# RKnot

Feb 23, 2021

# Contents

*A Simulation Space for Analyzing Viral Spread in Customized Populations*

```
<IPython.core.display.HTML object>
```

CHAPTER 1

# Introduction

**RKnot** is a simulation architecture for viral spread that aims to:

1. Faithfully replicate spread in real world conditions
2. Investigate the impacts of policy decisions and other interventions
3. Provide visualization tools for ease-of-presentation

**RKnot** attempts to distinguish itself from prevailing models by:

- allowing for customized population sizes and demographics
- supporting more realistic movement and contact patterns
- modulating transmission risk to account for varying subject behavior and location properties
- influencing interactions via an array of interventions that can be used in any combination

**RKnot** utilizes parallelization via Ray and JIT-compilation via Numba for performance improvements amd Matplotlib for visualizaitons.

```
<IPython.core.display.HTML object>
```

## 1.1 Basic Example

A simulation and visualization can be completed in a few quick lines of code. The user need only specify a dictionary describing the population, `group`, and a handful of parameters describing the simulation space and viral characteristics.

Below we simulate:

- a population of 1,000 subjects,
- beginning with five initially infected;
- a density of 1 subject per location
- a maximum simulation length of 150 days

  - *the simulation will automatically stop when there are no more infections*

- an initial reproduction number, $R_0$, of 3
- an infection duration of 14 days
- an immunity duration of 365

```
from rknot import Sim, Chart

group = {'n': 1000, 'n_inf': 5}
params = {'R0': 3,'imndur': 365, 'infdur': 14, 'density': 1, 'days': 150}


sim = Sim(groups=group, **params)
sim.run()

chart = Chart(sim)
chart.to_html5_video()
```

```
<IPython.core.display.HTML object>
```

Results:

| | |
|---|---|
| Peak | 52.1% |
| HIT | 72.7% |
| Total | 92.2% |
| Fatalities | 0.00% |
| % > 70 | nan% |
| IFR | 0.00% |
| Days to Peak | 36 |

As per the chart, this simulation results in a peak at 29 days, with 53% of the population infected at the peak and a Herd Immunity Threshold of 70%. In total 93% of the population was infected and 0.4% of the population, or 4 subjects, died.

## 1.2 Next Steps

- *Here you can explore how viral spread theory is incorporate into *RKnot*.*
- *Here you can learn about the core concepts on which the simulation architecture is built.*

Or you can jump right into the example simulations we have built and explore how different properities impact spread:

- *Factors Influencing Spread*
- *Impact of Dynamce Transmission Risk*

# CHAPTER 2

# Viral Spread Theory in RKnot

*RKnot* is built up from the contact level. A contact is an interaction between an infected person and a susceptible person that could result in transmission of the virus (more details *below*). Contacts occur in a 2d environment in which subjects move randomly according to pre-defined distributions or deterministically according to event subscription.

When a contact occurs, transmission is determined stochastically according to a multi-faceted likelihood of transmission tailored specifically to the infected and susceptible subjects in contact at the specific time at the specific location.

The expected number of new infections at any time, can be found as the sum of all contacts at that time multiplied each contact's specific transmission risk.

$$E(n_{infs}) = \sum_{i=1}^{N} c_i * \tau_i (2.1)$$

$\tau$ is a fundamental property of virus that can be further augmented for properties of the infected, or the subject, or the location of the contact.

$$\tau_i = \tau * T_{inf,i} * T_{sus,i} * T_{loc,i} (2.2)$$

Thus, the transmission risk of the virus is required in order to mimick its spread. This can be provided directly or derived from a virus' estimated $R_0$.

## 2.1 $R_0$

The propensity for a virus to spread is most-commonly referenced by its Basic Reproduction Number, $R_0$. $R_0$ is the number of new infections that will be caused by a single infected person in an entirely susceptible population.

At an individual level, $R_0$ is influenced by many factors: the number of contacts the infected person has, the location of contacts, the social setting of contacts, etc.

At a population level, many models of $R_0$ assume that the individual factors average out, thus leaving us with a property that is fundamental to the virus itself. We can see there is a broad range of $R_0$ values for different viruses in the wikipedia entry.

$R_0$ can be described as:

$R_0 = \beta * d$(2.3)

*where:* $\beta$ *= transmission rate (infections / day)*
$d$ *= duration of infection (days)*(2.4)

The above ignores the number of contacts made, however, $\beta$ can be further broken down as:

$\beta = \tau * \overline{c}$(2.5)

*where:* $\tau$ *= probability of transmission, or transmission risk*
$\overline{c}$ *= contacts per day*(2.6)

which yields:

$R_0 = \tau * \overline{c} * d$(2.7)

Using the relationship above, we can simulate viral spread contact by contact, however, we must have a value(s) for $\tau$.

## 2.2 Static Transmission Risk

There are many mathematical models used to describe viral transmission, the most commonly-referenced being the SIR model (Suscepitble-Infected-Recovered).

The SIR model makes several simplifying assumptions:

- Closed, well-mixed population with no demography
- Constant Rates

This allows viral spread to be described in 3 equations.

$$\frac{ds}{dt} = \beta si$$
$$\frac{di}{dt} = \beta si - id^{-1}$$
$$\frac{dr}{dt} = di(2.8)$$

$$\textit{where: } s = S/N = \textit{number of susceptible / total population size}$$
$$\textit{where: } i = I/N = \textit{number of infected / total population size}$$
$$\textit{where: } s = R/N = \textit{number of recovered / total population size}$$

The simplified model allows for some quick estimates of various spread characteristics. Herd Immunity threshold, for instance, can be found as:

$$\text{HIT} = 1 - 1/R_0(2.9)$$

For instance, $R_0$ of 2.5x, the prevailing estimate for sars-cov-2, yields a HIT of 60%.

