STAT3004

Project One

May 19, 2020

Contents

Question One

Part A

We want to show that:

$$\begin{aligned}\mathbb{E}\left[X_t \mid X_0 = x_0\right] &= \alpha^t x_0 \\ \mathbb{E}\left[Y_t \mid X_0 = x_0\right] &= \alpha^{t-1}(1-\alpha)x_0\end{aligned}$$

First, we note the following one-step expectations:

$$\begin{aligned}\mathbb{E}\left[X_t \mid X_{t-1}\right] &= \alpha X_{t-1} \\ \mathbb{E}\left[Y_t \mid X_{t-1}\right] &= (1-\alpha)X_{t-1}\end{aligned}$$

These arise from the fact that α is the rate of non-infection from a single exposure, regardless of the number of people currently infected. Now, we will use $\mathbb{E}_{x_0}$ to denote the expectation given a particular starting condition x_0 and we find:

$$\begin{aligned}\mathbb{E}_{x_0} X_t &= \mathbb{E}_{x_0}\left(\mathbb{E}\left[X_t \mid X_{t-1}\right]\right) \\ &= \mathbb{E}_{x_0}\left[\alpha X_{t-1}\right] \\ &= \alpha \mathbb{E}_{x_0} X_{t-1}\end{aligned}$$

Iterating on this gives us:

$$\mathbb{E}_{x_0} X_t = \alpha^t x_0$$

as required.
Now consider:

$$\begin{aligned}\mathbb{E}_{x_0} Y_t &= \mathbb{E}_{x_0}\left(\mathbb{E}\left[Y_t \mid X_{t-1}\right]\right) \\ &= \mathbb{E}_{x_0}\left[(1-\alpha)X_{t-1}\right] \\ &= (1-\alpha)\mathbb{E}_{x_0} X_{t-1}\end{aligned}$$

Using the expectation for X_{t-1} as calculated above, this yields:

$$\mathbb{E}_{x_0} Y_t = (1-\alpha)\alpha^{t-1} x_0$$

as required.

Part B

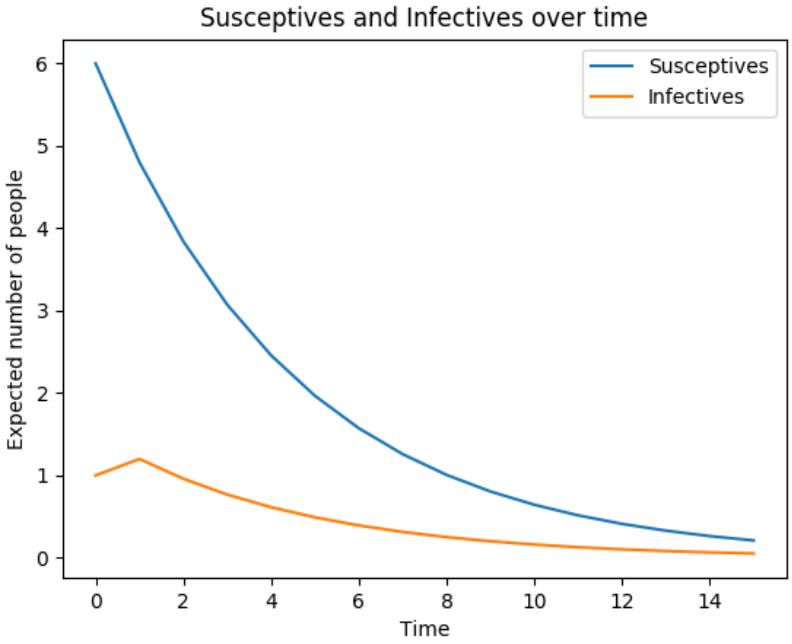
The following python code was used to plot the expectations from $t = 0, ..., 6$:

```python
import numpy as np
from matplotlib import pyplot as plt

t = np.linspace(1, 15, 15) # t = 0, ... , 15
x = []
y = []
x.append(6) # starting susceptibles
y.append(1) # starting infectives
alpha = 0.8 # rate of non-infection
for i in t:
    # calaculate expected values for each time interval
    x.append(alpha**i * x[0])
    y.append( (1 - alpha) * alpha **(i-1) * x[0])

# plot the expected values
t = np.linspace(0, 15, 16)
line1 = plt.plot(t, x, label="Susceptives")
line2 = plt.plot(t, y, label="Infectives")
plt.title("Susceptives and Infectives over time")
plt.legend(handles=[line1[0], line2[0]])
plt.ylabel("Expected number of people")
plt.xlabel("Time")
plt.show()
```

This then produced the following graph:



Question Two

Part A

We should note that we have X_{t+1} and Y_{t+1} as conditional binomial random variables. In particular, we have:

$$X_{t+1} \mid (X, Y)_t \sim \text{Bin}(X_t, \alpha^{Y_t})$$
$$Y_{t+1} \mid (X, Y)_t \sim \text{Bin}(X_t, 1 - \alpha^{Y_t})$$

From the expectation for a binomial distribution, we get:

$$\mathbb{E}\left[X_{t+1} \mid (X, Y)_t = (x, y)_t\right] = x_t \alpha^{y_t}$$
$$\mathbb{E}\left[Y_{t+1} \mid (X, Y)_t = (x, y)_t\right] = x_t (1 - \alpha^{y_t})$$

