Lecture Notes for Computational Modeling of Contagion and Epidemics: A companion to "The Rules of Contagion" by A. Kucharski

George Mohler

Introduction

These lecture notes serve as a companion to the text, "The Rules of Contagion" by Adam Kucharski [19]. The novel (which I will refer to as RoC) contains many great examples of contagion processes across various disciplines, from infectious diseases to the spreading of gun violence. The book also provides helpful intuition for understanding mathematical models of contagion. However, many of the details of these models are left out, since the book is intended for a wide audience. The purpose of these lecture notes is to provide more of the mathematical details of some of these models of contagion, to outline computational tools for the simulation of such models, and to give details on how they can be fit to data.

1 Difference equations, differential equations and the SIR Model

Consider the UK home ownership example on p32 of RoC (page numbers may vary based on your version of the book). In that model, each year a fixed fraction (which I will call α) of non-homeowners purchase a home. In each subsequent year, again an α fraction of non-homeowners purchase a home. Letting Y_n denote the fraction of homeowners in year n (with $Y_0 = 0$), we can model such a process with a difference equation:

$$Y_{n+1} = Y_n + \alpha(1 - Y_n). \tag{1}$$

Here the fraction of non-homeowners is $1 - Y_n$, and in year n α of them become homeowners in year $n + 1$ (which gets added to the number of current homeowners, Y_n).

In Figure 1, we plot the evolution of the number of homeowners Y_n over time, or in other words successive iterations of the process. We observe the flattening of the curve that Kucharski discusses, which is not due to contagion (this model is random), but is instead due to the fact that there are less non-homeowners

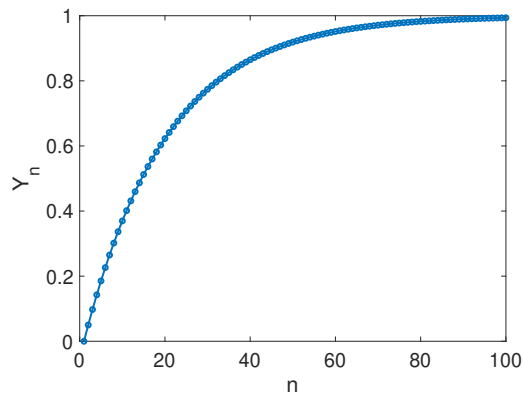

Figure 1: Plot of Y_n vs. n for the homeowner difference equation model with $\alpha = .05$.

that can become homeowners randomly at each subsequent iteration. You may have seen difference equations (also called recurrence equations) before. One famous example is the Fibonacci sequence $F_{n+1} = F_n + F_{n-1}$ where $F_1 = F_2 = 1$ (e.g. 1, 1, 2, 3, 5, 8, 13, ...). While this sequence keeps growing, the ratio F_{n+1}/F_n converges to the "golden ratio" $\frac{1+\sqrt{5}}{2}$.

A related type of model is called a differential equation. Whereas the difference equation models time over discrete steps (for example n above might denote a year), a differential equation models how a quantity changes over time continuously. There are several advantages of this approach, one being that we can easily change the time frame we wish to investigate (for example, what if we want to analyze home ownership per month instead of per year, we don't want to have to create a new model each time we change the timescale). Another advantage is that we can use tools for the theory of differential equations to analyze the models (which may allow us to answer questions such as, "when will the virus peak"). However, one thing to keep in mind is that we will often simulate the solutions to differential equations on a computer by approximating them with difference equations.

1.1 Susceptible-Infected (SI) Model

Kucharski notes that contagion processes tend to follow a S-shaped curve, unlike the curve in Figure 1. The SI model is maybe the simplest model that generates such a curve. Let $S(t)$ denote the number of susceptible individuals at time t and $I(t)$ denote the number of infected (who never recover), out of a population size of $N = S(t) + I(t)$. Here we could think of "infection" through Kucharski's VCR example, where we assume that initially very few people bought a VCR, then word spread and people began to purchase them because their friends had one, and finally once people buy a VCR they have it forever (with any model we could debate whether or not it's realistic). Let us assume that each VCR

owning individual has β contacts per unit time, of which the fraction $S(t)/N$ are susceptible, and therefore each VCR owner infects $dt\beta S(t)/N$ individuals who purchase a VCR in a small unit of time dt. Here we are assuming that the population is "well-mixed," and everyone can randomly come in contact with everyone else. We are also assuming that a contact between an I and S individual definitely leads to an infection.

Because there are $I(t)$ such VCR owning individuals, the total number of new VCR "infections" in a small unit of time is $dt\beta S(t)I(t)/N$. This leads to the following differential equation for the rate of change of infected individuals:

$$\frac{dI}{dt} = \beta \frac{SI}{N}. \tag{2}$$

In this example, there is conservation of the number of individuals, and therefore,

$$\frac{dS}{dt} = -\beta \frac{SI}{N}. \tag{3}$$

A few things to note. First, often $S(t)$ may be shortened to S, even though these mean the same thing. Also, sometimes these differential equations may be written in terms of the population fraction $s = S/N$ and $i = I/N$, yielding

$$\frac{ds}{dt} = -\beta si, \quad \frac{di}{dt} = \beta si.$$

1.2 Finite difference approximation

How do we know these equations yield the S-shaped curve discussed in the book? One way might be to analyze the equations and derivatives of the functions (for example, the sign of the second derivative can tell us about a switch from concave up to down). However, for many models this may be difficult or impossible, in which case we resort to computer simulation to gain insights into the model.

One method for approximating the solution to a differential equation is known as the Euler method. Suppose we are given the differential equation:

$$\frac{d\vec{x}(t)}{dt} = f(\vec{x}(t)). \tag{4}$$

The (forward) Euler method replaces the differential equation with the difference equation:

$$\frac{\vec{x}(t_{n+1}) - \vec{x}(t_n)}{\Delta t} = f(\vec{x}(t_n)), \tag{5}$$

where $\Delta t = t_{n+1} - t_n$. We can then move all the terms at time t_n to the right hand side and we are left with:

$$\vec{x}(t_{n+1}) = \vec{x}(t_n) + \Delta t f(\vec{x}(t_n)), \tag{6}$$

which we can iterate numerically on a computer starting with the initial condition $\vec{x}(t_0) = \vec{x}_0$.

Several things to note. First, the method is called forward Euler because we approximate the derivative at the previous time point and then project the solution forward. This approach can suffer from what's called instability, which shows up as high frequency oscillations that shouldn't be there. In such a scenario one should either reduce the time step Δt until the oscillations go away or switch to backward Euler (that involves solving a nonlinear equation). A second thing to note is that we are using Euler for its simplicity, however there are other methods that are more accurate and have other desired properties. For a good introduction to numerical approximation of differential equations, see Burden and Faires [12].

1.3 SIR Model

With the forward Euler method, we can now simulate solutions of the Kermack and McKendrick Susceptible-Infected-Removed (SIR) model [17]:

$$\frac{dS}{dt} = -\beta \frac{SI}{N}, \tag{7}$$

$$\frac{dI}{dt} = \beta \frac{SI}{N} - \gamma I, \tag{8}$$

$$\frac{dR}{dt} = \gamma I. \tag{9}$$

The SIR model is the same as the SI model, with the exception that infected individuals are removed at a rate γ (i.e. the probability of an individual being removed in a time interval δt is $\gamma \delta t$). In the VCR example above, this could reflect individuals replacing a VCR with a DVD player, or in the case of infectious diseases individuals may recover (or the disease could be fatal). The forward Euler iteration for approximating the solution is given by:

$$S_{n+1} = S_n - \Delta t \beta \frac{S_n I_n}{N}, \tag{10}$$

$$I_{n+1} = I_n + \Delta t (\beta \frac{S_n I_n}{N} - \gamma I_n), \tag{11}$$

$$R_{n+1} = R_n + \Delta t \gamma I_n, \tag{12}$$

which is found by plugging in the finite difference in Equation 5 and solving for the variables at time step $n+1$. One then just needs to specify the initial conditions S_0, I_0, R_0, the parameters β, γ, N and the discrete time step length Δt in order to start the iteration.

Sometimes SIR-type models are called "compartment" models and are represented by a picture as is done in Figure 2. Here each compartment represents each type of individual (susceptible, infected, removed) and arrows indicate movement of the population from one compartment to another at the specified

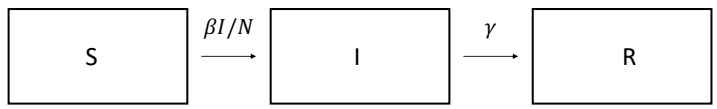

Figure 2: SIR model represented through compartments.

rate. Compartment diagrams can be useful when more complexity is added to the model.

In the mathematical epidemiology literature, models can often have many compartments. Consider a disease like COVID-19 and how SIR may be an over-simplification. First, COVID-19 has an incubation period before a person is able to transmit the disease to others. In this case it is common to add and "exposed" E compartment such that the flow is given by $S \to E \to I \to R$. Such a model is called "SEIR" and was a popular choice for a number of COVID-19 forecasting dashboards, given by the following equations:

$$\frac{dS}{dt} = -\beta \frac{SI}{N}, \tag{13}$$

$$\frac{dE}{dt} = \beta \frac{SI}{N} - \mu E, \tag{14}$$

$$\frac{dI}{dt} = \mu E - \gamma I, \tag{15}$$

$$\frac{dR}{dt} = \gamma I. \tag{16}$$

where E tracks the number of exposed and μ is the incubation rate. One can add more compartments to make the model more realistic, such as compartments for asymptomatic cases, quarantining cases, hospitalizations, and deaths. In the next chapter we will look at probabilistic models of contagion, but later on we will connect differential equation (compartment) models to their probabilistic counter parts.

1.4 Chapter 1 Exercises

1a. Simulate the SI model in Equations 2-3 with forward Euler using parameters $\beta = .005$, $N = 1000$, $S_0 = 999$, $I_0 = 1$ and $\Delta t = 1$ for 3000 iterations. Make a time series plot of S and I vs time and label the S and I curves.

1b. As mentioned in RoC, the SI model satisfies the property that new infections (or adoptions in marketing) grow fastest when 21% of the population have been infected (regardless of the parameters). Verify this is true numerically for the simulation in 1a. In particular, define the variable $P_n = \beta S_n I_n / N$ representing

new infections. This will be growing fastest when the first derivative has a maximum, where the derivative can be approximated by $dP_n = (P_{n+1} - P_{n-1})/(2\Delta t)$. So in other words, make a plot of dP_n and find the time when dP_n is largest. At that time check that $I_n/N \approx .21$.

1c. Prove that 1b is true, in particular that $I/N = (3-\sqrt{3})/6$, when the second derivative of dI/dt is zero. (Hint: try differentiating the SI equations and then plugging them back in to simplify).

2. Simulate the SIR model with $N = 75000$, $I_0 = .5$, $S_0 = N - I_0$, $R_0 = 0$, $\gamma = 5.6$, $\beta = N * 8 * 10^{-5}$, $\Delta t = 1/7$ (in weeks) and run the simulation for 50 weeks (parameters recommended from [1]). Next download the 1906 Bombay plague data from canvas (data can also be found in the R package "primer"). On the same graph, plot new deaths per week from the data alongside $\gamma I(t)$ to show the fit is reasonable. Note, however, that even though the model fits the data well, the parameters may not be realistic (for example the population was larger in 1906). We will see later how parameters can be estimated to fit a model to data.

3. Consider the SIR model and suppose now there are 2 types of infected individuals, those that are symptomatic and those that are asymptomatic. Let's assume that both are removed at the same rate, however the asymptomatic group have a different (potentially larger) contact rate with susceptible individuals. How could you modify the SIR equations to capture such a scenario?

2 Reproduction numbers and random networks

A critical parameter to understanding the spread of a contagion process is the "reproduction number" R. In the case of a single infected individual in a susceptible population, the initial reproduction number is called R_0, and represents the average number of people that first person will directly infect. For contagion processes, if $R_0 > 1$, then the epidemic will tend to grow (exponentially), if $R_0 < 1$ then the epidemic will tend to subside after a few generations.

Kucharski states four factors that can affect R: duration (how long a person is contagious), opportunities, transmission probability, and susceptibility. Different interventions and disease characteristics can impact these factors. For example, opportunities involve contacts between 2 or more individuals. When someone quarantines this reduces opportunities for transmission. On the other hand, wearing a N95 mask doesn't reduce contacts, however it can lower the transmission probability of a disease like COVID-19. When people are vaccinated, this may reduce the number of susceptible individuals.

In this chapter we will study simple branching processes, which are probabilistic versions of the SIR model that focus on individuals and random transmission related to the reproduction number, rather than tracking the deterministic population fraction of infected individuals, as is done with SIR. We will also

look at simple random graph models for social contacts on which a contagion processes might spread. First we will review some basic probability distributions.