The assumption of constant rates presents problems for simulating at the contact level, however. SIR assumes that transmission risk is constant during the infection period and that each subject in the population has the same number of contacts and so is able to ignore $\tau$ and $c$.

Referring to equation (5) above:

- $R_0$ is known (from external analysis and provided by the user)
- $d$ is known (from external analysis and provided by the user)

Thus, unknowns are $\tau$ & $\bar{c}$

As per above, we must know $\tau$ in order to simulate spread.

### 2.2.1 Expected Contact Rate

While we do not know $\bar{c}$, the simulation space is given a number of parameters that allow us to estimate the expected number of contacts. We know:

- The population size

- The number of locations
- The movement patterns of subjects
- The likelihood that a subject will be at a particular location at a particular time given 1/2/3 above

A simple method to estimate $\overline{c}$ is to assume that each subject is equally likely to be in any one location at any time. The probability of a single dot being in a singe location is:

$P(LOC_{xy}, DOT_i) = 1/N$(2.10)

*where:* $xy$ *= coordinates of the location*
$N$ *= number of locations*(2.11)

The probability of another dot being there at the same time:

$P(LOC_{xy}, DOT_{ij}) = 1/N * 1/N$(2.12)

The probability of $k$ dots being at the same location at the same time:

$P(LOC_{xy}, DOT_{ij}) = 1/N^k$(2.13)

Then, the number of ordered contacts is:

$\sum_{i=1}^{k}(1/N^k)$(2.14)

And for all possible orders:

$E(\overline{c}) = k * \sum_{i=1}^{k}(1/N^k)$(2.15)

> The result is a contact rate that is closely related to the density of the simulation space. For example, a population of 100 subjects in a space of 100 locations, would have an expected contact rate of 1.01 per day

In fact, we can prove that the contact rate for an equal mover is the density using probability theory. We can view the probability of contact with other dots in the space as a binomial random variable with the *trials* representing the number of dots at a location, so:

$$\begin{aligned} B(k-1, 1/N) \\ \textit{k is the number of dots} \\ \textit{N is the number of locations}(2.16) \end{aligned}$$

If there are three dots (and 3 locations), the distribution governing their interactions is the sum of the two Binomial functions:

$$B(2-1, 1/3) + B(2-1, 1/3)$$

We know the sum of two Binomial distributions is:

$$B(m, p) + B(n, p) = B(m+n, p)$$

so:

$$B(2-1, 1/3) + B(2-1, 1/3) = B(2, 1/3)$$

and for k dots:

$$\sum_k^i B(n_i, 1/N) = B(\sum_k^i n_i, 1/N) \text{ where: } n_i = 1(2.17)$$

which simplifies to:

$$B(k, 1/N)(2.18)$$

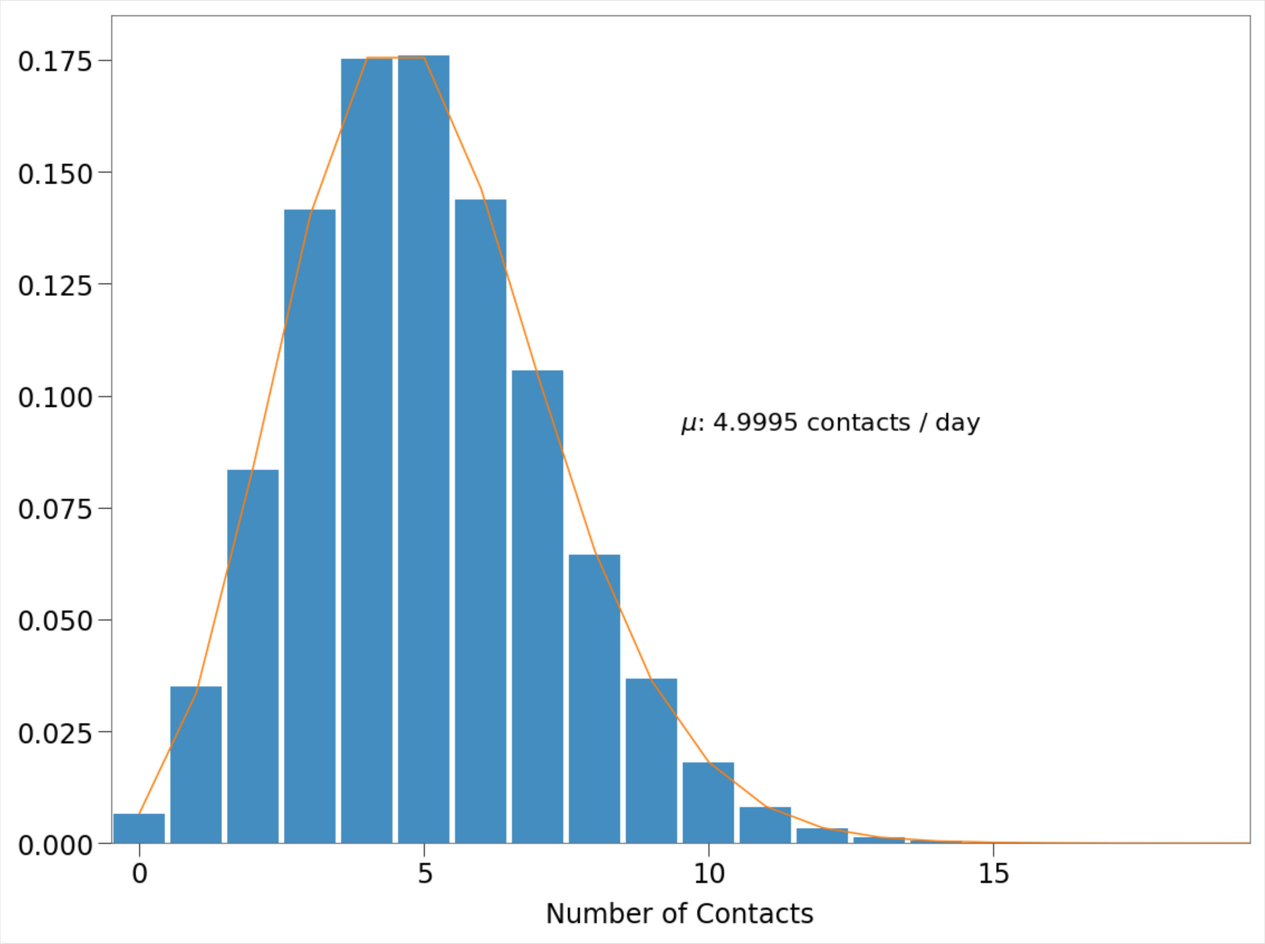
and we know the mean of a binomial distribution is the product $n * p$