Part B

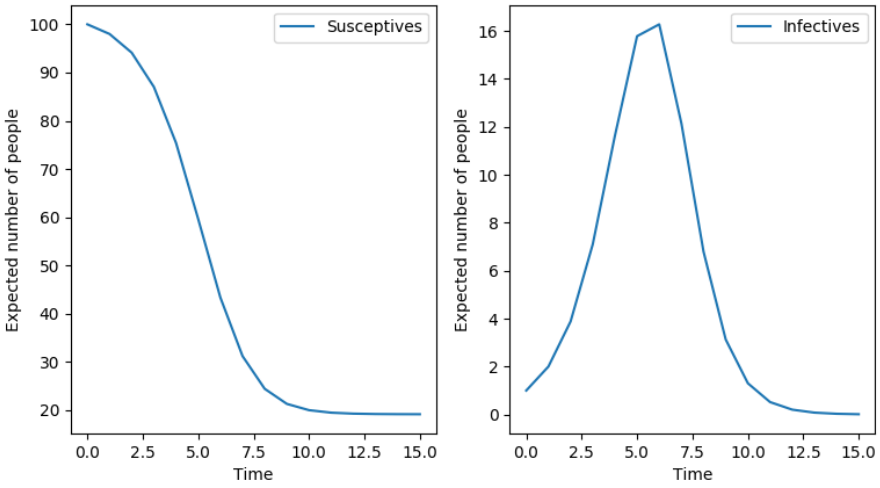
The following python code was used to reproduce the Figure 4.2 from [EM-4] and to generate a plot of the trajectory of expected values on the (X, Y) plane:

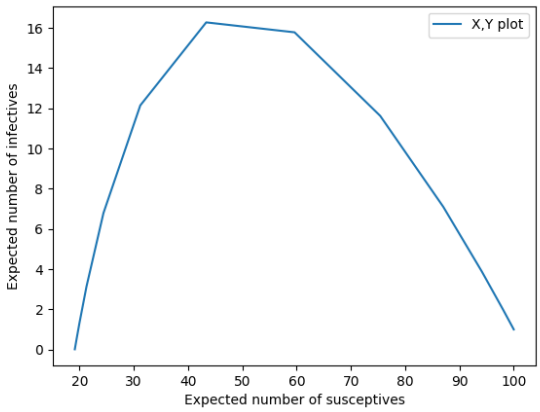
```
import numpy as np
from matplotlib import pyplot as plt

x = [100] # we have 100 susceptibles and 1 infective at t=0
y = [1]
t = np.linspace(1, 15, 15)
alpha = 0.98
for i in t:
    i = int(i)
    x.append(x[i-1] * alpha **(y[i-1]))
    y.append(x[i-1] * (1 - alpha ** (y[i-1])))

# plot the expected values
t = np.linspace(0, 15, 16)
f, (ax1, ax2) = plt.subplots(1, 2)
line1 = ax1.plot(t, x, label="Susceptives")
ax1.set_ylabel("Expected number of people")
ax1.set_xlabel("Time")
ax1.legend()
line2 = ax2.plot(t, y, label="Infectives")
ax2.legend()
ax2.set_ylabel("Expected number of people")
ax2.set_xlabel("Time")
plt.show()
f, ax3 = plt.subplots(1, 1)
line3 = ax3.plot(x, y, label="X,Y plot")
ax3.legend()
ax3.set_ylabel("Expected number of infectives")
ax3.set_xlabel("Expected number of susceptives")
plt.show()
```

This produced the following graphs:





Question Three

Part A

The following code was used to generate heatmaps of the transition probability matrix as in equation (4.1.5) for the Greenwood model for various values of x_0, ($x_0 = 5$, $x_0 = 10$, $x_0 = 20$ respectively):

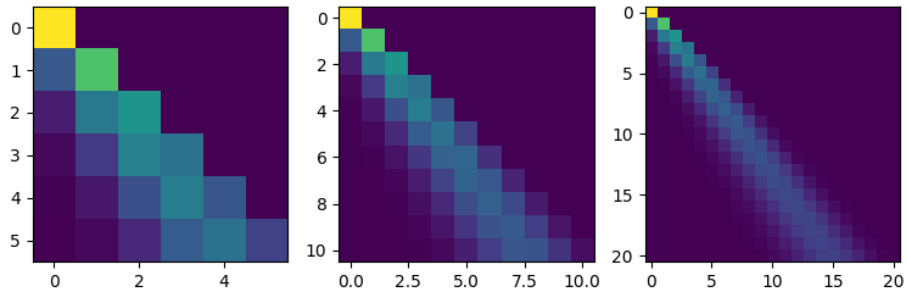
```python
import numpy as np
from scipy.special import comb
import matplotlib.pyplot as plt

alpha = 0.72 # arbitrary choice of alpha
x0 = 5 # set x0 here
matr1 = np.zeros((x0 + 1, x0 + 1)) # square matrix of order x0 + 1
f, (ax1, ax2, ax3) = plt.subplots(1, 3)
for i in range(x0 + 1): # rows go from 0 to x0 inclusive
    for j in range(x0 + 1): # same for columns
        if j <= i: # if j > i just leave it as zero
            matr1[i][j] = (alpha ** j) * ((1 - alpha) ** (i - j)) * comb(i, j)
ax1.imshow(matr1)

x0 = 10 # set x0 here
matr2 = np.zeros((x0 + 1, x0 + 1)) # square matrix of order x0 + 1
for i in range(x0 + 1): # rows go from 0 to x0 inclusive
    for j in range(x0 + 1): # same for columns
        if j <= i: # if j > i just leave it as zero
            matr2[i][j] = (alpha ** j) * ((1 - alpha) ** (i - j)) * comb(i, j)
ax2.imshow(matr2)

x0 = 20 # set x0 here
matr3 = np.zeros((x0 + 1, x0 + 1)) # square matrix of order x0 + 1
for i in range(x0 + 1): # rows go from 0 to x0 inclusive
    for j in range(x0 + 1): # same for columns
        if j <= i: # if j > i just leave it as zero
            matr3[i][j] = (alpha ** j) * ((1 - alpha) ** (i - j)) * comb(i, j)
ax3.imshow(matr3)
plt.show()
```