2.1 Review of probability

2.1.1 Binomial random variable

Suppose we flip a fair coin. Then the outcome will sometimes come up heads, and sometimes tails. If let X be 1 when the coin is heads and 0 when it lands tails, then X can be modeled by a Bernoulli random variable with parameter $p = 1/2$. Here we assume that the flips are iid, e.g. independent (they don't effect each other) and identically distributed (p doesn't change). In other situations p may not be 1/2, but the random variable we are interested in is still Bernoulli. For example if we buy a raffle ticket out of 1000 tickets, then $p = 1/1000$ (the probability that we win).

If we are interested in $Y = \sum_{i=1}^{N} X_i$, the sum of successes over N Bernoulli experiments, then that random variable is called Binomial. For example, if we are going to play roulette 10 times and bet on 17 each time, then $N = 10$ and $p = 1/38$ and Y is the number of times the wheel lands on 17 over the 10 spins.

Recall that the expected value of a random variable, is determined by

$$E[X] = \sum_x xp(x) \tag{17}$$

where the sum is taken over all possible outcomes x for X. In the case of the Bernoulli random variable, $E[X] = 1 \cdot p + 0 \cdot (1 - p) = p$. Given the expected value $\mu = E[X]$ or a random variable, the variance of a random variable is defined as:

$$\sigma^2 = Var[X] = E[(X - \mu)^2] \tag{18}$$

and measures how much the random variable tends to fluctuate about its mean. With a little algebra, one can show that for the Binomial random variable $p(x) = \binom{N}{x} p^x (1-p)^{N-x}$, $\mu = Np$ and $\sigma^2 = Np(1-p)$.

When we have data samples (observations) X_i, we often estimate the mean and variance. In that case we calculate the sample mean and sample variance:

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} X_i \tag{19}$$

$$\hat{\sigma^2} = \frac{1}{N-1} \sum_{i=1}^{N} (X_i - \hat{\mu})^2. \tag{20}$$

2.1.2 Poisson random variable

Imagine that a person is infected and comes in contact with 100 people in a day. Let us assume that for each contact the probability of transmission is $p = .05$

and that contacts are iid. This could maybe apply to situations like a grocery checkout, where each contact is similar in characteristics and duration. Let us further assume that the person is only infectious for that one day. Then in this simplified scenario, the number of secondary infections caused by the person is a Binomial random variable with $N = 100$ and $p = .05$. On average we would expect $\mu = Np = 100 \cdot .05 = 5$ secondary infections, so we would say the reproduction number is $R = 5$.

For a Binomial random variable when N is large, p is small, and $\lambda = Np$ is moderate, we often switch to using a Poisson random variable. The Poisson random variable takes on values that are non-negative integers with probability $p(x) = \lambda^x e^{-\lambda}/x!$. The reason we might use a Poisson random variable, rather than Binomial, is that we might not know precisely what N or p are, but we can observe the number of infections. The Poisson random variable can model a variety of phenomena, such as the number of soccer goals in a world cup match, the number of plane crashes in a year, etc. The defining characteristic of a process that is Poisson is that there are many opportunities (N) for something to happen, and a small independent probability (p) each time that they do occur. In the case of soccer, in each minute there is a small probability of a goal occurring and there are $N = 90$ minutes in a game. With some algebra you can show that for a Poisson random variable, $\mu = \sigma^2 = \lambda$.

2.1.3 Negative binomial random variable

Let's consider a random variable, X_i, representing the number of COVID-19 infections caused directly by person i in the population in Boston during 2020. One way to check whether this random variable is Poisson would be to computer the sample mean and variance, $\hat{\mu}$ and $\hat{\sigma}^2$, and see if they are about equal. However, my guess is that this would not be the case, and $\hat{\sigma}^2$ would likely be larger.

One reason for this is due to "super-spreaders," which Kucharski discusses in RoC. A 57 year old professor teaching remotely on zoom with no children in the house will not have the same risk of spreading COVID-19 as a 10 year old child that has in-person school, sports and play dates, some without a mask or distancing. In this situation, we may switch from a Poisson random variable to a negative Binomial random variable, where

$$p(x) = \binom{x + r - 1}{r - 1}(1 - p)^x p^r$$

and $\mu = pr/(1 - p)$, $\sigma^2 = pr/(1 - p)^2$. When the parameter r is large then the Negative Binomial approaches Poisson, but for small r then the Negative Binomial will be "over-dispersed", meaning the variance is larger than the mean.

For a further reference providing an introduction to probability, see for example [27].

2.2 Branching processes

The SI and SIR models in Chapter 1 are called "mean-field" models, because random fluctuations are averaged out and the remaining models are deterministic. Such models are appropriate when the number of infected and susceptible are large and the population is well mixed (everyone can come in contact with everyone else). However, initially after "patient-zero" is infected, what happens to the outbreak? Does it grow into a pandemic? Is it contained in one region and prevented from spreading? Does it fail to spread and die out on its own?

In the early stages of contagion, or in later stages near-elimination, it is useful to switch to stochastic (random) models that can help us quantify the probability of the above events. A branching process is a model in which infected individuals in one generation of disease transmission probabilistically infect more individuals in the next generation. Branching processes are also used to model population growth, where reproduction number could instead represent the average number of children a parent has.

In Figure 3, a tree depiction of 3 generations of a branching process is shown. In generation 0, one individual (patient 0) is infected. That individual infects 2 individuals, who make up generation 1. The first individual in generation 1 infects 3 others, whereas the second individual infects none (maybe they quarantined). The process can continue on indefinitely into generation 3, 4, 5..., although depending on the reproduction number it may end when one generation has no "children" (to borrow the terminology for population branching processes).

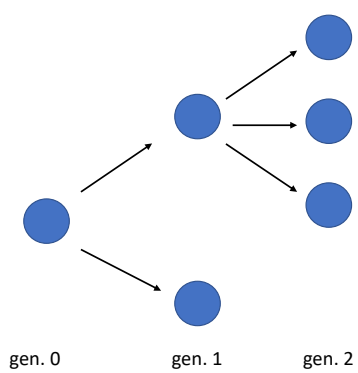

Figure 3: First 3 generations of a branching process.

In Algorithm 1 pseudo code for a branching process is shown. Here we have initialized the number of individuals in generation 0 as $N_0 = 1$, the reproduction number as $R_0 = 2$, and the number of generations to simulate, $G = 10$. The algorithm is a double for loop, where we loop over each generation g, and within each generation we loop over each individual and sample its number of direct

infections in the next generation $(g+1)$ as a Poisson random variable, $y \sim Pois(R_0)$, which we add to the running total of infections in the next generation, N_{g+1}. If at any point in the for loop over generations, the current generation has no infections, we then terminate the algorithm.

Algorithm 1 Branching process pseudo code

```
N_0 ← 1
R_0 ← 2
G ← 10
for g=0:G-1 do

    if N_g = 0 then Break
    end if

    N_{g+1} ← 0
    for i=1:N_g do
        y ~ Pois(R_0)
        N_{g+1} = N_{g+1} + y
    end for

end for
```

Note that the branching process in Algorithm 1 does not capture susceptibility as the SIR model does, nor is the population finite where at some point herd immunity could be reached. Instead, one of 2 possible scenarios will occur. If the reproduction number $R_0 < 1$, which is called "sub-critical", the branching process will die out at some generation. However, if the process is super-critical $(R_0 > 1)$, then the process will explode on average and continue to grow exponentially (if it makes it past the first few generations). We will see later how to extend the branching process in Algorithm 1 to include more realistic contagion characteristics, such as the time between infections and finite population effects.

2.3 Random networks

In the branching process in Algorithm 1, contacts between individuals are implicitly incorporated into the reproduction number parameter, but not explicitly part of the model. However, as Kucharski outlines in Chapter 2 of RoC, the network underlying social contacts can play an important role in how a contagion spreads. In this section we will introduce some terminology on networks (or "graphs") and introduce two random graph models.

In Figure 4, the same network is shown in three ways. On the left, the network is shown as a picture of four nodes connected by edges. Node 1 is connected to 2, 2 is connected to all the other nodes, and additionally node 3 is connected to 4. When dealing with networks on a computer, there are two useful data structures to represent a network. In the middle of Figure 4, an "adjacency matrix" A is shown for representing the same network. In that

matrix $a_{ij} = a_{ji} = 1$ if i is connected to j, and is 0 otherwise. Here we have assumed the network is "un-directed", meaning I am friends with you if you are friends with me. Instead, we could have represented a directed network, where the matrix A doesn't have to be symmetric (which may be more realistic for a network like Twitter where you follow someone who doesn't follow you). Finally, the network can instead be represented as an edge list, as shown on the right in Figure 4. The edge list can be a more efficient way to represent the network when the number of nodes is large, but the number of edges is small compared to the size of the adjacency matrix.

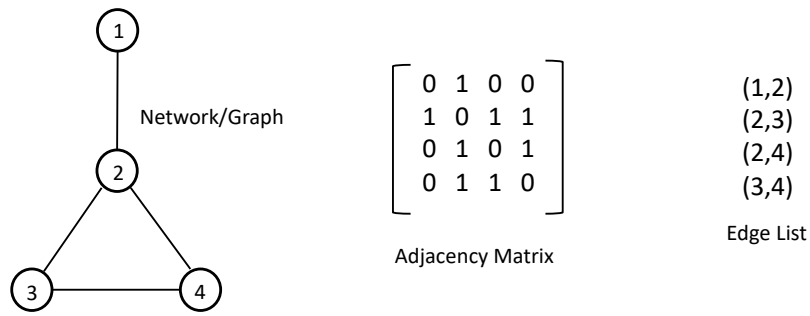

Figure 4: Three ways to represent the same undirected network.

One important quantity for the spread of contagion is the "degree" of the node, which is the number of nodes connected to a given node. For example, node 2 in Figure 4 has a degree of 3. A higher degree means more contacts, which could mean that if that node is infected then it will tend to spread a contagion to more nodes. One way to characterize a network is through its empirical "degree distribution":

$$p(k) = \frac{1}{N} \sum_{i=1}^{N} 1\{d_i = k\}, \tag{21}$$

where N is the number of nodes in the network and d_i is the degree of node i. So here $p(k)$ is the fraction of nodes that have degree k.

An interesting property of networks is that even if 2 have the same number of nodes, edges, and degree distributions, contagion processes can still behave differently between them. Kucharski discusses one such characteristic that can lead to different network behavior, namely "assortativity". First, the correlation between two random variables X and Y is given by,

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}, \tag{22}$$

and is a value that ranges from -1 to 1. When ρ is close to 1 then the random variables are highly positively correlated (they go up and down together), when

ρ is close to -1 they are negatively correlated and thus when one goes up the other goes down, and they are uncorrelated if $\rho = 0$. Back to degree assortativity, if we consider a randomly selected edge from a network, the degree assortativity coefficient is the correlation of the degrees of the 2 nodes connected by that randomly chosen edge (see [22] for details). When degree assortativity r is close to 1, we say the network is assortative and high degree nodes tend to be connected to other high degree nodes. For disassortative networks, r is closer to -1 and high degree nodes tend to be connected to low degree nodes.

2.3.1 Erdős-Rényi graphs

In the branching process Algorithm 1, we used a Poisson random variable to model the reproduction number. Recall that we could have used a Negative Binomial model instead, which would have greater variance and capture super-spreaders. So one question is how do different networks lead to different reproduction number distributions?

One type of random graph is called an Erdős-Rényi (ER) graph [11]. An ER graph is defined by two parameters which are the number of nodes N and a parameter p giving the probability that two nodes are connected. For each possible edge, the edge is sampled independently from a Bernoulli random variable with probability p. Thus, the number of edges follows a Binomial distribution with parameters N and p. Furthermore, the degree of a node follows a Binomial distribution with parameters $N - 1$ and p. Thus, if N is large and p is small, the degree distribution of an ER graph is approximately Poisson with parameter $\lambda = Np$. Therefore ER graphs will not tend to produce super-spreading, at least if transmission is assumed to be iid.

2.3.2 Barabási-Albert preferential attachment graph

The degree distribution of many real-world networks often follows what is called a "power law" distribution, e.g. $p(k) \sim 1/k^p$. These networks, such as the internet, are characterized by a small fraction of nodes having very high degree and a large fraction of nodes having low degree. The nodes with high degree can produce super-spreading in contagion processes that wouldn't be observed in an ER network.

One example of a random network with a power law degree is a Barabási-Albert (BA) preferential attachment network. The algorithm for generating a BA network is as follows [2]. Start with a network with m_0 nodes. Then add new nodes, one at a time, that are "preferentially attached" to existing nodes with higher degree. In particular, attach each new node to $m \leq m_0$ existing nodes, where the probability that the new node attaches to existing node i is given by $d_i / \sum_j d_j$ (here d_i is the degree of node i and the sum in the denominator is over all existing nodes).

