ASSIGNMENT 3, MATHS IN ACTION A, 2020

TIANWEI LIU

Abstract. This paper studies several mathematical models regarding epidemic spreading. Namely, this paper will be exploring the branching process model, susceptible-infected-recovered model (SIR) and the Yule process model. Extensive mathematical derivations and computer simulations have been done to investigate the characteristics of the above models.

1. Introduction

With the rapid spreading of COVID-19 and on the verge of a global pandemic outbreak, here I present my research on several epidemic spreading models that can be relevant and their implication. A branching process describes individuals which independently multiply or die. A SIR model has a fixed population of K individuals and they are either susceptible, infected or recovered. The Yule process model comes in handy for analysis of an unstoppable disease. See [1] for more details.

2. Branching Process Model

2.1. **Critical Case.** In the critical case of branching process model where $\alpha = \beta$, the probability of a certain outbreak size denoted by $P(Z)$ follows an asymptotic decay $A_n \sim n^{-3/2}/\sqrt{4\pi}$. In the simulation, 10^5 samples are generated from a single cell to estimate $P(Z)$ plotted on Figure 1.

From Figure 1, one can confirm graphically that the simulation concedes with the theoretical asymptotic decay. The Likelihood that a small outbreak will occur is much larger than a large outbreak. In the critical case, the probability that an endemic occurs is much larger than that of a severe epidemic or even a global pandemic.

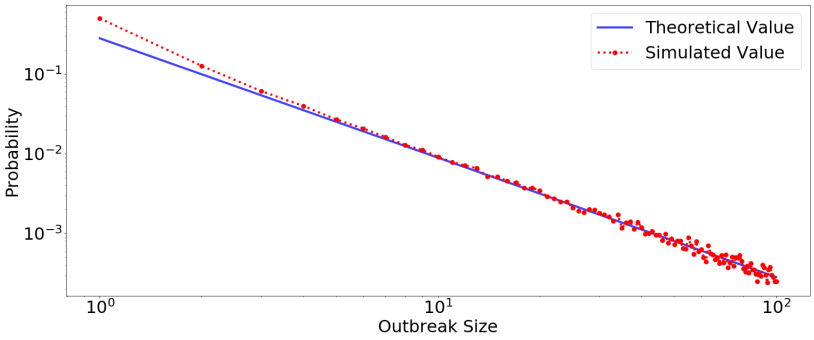

Figure 1. Simulation on the probability of reaching a given Z. The solid blue line is the slow asymptotic decay of the probability. The dotted red line with markers is plotted by the simulation on this probability. See Appendix A.

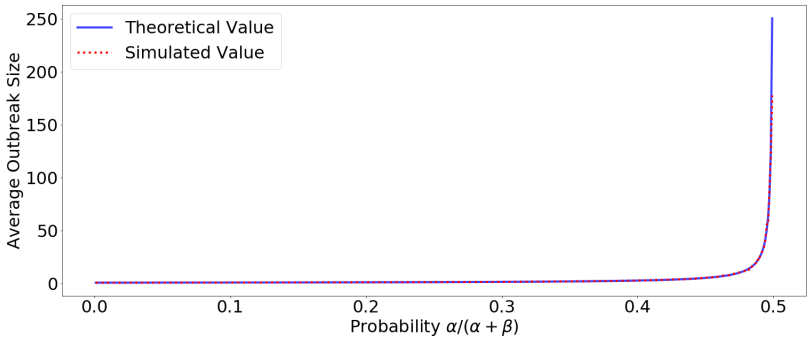

FIGURE 2. Simulation on $\hat{Z}$ w.r.t. the probability $\frac{\alpha}{\alpha+\beta}$. 10^4 samples are drawn for each probability value. The solid blue line is equation 1. The dotted red line is plotted by simulation. See Appendix B.

2.2. **Subcritical Case.** In the subcritical case, it is know that the average outbreak size $\hat{Z}$ satisfies the recursion equation $\hat{Z} = \frac{\beta}{\alpha+\beta} + 2\frac{\alpha}{\alpha+\beta}\hat{Z}$ where α is the rate at which each individual cell divides, and β is the rate at which each individual cell dies. Hence,

$$\hat{Z} = \frac{\beta}{\beta - \alpha} \tag{1}$$

is obtained. From Figure 2, the simulation agrees with the theory. $\hat{Z}$ remain very small in most regions of the subcritical case, but it shoots off quickly when approaching to the critical case.