These are the heatmaps generated by the code:



Part B

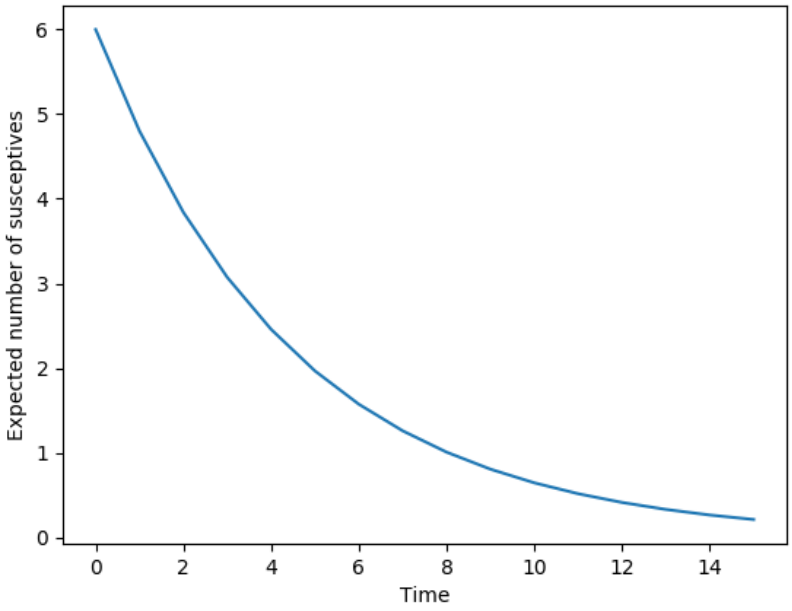
There are three communicating classes in this Markov Chain, which are the following:

$$C_1 = X_t = 0$$
$$C_2 = X_t = X_{t-1}$$
$$C_3 = \Omega - C_1 \cup C_2$$

C_1 is the class where the entire population has been infected and recovered, which means that the number of susceptibles will remain at zero. This is a recurrent class. C_2 is the class where no one in a single generation gets infected. Since the infectives are only infective for a single generation, the number of susceptibles will remain at its current level. This is a recurrent class. C_3 is the class where some susceptibles remain and some people are being infected each generation (i.e. normal epidemic behaviour). This is a transient class.

Part C

The plot below was generated using the estimation method in 3c:



This plot, matching the one given in 1b, was produced using the following code:

```
import numpy as np
from numpy.linalg import matrix_power
from scipy.special import comb
import matplotlib.pyplot as plt

alpha = 0.8 # same choice of alpha as in 1b
x0 = 6 # same as 1b
P = np.zeros((x0 + 1, x0 + 1)) # square matrix of order x0 + 1
e = np.zeros(x0 + 1)
e[x0] = 1
v = np.zeros((x0 + 1, 1))
for k in range(x0 + 1):
    v[k][0] = k
for i in range(x0 + 1): # rows go from 0 to x0 inclusive
    for j in range(x0 + 1): # same for columns
        if j <= i: # if j > i just leave it as zero
            P[i][j] = (alpha ** j) * ((1 - alpha) ** (i - j)) * comb(i, j)

x = []
T = np.linspace(0, 15, 16)
for t in T:
    if t == 0:
        x.append(x0)
    else:
        # generate expected values
        x.append(float(e.dot(matrix_power(P, int(t)).dot(v))))

plt.plot(T, x)
plt.show()
```

Why does this work? First, by raising P to the power t, each element presents the t-step transition probabilities. Then, the vector $e_{x_0+1}^T$ eliminates all rows but the lowest. These are the t-step transition probabilities from x_0 to the rest of the sample space. Multiplying this by the vector v gives the weighted sum of these probabilities, which is, in fact, the expectation.