2.4 SIR model on a network

Let's now consider a network, stochastic analog of the SIR model in Chapter 1. Again we will want to allow individuals to move from susceptible to infected to recovered/removed. However, now the population is no longer well-mixed (e.g. complete graph), but instead has a heterogeneous network structure. If a node is susceptible, but not connected to any infected nodes, then it shouldn't be infected in the near future. However, if a node has many connections to infected nodes, it's probability of infection should go up. In particular, let's assume that the probability of infection of susceptible node j over a time interval δt, who has n_j infected neighbors, is given by $p_j^I = 1 - e^{-\delta t n_j \beta}$ (we use this instead of $\delta t n_j \beta$ to make sure it's a probability between 0 and 1). As the time step δt, number of infected neighbors n_j, or transmission rate β increases, then the probability of infection increases. Finally, we assume that recovery is independent of links and occurs independently at some rate γ, e.g. the probability of recovery of infected node j is given by $p_j^R = 1 - e^{-\delta t \gamma}$.

We have provided pseudo code for the network SIR model in Algorithm 2. While one could imagine several possible data structures to use, we have chosen to use the adjacency matrix A along with time dependent vectors $\vec{S}^t$, $\vec{I}^t$, and $\vec{R}^t$. Here $S_i^t = 1$ means node i is susceptible at time t ($S_i^t = 0$ means it is either infected, $I_i^t = 1$, or removed, $R_i^t = 1$).

2.5 Latent homophily network

We will end this section with one more dynamic network model that appears to exhibit contagion, but actually is an example where homophily can be confused with contagion. Homophily in a social network is characterized by individuals with similar traits (for example demographics) being more likely to have edges between them in the network. There's an old saying that "birds of a feather flock together", which is essentially describing an example of homophily.

The network model, introduced by Shalizi and Thomas [29], goes as follows. First start with N nodes where each node i has a trait given by a uniform random variable, $x_i \sim U([0,1])$ (for example maybe this represents age in units of 100 years). Next an undirected network is created connecting the N nodes where an edge is assigned between nodes i and j with probability,

$$p_{ij} = \sigma(-3|x_i - x_j|). \tag{23}$$

Here $\sigma(x) = e^x/(1+e^x)$ is the logistic function, which is an increasing function from 0 to 1. Thus when i and j have similar traits, the difference $|x_i - x_j|$ is smaller and p_{ij} is larger. Next, from this undirected network, a directed friendship network is created where each node i chooses one node j among its connections to be a friend with probability proportional to $\sigma(-|x_j - .5|)$. If node j is chosen, then we let $A_{ij} = 1$ in the adjacency matrix (however j may not necessarily choose i, so A_{ji} could be zero). In Figure 5 we plot a simulation of the resulting network with latent homophily.

Algorithm 2 Network SIR pseudo code

Input: $N \times N$ adj. matrix A, parameters β, γ, δt, T, initial infected node j

```
S^0 ← 1
I^0 ← 0
R^0 ← 0
S_j^0 = 0
I_j^0 = 1
for t=1:(T/δt) do
    S^t = S^{t-1}
    I^t = I^{t-1}
    R^t = R^{t-1}
    for i=1:N do
        if S_i^{t-1} = 1 then
            n_i = Σ_j a_ij I_j^{t-1}
            if rand()< 1 − e^{−δt n_i β} then
                I_i^t = 1
                S_i^t = 0
            end if
        end if
        if I_i^{t-1} = 1 then
            if rand()< 1 − e^{−δt γ} then
                R_i^t = 1
                I_i^t = 0
            end if
        end if
    end for

end for
```

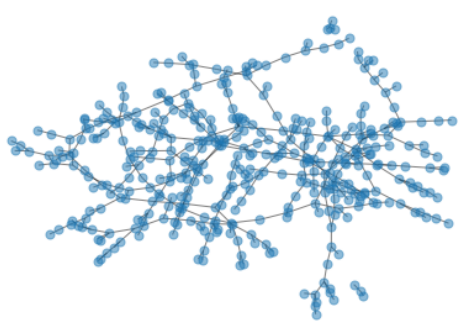

Figure 5: Latent homophily network of Shalizi and Thomas [29].

Now with the network defined by the adjacency matrix A, dynamics are added as follows (and actually the dynamics won't depend on A at all). We let $y_i(t)$ be an observable outcome at node i and time t, for example maybe a measure of smoking of individual i on day t. The dynamics proceed in 3 rounds:

- At time $t = 0$, $y_i(0) = (x_i - .5)^3 + \epsilon_0^i$
- At time $t = 1$, $y_i(1) = y_i(0) + .4x_i + \epsilon_1^i$
- At time $t = 2$, $y_i(2) = y_i(1) + .4x_i + \epsilon_2^i$

where ϵ_t^i are independent normal (Gaussian) random variables with mean 0 and standard deviation .02. The model is constructed such that the trend in y is larger when x is larger. In Figure 6, a "causal diagram" for the model is shown, where arrows indicate dependency between the random variables. We will see in Section 3 that when we attempt to estimate the causal dependencies from data, we often will detect arrows from $y_j(t-1)$ to $y_i(t)$, in other words contagion from j to i, even though such a direct dependency isn't present in the actual model.

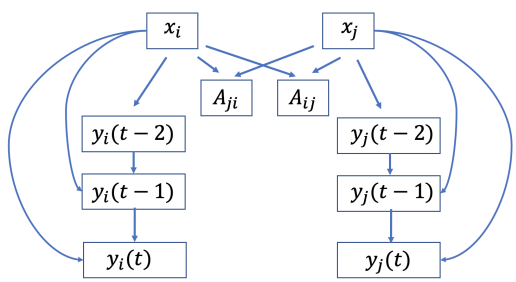

Figure 6: Causal diagram for Shalizi and Thomas homophily network model (reproduced from [29])

.

2.6 Chapter 2 Exercises

1a. Simulate 500 samples from a Binomial random variable with parameters $N = 100$ and $p = .1$. Create a histogram of the 500 samples.

1b. Next 500 samples from a Poisson random variable with parameter $\lambda = 10$. What is the sample mean and variance? Create a histogram of the 500 samples and compare to the histogram in 1a.

2a. Implement code for the branching process in Algorithm 1 with $N_0 = 1$ and $R_0 = .95$. Simulate the process 1000 times and make a plot of N_g vs g for the 1000 simulations. What is the average epidemic size (total infections) across the 1000 simulations?

2b. Next estimate the average epidemic size for the model in 2a using the geometric series $\sum_{i=0}^{\infty} R_0^i = 1/(1 - R_0)$. How does your estimate compare to the average you found in 2a?

3. Implement the Barabási-Albert preferential attachment network generation algorithm described in Section 2.3.2 from scratch (rather than using networkx). Then simulate a network with $N = 100$ nodes with 1 initial node and $m = 1$. Visualize the network (you can use draw_spring in networkx) and also plot the degree distribution (again you can use networkx functions for this).

4. In Chapter 2, Kucharski discusses assortative vs disassortative networks, and how contagion on the former spreads quickly but leads to smaller overall epidemics, whereas on the latter contagion spreads more slowly but creates larger epidemics. In this problem we will explore contagion on networks vs. assortativity using Algorithm 2.

a) First implement Algorithm 2 in Python.

b) Next use the networkx function "read_adjlist" from the networkx package to read in the files "ba_graph.adjlist" and "ws_graph.adjlist". Make a plot of the 2 networks using "networkx.draw_spring" and compute the assortativity of each using the networkx function "degree_assortativity_coefficient".

c) Run Algorithm 2 on the 2 networks using parameters $\delta t = .1$, $\beta = \gamma = 1$, and final time $T = 20$ and one randomly chosen node infected to start. You can extract the adjacency matrices from the networks using the functions "adjacency_matrix" and "todense". Try running the Algorithm several times with different random nodes chosen as infected initially. Make plots of the total infected vs. time for the simulations and interpret the results. Do you find that the disassortative network exhibits slower spreading, but a larger epidemic?

3 Estimating contagion models from data

Chapter 3 of RoC deals with social contagion and distinguishing contagion effects from those due to homophily and shared environmental exposures. In order to tackle these concepts we will first need to review regression and how to estimate models from data. We will also take the time to discuss how to estimate parameters of some of the models we saw in Chapter 1 and 2 and how we might quantify uncertainty in those models.

3.1 The likelihood function

Consider the Bernoulli random variable from Chapter 2 and the experiment of Boston College playing ice hockey against Boston University 5 times in a

row (let 1 indicate BC wins and 0 indicate they lose). Suppose we observe the sequence 1,1,0,0,1 and we want to estimate the probability p that BC will win if they play a sixth game (assuming games are iid). One way to estimate p (or the parameters of any model) is by maximizing the likelihood function.

The likelihood function is the probability of observing the data given the model parameters, $P(Data|Parameters)$. In the above example, this would be $P(1,1,0,0,1|p)$, and note that this depends on which value we plug in for p. If we plug in $p = .001$, then the probability of observing 3 wins is quite small (hence $p = .001$ is not very "likely"). For the case of independent Bernoulli trials, the likelihood simplifies into a product:

$$P(1,1,0,0,1|p) = p^1 p^1 (1-p)^0 (1-p)^0 p^1 = p^3(1-p)^2. \tag{24}$$

Then to estimate p, we use Maximum Likelihood Estimation (MLE), and recall from calculus we can maximize a function by setting its derivative to zero. In practice we maximize the log of the likelihood (called the log likelihood) because its easier to take a derivative of a sum than a product:

$$p_{mle} = \arg\max_p [\log(p^3(1-p)^2)] = \arg\max_p [3\log(p) + 2\log(1-p)]. \tag{25}$$

Differentiating, setting $3/p - 2/(1-p) = 0$, and solving for p we find that $p_{mle} = 3/5$ (which is a nice result, since 3/5 is what we would have guessed without using any probability theory).

3.2 Linear regression

As a second example, consider linear regression where we model a response variable y as a linear function of explanatory variables (sometimes called features) $x_1, ..., x_m$, plus some Gaussian noise ϵ:

$$y = \theta_1 x_1 + \theta_2 x_2 + ... + \theta_m x_m + \epsilon. \tag{26}$$

Here θ_i are coefficients of the features that are parameters that need to be estimated from data. For example, maybe we want to model the reproduction number of COVID-19 (y) in a zipcode as a function of population density (x_1) and fraction of the population vaccinated (x_2). Sometimes it is convenient to use vector notation, where we let $\vec{\theta}$ be a vector of coefficients and $\vec{x}$ be a vector of features, and thus $y = \vec{\theta}^t \vec{x} + \epsilon$.

Since we have assumed the noise is Gaussian, given data $(\vec{x}^j, y^j)_{j=1}^N$ then the likelihood is given by:

$$L = \prod_{j=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y^j - \vec{\theta}^t \vec{x}^j)^2/(2\sigma^2)}. \tag{27}$$

Taking the log of the likelihood we get:

$$logL = N\log(\frac{1}{\sqrt{2\pi\sigma^2}}) - \sum_{j=1}^{N} (y^j - \vec{\theta}^t \vec{x}^j)^2/(2\sigma^2). \tag{28}$$

Maximizing this function is equivalent to minimizing the square error (we can ignore the first term in the log likelihood because it doesn't depend on the parameters θ),

$$g(\vec{\theta}) = \sum_{j=1}^{N} (y^j - \vec{\theta}^t \vec{x}^j)^2, \tag{29}$$

which is why linear regression is sometimes called least-squares regression.

3.2.1 Linear regression applied to latent homophily network dynamics

Recall the network in Chapter 2 with latent homophily, where each node i has a trait x_i and nodes are more likely to be connected if they share a similar trait. In that model, the entries in the adjacency matrix satisfy $A_{ij} = 1$ when i declares j to be a friend. Then network dynamics were defined by an outcome variable $y_i(t)$ which evolved over 3 time steps $t = 0, 1, 2$. The dynamics at a node i only depended on the feature x_i and past values of the outcome variable at the node. However, Shalizi and Thomas showed that false contagion can be detected in such a model.

In particular, they define a regression (similar to the one used in Chistakis and Fowler [8]) as:

$$\begin{aligned} y_i(2) = \theta_0 + \theta_1 y_i(1) + \theta_2 \sum_j A_{ij} y_j(1) + \theta_3 \sum_j A_{ji} y_j(1) \\ + \theta_4 \sum_j A_{ij} y_j(0) + \theta_5 \sum_j A_{ji} y_j(0) + \epsilon_i. \end{aligned} \tag{30}$$

Here the first coefficient θ_0 is the intercept of the regression and θ_1 is a coefficient of the autocorrelation of the time series at node i. Then the next term captures the contagion effect of node j, who i declared to be their friend. If the coefficient θ_2 is positive and statistically significant, then a higher value of the outcome at j at time $t-1$ leads to a higher outcome at i at time t. They also include a potential effect from the "influenced to the influencer", the idea being that the coefficient (θ_3) should be smaller given the directionality of the friendship network (people who you consider a friend may influence you more than you influence them, especially if they do not consider you a friend). The next two terms are the contagion effects lagged one additional time step.

We have implemented the latent homophily network and the regression model in this Google colab notebook [1]. Notice that for the random seed chosen (simply the first possible), the coefficient θ_2 is estimated to be positive and statistically significant. Shalizi and Thomas ran thousands of such simulations and found that θ_2 is positive in a majority of runs [29]. It should be remarked that this toy model is not the same as the model estimated in [8]. However, the point is that one needs to be careful when attempting to disentangle contagion from homophily (or shared environmental influences).