2.3. **Supercritical Case.** In the supercritical case, it is not easy to derive $\hat{Z}$ analytically, and usually it is expressed as a ratio r to the system size M. However, there are sophisticated guesses that can be taken on this ratio, and this r can be approximated numerically. In Figure 3, the method that I propose is a polynomial regression with a degree of 3. A piece of a cubic function can properly fit the data in the current domain resulting:

$$r \approx -3.474 + 11.084P - 9.838P^2 + 3.231P^3 \tag{2}$$

where $P = \frac{\alpha}{\alpha+\beta}$. Scaling by M, r can be employed to compute $\hat{Z}$:

$$\hat{Z} = rM = (-3.474 + 11.084P - 9.838P^2 + 3.231P^3)M \tag{3}$$

3. The Susceptible-Infected-Recovered Model

3.1. **The Probability of Outbreak Sizes in the SIR Model.** The probability of different outbreak sizes in the SIR model are estimated via 10^5 simulations on outbreak sizes ranging from 1 to 100 inclusively. In Figure 4, as Z grows larger, $P(Z)$ first lies slightly above the asymptote and then diminishes rapidly. In the branching process model, the probability that an individual gets infected remains constant over time, whereas in the SIR model, this probability decreases over time. In the branching process model, individuals never develop immunity from the disease. On the other hand, individuals acquire immune in the SIR model after recovery. Therefore less individuals in the system can be victimized. When the outbreak reaches a certain size, a enough number of individuals have also become immune at the same time due to recovery, and the disease is very much likely to fade away soon.

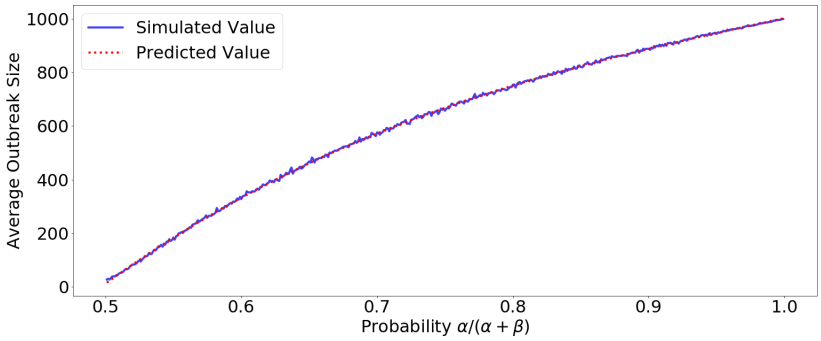

FIGURE 3. Similar plot to Figure 2 with $M = 10^3$. The solid blue line is obtained from simulation. The dotted red line is the polynomial regression model scaled by M. See Appendix C.

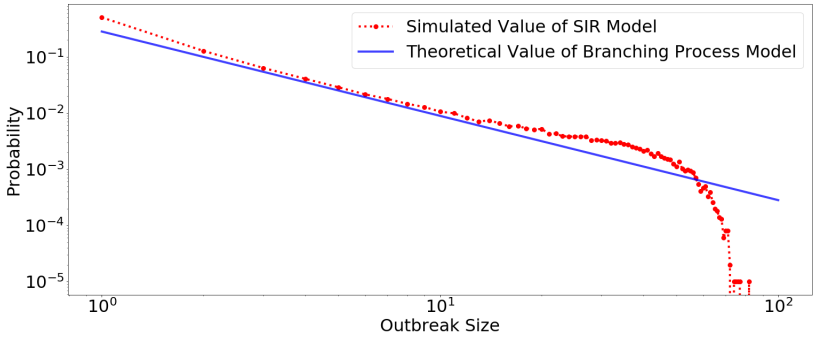

FIGURE 4. The Probability of Outbreak Sizes of The SIR Model Compared with that in Branching Process Model. The solid blue line is the asymptotic decay of the probability in the Branching Process Model. The dotted red line is the result from the simulation of the probability in the SIR model. See D.

