_modelruns(90/100,10/100,1,500,25000)
32  # #crossingtimes_d=SIR_modelruns(80/100,1/100,1,500,25000)
33  # #crossingtimes_e=SIR_modelruns(80/100,5/100,1,500,25000)
34  # #crossingtimes_f=SIR_modelruns(80/100,10/100,1,500,25000)
35  # #crossingtimes_g=SIR_modelruns(95/100,1/100,1,500,25000)
36  # #crossingtimes_h=SIR_modelruns(95/100,5/100,1,500,25000)
37  # #crossingtimes_i=SIR_modelruns(95/100,10/100,1,500,25000)
```

MiA

```
# ================================================================================
# print('a')
# a_1,a_2=best_fit_distribution(crossingtimes_a, bins=90, ax=None, typ=1)
# print('b')
# b_1,b_2=best_fit_distribution(crossingtimes_b, bins=90, ax=None, typ=1)
# print('c')
# c_1,c_2=best_fit_distribution(crossingtimes_c, bins=90, ax=None, typ=1)
# print('d')
# d_1,d_2=best_fit_distribution(crossingtimes_d, bins=90, ax=None, typ=1)
# print('e')
# e_1,e_2=best_fit_distribution(crossingtimes_e, bins=90, ax=None, typ=1)
# print('f')
# f_1,f_2=best_fit_distribution(crossingtimes_f, bins=90, ax=None, typ=1)
# print('g')
# g_1,g_2=best_fit_distribution(crossingtimes_g, bins=90, ax=None, typ=1)
# print('h')
# h_1,h_2=best_fit_distribution(crossingtimes_h, bins=90, ax=None, typ=1)
# print('i')
# i_1,i_2=best_fit_distribution(crossingtimes_i, bins=90, ax=None, typ=1)
# ================================================================================
#
#
# plt.figure(13)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_a,90,normed=True,alpha=0.3)
# crossingtimes_a.sort()
# ax1.plot(crossingtimes_a, scipy.stats.chi2.pdf(crossingtimes_a, ...
    *scipy.stats.chi2.fit(crossingtimes_a, 1,  scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{90}{100}, \mu = \frac{1}{100}$')
# plt.savefig('Q3b1.jpg')
# fig1.show()
#
# plt.figure(14)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_b,90,normed=True,alpha=0.3)
# crossingtimes_b.sort()
# ax1.plot(crossingtimes_b, scipy.stats.chi2.pdf(crossingtimes_b, ...
    *scipy.stats.chi2.fit(crossingtimes_b, 1,  scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{90}{100}, \mu = \frac{5}{100}$')
# plt.savefig('Q3b2.jpg')
# fig1.show()
#
```

MiA

```
# plt.figure(15)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_c,90,normed=True,alpha=0.3)
# crossingtimes_c.sort()
# ax1.plot(crossingtimes_c, scipy.stats.chi2.pdf(crossingtimes_c, ...
    *scipy.stats.chi2.fit(crossingtimes_c, 1,  scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{90}{100}, \mu = \frac{10}{100}$')
# plt.savefig('Q3b3.jpg')
# fig1.show()
#
# plt.figure(16)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_d,90,normed=True,alpha=0.3)
# crossingtimes_d.sort()
# ax1.plot(crossingtimes_d, scipy.stats.chi2.pdf(crossingtimes_d, ...
    *scipy.stats.chi2.fit(crossingtimes_d, 1,  scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{80}{100}, \mu = \frac{1}{100}$')
# plt.savefig('Q3b4.jpg')
# fig1.show()
#
# plt.figure(17)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_e,90,normed=True,alpha=0.3)
# crossingtimes_e.sort()
# ax1.plot(crossingtimes_e, scipy.stats.chi2.pdf(crossingtimes_e, ...
    *scipy.stats.chi2.fit(crossingtimes_e, 1,scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{80}{100}, \mu = \frac{5}{100}$')
# plt.savefig('Q3b5.jpg')
# fig1.show()
#
# plt.figure(18)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_f,90,normed=True,alpha=0.3)
# crossingtimes_f.sort()
# ax1.plot(crossingtimes_f, scipy.stats.chi2.pdf(crossingtimes_f, ...
    *scipy.stats.chi2.fit(crossingtimes_f, 1, scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{80}{100}, \mu = \frac{10}{100}$')
# plt.savefig('Q3b6.jpg')
# fig1.show()
#
```

MiA