[1] `https://colab.research.google.com/drive/1pXTMstAFgOKY74kIuEvlYUSFuTlZlw1r?usp=sharing`

3.3 Optimization techniques for maximum likelihood estimation

In Equation 29 our goal is to minimize the square error $g(\vec{\theta})$ and in general maximum likelihood estimation involves solving such an optimization problem. Recall from calculus that a function has a maximum or minimum when the gradient (vector of partial derivatives) is zero:

$$\nabla_\theta g(\vec{\theta}) = \vec{0}. \tag{31}$$

In the case of Equation 29, this results in a linear system that can be solved analytically. However, in many cases an analytical solution cannot be found. One popular approach to MLE is gradient descent, where we start with an initial guess for the model paramters $\vec{\theta}^0$ and then iterate:

$$\vec{\theta}^{k+1} = \vec{\theta}^k - \gamma \nabla_\theta g(\vec{\theta}^k). \tag{32}$$

Here γ is called the learning rate (which is similar to the timestep in forward Euler) and when this iteration converges notice that the gradient will be zero. The sign in front of the learning rate will depend on whether you are maximizing (+) or minimizing (-) g.

For some likelihoods there may be more than one local max, and in that case one might consider using a number of random initial guesses for the parameters and then picking the parameters that lead to the largest value of the likelihood. Also, sometimes the iteration can be sped up using second order derivatives, for example using Newton's Method. For more details on numerical optimization see [32]. For linear regression and other common likelihoods, the python packages "scikit-learn" and "statsmodels" have optimization functions implemented and many good examples on how to use them. In cases where you encounter a new model and likelihood, then you can use auto-differentiation software such as pytorch or tensorflow to solve the associated optimization problem in Equation 32.

3.4 Bayesian methods for parameter estimation

Suppose we were building a COVID-19 forecast for Boston during the initial surge in March 2020, when we wished to estimate the number of cases, hospitalizations, ventilators, etc that were needed. In that scenario, its important not to just estimate a single number of hospital beds, but instead to have best and worst case scenario estimates where we quantify the uncertainty in our model and our forecast. Bayesian estimation provides a set of methods for simultaneously optimizing the likelihood to find good parameters, while also providing estimates of the uncertainty of the model and its parameters.

Bayesian estimation comes from Bayes' Theorem, which states:

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}. \tag{33}$$

If we think of θ as our model parameters and X as the data, then notice that the first term in the numerator on the right handside is the likelihood. The term on the left handside is called the "posterior" and is the probability density of parameters θ given observing the data X. When you think about it, this is exactly what we want: we can use the mean of the posterior if we want a single "best" set of parameters, or we can compute the variance and 95% quantiles of the posterior to quantify uncertainty.

There are then two other terms on the right: $P(\theta)$ called the prior distribution for the parameters and a normalizing constant $P(X)$ that makes sure the posterior is a true probability density that integrates to 1. Here one must specify the prior $P(\theta)$, which is usually done by ether using domain knowledge (maybe we know the average recovery time is 10±3 days) or by using a density that is "uninformative", e.g. a prior that is very flat, allowing for a large range of possible parameters.

Unfortunately the constant $P(X)$ cannot be easily computed in practice. This is because $P(X) = \int P(X|\theta)P(\theta)d\theta$, and when the number of parameters is larger than a few this becomes a high dimensional integral that has exponential computational complexity to approximate. So what is done instead in practice is to use Monte Carlo methods to avoid this integral. One famous method for sampling from the posterior is called the Metropolis-Hastings algorithm shown in Algorithm 3.

Algorithm 3 Metropolis Hastings

```
Initialize θ^0
for K=0:N do
    θ* = θ^k + ε
    r = P(θ*|X) / P(θ^k|X)
    θ^{k+1} = θ^k
    if r>rand() then
        θ^{k+1} = θ*
    end if
end for
```

In the MH algorithm, we construct a sequence of parameters θ^k by adding noise (for example Gaussian noise ϵ) to generate a proposal set of new parameters θ^*. We then keep that new proposal as θ^{k+1} with probability equal to the ratio of the posteriors of the proposal and the previous parameters at iteration k, e.g. the ratio $r = P(\theta^*|X)/P(\theta^k|X)$. The neat thing about this ratio is that $P(X)$ on the top and bottom cancel out! So we avoid needing to compute the integral. Because the ratio r will tend to be larger if the likelihood of the new parameters is larger, the sequence θ^k will randomly make its way to larger values of the likelihood. This sequence of parameters is called a Markov Chain, and once it converges one can show that the sequence of parameters θ^k samples from the posterior $P(\theta|X)$.

Let's see how this works with an example. Again we will use the BC vs BU

hockey data $x = [1, 1, 0, 0, 1]$ where we want to estimate the probability p of BC winning the next game. We can use the package "pystan" which is a wrapper for the Bayesian programming language STAN [4]. In STAN we need to specify the data x, along with how many samples there are ($N = 5$). We also need to speficy the parameters that we want to estimate, in this case a single parameter p that is between 0 and 1 (because it's a probability). Finally, we specify the model. For the prior I used a beta distribution, which is typically used for probabilities because it's output is between 0 and 1. The code[2] is contained in Figure 7 and the output of the code is in Figure 8. In Figure 8, we see a plot of p^k vs MH iterations k, which is sampling from the posterior. We can compute the mean and 95% range of these values, which we see are 0.57 and $[0.23, 0.87]$ respectively. Note the mean is similar to the 0.6 value from MLE, however we also have an estimate of uncertainty. Because there were only 5 games, the Bayesian 95% "credible interval" is between 0.23 and 0.87. Thus based on the data we really can't say which team is better (we would need to collect more data).

It is worth noting that for higher dimensional problems with more parameters, Algorithm 3 can be quite slow. In this situation it is better to use information on derivatives of the likelihood to speed up convergence of the Markov Chain. Luckily STAN can do this for us. In practice one should use more samples than in the above example, perhaps 10,000, and also throw out samples early on before the Markov Chain converges (this period is called the burn-in). Note you can also repeatedly run STAN for several chains ("num_chains" in the code), to remove the dependence on the initial parameter guess (the parameters can potentially get stuck in a local maximum of the likelihood, as can happen with MLE). For further reference on Bayesian methods, see [14] which has a free online pdf version. For our class we will make use of STAN as a black box that can provide us parameter estimates and their uncertainty.

3.5 Using STAN to fit a SIR model

In the next example we use STAN to fit a SIR model to simulated data. We simulate the model in Equations 10-12 with parameters $\gamma = 1/8$, $\beta = .4$, $N = 100000$, $I_0 = 1$ and $\Delta t = 1$ day. We simulate the model for 30 days and then use the posterior parameters from STAN to forecast out an additional 30 days. We assume that the observed data for new daily infections is a Poisson random variable given by:

$$newInf \sim poisson(\beta SI/N), \tag{34}$$

which could alternatively can be specified as Gaussian or negative binomial. For the priors on the recovery rate, transmission rate, and initial infected we use

[2] `https://colab.research.google.com/drive/1ajm5DoPWAvsaKo0i1xyIgtaTG9wsxJcq?usp=sharing`

```
import nest_asyncio
nest_asyncio.apply()
import stan
import matplotlib.pyplot as plt
import numpy as np

model_code = """
data {
  int N;
  int x[N];
}
parameters {
  real<lower=0,upper=1> p;
}
model {
  p ~ beta(1, 1);
  x ~ bernoulli(p);
}
"""

hockey_data = {"N": 5,"x": [1,1,0,0,1]}

posterior = stan.build(model_code, data=hockey_data, random_seed=1)

fit = posterior.sample(num_chains=1, num_samples=1000)

plt.plot(fit["p"][0])
print(np.mean(fit["p"][0]))
print(np.percentile(fit["p"][0], [2.5, 97.5]))
```

Figure 7: Pystan code for Bernoulli hockey example.

```
mean of Bernoulli probability:  0.5687023654055571
95% Bayesian credible interval:  [0.22591212 0.87481003]
```

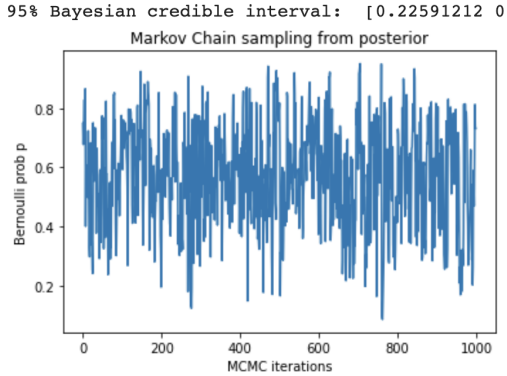

Figure 8: Output of pystan code for Bernoulli hockey example.

the following specifications recommended in [9]:

$$\gamma \sim lognormal(\log(1./8.), .2) \tag{35}$$
$$\beta \sim lognormal(\log(.4), .5) \tag{36}$$
$$I_0 \sim cauchy(0, 100). \tag{37}$$

The first two are informative priors based on domain knowledge of COVID-19 (recovery rate with a mean of 8 days, transmission rate with a mean of 0.4). The lognormal distribution is used to keep the parameters non-negative, since recovery and transmission rates must be positive. The cauchy distribution is an uninformative prior with large variance, indicating we do not have a good guess for the initial number of infected at day 0.

We plot the forecast in Figure 9, where we not only estimate the expected number of daily new infections, but we also provide a 99% credible interval for the number of new infections. A Google colab notebook with the code for the simulation and model fitting can be found here[3].

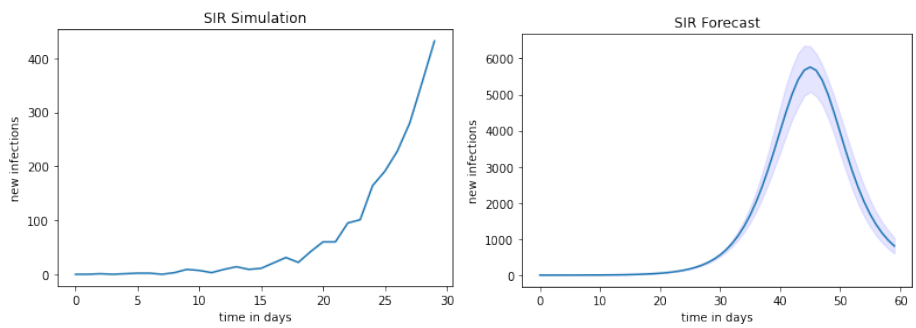

Figure 9: STAN forecast using data from a simulated SIR model.

3.6 Chapter 3 Exercises

1. Simulate 15 samples from a Poisson random variable with parameter $R_0 = 2.5$. Next implement STAN code to estimate the Poisson parameter R_0 from the simulated data. Use an uninformative prior for R_0 that is cauchy(0,100). Use 10000 MCMC iterations and find the posterior mean and 95% Bayesian credible interval for the estimated R_0. How do these agree with the true value of $R_0 = 2.5$?

2a. Modify the STAN code from Section 3.5 to fit a SEIR model instead of SIR. Note you will need to add a new exposed compartment variable, E, along with a new parameter μ, the incubation rate.

[3] `https://colab.research.google.com/drive/1tfx73t2Q_B_DxBXGxJwcEJ-Aauevk8Pc?usp=sharing`

2b. Next choose a prior for the incubation rate μ. To do this, do some research online for what a reasonable range for the COVID-19 incubation period is (note μ is the inverse of the average incubation period).

2c. Next use the STAN code to fit the SEIR model to Boston COVID-19 data from the spring of 2020 provided in "boston_covid.csv". Fit the model to the first 14 days of the data and then use the model to forecast out an additional 28 days. Plot the mean and 95% credible interval of the forecast and overlay on the plot the actual cases that occurred. How well did the forecast match the actual data? Interpret your results.

4 Spatial statistics and point process models of gun violence

Chapter 4 of RoC discusses gun violence and how it can be viewed as a contagion process. In this Section we will learn about branching point process models of gun violence that are well-suited for the scenario when the reproduction number is below 1. Here we will consider spatial aspects of gun violence in addition to contagious clustering in time. At the end of the Section we will also look at the Lum and Isaac analysis [20] discussed by Kucharski which investigates potential bias feedback loops that are possible when police use point process models for patrol allocation.

4.1 Mapping and analyzing point patterns

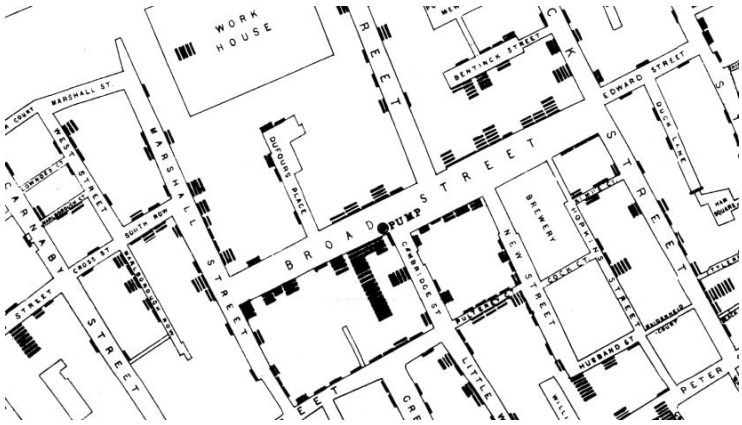

Figure 10: Cholera incidence map made by John Snow in 1854.