$$\mu = np = k/N = density$$

We can see that relationship below, where an environment with density of 5 generates a $\overline{k}$ of 5.

[85]:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt

density = 5
n_dots = 10000
n_locs = n_dots // density

# binom always returns n + 1
p = 1 / n_locs
n = n_dots - 1
x = np.arange(n + 1)

kontact = st.binom(n, p)

fig, ax = plt.subplots(1,1, figsize=(16,12))

counts, bins, _ = plt.hist(
    np.random.binomial(n, p, 100000),
    bins=x, density=True, rwidth=.9
)

ax.plot(x+.5, kontact.pmf(x))

ax.set_xticks([i+.5 for i in range(20) if not i % 5])
ax.set_xticklabels([i for i in range(20) if not i % 5])
ax.set_xlabel('Number of Contacts')

plt.text(.5, .5, f'$\mu$: {n*p} contacts / day', transform=ax.transAxes)

plt.xlim(0,20)

plt.gcf().set_facecolor('white') # Or any color
plt.tight_layout()
plt.savefig('imgs/binom.png')
```



### 2.2.2 Likelihood of Transmission

With the expected contact rate known, probability of transmission under the SIR model is found as:

$\tau = \frac{R_0}{* \overline{c}}$(2.19)

So, again, If a susceptible subject has contact with an infected at the same location, at any time, its probability of infection is:

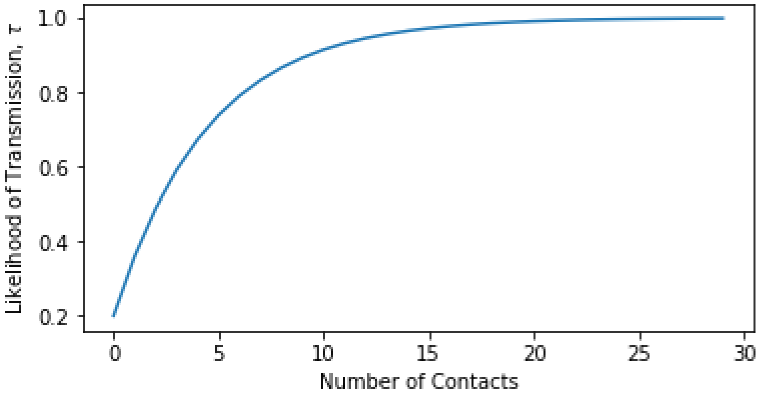
$\tau_i = \tau * T_{inf,i} * T_{sus,i} * T_{loc,i}$(2.20)

If a susceptible comes in contact with multiple infected, we assume that this results in multiple contacts that occur in succession. So we must sum all the branches of the probability tree that end in an infection:

$$\sum_{i=1}^{N}(1-\tau_i)^n * \tau_i(2.21)$$

$$\begin{aligned}\textit{where: } \tau_i &= \textit{likelihood of transmission for contact i}\\ N &= \textit{number of infected dots}(2.22)\end{aligned}$$

This ensures that the likelihood of transmission is asymptotic to 1, as follows:



## 2.3 Dynamic Transmission Risk

The SIR model makes several assumptions that make for simpler math, but that do not map well onto reality.

In particular, SIR assumes constant transmission risk,

$$\tau$$

, when in reality, we know that the infectiousness of an individual changes over time as a function of viral load. It takes time for the virus to accumulate in the subject and, then, in turn it takes time for the subject to dimish the virus via its immune response.

Generally, the greater the viral load, the greater the transmission risk. And so the likelihood of transmission should follow a similar pattern as the viral load (or vice versa). This New York Times piece has a nice visualisation of this concept for sar-cov-2.

There are several techinques available for incorporating viral load in a viral spread model including serial interval, explored here and generation time, explored here. These techniques, however, again tend to ignore $\bar{c}$ and focus on $\beta$.

### 2.3.1 Hutch Model

A paper from a team at the Fred Hutchinson Cancer Research Center, however, maps transmission risk directly onto an estimated viral load curve and infectiousness factor and optimizes it at a specfic contact rate and contact variance (hence forth known as the Hutch model).

First, we will show how the dynamic transmission risk curve is derived. The goal is to produce an array of non-zero probabilites reflecting the likelihood of transmission of virus from one infected to a susceptible in a single contact, based on the known viral load characteristics of the virus.

The Fred model combines quantities of infectiousness and viral load. Viral load is found via a system of 6 differential equations as follows:

$$\begin{aligned}\frac{ds}{dt} &= -\beta v s \\ \frac{di}{dt} = \beta v s - \delta i^k i - mi\frac{e^r}{e^r+\phi^r} \\ \frac{dv}{dt} &= \pi i - \gamma v \\ \frac{dM_1}{dt} &= \omega i M_1 - q M_1 \\ \frac{dM_2}{dt} &= q(M_1 - M_2) \\ \frac{de}{dt} &= q(M_2 - \delta_e E (2.23)\end{aligned}$$

The variable v is the viral load.

With viral load known, the transmission risk, $\tau$, can be found as:

$$\tau_t = \frac{v(t)^\alpha}{\gamma^\alpha + v(t)^\alpha}(2.24)$$

The paper provides estimates for several of the parameters including $\gamma$, $\alpha$, etc. The system can be described and solved using the `odeint` method in `scipy`.

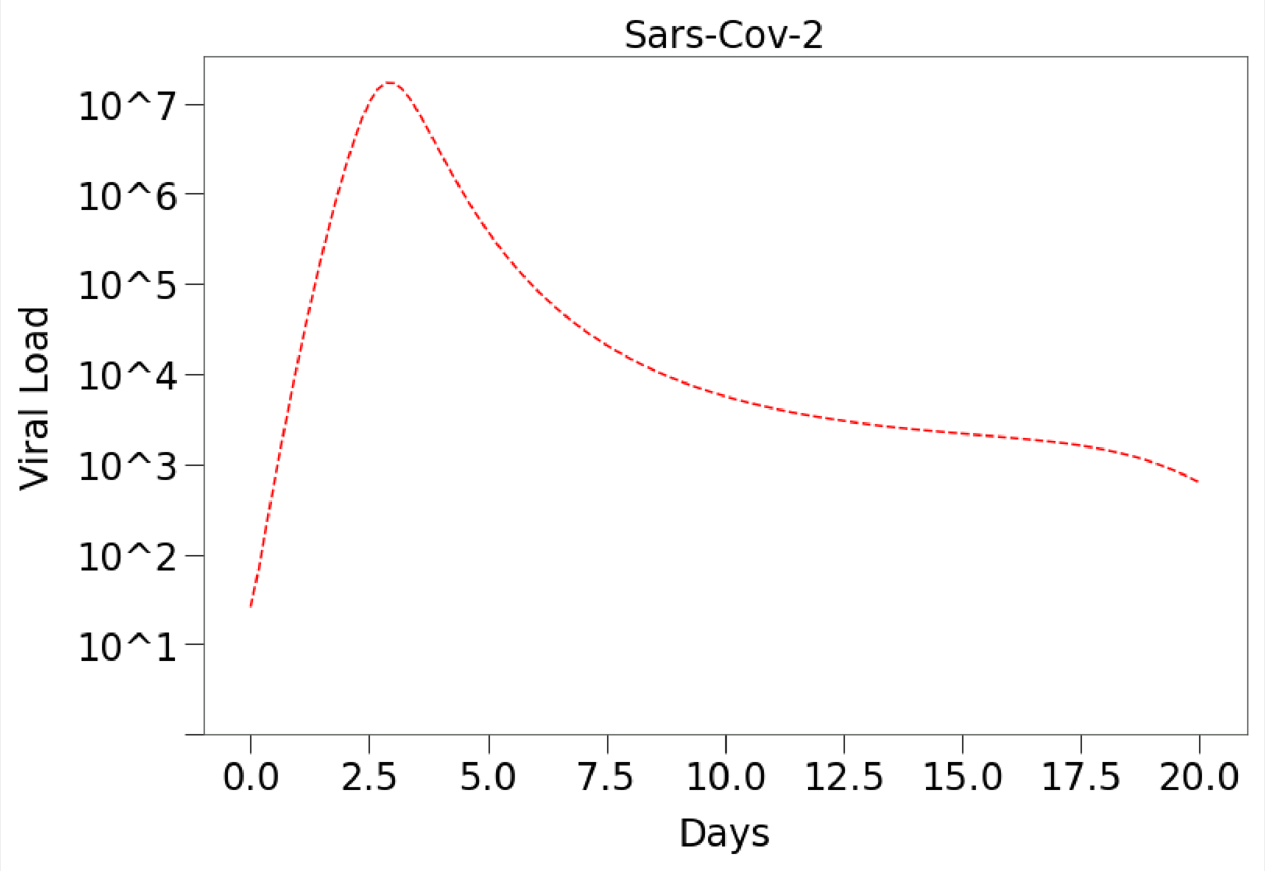
[65]:
```
import math
import numpy as np
from scipy.integrate import odeint

def inf_rate(beta,v,s):
    return beta*v*s

def sfunc(beta, v, s):
    return -inf_rate(beta, v, s)

def vfunc(pi,i,c,v):
    return pi*i - c*v
```

(continues on next page)

(continued from previous page)