```
# plt.figure(19)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_g,90,normed=True,alpha=0.3)
# crossingtimes_g.sort()
# ax1.plot(crossingtimes_g, scipy.stats.chi2.pdf(crossingtimes_g, ...
    *scipy.stats.chi2.fit(crossingtimes_g, 1, scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{95}{100}, \mu = \frac{1}{100}$')
# plt.savefig('Q3b7.jpg')
# fig1.show()
#
# plt.figure(20)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_h,90,normed=True,alpha=0.3)
# crossingtimes_h.sort()
# ax1.plot(crossingtimes_h, scipy.stats.chi2.pdf(crossingtimes_h, ...
    *scipy.stats.chi2.fit(crossingtimes_h, 1, scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{95}{100}, \mu = \frac{5}{100}$')
# plt.savefig('Q3b8.jpg')
# fig1.show()
#
# plt.figure(21)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_i,90,normed=True,alpha=0.3)
# crossingtimes_i.sort()
# ax1.plot(crossingtimes_i, scipy.stats.chi2.pdf(crossingtimes_i, ...
    *scipy.stats.chi2.fit(crossingtimes_i, 1, scale=2, loc=1)), 'b--')
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency',title=r'$\beta= ...
    \frac{95}{100}, \mu = \frac{10}{100}$')
# plt.savefig('Q3b9.jpg')
# fig1.show()
```

MiA

```
# plt.figure(22)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_g, 90, normed=True,alpha=0.3)
# ax1.hist(crossingtimes_h, 90, normed=True,alpha=0.3)
# ax1.hist(crossingtimes_i, 90, normed=True,alpha=0.3)
# ax1.plot(crossingtimes_g, scipy.stats.chi2.pdf(crossingtimes_g, ...
    *scipy.stats.chi2.fit(crossingtimes_g, 1, scale=2, loc=1)), 'b—-')
# ax1.plot(crossingtimes_h, scipy.stats.chi2.pdf(crossingtimes_h, ...
    *scipy.stats.chi2.fit(crossingtimes_h, 1, scale=2, loc=1)), 'r—-')
# ax1.plot(crossingtimes_i, scipy.stats.chi2.pdf(crossingtimes_i, ...
    *scipy.stats.chi2.fit(crossingtimes_i, 1, scale=2, loc=1)), 'g—-')
# green_patch = mpatches.Patch(color='green',label=r'Mutation Rate of ...
    $\frac{1}{10}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Mutation Rate of ...
    $\frac{1}{100}$')
# red_patch = mpatches.Patch(color='red',label=r'Mutation Rate of ...
    $\frac{1}{20}$')
# plt.legend(handles=[green_patch,red_patch,blue_patch])
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency')
# plt.savefig('Q3b10.jpg')
# fig1.show()
#
# plt.figure(23)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_a,90,normed=True,alpha=0.3)
# ax1.hist(crossingtimes_b,90,normed=True,alpha=0.3)
# ax1.hist(crossingtimes_c,90,normed=True,alpha=0.3)
# ax1.plot(crossingtimes_a, scipy.stats.chi2.pdf(crossingtimes_a, ...
    *scipy.stats.chi2.fit(crossingtimes_a, 1, scale=2, loc=1)), 'b—-')
# ax1.plot(crossingtimes_b, scipy.stats.chi2.pdf(crossingtimes_b, ...
    *scipy.stats.chi2.fit(crossingtimes_b, 1, scale=2, loc=1)), 'r—-')
# ax1.plot(crossingtimes_c, scipy.stats.chi2.pdf(crossingtimes_c, ...
    *scipy.stats.chi2.fit(crossingtimes_c, 1, scale=2, loc=1)), 'g—-')
# green_patch = mpatches.Patch(color='green',label=r'Mutation Rate of ...
    $\frac{1}{10}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Mutation Rate of ...
    $\frac{1}{100}$')
# red_patch = mpatches.Patch(color='red',label=r'Mutation Rate of ...
    $\frac{1}{20}$')
# plt.legend(handles=[green_patch,red_patch,blue_patch])
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency')
# plt.savefig('Q3b11.jpg')
# fig1.show()
#
```

MiA