Part D

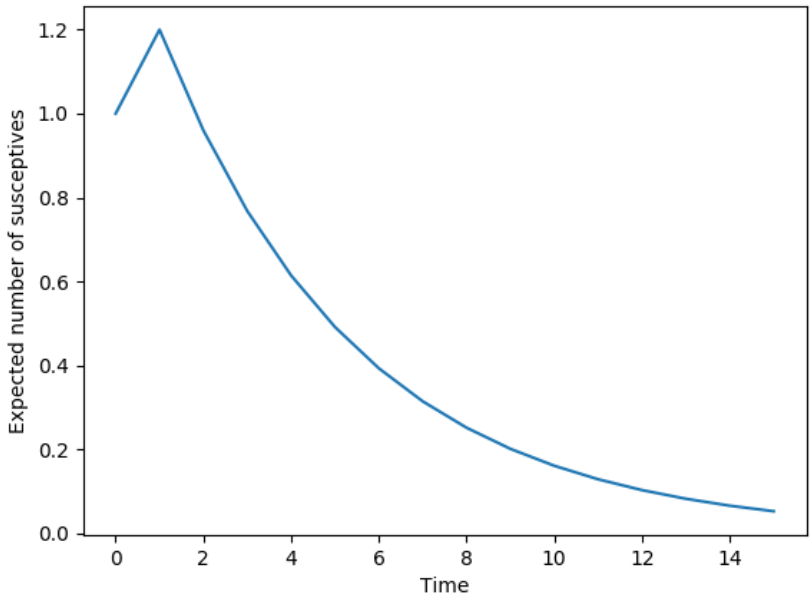
We can do a similar thing for Y_t using a slightly different construction. Consider a matrix Q defined as the matrix of one-step probabilities from X to Y. That is:

$$\begin{aligned} q_{ij} &= \mathbb{P}\left(Y_{t+1} = j \mid X_t = i,\ Y_t > 0\right) \\ &= \alpha^{i-j}\left(1-\alpha\right)^j \binom{i}{j} \end{aligned}$$

Now, we can apply the following to determine the expected values for Y_t in the following way:

$$\mathbb{E}_{x_0}(Y_t) = e_{x_0+1}^T P^{t-1} Q v$$

This technique generates the following plot:



The plot can be generated using the following code:

```python
import numpy as np
from numpy.linalg import matrix_power
from scipy.special import comb
import matplotlib.pyplot as plt
alpha = 0.8 # same choice of alpha as in 1b
x0 = 6 # same as 1b
P = np.zeros((x0 + 1, x0 + 1)) # square matrix of order x0 + 1
Q = np.zeros((x0 + 1, x0 + 1))
e = np.zeros(x0 + 1)
e[x0] = 1
v = np.zeros((x0 + 1, 1))
for k in range(x0 + 1):
    v[k][0] = k
for i in range(x0 + 1): # rows go from 0 to x0 inclusive
    for j in range(x0 + 1): # same for columns
        if j <= i: # if j > i just leave it as zero
            P[i][j] = (alpha ** j) * ((1 - alpha) ** (i - j)) * comb(i, j)
            Q[i][j] = (alpha ** (i - j)) * ((1 - alpha) ** j) * comb(i, j)
y = []
T = np.linspace(0, 15, 16)
for t in T:
    if t == 0:
        y.append(1)
    else:
        # generate expected values
        eP = e.dot(matrix_power(P, int(t) - 1))
        ePQ = eP.dot(Q)
        ePQv = float(ePQ.dot(v))
        y.append(ePQv)

plt.plot(T, y)
plt.xlabel("Time")
plt.ylabel("Expected number of susceptives")
plt.show()
```

Question Four

NB - All the below numerical calculations were performed with $\alpha = 0.75$, and assuming $Y_0 = 1$

Part A

There are several key things to understand about equation 4.1.6. First, a conceptual overview of the meaning of the question. The probability of there being j susceptibles at time t is the sum of the probabilities of all susceptibles there could have been at time $t-1$ multiplied by the probability of going from that number of susceptibles to j susceptibles at time t.
The other thing to understand is the bounds on the sum. Note that if the number of susceptibles does not decrease at all between generations, then $Y_t = 0$ at that time and the infection has stopped. Thus, because the infection is still going, the highest amount of susceptibles there could be at time $t-1$ is $x_0 - (t-1)$. This is thus the top bound for how many susceptibles there be in the generation before the current one.
The lower bound is given as $i = j + 1$. Since the infection is still going at time t, if there are to be j susceptibles at time t, there must be at least $j+1$ susceptibles at time $t-1$. Thus, this equation covers all possible "paths" to there being j susceptibles at time t.

Part B

We want all of the possible cases where we have $W = 5$ or $W = 6$. The slowest that the infection can progress is with one person becoming infected each generation, and the infection only ending once everyone has been infected. This means that $t \leq 7$. Now let:

$$q_{kni} = \mathbb{P}((W,T) = (k,n) \mid X_0 = i, Y_0 > 0)$$

So we consider:

$$\mathbb{P}(W > 4) = \sum_{t=1}^{7} q_{5t6} + \sum_{t=1}^{7} q_{6t6}$$

Now, there are several cases that we can ignore. In fact, if we have $n = 1,\ k \neq 0$ then $p_{kni} = 0$ for any i, and other similar restrictions. For $\alpha = 0.75,\ x_0 = 6$, this was calculated to be 0.2407 using the displayed code:

```python
import numpy as np
from numpy.linalg import matrix_power
from scipy.special import comb
import matplotlib.pyplot as plt

def p_ij(i, j, alpha):
    # probability of going from i susceptibles to j susceptibles given alpha
    if j <= i:
        return comb(i, j) * ((1 - alpha) ** (i - j)) * (alpha ** j)
    return 0

def p_jt(j, t, x0, alpha):
    # probability of having j susceptibles at time t given starting susceptibles x0 and alpha
    s = 0
    if t == 0 and j == x0:
        return 1
    elif t == 0 and j != x0:
        return 0
    for i in range(j+1, x0 - t + 2):
        s += p_ij(i, j, alpha) * p_jt(i, t - 1, x0, alpha)
    return s

def q_kni(k, n, alpha, i):
    # probability that T = n and W = k given x0 and alpha
    if k == 0 and n != 1: # if nobody got infected, it must have died in a single generation
        return 0
    elif k >= 1:
        if n < 2 or n > k + 1:
            return 0
    ret = p_jt(i - k, n - 1, i, alpha) * (alpha ** (i - k))
    return ret

su = 0
for k in range(5, 7):
    for n in range(1, 8):
        su += q_kni(k, n, 0.75, 6)
print(su)
```