In Chapter 4 of RoC we saw the map John Snow made of cholera cases in London, which helped him determine that a source of water was the cause

of the outbreak (Figure 10). Similar maps are used in analyzing urban event data today to identify clusters of crimes, overdoses, traffic crashes, and other types of social harm. When we map a set of locations (that could consist of a latitude and longitude), we call the resulting pattern a "point pattern." Examples of point patterns in the unit square are shown in Figure 11. On the left, is a completely random set of points where the x and y coordinates are independent uniform random variables. It's worth noting that clusters still form by random chance, so when we say something "clusters" what we will mean is that it clusters *more* than you would expect from random chance. In the middle we have plotted a 'regular" or "self-avoiding" point pattern where points are randomly distributed, but no two points are with a fixed radius of each other. For example, the location of apple trees in a commercial orchard might follow a regular point pattern.

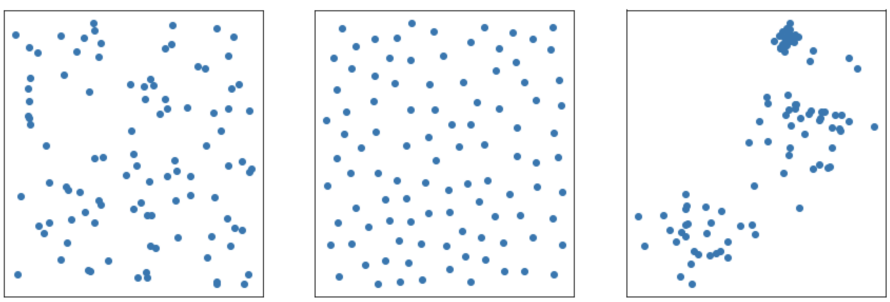

Figure 11: Random (left), regular (middle) and clustered (right) point patterns.

On the right is a clustered point pattern that exhibits more clustering than would occur by random chance. Gun violence, crime, and other urban events tend to be clustered. In part, this is due to geographical constraints (burglary occurs in houses, and not in parks and lakes), but it is also due to human behavior and factors such as contagion.

4.1.1 Ripley's K function

How can we tell whether a point pattern is regular, random or clustered? In some cases it may be visually clear, but we can also use quantitative measures of clustering. One spatial statistic we can use for this purpose is called Ripley's K function [25]. Consider a randomly distributed set of points $\vec{x}_i$, $i = 1, ..., N$, in a spatial region with area A. Next pick a random point $\vec{x}_j$, and a radius r around that point. How many points would we expect to be within that radius? If the points are randomly distributed, then we would expect there to be $N \cdot \pi r^2/A$ points in that radius (ignoring boundary effects). The K function is then defined to be

$$K(r) = \lambda^{-1} E[\text{number of extra points w/in radius r of a random point}],$$

where $\lambda = N/A$. So for a uniformly random point pattern, $K(r) = \pi r^2$.

For a given dataset of points, we can estimate the K function with a sample average:

$$\hat{K}(r) = \frac{A}{N(N-1)} \sum_{i \neq j} 1\{\|\vec{x}_i - \vec{x}_j\| < r\}. \tag{38}$$

Then we can compare that empirical curve against the curve πr^2. If it is higher, then that means clustering at that radius, if it is lower, then that means self-avoidance. Of course even if we simulate data completely at random and compute $\hat{K}(r)$, by random chance the empirical K function will be slightly higher or lower than the theoretical curve. Therefore we typically simulate a large number (several thousand) of uniformly random point patterns on the same domain as our data, with the same number of points, and then compute a confidence interval $[\hat{K}_{low}(r), \hat{K}_{high}(r)]$ from that data. If the empirical K function of the data falls outside this interval, then we may reject the null hypothesis that are data is completely random and claim there is evidence of clustering or avoidance. A Google colab notebook with code for generating the point patterns in Figure 11 and the K function of the point patterns can be found here[4].

4.1.2 Hotspot maps

While the K function, and similarly defined spatial statistics, can help us determine whether our data exhibits clustering, it doesn't give us an intuitive sense of *how much* clustering is present. Furthermore, we would like to identify clusters so that we can intervene and stop an outbreak from spreading.

John Snow's map in Figure 10 gives us a great starting point for what is called a "hotspot map" by criminologists [5]. Starting with a point pattern (locations of cholera incidence), Snow then aggregated those events by dividing street segments into bins and counting the number of events in each bin. The result was a 2-dimensional barchart on the London street network showing that street segments near a particular water pump had an unusually high incidence of cholera.

How can we create such a visualization on a computer? First we need to define spatial units over which we will aggregate data. Street segments are a popular choice today, just as they were for John Snow. Alternatively, polygons such as those defined by census blocks are also used when one would like to associate demographic variables with hotspots of incidents. Suppose we have divided a city (or street network) into M spatial sub-regions U_m, $m = 1, ..., M$, and we have incident data $\vec{x}_i$, $i = 1, ..., N$. To create a hotspot map, we then just need to count the number of incidents, y_m, in each sub-region, U_m:

$$y_m = \sum_{i=1}^{N} 1\{\vec{x}_i \in U_m\}. \tag{39}$$

If $\vec{c}_m$ is the centroid coordinates of region U_m, then we have all we need to visualize the data. For example, we can:

[4] `https://colab.research.google.com/drive/1g16zWxZbtf_ad9PEYWgrO35z9PRDr4H6?usp=sharing`

- place a marker at $\vec{c}_m$ with size proportional to y_m,
- color code the entire region U_m with colors indicating how large y_m is,
- or select the top k largest values of y_m and plot only those corresponding sub-regions.

We plot an example of the third type of hotspot in the left of Figure 12, where we have used squares as the sub-region and we have selected the top k hotspots to plot. The color of the hotspot indicates what type of incident is at risk: vehicle crash (red), burglary (green), larceny (blue) and assault (purple).

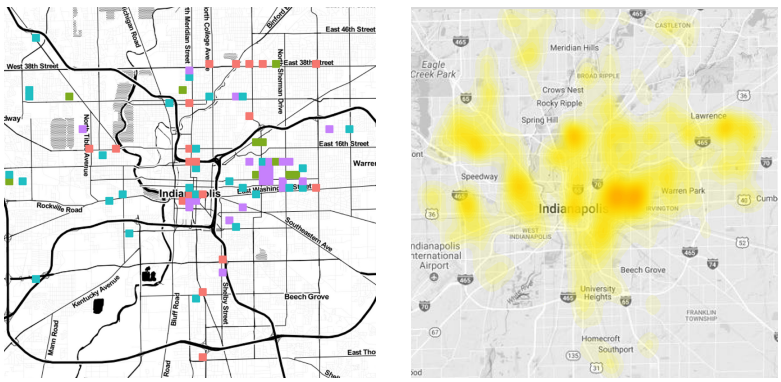

Figure 12: Example hotspot maps. Histogram based hotspots (left) and kernel density based hotspot map (right).

One draw back of these counting based hotspot maps is that we are treating each spatial sub-region U_m as independent of all others. In reality, the counts y_m are often correlated and a hotspot may be comprised of multiple sub-regions. Kernel density estimation (KDE) [30] is one way to account for these correlations between sub-regions and to connect hotspots across the regions:

$$y_m = \sum_{i=1}^{N} g(\vec{c}_m - \vec{x}_i; \sigma). \tag{40}$$

In Equation 40, we center a kernel function g at each point $\vec{x}_i$ that decays the farther away you get from $\vec{x}_i$. A typical choice is a Gaussian density function, where σ is the "bandwidth" that controls how quickly the function decays. Then these functions are summed and evaluated at each sub-region center $\vec{c}_m$. The effect is you get a smoother function representing the level of incidence over space that can be visualized as a "heat map". In example of such a map is shown on the right in Figure 12.

Both the counting based histogram in Equation 39 and KDE in Equation 40 can be viewed as estimators of an underlying probability density of events. There are techniques for selecting how large the spatial units U_m should be, or

for how large to select the bandwidth σ in KDE. There is a balance to achieve, as too large a bandwidth creates a very blurry plot that is not useful and too small a bandwidth will yield hotspots that are due to noise rather than real effects. See [30] for a discussion of bandwidth selection.

4.1.3 The lorenz curve and the gini coefficient

Criminologist David Weisburd posits that there is a "law of crime concentration" [31] which states that over 50% of crime occurs in only a few percent of a city, and that this observation is consistent across many cities around the world. For such an analysis, Weisburd uses street segments as the sub-regions U_m, then calculates the histogram estimator y_m in Equation 39, and sorts the counts y_m to calculate what fraction of sub-regions are needed to contain 25% or 50% of crime. If we denote the sorted counts (ordered greatest to least) as $y_{(m)}$, then

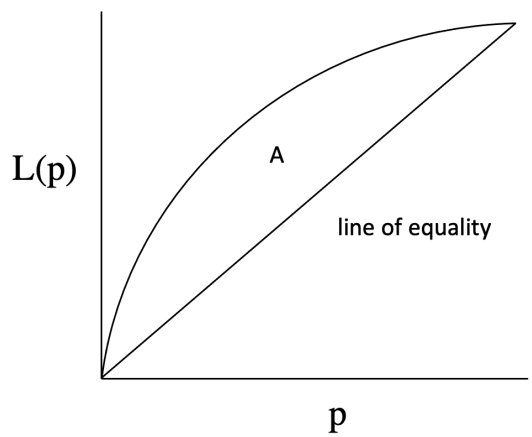

Figure 13: Lorenze curve and the gini coefficient. Gini coefficient G is twice the area between $L(p)$ and the line of inequality, e.g. $G = 2A$.

Weisburd's statistic is a point on the lorenz curve:

$$L(p) = \frac{\sum_{m=1}^{k} y_{(m)}}{\sum_{m=1}^{M} y_{(m)}}, \tag{41}$$

where $p = k/M$ is the fraction of regions or street segments flagged as hotspots. For example, $L(.03) = .5$ can be interpreted as 3% of the city containing 50% of events. We can think of the lorenz curve as a measure of inequality, for if all regions had an equal rate of events then is should be the case that $L(.03) = .03$. It should be noted that in economics the counts are by convention sorted least to greatest, and that the lorenz curve is then below the line of equality.

A related measure of inequality is the Gini coefficient, which is twice the area between the lorenz curve and the line of equality (see Figure 13). The Gini coefficient can be estimated by approximating the integral of the lorenz curve:

$$G = \frac{1}{M}\left(2\left(\frac{\sum_{m=1}^{M}(M+1-m)y_{(m)}}{\sum_{m=1}^{M} y_{(m)}}\right) - M - 1\right). \tag{42}$$

The Gini coefficient can be thought of as an average value of the lorenz curve, and its value ranges from 0 (total equality) to 1 (total inequality). Larger values of G indicate that incidents are more clustered and that a large fraction of events are contained in only a few hotspots.

4.2 Point process models

In addition to clustering in space, gun violence may also cluster in time. In Figure 14, we plot the times of attacks between Locke St. and Lowell St. gangs in Los Angeles. The events cluster more than would be expected by random chance, with periods of no events in between clusters. In this section we will model such clustering with a branching process called a "Hawkes point process" and then use that model to estimate contagion parameters such as the reproduction number and serial interval between events. First we will introduce point processes.

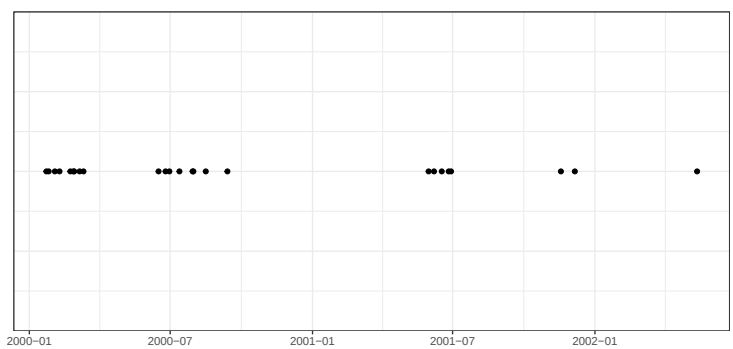

Figure 14: Times of attacks between Locke St. and Lowell St. gangs in Los Angeles from 1999-2002.

4.2.1 Poisson process

Consider a time interval $[0, T]$ and suppose events occur randomly at a constant rate, or "intensity", $\lambda(t) = \mu$. Such a process is called a Poisson process, which is an example of a more general class of models called point processes. Why is the model called a "Poisson" process? There are several ways to simulate data from the model, but one way involves:

1. Draw a Poisson random variable $N \sim Pois(\mu T)$ specifying the number of events.

2. Distribute N event times t_i randomly and independently according to $t_i \sim Unif([0, T])$.

As we can see in the above algorithm, the number of events is Poisson, and then the times of events are random in the time interval. Alternatively, a Poisson process can be simulated sequentially with iid exponential random variables as the inter-event times, e.g. $t_1 \sim exp(\mu^{-1})$, $t_2 \sim t_1 + exp(\mu^{-1})$,...,$t_i \sim t_{i-1} + exp(\mu^{-1})$, which then ceases when $t_i > T$ (so an interesting fact is that, for a uniform distribution of points, the inter-point distances are exponential).