```
# plt.figure(24)
# fig1,ax1=plt.subplots()
# ax1.hist(crossingtimes_d,90,normed=True,alpha=0.3)
# ax1.hist(crossingtimes_e,90,normed=True,alpha=0.3)
# ax1.hist(crossingtimes_f,90,normed=True,alpha=0.3)
# ax1.plot(crossingtimes_d, scipy.stats.chi2.pdf(crossingtimes_d, ...
    *scipy.stats.chi2.fit(crossingtimes_d, 1, scale=2, loc=1)), 'b--')
# ax1.plot(crossingtimes_e, scipy.stats.chi2.pdf(crossingtimes_e, ...
    *scipy.stats.chi2.fit(crossingtimes_e, 1, scale=2, loc=1)), 'r--')
# ax1.plot(crossingtimes_f, scipy.stats.chi2.pdf(crossingtimes_f, ...
    *scipy.stats.chi2.fit(crossingtimes_f, 1, scale=2, loc=1)), 'g--')
# green_patch = mpatches.Patch(color='green',label=r'Mutation Rate of ...
    $\frac{1}{10}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Mutation Rate of ...
    $\frac{1}{100}$')
# red_patch = mpatches.Patch(color='red',label=r'Mutation Rate of ...
    $\frac{1}{20}$')
# plt.legend(handles=[green_patch,red_patch,blue_patch])
# ax1.set(xlabel='Time Untill Crossing', ylabel='Frequency')
# plt.savefig('Q3b12.jpg')
# fig1.show()
#
# ================================================================================
```

MiA

```
def SIR_model_mutant(probinfbase,probmutbase,startsize,totalpop,runs):
    muthappen=[]
    muthappentime=[]
    k=1
    while k<runs:
        numIinf = [startsize]
        numJinf = [0]
        numIcured = [0]
        numJcured = [0]
        numIsus= [totalpop-startsize]
        numJsus= [totalpop]
        time=[0]
        crossed=0
        while (totalpop>numIinf[-1] and numIinf[-1]>0) or ...
            (totalpop>numJinf[-1] and numJinf[-1]>0):
            if numJinf[-1] == 1 and crossed==0:
                crossed==1
                muthappen.append(1)
                muthappentime.append(time[-1])
                k+=1
                if (k % 100) == 0:
                    print('{}% complete'.format(100*k/runs))
                break
            time[-1]=time[-1]+np.random.exponential(1/(numIinf[-1]+numJinf[-1]))
            alpha=(probinfbase*numIsus[-1]*numIinf[-1]/totalpop)
            beta=(1-probinfbase)*numIinf[-1]
            mut=(probmutbase*numIinf[-1])
            if numIinf[-1] == 0 or numIsus[-1]≤0:
                probIinf=0
            else:
                probIinf= alpha/(alpha+beta+mut)
            if numIinf[-1] == 0 or numIsus[-1]≤0:
                probImut=0
            else:
                probImut=mut/(alpha+beta+mut)
```

MiA