```
def ifunc(beta, v, s, delta, i, k, m, e, r, phi):
    inf_r = inf_rate(beta,v,s)
    dens_rate = delta*(i**k)
    acq_res = (m * e**r) / (e**r + phi**r)
    return inf_r - dens_rate*i - acq_res*i

def m1func(omega, i, m1, q):
    return omega*i*m1 - q*m1

def m2func(q, m1, m2):
    return q * (m1-m2)

def efunc(q, m2, dE, e):
    return q*m2 - dE*e

def model(z,t):
    beta=10**-7.23
    k=0.08
    delta= 3.13
    pi=10**2.59
    m=3.21
    omega=10**-4.55
    r=10
    dE=1
    q=2.4*10**-5
    c=15

    s, i, v, m1, m2, e = z[0], z[1], z[2], z[3], z[4], z[5]

    dsdt = sfunc(beta, v, s)
    didt = ifunc(beta, v, s, delta, i, k, m, e, r, phi)
    dvdt = vfunc(pi,i,c,v)
    dm1dt = m1func(omega, i, m1, q)
    dm2dt = m2func(q, m1, m2)
    dedt = efunc(q, m2, dE, e)

    dzdt = [dsdt,didt,dvdt,dm1dt,dm2dt,dedt]
    return dzdt

# FROM PRIOR RESEARCH
pi=10**2.59
c = 15
S0=10**7