4.2.2 Hawkes process

In order to model events such as those in Figure 14, we need to allow the intensity of events $\lambda(t)$ to change in time. In the case of gang retaliations, we would expect the rate of events to increase after an initial event. One type of point process that models this type of behavior is called a "Hawkes process", where the intensity is modeled as:

$$\lambda(t) = \mu + \sum_{t > t_i} R_0 \gamma e^{\gamma(t - t_i)}. \tag{43}$$

The Hawkes process in Equation 43 has three parameters. The first parameter μ is called the "baseline" or "background" rate, and it models events that occur spontaneously as a Poisson process with rate μ. In the case of gun violence, these events might be an initial random event that then sparks an outbreak of retaliations. The parameter R_0 is the reproduction number of the contagion process and the parameter γ is the recovery rate. An example of such an intensity is shown in Figure 15, where we see after each event time the intensity increases by an amount $R_0\gamma$, and then in the absence of events the intensity decays exponentially back to the baseline rate μ. The Hawkes process can be simulated as a branching process [16]:

1. Draw a Poisson random variable $N \sim Pois(\mu T)$ specifying the number of spontaneous background events.

2. Distribute N background event times t_i^0 randomly and independently according to $t_i^0 \sim Unif([0, T])$. These comprise generation $g = 0$.

3. Given events t_i^g from generation g, simulate events in generation $g + 1$ as:

   - For each event t_i^g, draw a Poisson random variable $N_i^g \sim Pois(R_0)$ specifying the number of direct ancestor events of t_i^g.
   - Add N_i^g events to generation $g + 1$ with event times distributed as $t \sim t_i^g + exp(\gamma)$.

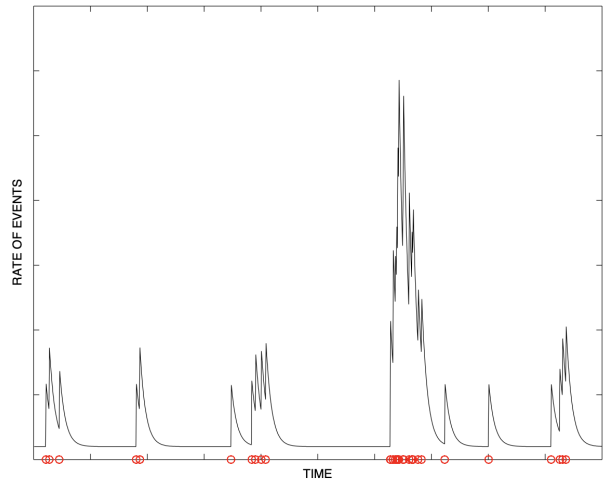

Figure 15: Simulation of a Hawkes process.

4. Stop the simulation when all events in a generation are past the event window ($> T$) or when a generation has no events.

The Hawkes process is connected to the SIR model in Chapter 1 and can be viewed as a stochastic version that models event times a continuous process, rather than counts of events over discrete time intervals [26] (and the Hawkes intensity can be modified to account for finite population effects). An incubation period can also be added to the Hawkes process model by replacing the exponential kernel in Equation 43 with an alternative function with a peak to the right of zero, such as a gamma, weibull, or log-normal density [7].

4.2.3 Estimating a Hawkes process in STAN

Recall from Chapter 3 that we can use the log likelihood function to estimate model parameters. In the case of the Hawkes process, the log-likelihood is given by [23]:

$$logL = \sum_{i=1}^{N} \log(\lambda(t_i)) - \int_0^T \lambda(t)dt. \tag{44}$$

We can also approximate the integral in the likelihood formula as [28]:

$$\int_0^T \lambda(t)dt \approx \mu T + R_0 N. \tag{45}$$

Because the Hawkes process is not included in the STAN library, we need to specify this custom likelihood in our STAN code. In STAN, the log likelihood is called the "target" and we can add this log-likelihood function to the target in

the model section of the STAN code (see Figure 16). A Google colab notebook for simulating and estimating a Hawkes process in STAN can be found here[5].

```
model_code = """
data {
  int N;
  real t[N];
  int T;
}
parameters {
  real<lower=0> gamma;
  real<lower=0,upper=1> R;
  real<lower=0> mu;
}
transformed parameters {
  vector[N] lam;

  lam[1] = mu;

  for (j in 2:N){
    lam[j]=mu;
    for(k in 1:(j-1)){
     lam[j] = lam[j]+R*gamma*exp(-gamma*(t[j]-t[k]));
    }
  }
}
model {
  R ~ beta(1,1);
  mu ~ exponential(.1);
  gamma ~ exponential(.1);
  for (j in 1:N){
    target+=log(lam[j]);
  }
  target+=-mu*T-R*N;
}
"""
```

Figure 16: STAN code with custom likelihood for estimating the Hawkes process in Equation 43.

4.3 Hotspot policing: deterrence, discovery and biased feedback loops

Earlier in this section we saw how hotspot maps can be constructed and used to identify areas of a city with higher than usual rates of harmful events, such as cases of a disease, traffic crashes, crime or shootings. These maps can then

[5] https://colab.research.google.com/drive/1OdZ3AAOORUQwdXGmO8S_NghYrIoUc5wG?usp=sharing

be used for the allocation of resources to perform an intervention with the goal of reducing the risk of harm in the identified hotspots.

One type of intervention is known as "hotspot policing," where police officers are sent to hotspots to conduct a range of activities that can include vehicle patrols, foot patrols, and community engagement. They can also range from very passive activities, such as parking unmanned vehicles with police markings to deter activity, to activities such as stop, question and frisk of individuals passing through hotspots. For example, in the Philadelphia foot patrol experiment in 2006 [24], 200 officers conducted foot patrols in 60 violent crime hotspots for 16 hours a day. The experiment showed a 23% reduction in violent crime (compared to 60 randomized control hotspots). In an experiment in Los Angeles [21], our research showed that one property crime (burglary/vehicle theft) is prevented for every 1000 minutes of police patrol time in hotspots (defined using a Hawkes process similar to that in Equation 43). However, it should be noted that a consequence of hotspot policing is increased encounters between police and citizens. For example, the Philadelphia experiment led to 15% greater drug related incidents and 64% more pedestrian stops. These types of stops and interactions tend to disproportionally impact minority populations [6]. Thus hotspot policing has the potential to *deter* harmful events in hotspots, but it also has the potential to lead to *detection* of things like drug use, broken tail lights, or other minor offenses that allow for police discretion and tends to unfairly target people of color.

As discussed in RoC, Lum and Isaac [20] showed how hotspot policing (or spatial predictive policing) has the potential to lead to feedback loops of police bias. Using drug use data from Oakland California, they showed how a Hawkes process that takes drug crime data as input would direct police to areas with higher rates of detection of drug use in the past. That in turn would lead to more drug arrests in those areas, which would then be fed back into the Hawkes process algorithm, directing more police to those areas tomorrow (hence a feedback loop).

Here we will consider a simplified version of this model. Suppose there are 2 neighborhoods, A and B, and that the Poisson rates of reported crime in those neighborhoods are given by $\lambda_A = \lambda_B$ (thus they have equal rates of reported crime). Next suppose that each day, the police department creates a 10-day hotspot map which is the total number of crime in a neighborhood in the past 10 days. Then, assuming limited resources, they send a patrol today to the neighborhood with the largest value of the hotspot map. Finally, we assume that the police presence influences the rate of crime through detection, such that for today the rate of crime in the neighborhood with patrol is given by $\rho\lambda_A$ (or $\rho\lambda_B$ if B is chosen). Here the factor ρ could reflect deterrence ($\rho < 1$) or detection ($\rho > 1$). Lum and Isaac used a value of $\rho = 1.2$.

In Figure 17, we plot the cumulative number of crimes in each neighborhood from a simulation of this model for 200 days, where at day 100 police are introduced into the model. On the left, we use $\rho = 0.1$, which reflects deterrence. In that simulation the slope of the curves decreases, but both neighborhoods have similar rates of events. This is because the deterrence reduces the hotspot

map value, leading to police alternating between visits to neighorhood A and B. On the right, we show results from a simulation with $\rho = 2.0$. Here we see that after police are introduced at time 100, neighborhood B becomes the main focus of police in perpetuity (even though the rates of reported crime are equal between the two neighborhoods). Depending on the random seed of the simulation, neighborhood A could have become the focus of police. A Google colab notebook with the simulation can be found here[6].

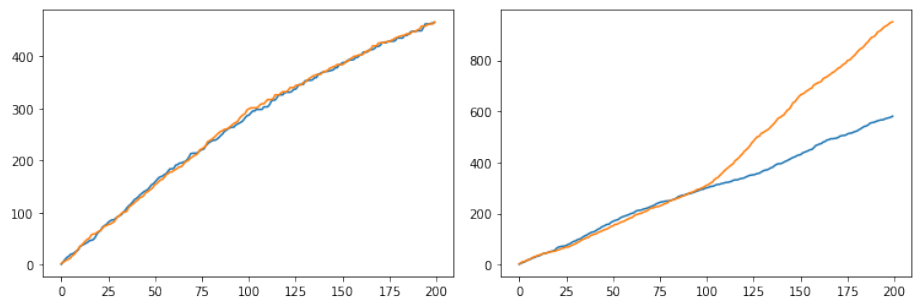

Figure 17: Cumulative crime in neighborhood A (blue) and B (orange) vs time in two simulations of hotspot policing. The simulation on the left is of deterrence ($\rho = .1$) and on the simulation on the right is of detection ($\rho = 2.0$).

Of course, this is an overly-simplified model. In reality, deterrence and detection are both happening during hotspot policing interventions. Deterrence tends to reduce the risk of more serious crime, but at the expense of more pedestrian and traffic stops that disproportionately and negatively impact minority individuals [6]. Research is needed on how to design interventions, whether by police or alternative organizations, that have the benefits of harm reduction while reducing negative side effects, such as those that stem from proactive policing.

4.4 Models that interact with their environment

Much of Chapter 4 in RoC discusses how models can influence policy, thus altering the system they are attempting to model. In the case of COVID-19, early SEIR forecasts showed dire consequences if public health interventions were not quickly employed. Of course, those interventions were implemented, and cases quickly fell, causing the forecast to be "wrong". In the case of hotspot policing above, the model may be used to direct police to certain areas of a city, thus altering what crime data is ultimately fed back into the model.

Reinforcement learning (RL) is a branch of artificial intelligence and machine learning research that attempts to take into account how models interact with their environment, and how models should change based on the feedback they

[6] https://colab.research.google.com/drive/1sbI7H4DQL1-1u-Ys3goFWC2jmP4IuuyK?usp=sharing

get from their environment. While RL could be the subject of an entire semester long course, we will give one simple example here related to the hotspot maps and detection of incidents.

Suppose there are N dormitories at a college each with 1000 students. We believe there is asymptomatic spreading of a virus on campus, and we have a limited supply of tests that can be used to detect cases. In particular, we have enough tests for 10 individuals in a single dormitory per day. How might we choose which dorm to test each day to most quickly learn which has the highest prevalence of the disease? The problem above is known as a multi-armed bandit problem. The name comes from slot machines at casinos, which have an "arm" that you pull to play the game (and are called bandits because they take your money). Different machines are set to somewhat different average payouts, so the goal is to quickly find the best machine to play.

We conclude this section with the ϵ-greedy multi-armed bandit algorithm shown in Algorithm 4. In each round we choose an arm k to pull. With probability ϵ we choose an arm randomly, which allows us to explore arms we have yet to visit (and which could have a higher payoff). Or, with probability $1 - \epsilon$, we choose to play the arm with the largest average payoff based on our historical observations. The parameter ϵ can be tuned to balance exploration vs. exploitation of existing knowledge. These types of algorithms can be useful when dealing with partially observed events, and when we have to take a particular action to observe data. We leave it as an exercise to apply the ϵ-greedy algorithm to the random dormitory testing scenario above.

Algorithm 4 MAB ϵ-greedy

```
Initialize mean reward m_k = 0 at each arm k
for t=1:T do
    if ϵ >rand() then
        pick arm k randomly
        else
        pick arm k =argmax_j m_j
    end if
    observe reward r_k^t of arm k at time t
    update m_k as the average of all rewards observed at arm k
end for
```

4.5 Chapter 4 Exercises

1a. Import the data "chicago_gun_violence_reports.csv" from canvas into python, which contains reports of batteries in Chicago involving a gun from 2016 to 2021. Calculate the total number of events y_m in each police "Beat" m in the data. Which beat has the most events and how many does it contain?

1b. Next order the event counts y_m in descending order and calculate the lorenz curve and gini index. Make a plot of the lorenz curve. What percent of events

are captured in the top 10% of beats?