```
            if numIinf[-1]>0:
                change=np.random.choice([0,1,2],p=[probIinf, ...
                    probImut,1-probIinf-probImut])
                if change == 0:
                    numIinf[-1]=numIinf[-1]+1
                    numIsus[-1]=numIsus[-1]-1
                elif change==1:
                    numJinf[-1]=numJinf[-1]+1
                    numIinf[-1]=numIinf[-1]-1
                    numJsus[-1]=numJsus[-1]-1
                else:
                    numIinf[-1]=numIinf[-1]-1
                    numIcured[-1]=numIcured[-1]+1
            if numJinf[-1]>0:
                if numJinf[-1] == 0 or numJsus[-1]<=0:
                    probinf=0
                else:
                    probinf= ...
                        (probinfbase*numJsus[-1]*numJinf[-1]/totalpop)/((probinfbase*numJsu
                changeJ=np.random.choice([0,1],p=[probinf,1-probinf])
                if changeJ == 0:
                    numJinf[-1]=numJinf[-1]+1
                    numJsus[-1]=numJsus[-1]-1
                else:
                    numJinf[-1]=numJinf[-1]-1
                    numJcured[-1]=numJcured[-1]+1
            if numIinf[-1]<0:
                numIinf[-1]=0
            if numJcured[-1]>totalpop:
                numJcured[-1]=totalpop
            if numIcured[-1]>totalpop:
                numIcured[-1]=totalpop
            if numJinf[-1]<0:
                numJinf[-1]=0
            if numIsus[-1]<0:
                numIsus[-1]=0
            if numJsus[-1]<0:
                numJsus[-1]=0
        if numJinf[-1] == 0 and crossed==0:
            crossed==0
            muthappen.append(0)
            muthappentime.append(0)
            k+=1
            if (k % 100) == 0:
                print('{}% complete'.format(100*k/runs))
    return muthappen, muthappentime
```

MiA

```
#mut_avg=[]
#mut_time=[]
#for j in list(np.arange(0,1,0.01)):
  #  muthappen_a,muthappentime_a=SIR_model_mutant(25/100,j,1,1000,5000)
 #   mut_avg.append(statistics.mean(muthappen_a))
 #   mut_time.append(statistics.mean(muthappentime_a))

#mut_avg_b=[]
#mut_time_b=[]
#for j in list(np.arange(0,1,0.01)):
 #   muthappen_a,muthappentime_a=SIR_model_mutant(50/100,j,1,1000,5000)
  #  mut_avg_b.append(statistics.mean(muthappen_a))
   # mut_time_b.append(statistics.mean(muthappentime_a))

#mut_avg_c=[]
#mut_time_c=[]
#for j in list(np.arange(0,1,0.01)):
 #   muthappen_a,muthappentime_a=SIR_model_mutant(75/100,j,1,300,1000)
  #  mut_avg_c.append(statistics.mean(muthappen_a))
   # mut_time_c.append(statistics.mean(muthappentime_a))

# ================================================================================
# plt.figure(21)
# fig1,ax1=plt.subplots()
# ax1.plot(list(np.arange(0,1,0.01)),mut_avg,'b')
# ax1.plot(list(np.arange(0,1,0.01)),mut_avg_b,'r')
# ax1.plot(list(np.arange(0,1,0.01)),mut_avg_c,'g')
# green_patch = mpatches.Patch(color='green',label=r'Infection Rate of ...
    $\frac{75}{100}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Infection Rate of ...
    $\frac{25}{100}$')
# red_patch = mpatches.Patch(color='red',label=r'Infection Rate of ...
    $\frac{50}{100}$')
# plt.legend(handles=[green_patch,red_patch,blue_patch], loc='upper right')
# ax1.set(xlabel='Muatation Rate', ylabel='Probability')
# plt.savefig('Q3mu.jpg')
# fig1.show()
```

MiA

```
# plt.figure(22)
# fig1,ax1=plt.subplots()
# ax1.plot(list(np.arange(0,1,0.01)),mut_time,'b')
# ax1.plot(list(np.arange(0,1,0.01)),mut_time_b,'r')
# ax1.plot(list(np.arange(0,1,0.01)),mut_time_c,'g')
# green_patch = mpatches.Patch(color='green',label=r'Infection Rate of ...
    $\frac{75}{100}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Infection Rate of ...
    $\frac{25}{100}$')
# red_patch = mpatches.Patch(color='red',label=r'Infection Rate of ...
    $\frac{50}{100}$')
# plt.legend(handles=[green_patch,red_patch,blue_patch], loc='upper right')
# ax1.set(xlabel='Muatation Rate', ylabel='Mean Time to Mutation')
# plt.savefig('Q3mu1.jpg')
# fig1.show()
# ================================================================================