I0=1
V0=pi*I0/c
M10=1
M20=0
E0=0
phi=100
z0 = [S0, I0, V0, M10, M20, E0]

t = np.linspace(0,20,30*4)

z = odeint(model,z0,t)
v= z[:, 2]
```

This results in the curve below, which shows the level of virus present in an infected person over time.



We can see that sars-cov-2 viral load can have a long tail, however, the Hutch paper showed that the amount of virus present during the tail is unlikely to result in high transmission, as per below.

Transmission risk is derived from the viral load above as well as two properties of the infectiousness of the subjects in the contact, $\alpha$ and $\gamma$ (see the paper for more details).

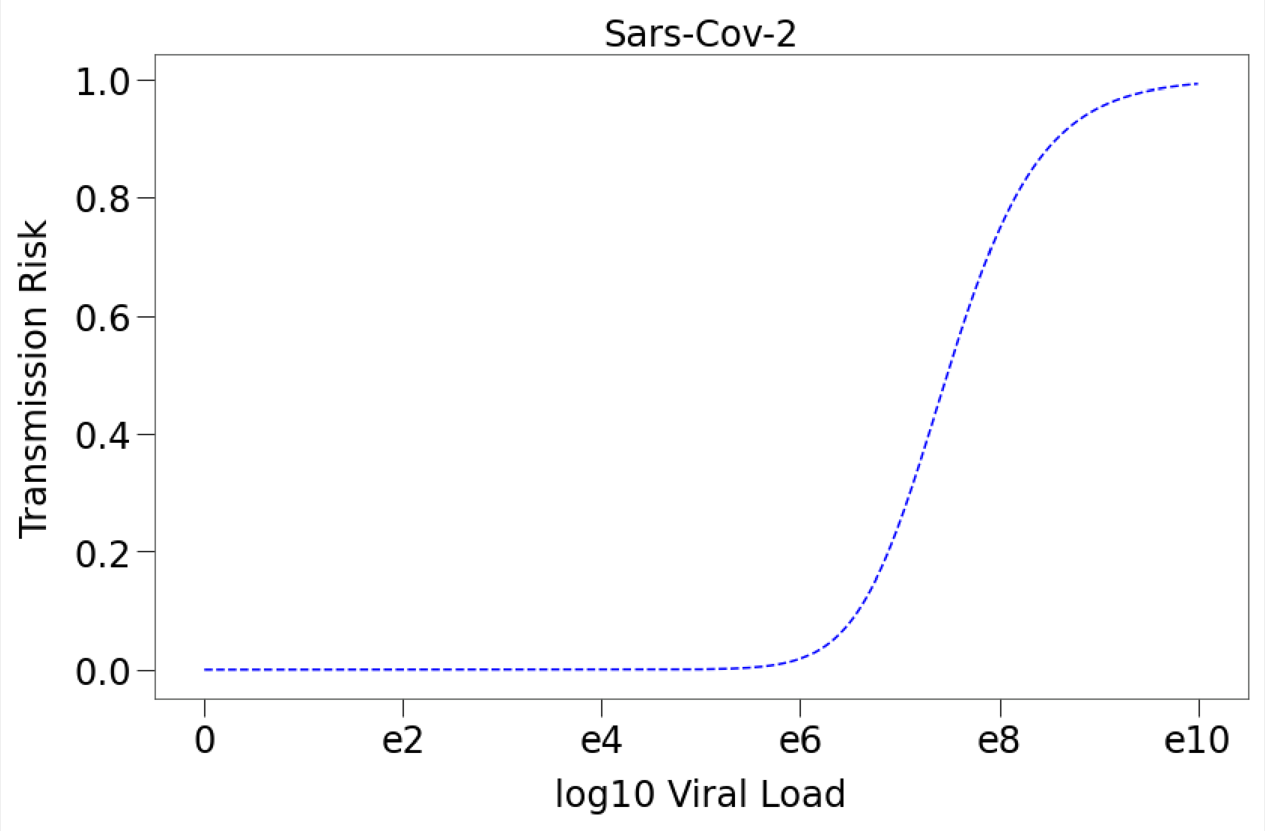
The paper estimated $\alpha = 0.8$ and $\gamma = 10^7$.

[67]:

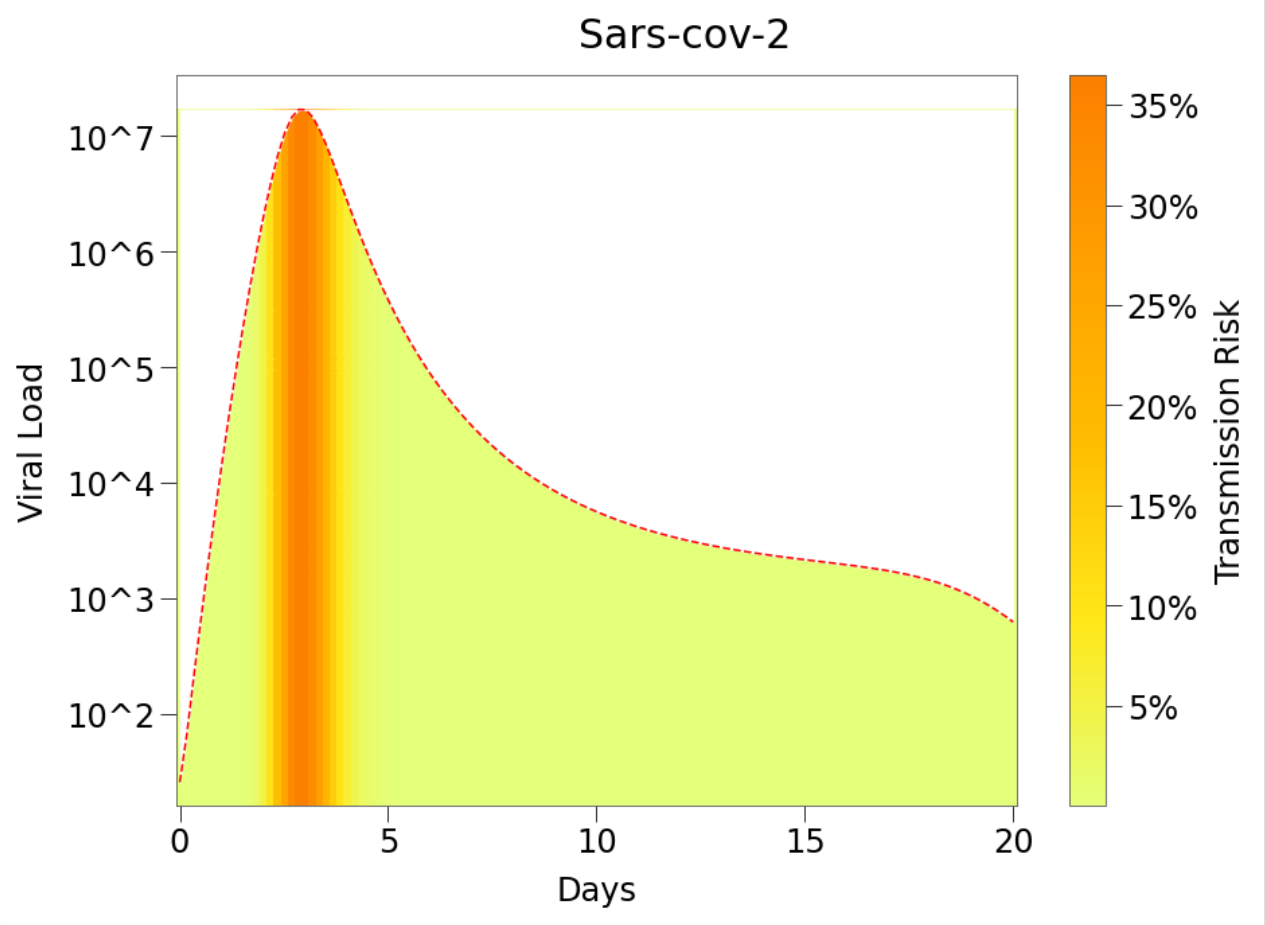
```
def infness(v, alpha, gamma):
    num = v**alpha
    den = gamma**alpha + v**alpha
    return num/den