Part C

I carried out a Monte Carlo simulation using 10^6 simulations of the epidemic with $\alpha = 0.75$, $x_0 = 6$, and calculated a proportion of times that $W > 4$, and got a value of 0.2405, which validates the model. This was caluclated using the displayed code:

```python
import numpy as np

def single_attempt(x0, alpha, W):
    # one walk through with the given starting conditions
    if W > x0:
        return 0
    w = 0
    x = x0
    while x > 0:
        sg = single_generation(x, w, alpha)
        if sg == 0:
            break
        else:
            x = x - sg
            w = w + sg

    if w > W:
        return 1
    else:
        return 0

def single_generation(x, w, alpha):
    ret = 0
    for i in range (x):
        r = np.random.uniform()
        if r >= alpha:
            ret += 1
    return ret

N = 1000000
s = 0
for i in range(N):
    s += single_attempt(6, 0.75, 4)
print(float(s)/float(N))
```

Part D

Recall P, the matrix of one-step transition probabilities for X_t:

$$P = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1-\alpha & \alpha & 0 & \dots & 0 \\ (1-\alpha)^2 & 2(1-\alpha)\alpha & \alpha^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (1-\alpha)^{x_0} & x_0(1-\alpha)^{x_0-1}\alpha & \binom{x_0}{2}(1-\alpha)^{x_0-2}\alpha^2 & \dots & \alpha^{x_0} \end{bmatrix}$$

Since we are interested in the time T for which the infection lasts, we are interested in the probabilities of non-repeating steps. So, consider the matrix of non-repeating steps as follows:

$$\bar{P} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 1-\alpha & 0 & 0 & \dots & 0 \\ (1-\alpha)^2 & 2(1-\alpha)\alpha & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (1-\alpha)^{x_0} & x_0(1-\alpha)^{x_0-1}\alpha & \binom{x_0}{2}(1-\alpha)^{x_0-2}\alpha^2 & \dots & 0 \end{bmatrix}$$

Now, we have that $\mathbb{P}(T = t)$ is the probability of taking $t-1$ non-repeating steps starting with $X_0 = x_0$, followed by one repeating step. To factor the repeating step, consider:

$$v = \begin{bmatrix} 1 \\ \alpha \\ \alpha^2 \\ \vdots \\ \alpha^{x_0} \end{bmatrix}$$

(The book has this as a diagonal matrix multiplied by a unit vector, but it can also be considered as a vertical vector since the destination of the one- step is unimportant, so long as we know that it has, in fact, repeated). Finally, we take the vector $A = (0, 0, 0, \dots, 1)$ since $\mathbb{P}(X_0 = x_0) = 1$ and we have:

$$\mathbb{P}_T(t) = A\bar{P}^{t-1}v$$

Now, the PGF of T is given by:

$$\begin{aligned} G_T(z) &= \mathbb{E}z^T \\ &= \sum_{t=1}^{\infty} z^t \mathbb{P}_T(t) \\ &= A \sum_{t=1}^{\infty} \left(z^t \bar{P}^{t-1} \right) v \end{aligned}$$

The above equation will be left in that form for ease of differentiation later, (although it is possible to use a geometric series trick to simplify it further) . Now, W represents the eventual size of the epidemic, that is, the total number of infectives. Since at each time step there may be a variable number of infectives added to the count of infectives with each time step, we must first create a matrix of one-step PGFs for the new infectives in the following way:

$$\bar{P}_\phi = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 1-\alpha\phi & 0 & 0 & \dots & 0 \\ [(1-\alpha)\phi]^2 & 2(1-\alpha)\alpha\phi & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ [(1-\alpha)\phi]^{x_0} & x_0[(1-\alpha)\phi]^{x_0-1}\alpha & \binom{x_0}{2}[(1-\alpha)\phi]^{x_0-2}\alpha^2 & \dots & 0 \end{bmatrix}$$

We see that for each matrix element, we have φ^{i-j} due to there being $i-j$ new infectives at this step. We can thus deduce the joint pdf as:

$$G_{W,T}(\phi, z) = A \sum_{t=1}^{\infty} \left(\bar{P}_\phi^{t-1} z^t \right) v$$

However, for our purpose, we are interested only in the marginal PGF of W, and so we take:

$$G_{W,T}(\phi, 1) = G_W(\phi) = A \sum_{t=1}^{\infty} \left(\bar{P}_\phi^{t-1} \right) v$$