def SIR_model_mutant_numinf(probinfbase,probmutbase,startsize,totalpop,runs):
    muthappen=[]
    muthappentime=[]
    k=1
    while k<runs:
        numIinf = [startsize]
        numJinf = [0]
        numIcured = [0]
        numJcured = [0]
        numIsus= [totalpop-startsize]
        numJsus= [totalpop]
        time=[0]
        crossed=0
        while (totalpop>numIinf[-1] and numIinf[-1]>0) or ...
            (totalpop>numJinf[-1] and numJinf[-1]>0):
            if numJinf[-1] == 1 and crossed==0:
                crossed==1
                muthappen.append(numIinf[-1])
                muthappentime.append(time[-1])
                k+=1
                if (k % 100) == 0:
                    print('{}% complete'.format(100*k/runs))
                break
```

MiA

```
            time[-1]=time[-1]+np.random.exponential(1/(numIinf[-1]+numJinf[-1]))
            alpha=(probinfbase*numIsus[-1]*numIinf[-1]/totalpop)
            beta=(1-probinfbase)*numIinf[-1]
            mut=(probmutbase*numIinf[-1])
            if numIinf[-1] == 0 or numIsus[-1]<=0:
                probIinf=0
            else:
                probIinf= alpha/(alpha+beta+mut)
            if numIinf[-1] == 0 or numIsus[-1]<=0:
                probImut=0
            else:
                probImut=mut/(alpha+beta+mut)
            if numIinf[-1]>0:
                change=np.random.choice([0,1,2],p=[probIinf, ...
                    probImut,1-probIinf-probImut])
                if change == 0:
                    numIinf[-1]=numIinf[-1]+1
                    numIsus[-1]=numIsus[-1]-1
                elif change==1:
                    numJinf[-1]=numJinf[-1]+1
                    numIinf[-1]=numIinf[-1]-1
                    numJsus[-1]=numJsus[-1]-1
                else:
                    numIinf[-1]=numIinf[-1]-1
                    numIcured[-1]=numIcured[-1]+1
            if numJinf[-1]>0:
                if numJinf[-1] == 0 or numJsus[-1]<=0:
                    probinf=0
                else:
                    probinf= ...
                        (probinfbase*numJsus[-1]*numJinf[-1]/totalpop)/((probinfbase*numJsu
                changeJ=np.random.choice([0,1],p=[probinf,1-probinf])
                if changeJ == 0:
                    numJinf[-1]=numJinf[-1]+1
                    numJsus[-1]=numJsus[-1]-1
                else:
                    numJinf[-1]=numJinf[-1]-1
                    numJcured[-1]=numJcured[-1]+1
```

MiA

```
            if numIinf[-1]<0:
                numIinf[-1]=0
            if numJcured[-1]>totalpop:
                numJcured[-1]=totalpop
            if numIcured[-1]>totalpop:
                numIcured[-1]=totalpop
            if numJinf[-1]<0:
                numJinf[-1]=0
            if numIsus[-1]<0:
                numIsus[-1]=0
            if numJsus[-1]<0:
                numJsus[-1]=0
        if numJinf[-1] == 0 and crossed==0:
            crossed==0
            muthappen.append(0)
            muthappentime.append(0)
            k+=1
            if (k % 100) == 0:
                print('{}% complete'.format(100*k/runs))
    return muthappen, muthappentime
# ================================================================================
#
# mut_avginf=[]
# mut_timeinf=[]
# for j in list(np.arange(0,1,0.01)):
#     muthappen_a,muthappentime_a=SIR_model_mutant_numinf(25/100,j,1,500,5000)
#     mut_avginf.append(statistics.mean(muthappen_a))
#     mut_timeinf.append(statistics.mean(muthappentime_a))
#
# mut_avginf_b=[]
# mut_timeinf_b=[]
# for j in list(np.arange(0,1,0.01)):
#     muthappen_a,muthappentime_a=SIR_model_mutant_numinf(50/100,j,1,500,5000)
#     mut_avginf_b.append(statistics.mean(muthappen_a))
#     mut_timeinf_b.append(statistics.mean(muthappentime_a))
#
# mut_avginf_c=[]
# mut_timeinf_c=[]
# for j in list(np.arange(0,1,0.01)):
#     muthappen_a,muthappentime_a=SIR_model_mutant_numinf(75/100,j,1,500,5000)
#     mut_avginf_c.append(statistics.mean(muthappen_a))
#     mut_timeinf_c.append(statistics.mean(muthappentime_a))
```

MiA