def taufunc(v, alpha, gamma):
    return infness(v, alpha, gamma)**2

alpha = 0.8
gamma = 10**7

vlin = np.linspace(1, 10**10, 10**5)
tmr = taufunc(vlin, alpha, gamma)
```

`tmr` is the transmission risk curve that we will utilize in our simulations. It is a 1d array with each element representing the likelihood of transmission of the virus at that point in the infection's life cycle.

We can see from the plot below, that transmission risk only increases materially at exponentially higher viral loads:

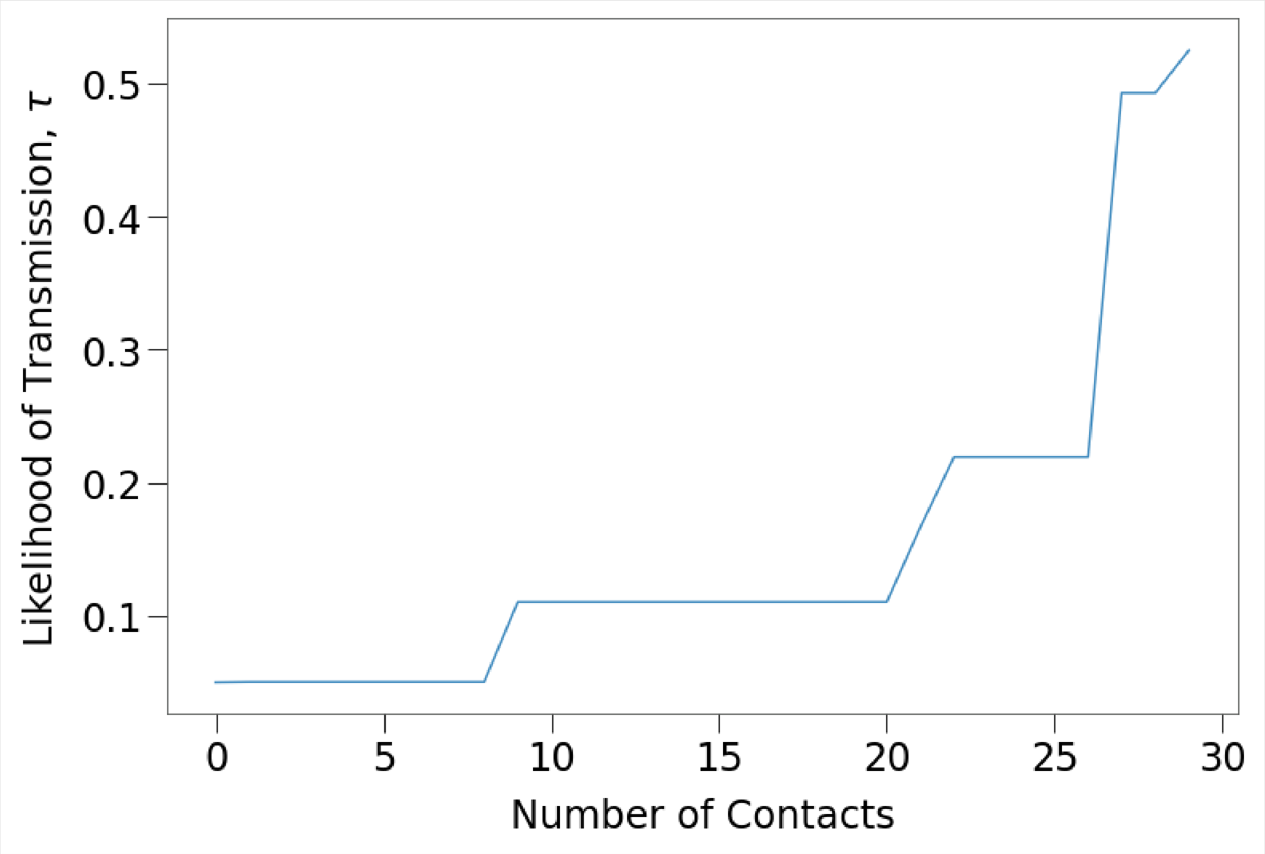


We can further combine Chart 1 and Chart 2 above to show that transmission of sars-cov-2 is likely only during a very narrow range in the early stage of infection.



Note that there is only a significant risk of transmission of the virus for during the first few days of the infection period (shown as the more orange color under the curve).

If we assume random $\tau$ values from `tmr` curve above for 30 different infected dots, the likelihood of transmission to a susceptible as a function of the number of dots at the same location, scales as follows:



## 2.4 Other Forms of Heterogeneity

**RKnot** seeks to address the shortfalls in $R_0$ models by allowing the user to introduce customized, heterogeneous populations across several axis including:

- **Fatality Rate**
- **Population Density**
- **Movement** - frequency and distance of location changes in space according to different probability distributions.
- **Events** - in the real world, people do not move and interact according to smooth probability functions. In fact, they typically have a small subset of movements that are huge outliers from any distribution. These are the professional sporting events, vacation trips, church functions, house parties, etc. that are scheduled and often times recurring. Thankfully, they are more often than not deterministic, which allows us to incorporate them in a simulation.
- **Susceptibility** - segments of population can be made immune (without requiring vaccination) to mimick phenomenon like possible T-cell immunity.
- **Subject Transmission Factor**, $T_i$: $R_0$ assumes that all contacts have the same transmission risk, $\tau$ (subject to the viral load at the time of the interaction). **RKnot** introduces a unitless Transmission Factor, $T$, for each subject at each contact that can modulate $\tau$. This can be used to mimick social distancing or mask wearing or different socio-cultural norms that may impact spread (i.e. east Asian bows versus southern European double-kisses).

*Still To Be Incorporated*

- **Location Transmission Factor**, $T_{\text{xy}}$: similar to $T_i$ above, we can introduce a transmission factor to specific locations that might result in higher or lower likelihood of spread. This could be used to simulate certain work environments (like enclosed office spaces or meat-packing plants). It can also be used to mimic seasonality, by changing $T_{xy}$ over time to account for, say, more time outdoors in temperate seasons.
- **Testing and Isolation** - with the heightened awareness of a pandemic, individuals in population are more likely to self-isolate or quarantine themselves upon sympton onset, thereby helping to reduce spread.

## 2.5 The Average Contact

Currently, **RKnot** assumes that each and every contact is an *Average Contact*.

The average contact is a purely theoretical interaction that would result in about an average likelihood of transmission relative to all other possible interactions. It is not influenced by external factors such as the demographics of the subjects, the properties of the location, etc. Thus, the $\tau$ of an Average Contact is a fundamental property of the virus.