3.2. **Approximating the Average Outbreak Size in the SIR Model.** In Figure 5, Although it seems that the simulated value is bounded a hyperbolic asymptotic, this is barely solvable either analytically or numerically. Instead, I propose another polynomial regression model of degree 5 to fit on the data. This provides a good fit of the data, and yields to:

(4)
$$\hat{Z} \approx 3.2539 + .10299M - 3.2419 * 10^{-4}M^2 + 5.9877 * 10^{-7}M^3 - 5.3877 * 10^{-10}M^4 + 1.8491 * 10^{-13}M^5$$

4. YULE PROCESS MODEL

4.1. **The Average Outbreak Size of the Yule Process Model.** At the early stages of each global pandemic, the disease is usually at its deadliest time, and the momentum is unstoppable for some time. The Yule process is perfect for modelling such situation. In theory,

(5)
$$\hat{Z} = e^{\alpha t}$$

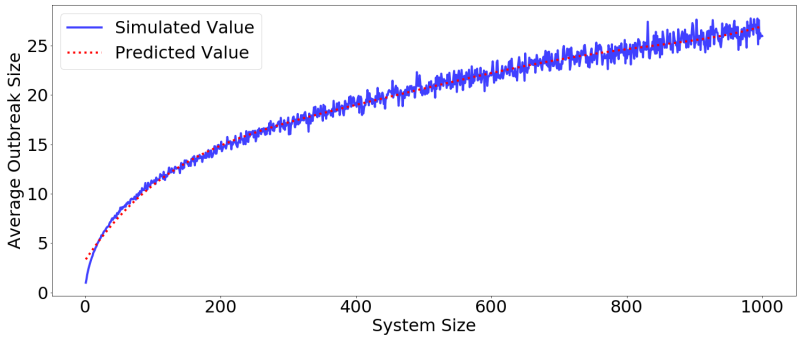

FIGURE 5. $\hat{Z}$ in the SIR model w.r.t. M. The solid blue line is obtained from simulation. The dotted red line is the result from the polynomial regression model. See Appendix E.

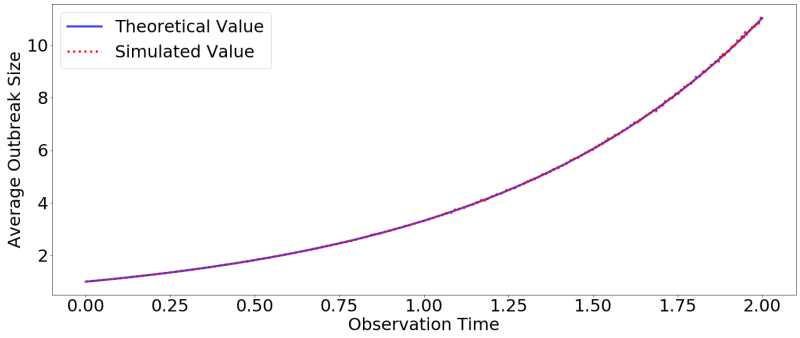

FIGURE 6. $\hat{Z}$ in the Yule process w.r.t. t. The solid blue line is 5 with $\alpha = 1.2$. The dotted red line is $\hat{Z}$ simulated from $5 * 10^4$ samples. See F.

where α denotes the birthrate of the bacteria or virus and t denotes the time. In Figure 6, the simulation concurs with the theory. Currently, the global outbreak of COVID-19 is following a Yule process with exponential growth. Necessary measures should be taken as soon as possible to slow down the outbreak. With time goes on, the spreading of COVID-19 will be transitioning into the SIR model, and eventually cease.

5. Discussion and Conclusion

The branching process model is more of a toy model in epidemiology, whereas the SIR model, and the Yule process model have their own unique application in the real world epidemics. They provide us invaluable insights on many characteristics of epidemic outbreak. Researching and utilizing those models will not only further mathematical understanding, but also save human lives.

References

[1] Lecture notes.