```
# plt.figure(22)
# fig1,ax1=plt.subplots()
# ax1.plot(list(np.arange(0,1,0.01)),mut_avginf,'b')
# ax1.plot(list(np.arange(0,1,0.01)),mut_avginf_b,'r')
# ax1.plot(list(np.arange(0,1,0.01)),mut_avginf_c,'g')
# green_patch = mpatches.Patch(color='green',label=r'Infection Rate of ... $\frac{75}{100}$')
# blue_patch = mpatches.Patch(color='blue',label=r'Infection Rate of ... $\frac{25}{100}$')
# red_patch = mpatches.Patch(color='red',label=r'Infection Rate of ... $\frac{50}{100}$')
# plt.legend(handles=[green_patch,red_patch,blue_patch], loc ='upper right')
# ax1.set(xlabel='Muatation Rate', ylabel='Average Number of Infected at ... Mutation')
# plt.savefig('Q3mu1i.jpg')
# fig1.show()
#
# ... numIinf_max,numIsus_max,numIcured_max,numJinf_max,numJsus_max,numJcured_max,time_max=SI
# while len(numIinf_max)<2000:
#     numIinf_max.append(numIinf_max[-1])
# while len(numJinf_max)<2000:
#     numJinf_max.append(numJinf_max[-1])
# while len(time_max)<2000:
#     time_max.append(time_max[-1])
# numtotinf_max=[numIinf_max+numJinf_max]
# numIinf_max=[numIinf_max]
# numJinf_max=[numJinf_max]
# time_max=[time_max]
# k=1
# while k<2000:
#     ... numIinf,numIsus,numIcured,numJinf,numJsus,numJcured,time=SIR_model(3/4,1/100,1,1000)
#     while len(numIinf)<2000:
#         numIinf.append(numIinf[-1])
#     while len(numJinf)<2000:
#         numJinf.append(numJinf[-1])
#     while len(time)<2000:
#         time.append(time[-1])
#     numIinf_max.append(numIinf)
#     numJinf_max.append(numJinf)
#     time_max.append(time)
#     holder=[]
#     holder.append(numIinf)
#     holder.append(numJinf)
#     arrays = [np.array(x) for x in holder]
#     numtotinf_max.append([np.sum(k) for k in zip(*arrays)])
#     k+=1
```

MiA

```
# arraystot = [np.array(x) for x in numtotinf_max]
# avgtot=[np.mean(k) for k in zip(*arraystot)]
# arraysI = [np.array(x) for x in numIinf_max]
# avgI=[np.mean(k) for k in zip(*arraysI)]
# arraysJ = [np.array(x) for x in numJinf_max]
# avgJ=[np.mean(k) for k in zip(*arraysJ)]
# arraystime = [np.array(x) for x in time_max]
# avgtime=[np.mean(k) for k in zip(*arraystime)]

# plt.figure(22)
# fig1,ax1=plt.subplots()
# plt.plot(avgtime,avgI,'b')
# plt.plot(avgtime,avgJ,'r')
# plt.plot(avgtime,avgtot,'g')
# green_patch = mpatches.Patch(color='green',label='Total Infections')
# blue_patch = mpatches.Patch(color='blue',label='Strain I Infections')
# red_patch = mpatches.Patch(color='red',label='Strain J Infections')
# plt.legend(handles=[green_patch,red_patch,blue_patch])
# ax1.set(xlabel='Time', ylabel='Average Number of Infections')
# plt.savefig('Q3mu1iii.jpg')
# fig1.show()
# ================================================================================

def ...
    SIR_model_vac(vac_startwork,vac_strength,vac_time,probinfbase,probmutbase,startsize,tot
    numIinf = [startsize]
    numJinf = [0]
    numIcured = [0]
    numJcured = [0]
    numIsus= [totalpop-startsize]
    numJsus= [totalpop]
    time=[0]
    vacI_on=0
    vacJ_on=0
    J_trigger=0
    I_trigger=0
```

MiA