I like to think of the Average Contact as the *Elevator Case*, i.e.:

- Two people on an elevator, standing three feet apart, having a conversation for several minutes before one person exits. No masks nor other conscious social distancing, but no particularly reckless behaviour either.

Every other conceivable interaction can now be scaled relative to the Elevator Case on a continuum of higher or lower probability of transmission using transmission factors, $T$. For instance:

- two college students pressed closely together on a concert floor and yelling at the band on stage would have $T >>> 1x$
- two people standing in a open field, 6 feet apart with masks on exchanging limited pleasantries would have $T <<< 1x$

---

## 2.6 References

- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3782273/
- https://www.bmj.com/content/370/bmj.m3563
- https://www.medrxiv.org/content/10.1101/2020.06.28.20142190v1
- https://fivethirtyeight.com/features/without-a-vaccine-herd-immunity-wont-save-us/
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3935673/
- https://web.stanford.edu/~jhj1/teachingdocs/Jones-on-R0.pdf
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1804098/
- https://www.ijidonline.com/article/S1201-9712%2820%2930119-3/pdf
- https://www.nytimes.com/interactive/2020/10/02/science/charting-a-coronavirus-infection.html?s=03
- https://www.ijidonline.com/article/S1201-9712%2820%2930119-3/pdf
- https://www.medrxiv.org/content/10.1101/2020.03.08.20032946v1.full.pdf

# CHAPTER 3

## Key Concepts

RKnot builds a simulation across four dimensions of global properties:

- **Time**
  - The fundamental unit of time is a `tick`.
  - Each iteration of the simulation is one tick. During a tick, the following occurs:
    - subjects can move to new locatioccns
    - subjects can contact other subjects
    - attributes of the subjects can change
  - Many of the fundamental properties of a virus are measured in days. RKnot translates daily inputs into ticks.
  - Currently, only one `tick` per day is supported. The goal is to support any number of ticks during a day.
- **Space**
  - Subjects interact in an two-dimensional environment called the Grid.
  - The Grid *must* be a square. The Grid size can be passed manually or it can be determined automatically for a specified density level.
  - Each pair of xy coordinates in the Grid is a location.
  - A *contact* occurs when an infected subject and a susceptible subject occupy the same location at the same tick.
  - There is no limit to the number of subjects that can occupy a single location at the same time.
  - Subjects move through the Grid according to user-specified `mover` *functions*. These functions typically incorporate a degree of randomness.
  - A subject can also move by attending an *Event*.
  - Portions of the Grid may be restricted by *Boxes and/or Gates*.
- **Subjects**

  - subjects (also referred to as “dots”) are the analog of people in the simulation.
  - subjects carry many attributes through the life of the simulation that are updated and changed as required *see Dot Matrix*).

- **Virus**
  - the user may pass several characteristics fundamental to the simulated virus. RKnot may infer others. Virus characteristics include:
    - $R_0$
    - Duration of Infection
    - Transmission Risk
    - Duration of Immunity
    - Infection Fatality Rate

## 3.1 The Sim

The `Sim` object is the user interface for the RKnot simulation package and acts as a thin wrapper for the Server and Worker classes of **Ray** actors that form the core of a simulation.

A Sim object is instantiated with pre-defined characteristics of the space, the subjects, and the virus.

For demonstration purposes, a quick default simulation can be run by simply providing a few parameters.