Appendix A. The Critical Case of the Branching Process Model

```
# Setup
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import time
from itertools import product

plt.rcParams["figure.figsize"] = (25, 10)
plt.rcParams["axes.labelsize"] = 32
plt.rcParams["xtick.labelsize"] = 32
plt.rcParams["ytick.labelsize"] = 32

prob_inf_critical = 0.5
#power = 10
outbreaksize = np.arange(1, 101).astype(int)
runs = 10**5
prob_theory_INF = outbreaksize ** (-3/2) / np.sqrt(4*np.pi)
prob_estimator_INF = np.zeros(outbreaksize.shape[0])

start = time.time()
for i, obs_size in enumerate(outbreaksize):
    if i % 25 == 0:
        print("Simulating for outbreak size:", obs_size,
            "\t Time Elapsed:", time.time() - start)
    num_inf = np.ones(runs)
    obs = np.ones(runs)
    while (num_inf > 0).any() and (obs <= obs_size).any():
        rng = np.random.uniform(size=runs)
        num_inf = np.where(obs > obs_size, 0, num_inf)
        num_inf = np.where(np.logical_and(rng < prob_inf_critical, num_inf > 0),
                           num_inf + 1, num_inf - 1)
        obs = np.where(np.logical_and(rng < prob_inf_critical, num_inf > 0),
                       obs + 1, obs)
        num_inf = np.where(num_inf < 0, 0, num_inf)
    prob_estimator_INF[i] = np.sum(obs == obs_size)

print("Time Elapsed:", time.time() - start)
prob_estimator_INF /= runs

plt.loglog(outbreaksize, prob_theory_INF, linewidth = 4, color="b",
            alpha=0.75, label="Theoretical Value")
plt.loglog(outbreaksize, prob_estimator_INF, linestyle=":", linewidth=4,
           marker=".", markersize=16, color="r", label="Simulated Value")
plt.xlabel("Outbreak Size")
plt.ylabel("Probability")
```

```
plt.legend(prop={"size" : 32})
plt.savefig("p1_prob.png")
```

Appendix B. The Subcritical Case of the Branching Process Model

```
# Setup
import numpy as np
import matplotlib.pyplot as plt
import time
from itertools import product

plt.rcParams["figure.figsize"] = (25, 10)
plt.rcParams["axes.labelsize"] = 32
plt.rcParams["xtick.labelsize"] = 32
plt.rcParams["ytick.labelsize"] = 32

runs = 10**4
prob_inf_subcritical = np.arange(1, 500) * 0.001
obs_sub_theory = (1 - prob_inf_subcritical) / (1 - 2 * prob_inf_subcritical)
obs_sub_sim = np.zeros(prob_inf_subcritical.shape)

start = time.time()
for i, prob in enumerate(prob_inf_subcritical):
    if i % 100 == 0:
        print("Simulating for alpha =", "{:.3f}".format(prob),
                "\t Time Elapsed:", time.time() - start)
    num_inf = np.ones(runs)
    obs = np.ones(runs)
    while (num_inf > 0).any():
        rng = np.random.uniform(size=runs)
        num_inf = np.where(np.logical_and(num_inf > 0, rng < prob),
                           num_inf + 1, num_inf - 1)
        obs = np.where(np.logical_and(num_inf > 0, rng < prob), obs + 1, obs)
        num_inf = np.where(num_inf < 0, 0, num_inf)
    obs_sub_sim[i] = np.mean(obs)
print("Time Elapsed:", time.time() - start)

plt.plot(prob_inf_subcritical, obs_sub_theory,
        linewidth = 4, color="b", alpha=0.75, label="Theoretical Value")
plt.plot(prob_inf_subcritical, obs_sub_sim,
        linestyle=":", linewidth=4, color="r", label="Simulated Value")
plt.xlabel("Probability $\\alpha / (\\alpha + \\beta)$")
plt.ylabel("Average Outbreak Size")
plt.legend(prop={"size" : 32})
plt.savefig("p2_subcritical.png")
```