```
    while (totalpop>numIinf[-1] and numIinf[-1]>0) or (totalpop>numJinf[-1] ...
        and numJinf[-1]>0):
        time.append(time[-1]+np.random.exponential(1/(numIinf[-1]+numJinf[-1])))
        numIinf.append(numIinf[-1])
        numJinf.append(numJinf[-1])
        numIsus.append(numIsus[-1])
        numJsus.append(numJsus[-1])
        numIcured.append(numIcured[-1])
        numJcured.append(numJcured[-1])
        if numIinf[-1]==vac_startwork and I_trigger==0:
            I_trigger=time[-1]
        if time[-1] ≥ I_trigger+vac_time and I_trigger>0:
            vacI_on = 1
        if time[-1] ≥ J_trigger+vac_time and J_trigger>0:
            vacJ_on = 1
        if numJinf[-1]==vac_startwork and J_trigger==0:
            J_trigger=time[-1]
        if vacI_on == 1:
            b=numIsus[-1]*vac_strength
            numIsus[-1]=numIsus[-1]-b
            numIcured[-1]=numIcured[-1]+b
        if vacJ_on ==1:
            b=numJsus[-1]*vac_strength
            numJsus[-1]=numJsus[-1]-b
            numJcured[-1]=numJcured[-1]+b
        alpha=(probinfbase*numIsus[-1]*numIinf[-1]/totalpop)
        beta=(1-probinfbase)*numIinf[-1]
        mut=(probmutbase*numIinf[-1])
        if numIinf[-1] == 0 or numIsus[-1]≤0:
            probIinf=0
        else:
            probIinf= alpha/(alpha+beta+mut)
        if numIinf[-1] == 0 or numIsus[-1]≤0:
            probImut=0
        else:
            probImut=mut/(alpha+beta+mut)
```

MiA

```
        if numIinf[-1]>0:
            change=np.random.choice([0,1,2],p=[probIinf, ...
               probImut,1-probIinf-probImut])
            if change == 0:
                numIinf[-1]=numIinf[-1]+1
                numIsus[-1]=numIsus[-1]-1
            elif change==1:
                numJinf[-1]=numJinf[-1]+1
                numIinf[-1]=numIinf[-1]-1
                numJsus[-1]=numJsus[-1]-1
            else:
                numIinf[-1]=numIinf[-1]-1
                numIcured[-1]=numIcured[-1]+1
        if numJinf[-1]>0:
            if numJinf[-1] == 0 or numJsus[-1]<=0:
                probinf=0
            else:
                probinf= ...
                    (probinfbase*numJsus[-1]*numJinf[-1]/totalpop)/((probinfbase*numJsus[-1
            changeJ=np.random.choice([0,1],p=[probinf,1-probinf])
            if changeJ == 0:
                numJinf[-1]=numJinf[-1]+1
                numJsus[-1]=numJsus[-1]-1
            else:
                numJinf[-1]=numJinf[-1]-1
                numJcured[-1]=numJcured[-1]+1
        if numIinf[-1]<0:
            numIinf[-1]=0
        if numJcured[-1]>totalpop:
            numJcured[-1]=totalpop
        if numIcured[-1]>totalpop:
            numIcured[-1]=totalpop
        if numJinf[-1]<0:
            numJinf[-1]=0
        if numIsus[-1]<0:
            numIsus[-1]=0
        if numJsus[-1]<0:
            numJsus[-1]=0
    return numIinf,numIsus,numIcured,numJinf,numJsus,numJcured,time
```

MiA