Now we are ready to apply this to our case. Note that $\bar{P}_\phi^k = 0$ if we have $k > x_0$. In our case, we have $x_0 = 6$ and $\alpha = 0.75$. We also note that as raise $\bar{P}_\phi$ to various powers, the power of ϕ in each position is unchanged. So we can calculate the various powers of $\bar{P}$ and then "add back" the ϕs. So, we can now calculate the various powers of $\bar{P}$ numerically in a relatively simple way with python, using the following code.

```python
import numpy as np
from numpy.linalg import matrix_power
from scipy.special import comb

x0 = 6
alpha = 0.75
Pbar = np.zeros((x0 + 1, x0 + 1))
for i in range(x0 + 1):
    for j in range(x0 + 1):
        if j < i:
            Pbar[i][j] = comb(i, j) * ((1 - alpha) ** (i - j)) * (alpha ** j)

Pbars = [np.identity(x0 + 1), Pbar]
for k in range(2, x0 + 1):
    Pbars.append(matrix_power(Pbar, k))
```

To give the first couple, we have:

$$\bar{P}^0_\phi = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\bar{P}^1_\phi = \begin{bmatrix} 0 & 0 & \dots & 0 \\ (1-\alpha)\phi & 0 & \dots & 0 \\ [(1-\alpha)\phi]^2 & 2\alpha(1-\alpha)\phi & \dots & 0 \\ [(1-\alpha)\phi]^3 & 3\alpha[(1-\alpha)\phi]^2 & \dots & 0 \\ [(1-\alpha)\phi]^4 & 4\alpha[(1-\alpha)\phi]^3 & \dots & 0 \\ [(1-\alpha)\phi]^5 & 5\alpha[(1-\alpha)\phi]^4 & \dots & 0 \\ [(1-\alpha)\phi]^6 & 6\alpha[(1-\alpha)\phi]^5 & \dots & 0 \end{bmatrix}$$

If we take a sum of these for $\alpha = 0.75$ numerically (and rounding off each entry

to two decimal places for display reasons), we obtain:

$$\sum_{t=1}^{x_0+1} \bar{P}^{t-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.25\phi & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.16\phi^2 & 0.38\phi & 1 & 0 & 0 & 0 & 0 \\ 0.12\phi^3 & 0.30\phi^2 & 0.42\phi & 1 & 0 & 0 & 0 \\ 0.10\phi^4 & 0.25\phi^3 & 0.39\phi^2 & 0.42\phi & 1 & 0 & 0 \\ 0.09\phi^5 & 0.23\phi^4 & 0.35\phi^3 & 0.43\phi^2 & 0.40\phi & 1 & 0 \\ 0.08\phi^6 & 0.21\phi^5 & 0.33\phi^4 & 0.41\phi^3 & 0.44\phi^2 & 0.36\phi & 1 \end{bmatrix}$$

Pre-multiplying this by A gives us:

$$A \sum_{t=1}^{x_0+1} \bar{P}^{t-1} = \begin{bmatrix} 0.08\phi^6 & 0.21\phi^5 & 0.33\phi^4 & 0.41\phi^3 & 0.44\phi^2 & 0.36\phi & 1 \end{bmatrix}$$

We can multiply by v to finally get (rounded to 4 decimal places) that:

$$\begin{aligned} G_W(\phi) =& 0.0822\phi^6 + 0.1585\phi^5 + 0.1854\phi^4 + 0.1731\phi^3 + 0.1384\phi^2 + \\ & 0.0845\phi + 0.1780 \end{aligned}$$

Now, of particular interest to us, we have that:

$$\begin{aligned} \mathbb{P}(W = 5) &= 0.1585 \\ \mathbb{P}(W = 6) &= 0.0822 \end{aligned}$$

Summing these gives that $\mathbb{P}(W > 4) = 0.2407$ as predicted before.

Question Five

Part A

Considering that we have $X_0 = x_0$ for some $x_0 < \infty$, we have the following state space for X_t:

$$X_t \in [0, x_0] \cap \mathbb{Z}$$

Alternatively, we can consider a state space for the conditional $X_t \mid X_{t-1}$:

$$X_t \in [0, X_t - 1] \cap \mathbb{Z}$$

This state space completely defines the process, since Y_t is defined by X_t and X_{t-1} (namely, that $Y_t = X_{t-1} - X_t$).

Part B

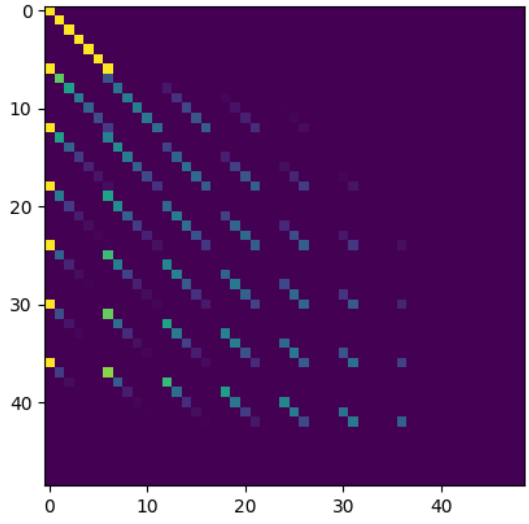
There are three distinct communicating classes for (X_t, Y_t):

$$C_1 = \{(X_t, Y_t) \mid X_t = 0\}$$
$$C_2 = \{(X_t, Y_t) \mid X_t > 0, Y_t > 0\}$$
$$C_3 = \{(X_t, Y_t) \mid X_t > 0, Y_t = 0\}$$

C_1 is a closed class; this is the case where everyone in the community has been sick and recovered; thus, there can be no new infections or change of state. C_2 is an open class; this is the case where the infection is still going. At any step, we could go from this class to either C_1 or C_3. C_3 is actually a set of closed classes, (since the value of X_t may vary). Each of these classes is a closed class, where the infection has been quashed in the population before everyone contracted it.