Appendix C. The Supercritical Case of the Branching Process Model

```
    # Setup
import numpy as np
import matplotlib.pyplot as plt
import time
from itertools import product

plt.rcParams["figure.figsize"] = (25, 10)
plt.rcParams["axes.labelsize"] = 32
plt.rcParams["xtick.labelsize"] = 32
plt.rcParams["ytick.labelsize"] = 32

runs = 10**4
pop_size = 10**3
prob_inf_supcritical = np.arange(501, 1000) * 0.001
obs_sup_theory = (1 - prob_inf_supcritical) / (1 - 2 * prob_inf_supcritical)
obs_sup_sim = np.zeros(prob_inf_supcritical.shape)

start = time.time()
for i, prob in enumerate(prob_inf_supcritical):
    if i % 100 == 0:
        print("Simulating for alpha =",  "{:.3f}".format(prob),
                "\t Time Elapsed:", time.time() - start)
    num_inf = np.ones(runs)
    obs = np.ones(runs)
    while np.logical_and(num_inf > 0, obs <= pop_size).any():
        rng = np.random.uniform(size=runs)
        num_inf = np.where(np.logical_and(num_inf > 0, rng < prob),
                           num_inf + 1, num_inf - 1)
        num_inf = np.where(num_inf < 0, 0, num_inf)
        obs = np.where(np.logical_and(num_inf > 0, obs <= pop_size, rng < prob),
                       obs + 1, obs)
        obs = np.where(obs >= pop_size, np.inf, obs)
    obs = np.where(obs > pop_size, pop_size, num_inf)
    obs_sup_sim[i] = np.mean(obs)
print("Time Elapsed:", time.time() - start)

import sklearn as sklearn
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

X = PolynomialFeatures(degree = 3).fit_transform(prob_inf_supcritical.reshape(-1,1))
y = obs_sup_sim / pop_size
l = LinearRegression(fit_intercept = False).fit(X,y)
print(l.coef_)
```

```
plt.plot(prob_inf_supcritical, obs_sup_sim,
        linewidth = 4, color="b", alpha=0.75, label="Simulated Value")
plt.plot(prob_inf_supcritical, l.predict(X) * pop_size,
        linestyle=":", linewidth=4, color="r", label="Predicted Value")
plt.xlabel("Probability $\\alpha / (\\alpha + \\beta)$")
plt.ylabel("Average Outbreak Size")
plt.legend(prop={"size" : 32})
plt.savefig("p2_supcritical_fit.png")

np.sqrt(mean_squared_error(l.predict(X) * pop_size, obs_sup_sim))
```

Appendix D. Probability SIR Model

```
# Setup
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import time
from itertools import product

plt.rcParams["figure.figsize"] = (25, 10)
plt.rcParams["axes.labelsize"] = 32
plt.rcParams["xtick.labelsize"] = 32
plt.rcParams["ytick.labelsize"] = 32

outbreaksize = np.arange(1, 101).astype(int)
runs = 10**5
K = outbreaksize[-1]
prob_estimator_SIR = np.zeros(outbreaksize.shape[0])
prob_theory_INF = outbreaksize ** (-3/2) / np.sqrt(4*np.pi)

start = time.time()
for i, obs_size in enumerate(outbreaksize):
    if i % 25 == 0:
        print("Simulating for outbreak size:", obs_size,
                "\t Time Elapsed:", time.time() - start)
    num_inf = np.ones(runs)
    num_imu = np.zeros(runs)
    obs = np.ones(runs)
    while (num_inf > 0).any() and (obs <= obs_size + 1 ).any():
        rng = np.random.uniform(size=runs)
        num_inf = np.where(obs > obs_size, 0, num_inf)
        prob = (K - num_inf - num_imu) / (2 * K - num_inf - num_imu)
        num_inf = np.where(np.logical_and(num_inf > 0, rng < prob),
                           num_inf + 1, num_inf - 1)
```

```
        num_imu = np.where(np.logical_not(np.logical_and(num_inf > 0, rng < prob)),
                           num_imu + 1, num_imu)
        obs = np.where(np.logical_and(num_inf > 0, rng < prob, ), obs + 1, obs)
        num_inf = np.where(num_inf < 0, 0, num_inf)
    prob_estimator_SIR[i] = np.sum(obs == obs_size)

print("Time Elapsed:", time.time() - start)
prob_estimator_SIR  /= runs

plt.loglog(outbreaksize, prob_estimator_SIR, linestyle=":", linewidth=4,
           marker=".", markersize=16, color="r",
           label="Simulated Value of SIR Model")
plt.loglog(outbreaksize, prob_theory_INF, linewidth = 4, alpha=0.75, color="b",
           label="Theoretical Value of Branching Process Model")
plt.xlabel("Outbreak Size")
plt.ylabel("Probability")
plt.legend(prop={"size" : 32})
plt.savefig("p3_prob.png")
```