1c. Create a hotspot map of the top 10% of beats with the most reported batteries involving a gun. Place a circle marker at the center of each beat (as defined by the average or median of the incident locations in the beat) and make the size of the marker proportional to the number of incidents. You can use plotly and mapbox to place these markers on the map.

2a. Download the data "LockeLowell.csv" containing the event times of attacks between Locke St. and Lowell St. gangs in Hollenbeck Los Angeles from 1999-2002. Use the STAN code provided in Section 4.2.3 to estimate the reproduction number and recovery rate parameters of a Hawkes process fit to the data. How do these values compare to the parameters $R_0 = 0.63$ and $\gamma = (125days)^{-1}$ from Chicago discussed in RoC?

2b. Using the posterior mean parameters for μ, R_0 and γ found in 2a, simulate data from a Hawkes process for 3 years using these parameters and make a plot similar to Figure 14. Qualitatively, how does the clustering pattern of these simulated events compare to the Locke-Lowell data?

3. Suppose there are $N = 10$ dormitories at a college each with 1000 students. We believe there is asymptomatic spreading of a virus on campus, and we have a limited supply of tests that can be used to detect cases. In particular, we have enough tests for 10 individuals in a single dormitory per day. Finally, suppose that the true prevalence rates of the dorms $k = 1, ..., 10$ are $p = k/100$ (note that observations at a dorm thus are a binomial with parameters 1000 and p). Implement the ϵ-greedy algorithm to optimize the average number of detected cases over 100 days of testing. See how the results change for different $\epsilon = 0, .05, .1, .2$. For each of these ϵ, make a plot of number of cases detected vs time for 10 simulations.

5 Online viral processes

In Chapter 5, Kucharski explores a variety of examples of online contagion processes. These include viral social media cascades, online opinion formation, and contagion of emotion. In this type of research, we often need to analyze the text of social media posts to understand what is being said, what opinion the poster may hold and what their sentiment towards an issue is. In this section we first introduce some basic tools for analyzing text data. We then explore a dynamic network model for the formation of echo chambers.

5.1 Text analytics

Consider the data in Figure 18. It contains (fake) data on post users made online about whether they prefer ice cream or frozen yogurt. We will explore

some basic concepts in text analytics using this data as an example.

```
import pandas as pd
df=pd.DataFrame({'post_id':[1,2,3,4],
                 'post_text':['I love ice cream',
                              'I HATE frozen yogurt',
                              'Chocalate ice cream is my favorite',
                              'I eat froyo all the time'],
                 'label':['ice cream','ice cream','ice cream','frozen yogurt']})
print(df)
```

```
   post_id                          post_text          label
0        1                   I love ice cream      ice cream
1        2                I HATE frozen yogurt      ice cream
2        3  Chocalate ice cream is my favorite      ice cream
3        4            I eat froyo all the time  frozen yogurt
```

Figure 18: Pandas dataframe containing text information.

First, if our dataset is large, we may wish to get a sense of what individuals are talking about in the data by looking at the frequency of keywords. To do this, we can first split each sentence into a list of separate words. We can then create a long dataframe (using panda's explode function), where each word is given a new row in the data. Finally, we can use pandas group by and aggregation to count the word frequencies. An example is shown in Figure 19.

Note in Figure 19 that the word HATE is capitalized, and that if it appeared again in lower case that it would be counted as two separate words. So it is common practice to convert words to lower case before analyzing text data, for example this can be done in python as "HATE".lower(). We might also want to remove "stop words" such as "the", "and", etc. that do not contain much information on what topics are contained in the text. The python package "nltk" contains a list of stopwords you can use to remove them from a dataset of text (once phrases are split into separate words).

5.1.1 Sentiment of text

In Chapter 5 of RoC, we read about the controversial Facebook emotion experiment [18], where users in treatment groups were either shown more positive posts (resulting in the users subsequently posting more positive comments), or more negative posts (resulting in the users posting more negative content). The controversy stemmed from the researchers failing to get informed consent from users ahead of time.

But how does one even figure out if a post is negative or positive? It would be impractical for humans to label millions of posts from users, so an algorithm is typically used to make such an assessment. One way to do this is to use a manually coded dictionary of words where humans have assigned a "polarity" score to each word indicating whether it is negative, positive or neutral. For example, in Figure 20 we apply sentiment analysis provided by the textblob package to the sentence "I hate cold frozen yogurt." In textblob's sentiment

```
df['word']=df['post_text'].str.split(" ")
df_long=df.explode(['word']).reset_index()
print(df_long[['post_id','word']].head(10))
```

```
   post_id       word
0        1          I
1        1       love
2        1        ice
3        1      cream
4        2          I
5        2       HATE
6        2     frozen
7        2     yogurt
8        3  Chocalate
9        3        ice
```

```
dfcount=df_long.groupby(['word']).\
    size().reset_index(name='count').\
    sort_values(['count'], ascending=False)
print(dfcount.head(10))
```

```
        word  count
2          I      3
4      cream      2
9        ice      2
0  Chocalate      1
1       HATE      1
3        all      1
5        eat      1
6   favorite      1
7      froyo      1
8     frozen      1
```

Figure 19: Counting word frequency using pandas.

word library, “cold” is assigned a polarity value of -0.6 and “hate” is assigned a value -0.8. All the other words are deemed neutral, with a polarity value of 0.0. Then the non-zero polarity values in the sentence are averaged, such that the polarity of the sentence is assigned a score of -0.7.

```
from textblob import TextBlob

text="I hate cold frozen yogurt"
print(TextBlob(text).sentiment)

text="hate"
print(TextBlob(text).sentiment)

text="cold"
print(TextBlob(text).sentiment)
```

```
Sentiment(polarity=-0.7, subjectivity=0.95)
Sentiment(polarity=-0.8, subjectivity=0.9)
Sentiment(polarity=-0.6, subjectivity=1.0)
```

Figure 20: Sentiment analysis using textblob.

5.1.2 Text classification

Another task we often need to perform with text data is classification, where we might want to classify whether a post is on a particular topic (maybe we would like to find the subset of Tweets related to COVID-19 vaccination), or we might want to classify the opinion expressed in the post (whether the post is pro- or anti-vaccination). For example, in the article by Flaxman et al [13] on echo chambers discussed in RoC, the authors needed to classify news articles as regular articles vs. opinion pieces.

One approach to this problem (and that taken by Flaxman et al), is to first create a document-term matrix. This is a matrix where each row corresponds to a “document,” which could be a Facebook post or a news article, and each column corresponds to a word. Then the matrix is filled with zeros and ones, where zero indicates the word was not present in that document and one indicates that the word is present (or instead the entry can be the count of the word if it appeared multiple times). An example is shown in Figure 21.

Then, given labels for each document (whether an article is news or opinion, or whether a post is pro- or anti-ice cream), we can then train a classifier using the doc-term matrix as features. For example, given two categories labeled as

```python
from sklearn.feature_extraction.text import CountVectorizer

docs = ['I love frozen bananas','I hate bananas in ice cream']

vec = CountVectorizer()
X = vec.fit_transform(docs)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
print(df)
```

```
   bananas  cream  frozen  hate  ice  in  love
0        1      0       1     0    0   0     1
1        1      1       0     1    1   1     0
```

Figure 21: Document-term matrix.

$y = 1$ and $y = -1$, we could form a regression as:

$$y = a_0 + a_1 1\{word_1\} + a_2 1\{word_2\} + ... + a_n 1\{word_n\}. \tag{46}$$

Here the coefficient a_i multiplies an indicator variable for whether $word_i$ is present or absent in the document. If a_i is positive, that means that $word_i$ tends to be present when $y = 1$ is the label of the document. If a_i is negative, that means that $word_i$ tends to be present when the document is of the other type ($y = -1$).

There is a problem with this approach however, namely that there is over a hundred thousand possible words. So depending on the number of rows in our data, we may be very likely to overfit noise in the data. For this reason we often reduce the number of possible words. One approach is to remove stop words, or also use words if they appear frequently enough, but also not too frequently. Suppose we are trying to decide if a person is pro- or anti-mask useage based on posts. A word such as COVID-19 may appear quite frequently, in fact it may be in almost every post. So it won't be very useful in classifying between the two categories. On the other hand, if a word such as a proper noun only appears once, then it also won't be useful. We can quantify this using "term frequency-inverse document frequency" (tf-idf), which is the frequency of a word in a document divided by the percentage of documents in which the word appears. Thus we often use a tf-idf doc-term matrix, which is smaller than the full doc-term matrix. An example of using tf-idf and linear regression to classify news articles can be found here[7].

The keyword-regression approach has two problems that can't be solved by reducing the number of words or using tf-idf. One problem is how to deal with synonyms, because if we say "agriculture" or "farming", these are basically the same thing for classification purposes but they will be treated as two separate words. Neural networks have an advantage in that they "embed" (or "represent") words as a vector, in a way such that two words that are similar have

[7] https://scikit-learn.org/stable/auto_examples/text/plot_document_classification_20newsgroups.html#sphx-glr-auto-examples-text-plot-document-classification-20newsgroups-py

vectors that are nearby in euclidean space. Second, it is often not just a single word, but context and dependencies between words that give text meaning. If I say "I hate that I have to wait so long to get ice cream", the regression approach may classify this as me hating ice cream, but the context of me waiting really means that I like ice cream. Large neural network based language "transformer" models [10] are able to learn these types of dependencies when trained on millions of documents and are currently achieving state of the art performance on a number of language tasks (but they are outside the scope of these lecture notes).

5.2 A dynamic network model of echo chamber formation

In Section 2, we saw the network SIR model where a contagion process dynamically evolved on a network. In that model, the network was fixed ahead of time and then the dynamics evolved on top of the network.

However, in an online system such as Facebook or Twitter, our behavior can actually change the network. For example, which friends posts you "like" on Facebook is fed into an algorithm that determines which friends posts you see (or don't see) in the future, thus changing the strength of friendship ties on the network. In [3], a dynamic network model for how echo chambers might form is presented and we will consider that model in this section.

Consider N nodes $i = 1, ..., N$ and suppose each individual has a parameter $x_i \in [-\infty, \infty]$ that characterizes that individuals opinion about a particular issue. There are two opinions, where $x_i < 0$ indicates one side, and $x_i > 0$ the other opinion (where the more negative or positive x_i is, the more strongly the opinion is held). Additionally, each individual has a parameter a_i that determines how active individual i is online. Initially we draw random variables for the opinion and activity variables: $x_i \sim Unif([0, 1])$, $a_i \sim Pareto(\gamma)$.

Next, we simulate the dynamics for discrete time steps $t = 1, ..., T$, where at each time step:

1. User i is activated with probability a_i.

2. If user i is active, they pick m friends to influence. The probability individual j is chosen as a friend is determined by $p_{ij} \propto |x_i - x_j|^{-\beta}$. If individual j is chosen we set the current adjacency value at time t to $A_{ij}(t) = 1$ ($A_{ij}(t) = 0$ otherwise). Thus when the opinions are more similar, the probability of a link is greater at that time.

3. If $A_{ij}(t) = 1$, with probability r we set $A_{ji}(t) = 1$. Thus friends you choose to interact with can also influence you.

4. Finally, the opinion variable is updated in time according to Equation 47.

$$\frac{dx_i}{dt} = -x_i + K \sum_j A_{ij}(t) \tanh(\alpha x_j) \tag{47}$$

In Equation 47, the first term on the right leads to the opinion becoming less strong (closer to zero) in the absence of friend influence. The second term models the influence of friends at time t, where the tanh function saturates at -1 or 1 depending on the opinion x_j of friend j. The parameters α and K determine how much influence the friends' opinions have on i at the next time step. In Figure 22, we plot a visualization of the network at the beginning of the simulation and after 1000 time steps. We can see that initially there is polarization, but that there are still quite a few connections between individuals of differing opinions. Later, as opinions become more polarized, echo chambers form where individuals almost only interact with individuals of similar opinions.

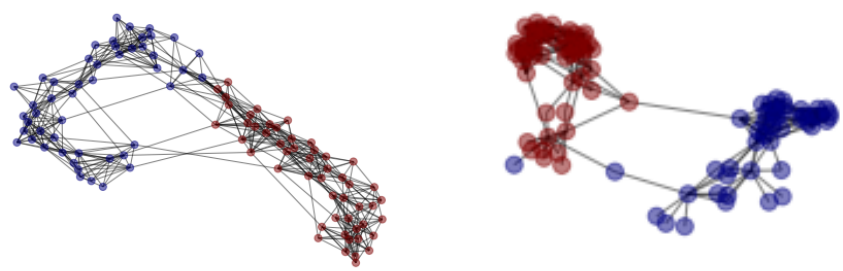

Figure 22: Network with nodes colored by opinion variable (blue vs. red) at start (left) and end (right) of simulation of the echo chamber model in [3]

.