Part C

We can plot the matrix of transition probabilities (as in 4.2.2) in the following way:



This plot was generated using the following python code:

```python
import numpy as np
from scipy.special import comb
from matplotlib import pyplot as plt

alpha = 0.75
x0 = 6
dim = (x0 + 1) ** 2
P = np.zeros((dim, dim))
ue = 0
oe = 0
for i in range(x0 + 1):
    for j in range(x0 + 1):
        for k in range(x0 + 1):
            for l in range(x0 + 1):
                if j + l == k and (i != 0 or j == 0):
                    P[i*x0 + k][j*x0 + l] = comb(k, l) * (( 1 - (alpha ** i)) ** j) * (alpha ** (i * l))
plt.imshow(P)
plt.show()
```

Part D

A Monte Carlo simulation found that $\mathbb{P}(W > 4) = 0.4396$. This is significantly higher than the probability calculated in 4c. This is because, under the Reed-Frost model, the infectivity of the disease rises exponentially with the number of infectives in the population, creating a greater tendency for the disease to spiral further out of control (and thus affect a greater proportion of the population before it has run its course).
The python code used to perform this simulation is located overleaf:

```python
import numpy as np

def single_attempt(x0, y0, alpha, W):
    # one walk through with the given starting conditions
    if W > x0: # if there's not enough people in the population, prob = 0
        return 0
    w = 0
    x = x0
    y = y0
    while x > 0:
        sg = single_generation(x, y, w, alpha) # how many infected in a single generation
        if sg == 0: # it ended here
            break
        else:
            x = x - sg
            w = w + sg
            y = sg
    if w > W:
        return 1
    return 0

def single_generation(x, y, w, alpha):
    ret = 0
    for i in range (x): # if x is zero, this just returns zero
        r = np.random.uniform()
        if r >= alpha ** y:
            ret += 1
    return ret

N = 1000000
s = 0
for i in range(N):
    s += single_attempt(6, 1, 0.75, 4)
print(float(s)/float(N))
```

Question Six

Before proceeding with the analysis, it will be helpful to lay out some assumptions for the sake of the model.

Assumptions

This model (the Reed-Frost model) and this situation functions on a series of assumptions:

- There is one (and only one) opportunity for contact during the day (this might be when everyone is getting tested or something similar).

- During this time of contact, the probability of coming into contact with a given person is $p = 0.1$. If that person is sick, the infection transmits with probability $\beta = 0.05$.
- Infectious contact with one individual is sufficient to transmit the disease. Thus, to avoid infection, an individual must avoid infectious contact with ALL currently infected individuals.
- We assume that infectious contact with an infected individual is the ONLY method of transmission.
- We assume that there is always enough demand on travel for the motel to always be filled.
- We assume that the flow into the motel represents a standard sampling of individuals wanting to come into the country (i.e. there is not a higher or lower chance of people coming to the motel having the infection).
- We assume that whenever all well people are released from the motel, the next group of people enters and is tested that day.

With these assumptions, we proceed with some preanalysis before performing Monte Carlo simulation to determine the long-term proportion of infections coming out of the motel. First, for the same of simplicity, we calculate the probability of an individual avoiding infectious contact with a single infected individual:

$$\begin{aligned}\alpha &= 1 - p + p(1 - \beta) \\ &= 1 - p\beta \\ &= 0.995\end{aligned}$$

We can split the population currently in the motel at time t into susceptives and infectives, denoted by X_t and Y_t respectively. Here, we have the following initial conditions:

$$\begin{aligned}X_0 &\sim \text{Bin}(x_0, 1 - \eta) \\ Y_0 &= x_0 - X_0\end{aligned}$$

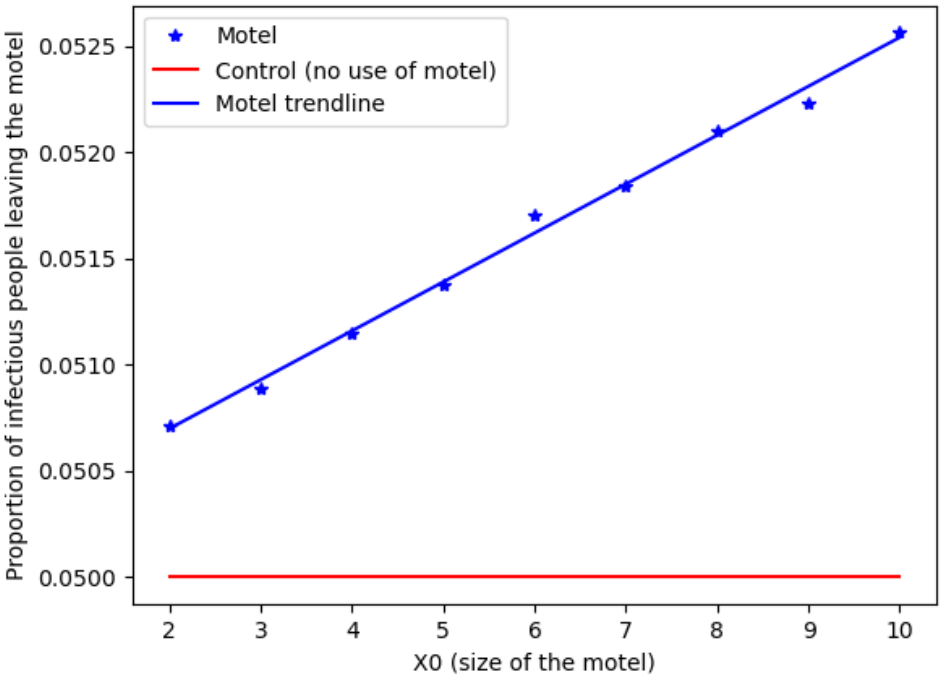
Next, we have the one-step transition probability:

$$\mathbb{P}\left((X, Y)_{t+1} = (x, y)_{t+1} \mid (X, Y)_t = (x, y)_t\right) = \binom{x_t}{x_{t+1}} \alpha^{y_t x_{t+1}} (1 - \alpha^{y_t})^{y_{t+1}}$$

Essentially this means that we have:

$$X_{t+1} \mid (X, Y)_t \sim \text{Bin}(X_t, 1 - \alpha^{Y_t})$$
$$Y_{t+1} = X_t - X_{t+1}$$

However, this is somewhat different from the standard Reed-Frost model, because the model is "restarted" each time the infection peters out.
For each value of $x_0 \in \{2, 3, 4, \ldots, 10\}$ I conducted 10 000 one hundred day trials and averaged the rate of infectious people leaving the motel, and compared it to what the value would be if the motel was not operating $\eta = 0.05$. This generated the following plot:



As can be seen here, there appeared to be a strong, linearly increasing relationship between motel size and the proportion of infectious people leaving the motel, $R^2 = 0.995$. The code to perform this simulation is included in the Appendix. This linear model had a gradient of $m = 0.0002301$ and an intercept of $c = 0.050$.

Appendix

This is the code used to perform the analysis found in Question Six.

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression

def ndayattempt(x0, eta, p, beta, n):
    """
    Simulates the motel for n days, returning the
    proportion of unwell individuals that leave
    the motel for a given set of initial conditions.
    Parameters:
        - x0: the number of people who enter the motel
              each time
        - eta: the probability of a person coming in
               being ill with COVID-19
        - p: the probability of having contact with
             any one person in the hotel
        - beta: the probability of a contact with an
                infective person being infectious
        - n: the number of days to run the simulation
    Returns:
        the proportion of individuals leaving the
        motel who do so sick
    """
    alpha = 1 - p*beta
    d = n
    totalOut = 0
    infectiousOut = 0
    while d > 0:
        i, j, k = single_attempt(x0, eta, alpha, d)
        infectiousOut += i
        totalOut += i + j
        d = k

    output = float(infectiousOut)/float(totalOut)
```

```
    return(output)

def single_attempt(x0, eta, alpha, d):
    """
    Simulates a single "run-through" of the model,
    with x0 people arriving at the model and it
    continuing along until the infection peters out,
    or the simulation has run longer than the
    days remaining.
    Parameters:
        - x0: as above
        - eta: as above
        - alpha: the probability of non-infectious
                 contact
        - d: the number of days remaining in the
             current simulation
    Returns:
        the number of sick people who left the motel
        in this iteration, the number of well people
        who left, as well as how many days are
        remaining in this simulation.
    """

    if d <= 0:
        return(0, 0)
    X = np.random.binomial(x0, 1 - eta)
    Y = x0 - X
    totalInfectious = 0
    while d > 0 and Y > 0:
        totalInfectious += Y
        X, Y = single_generation(X, Y, alpha)
        d -= 1
    if Y == 0:
        return(totalInfectious, X, d)
    elif d == 0:
        return(totalInfectious, 0, d)
```

```
def single_generation(X, Y, alpha):
    """
    Conducts a single generation (read: single day)
    of infections at the motel
    Parameters:
        - X: the number of susceptibles coming into
             the day
        - Y: the number of infectives coming into
                the day
        - alpha: the probability of avoiding
                        infectious contact
    Returns:
        the values for X and Y after the day
    """
    p = alpha ** Y
    n = X
    X = np.random.binomial(n, p)
    Y = n - X
    return (X, Y)

o = []
X0 = [2, 3, 4, 5, 6, 7, 8, 9, 10]
for x0 in X0:
    s = []
    for i in range(10000):
        s.append(ndayattempt(x0, 0.05, 0.1, 0.05, 100))
    o.append(sum(s)/len(s))
Y0 = np.array([0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05])
X0 = np.array(X0)
o = np.array(o)
f, ax1 = plt.subplots(1, 1)
data1 = ax1.plot(X0, o, 'b*', label="Motel")
data2 = ax1.plot(X0, Y0, 'r-', label="Control (no use of motel)")
regress = LinearRegression()
regress.fit(X0.reshape(-1, 1), o.reshape(-1, 1))
R = regress.score(X0.reshape(-1,1), o.reshape(-1, 1))
```

```
print("R^2 value is:", R)
print("M=", regress.coef_)
print("c=", regress.intercept_)
Yp = regress.predict(X0.reshape(-1, 1))
data3 = ax1.plot(X0, Yp, 'b-', label="Motel trendline")
ax1.set_ylabel("Proportion of infectious people leaving the motel")
ax1.set_xlabel("X0 (size of the motel)")
ax1.legend()
plt.show()
```