Appendix E. Outbreak Size SIR Model

```
sys_size = np.arange(1, 1001)
obs_sir_sim = np.zeros(sys_size.shape[0])
runs = 10**4

start = time.time()
for i, pop in enumerate(sys_size):
    if i % 250 == 0:
        print("Simulating for population size:", pop,
                "\t Time Elapsed:", time.time() - start)
    num_inf = np.ones(runs)
    num_imu = np.zeros(runs)
    obs = np.ones(runs)
    while (num_inf > 0).any() and (obs <= pop).any():
        rng = np.random.uniform(size=runs)
        prob = (pop - num_inf - num_imu) / (2 * pop - num_inf - num_imu)
        num_inf = np.where(np.logical_and(num_inf > 0, rng < prob),
                           num_inf + 1, num_inf - 1)
        obs = np.where(np.logical_and(num_inf > 0, obs <= pop, rng < prob),
                           obs + 1, obs)
        num_imu = np.where(np.logical_not(np.logical_and(num_inf > 0, rng < prob)),
                           num_imu + 1, num_imu)
        num_inf = np.where(num_inf < 0, 0, num_inf)
        obs = np.where(obs >= pop, np.inf, obs)
        num_inf = np.where(obs > pop, 0, num_inf)
    obs = np.where(obs > pop, pop, obs)
```

```
    obs_sir_sim[i] = np.mean(obs)

print("Time Elapsed:", time.time() - start)

import sklearn as sklearn
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

X = PolynomialFeatures(degree = 5).fit_transform(sys_size.reshape(-1,1))
y = obs_sir_sim
l = LinearRegression(fit_intercept = False).fit(X,y)
print(l.coef_)

plt.plot(sys_size, obs_sir_sim, linewidth = 4,
        color="b", alpha=0.75, label="Simulated Value")
plt.plot(sys_size, l.predict(X), linestyle=":",
        linewidth=4, color="r", label="Predicted Value")
plt.xlabel("System Size")
plt.ylabel("Average Outbreak Size")
plt.legend(prop={"size" : 32})
plt.savefig("p3_obs_fit.png")

np.sqrt(mean_squared_error(l.predict(X), obs_sir_sim))
```

Appendix F. Yule Process

```
# Setup
import numpy as np
import matplotlib.pyplot as plt
import time
from itertools import product

plt.rcParams["figure.figsize"] = (25, 10)
plt.rcParams["axes.labelsize"] = 32
plt.rcParams["xtick.labelsize"] = 32
plt.rcParams["ytick.labelsize"] = 32

runs = 5 * 10**4
alpha = 1.2
time = np.linspace(0, 2, num=1001)
yule_theory = np.exp(alpha * time)
yule_sim = np.zeros(time.shape[0])

for i, t in enumerate(time):
    cells_counter = np.zeros(runs)
```

```
    for j in range(runs):
        numcell = 1
        clock = np.random.exponential(1/(numcell*alpha))
        while clock < t:
            numcell += 1
            clock += np.random.exponential(1/(numcell*alpha))
        cells_counter[j] = numcell
    yule_sim[i] = np.mean(cells_counter)

plt.plot(time, yule_theory, linewidth = 4,
        color="b", alpha=0.75, label="Theoretical Value")
plt.plot(time, yule_sim, linestyle=":",
        linewidth=4, color="r", label="Simulated Value")
plt.xlabel("Observation Time")
plt.ylabel("Average Outbreak Size")
plt.legend(prop={"size" : 32})
plt.savefig("p4_yule.png")
```