5.3 Chapter 5 Exercises

1. In this assignment we will analyze short descriptions of news articles from the Huffington Post (provided in the data huffpost.csv on Canvas).

a. First import the data using pandas and inspect the columns and the first few rows.

b. Next let's find the top 10 most common words by "category" of article. First use "str.split" on the "short_description" column to break each paragraph into separate words. Next use pandas "explode" to create a long dataframe where each word is in a new row. Convert each word to lower case using "str.lower" and then finally use a groupby aggregation (similar to the Chicago hotspot assignment) to count the frequency of words.

c. Repeat b), but this time with "stop words" removed.

d. Use the sentiment function in the textblob package to computer the sentiment of each short description. What is the average sentiment of U.S. NEWS and ENTERTAINMENT categories?

e. Fit and assess a regression model to predict U.S. NEWS vs ENTERTAINMENT articles. Subset the data to only those categories. Use sklearn's

TfidfVectorizer to convert the text data to keyword features. Fit a ridge regression classifier on 2/3 of the data. Assess the accuracy of the classifier on the remaining 1/3 of the data. What keywords correspond to the largest coefficients of the model?

2. Simulate the dynamic model for echo chamber formation in Section 5.2 using the parameters $K = 3$, $\beta = 3$, $\alpha = 3$, $r = 0.5$, $m = 10$, $\gamma = 1.1$ for $N = 100$ individuals. Use forward Euler with $dt = 0.01$ to solve Equation 47 for $T = 1000$ time steps. Make a plot of the final network with adjacency matrix $A(1000)$, where the nodes are colored according to their final opinion variables x_i. Also keep track at each time of how many connections there are in the network between individuals whose opinions are different, e.g. $x_i x_j < 0$. Make a plot over time of the number of such connections.

6 Phylogenetic analysis

In Chapter 7 of RoC, we saw how phylogenetic analysis of DNA from viruses can help us track outbreaks and answer questions related to when an epidemic first arrived in a region and the history of its evolution. For example, if a virus is fast-evolving, and DNA sequencing in a region shows little variation across cases detected, that would indicate the virus is relatively new to the area. We also saw that phylogenetic analysis can be applied to written text, for example to trace the origin and evolution of folk tales.

6.1 Hamming distance

Consider the two genetic sequences below, consisting of a sequence of characters A, C, G, T representing the four nucleotide base pairs of DNA:

$$ACTGAATC$$
$$AATGCATC$$

We can calculate the "Hamming distance" between the two sequences, which is the number of characters that differ between the sequences. We have shown in red the two characters that are not the same, indicating that the Hamming distance is 2. Sometimes the Hamming distance is normalized by dividing by the length of the sequence.

The Hamming distance tells us how close two sequences from two different samples of a virus are genetically. If the Hamming distance is small for two SARS-COV-2 RNA sequences, for example, then they are more likely to be the same variant (e.g. both omicron or both delta). We can use the biopython package[8] to calculate the Hamming distance between sequences and construct a phylogenetic tree.

[8] `https://biopython.org/`

6.2 Sequence alignment

Often a pre-processing step called sequence "alignment" is required before comparing two or more genetic sequences. For example, consider the following two sequences:

$$ACTAGC$$
$$TACTGC$$

where an A has been inserted in the first sequence between T and G and a T has been inserted at the beginning of the second sequence. These sequences can then be aligned as:

-ACTAGC
TACT-GC

where a dash has been inserted for the missing T in the first sequence and the missing A in the second sequence. We then can compute the similarity of the aligned sequences (for example using Hamming distance).

A number of algorithms exist for sequence alignment (which is called multiple sequence alignment in the case when three or more sequences are aligned at the same time). Roughly speaking, insertions can be added through trial and error, where the similarity between sequences can be computed for proposed insertions, and then if the sequences are more similar with an insertion, we keep it in the sequence (otherwise we reject the proposal). This type of algorithm falls under the category of dynamic programming. Other heuristic based algorithms attempt to match patterns between the sequences (for example, the "ACT" and the "GC" patterns in the example above). The biopython package has several algorithms implemented for pairwise and multiple sequence alignment.

6.3 Phylogenetic tree and UPGMA algorithm

With many different DNA/RNA sequences, for example coming from different virus variants or different species of animals, it can be useful to group or "cluster" the sequences in a phylogenetic tree. The tree can be thought of in the same way as a family tree, where ancestors are placed earlier in the tree, and then sequences branch-off to form separate lineages. The closer two sequences are in the tree, the more closely related are the virus variants or species. We show an example tree computed using biopython in Figure 23.

There are a number of different algorithms for clustering genetic (or textual) sequences, so we will just present one here to give you an idea of how it can be done. The algorithm is called Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [15].

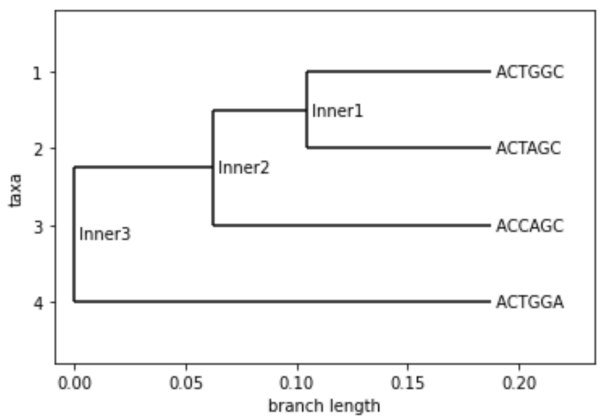

Figure 23: Phylogenetic tree of four sequences.

Suppose we wish to cluster the following four sequences:

CTAG
CTAA
ATGA
TAGA

The first step is to construct a matrix M where entry m_{ij} is the Hamming distance between sequence i and j. For the above four sequences we get:

$$\begin{bmatrix} 0 & 1 & 3 & 4 \\ 1 & 0 & 2 & 3 \\ 3 & 2 & 0 & 2 \\ 4 & 3 & 2 & 0 \end{bmatrix} \tag{48}$$

For example, sequence 1 and 2 differ by 1 character, so $m_{12} = m_{21} = 1$. However, sequence 1 and 4 differ by 4 characters, so $m_{14} = m_{41} = 4$. Note that in biopython these values are divided by the sequence length (which in this case if 4).

The UPGMA clustering works iteratively, where given the current matrix, we pick the two sequences that are closest and combine them into a group. In this case we combine 1 and 2, and thus are left with groups $(1, 2), 3, 4$. We then update the distances in a new matrix, which is the distance between these three groups. The distance between sequence 3 and 4 is still the same, but now we find the distance between the group $(1, 2)$ and the other sequences using the arithmetic mean of the distances of 1 and 2 to the other sequences in the matrix (hence the name UPGMA). Because sequence 1 is distance 3 to sequence 3, and sequence 2 is distance 2, the distance from $(1, 2)$ to three is the average: 2.5. Similarly, the distance of $(1, 2)$ to sequence 4 is 3.5. We then get the following

matrix representing the distances from the three groups $(1,2), 3, 4$:

$$\begin{bmatrix} 0 & 2.5 & 3.5 \\ 2.5 & 0 & 2 \\ 3.5 & 2 & 0 \end{bmatrix} \tag{49}$$

We then proceed iteratively, where we pick the 2 closest groups in this matrix, which are sequences 3 and 4 (that are 2 apart). We thus combine them into a group, $(3,4)$, resulting in two remaining groups $(1,2)$ and $(3,4)$ and matrix:

$$\begin{bmatrix} 0 & 3 \\ 3 & 0 \end{bmatrix} \tag{50}$$

Here the distance of group $(3,4)$ to $(1,2)$ is the average of 3 to $(1,2)$ and 4 to $(1,2)$, which is $(2.5+3.5)/2 = 3$. Because there is only one remaining pair of groups, the final tree is given by $((1,2),(3,4))$.

This python colab notebook[9] contains example code for creating sequence objects, computing Hamming distance, aligning sequences, and performing UPGMA clustering.

6.4 Chapter 7 Exercises

1. The CSV file from canvas "covid_sequences.csv" contains 5 sequences labeled omicron, mu, delta, alpha and gamma. The CSV also contains an unlabeled 6th sequence. Compute the Hamming distance of the the 6th sequence to the first 5 labeled sequences to determine which variant the new sequence is likely to be.

2. Cluster by hand the following four sequences using the UPGMA algorithm. Then use the biopython algorithm to check your work.

AACT
CTAG
AACA
CGAA

3. Run the UPGMA algorithm in biopython on the data from question 1 and create a plot of the phylogenetic tree.

Acknowledgements

Development of these lecture notes was supported in part by NSF grant ATD-2124313 and ATD-2317397.

[9] `https://colab.research.google.com/drive/1kPGAB0y2HhssRKqBbFZvaZeT1XR9UrAD?usp=sharing`

References

[1] https://statisfaction.wordpress.com/2020/04/09/sir-models-with-kermack-and-mckendrick/.

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[3] Fabian Baumann, Philipp Lorenz-Spreen, Igor M Sokolov, and Michele Starnini. Modeling echo chambers and polarization dynamics in social networks. *Physical Review Letters*, 124(4):048301, 2020.

[4] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

[5] Spencer Chainey and Jerry Ratcliffe. *GIS and crime mapping.* John Wiley & Sons, 2013.

[6] Aaron Chalfin, Benjamin Hansen, Emily K Weisburst, and Morgan C Williams Jr. Police force size and civilian race. *American Economic Review: Insights*, 4(2):139–58, 2022.

[7] Wen-Hao Chiang, Xueying Liu, and George Mohler. Hawkes process modeling of covid-19 with mobility leading indicators and spatial covariates. *International journal of forecasting*, 38(2):505–520, 2022.

[8] Nicholas A Christakis and James H Fowler. The spread of obesity in a large social network over 32 years. *New England journal of medicine*, 357(4):370–379, 2007.

[9] Jonas Dehning, Johannes Zierenberg, F Paul Spitzner, Michael Wibral, Joao Pinheiro Neto, Michael Wilczek, and Viola Priesemann. Inferring change points in the spread of covid-19 reveals the effectiveness of interventions. *Science*, 369(6500):eabb9789, 2020.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[12] J Douglas Faires and Richard L Burden. *Numerical methods.* Thomson, 2003.

[13] Seth Flaxman, Sharad Goel, and Justin M Rao. Filter bubbles, echo chambers, and online news consumption. *Public opinion quarterly*, 80(S1):298–320, 2016.

[14] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

[15] Ilan Gronau and Shlomo Moran. Optimal implementations of upgma and other common clustering algorithms. *Information Processing Letters*, 104(6):205–210, 2007.

[16] Alan G Hawkes and David Oakes. A cluster process representation of a self-exciting process. *Journal of Applied Probability*, 11(3):493–503, 1974.

[17] William O Kermack and Anderson G McKendrick. Contributions to the mathematical theory of epidemics–i. 1927. *Bulletin of mathematical biology*, 53(1-2):33–55, 1991.

[18] Adam DI Kramer, Jamie E Guillory, and Jeffrey T Hancock. Experimental evidence of massive-scale emotional contagion through social networks. *Proceedings of the National Academy of Sciences*, 111(24):8788–8790, 2014.

[19] Adam Kucharski. *The Rules of Contagion: Why Things Spread–And Why They Stop*. Basic Books, 2020.

[20] Kristian Lum and William Isaac. To predict and serve? *Significance*, 13(5):14–19, 2016.

[21] George O Mohler, Martin B Short, Sean Malinowski, Mark Johnson, George E Tita, Andrea L Bertozzi, and P Jeffrey Brantingham. Randomized controlled field trials of predictive policing. *Journal of the American statistical association*, 110(512):1399–1411, 2015.

[22] Mark EJ Newman. Mixing patterns in networks. *Physical review E*, 67(2):026126, 2003.

[23] Tohru Ozaki. Maximum likelihood estimation of hawkes' self-exciting point processes. *Annals of the Institute of Statistical Mathematics*, 31(1):145–155, 1979.

[24] Jerry H Ratcliffe, Travis Taniguchi, Elizabeth R Groff, and Jennifer D Wood. The philadelphia foot patrol experiment: A randomized controlled trial of police patrol effectiveness in violent crime hotspots. *Criminology*, 49(3):795–831, 2011.

[25] Brian D Ripley. The second-order analysis of stationary point processes. *Journal of applied probability*, 13(2):255–266, 1976.

[26] Marian-Andrei Rizoiu, Swapnil Mishra, Quyu Kong, Mark Carman, and Lexing Xie. Sir-hawkes: linking epidemic models and hawkes processes to model diffusions in finite populations. In *Proceedings of the 2018 world wide web conference*, pages 419–428, 2018.

[27] Sheldon Ross. *A first course in probability*. Pearson, 2010.

[28] Frederic Paik Schoenberg. Facilitated estimation of etas. *Bulletin of the seismological Society of America*, 103(1):601–605, 2013.

[29] Cosma Rohilla Shalizi and Andrew C Thomas. Homophily and contagion are generically confounded in observational social network studies. *Sociological methods & research*, 40(2):211–239, 2011.

[30] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[31] David Weisburd. The law of crime concentration and the criminology of place. *Criminology*, 53(2):133–157, 2015.

[32] Stephen Wright, Jorge Nocedal, et al. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.