```
numIinf=[1]
while max(numIinf)<10:
    numIinf,numIsus,numIcured,numJinf,numJsus,numJcured,time=SIR_model_vac(5,0.05,0.5,3/4,
print('Did it, *snap*, Its a thing now, did it, *snap*')
plt.figure(31)
fig1,ax1=plt.subplots()
ax1.plot(time, numIinf,'r', label='Infected with I')
ax1.plot(time, numIsus, 'y', label='Suseptible to I')
ax1.plot(time, numIcured, 'g', label='Immune to I')
ax1.plot(time, numJinf, 'r--', label='Infected with J')
ax1.plot(time, numJsus, 'y--', label='Suseptible to J')
ax1.plot(time, numJcured, 'g--', label='Immune to J')
ax1.set(xlabel='Time', ylabel='Population in Group')
plt.gcf().subplots_adjust(left=0.15)
legend = ax1.legend()
legend.get_frame().set_alpha(0.5)
#plt.savefig('Q4aa55aa.jpg')
fig1.show()

numIinf=[1]
while max(numIinf)<50:
    numIinf,numIsus,numIcured,numJinf,numJsus,numJcured,time=SIR_model_vac(50,0.05,3,3/4,1
print('Did it, *snap*, Its a thing now, did it, *snap*')
plt.figure(31)
fig1,ax1=plt.subplots()
ax1.plot(time, numIinf,'r', label='Infected with I')
ax1.plot(time, numIsus, 'y', label='Suseptible to I')
ax1.plot(time, numIcured, 'g', label='Immune to I')
ax1.plot(time, numJinf, 'r--', label='Infected with J')
ax1.plot(time, numJsus, 'y--', label='Suseptible to J')
ax1.plot(time, numJcured, 'g--', label='Immune to J')
ax1.set(xlabel='Time', ylabel='Population in Group')
plt.gcf().subplots_adjust(left=0.15)
legend = ax1.legend()
legend.get_frame().set_alpha(0.5)
#plt.savefig('Q4aa5yy5aa.jpg')
fig1.show()
```

MiA

```
numIinf_max,numIsus_max,numIcured_max,numJinf_max,numJsus_max,numJcured_max,time_max=SIR_m
while len(numIinf_max)<20000:
    numIinf_max.append(numIinf_max[-1])
while len(numJinf_max)<20000:
    numJinf_max.append(numJinf_max[-1])
while len(time_max)<20000:
    time_max.append(time_max[-1])
numtotinf_max=[numIinf_max+numJinf_max]
numIinf_max=[numIinf_max]
numJinf_max=[numJinf_max]
time_max=[time_max]
k=1
while k<2000:
    print(k)
    numIinf,numIsus,numIcured,numJinf,numJsus,numJcured,time=SIR_model_vac(10,0.05,2,3/4,1
    while len(numIinf)<20000:
        numIinf.append(numIinf[-1])
    while len(numJinf)<20000:
        numJinf.append(numJinf[-1])
    while len(time)<20000:
        time.append(time[-1])
    numIinf_max.append(numIinf)
    numJinf_max.append(numJinf)
    time_max.append(time)
    holder=[]
    holder.append(numIinf)
    holder.append(numJinf)
    arrays = [np.array(x) for x in holder]
    numtotinf_max.append([np.sum(k) for k in zip(*arrays)])
    k+=1
arraystot = [np.array(x) for x in numtotinf_max]
avgtot=[np.mean(k) for k in zip(*arraystot)]
arraysI = [np.array(x) for x in numIinf_max]
avgI=[np.mean(k) for k in zip(*arraysI)]
arraysJ = [np.array(x) for x in numJinf_max]
avgJ=[np.mean(k) for k in zip(*arraysJ)]
arraystime = [np.array(x) for x in time_max]
avgtime=[np.mean(k) for k in zip(*arraystime)]

plt.figure(22)
fig1,ax1=plt.subplots()
plt.plot(avgtime,avgI,'b')
plt.plot(avgtime,avgJ,'r')
plt.plot(avgtime,avgtot,'g')
green_patch = mpatches.Patch(color='green',label='Total Infections')
blue_patch = mpatches.Patch(color='blue',label='Strain I Infections')
red_patch = mpatches.Patch(color='red',label='Strain J Infections')
plt.legend(handles=[green_patch,red_patch,blue_patch])
ax1.set(xlabel='Time', ylabel='Average Number of Infections')
plt.savefig('Q4mu1di55ii.jpg')
fig1.show()
```