```
from rknot import Sim, Chart

params = {'square': 4, 'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365}
group = {'n': 2, 'n_inf': 1}
sim = Sim(groups=group, **params)
```

`run` is the main method of the `Sim` object. `run` iterates through each `tick` in the simulation. Currently, one day == one tick.

```
sim.run()
```

`sim.run()` does not return any values, but it does update various attributes of the `sim` object. After calling `run`, you can then pass `sim` to the `Chart` object, which will generate an animation of the simulation across time.

```
chart = Chart(sim, dotsize=2000, interval=200, show_intro=False, use_init_func=False)
chart.to_html5_video()
```

alternative text

The animation is split in 3 sections: * **Interactions** * the visual representation of subjects in the Grid. Each marker is a subject and each cross-section of gridlines is a point (for larger grids the lines are removed). * **Details** * provides several on-the-run statistics including Effective Reproduction Number, total fatalities, and fatalities by group. * **Infections** * shows the change in infection level over each day, showing both current infection level and total penetration (“Ever” in the legend)

The animation is built on AxesSubPlot components that can be arranged in any fashion desired, including a handful of preset layouts. *see Chart for more details*.

As per the animation above, the default simulation is of a single infected subject, moving across a 4x4 two-dimensional space according to the `equal` mover function. The subject is equally likely to move to any location in the Grid on any tick.

## 3.2 Subjects

### 3.2.1 Dots

Dots are subjects/people in the simulation space. A subject has many attributes that are adjusted over time, including:

- which Group it belongs to
- if it is alive
- if it is infected
- if it is susceptible
- its location
- any restricted areas that apply to it (*see Boxes and Gates*)
- if infected, when it will recover (or when it will succumb)
- if recovered, when it will again become susceptible
- its fatality rate
- its `mover` function

*see Dot Matrix* for a more fulsome discussion.

### 3.2.2 Groups

Dots are the fundamental subjects of the simulation, but dots can **only** be created via a Group object.

To create our group objects, we can pass a list of dictionaries to the `groups` parameter of Sim. The dictionaries correspond to the attributes of the group, which in turn correspond to the attributes of its constituent dots at initiation.

To create a group, you need only provide two parameters:

- `n`, population size of the group at initiation
- `n_inf`, number of infected subjects in the group at initiation

*If only one group is being provided, you can pass a dictionary. With multiple groups, pass an iterable of dicts.*

```
params = {'square': 10, 'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365}
group = dict(n=2, n_inf=1)

sim = Sim(groups=group, **params)
sim.run()

chart = Chart(
    sim, figsize=(16,8), dotsize=2000, interval=200,
    layout='dots_only', show_intro=False
)
chart.to_html5_video()
```

Below we see this structure creates a 10x10 Grid with two subjects, only one of which is infected at the outset.

alternative text

There are many other parameters and customizations that can provided:

- `name`
  - if not provided, Sim will create one
- `box` & `box_is_gate`
  - see *Boxes and Gates*
- `mover`
  - function used to dictate a dot's movement (*see Mover Functions*)
- `tmf`
  - *transmission factor*, $T$, applied to each of the dots interactions.
  - default: 1
- `susf`
  - *susceptiblity factor*; the fraction of subjects in a group that will be made susceptible to the virus at initiation
  - the inverse of `susf` $(1/susf)$ is the number of subjects in a group that already have immunity.
- `ifr`
  - *infection fatality rate*; or the likelihood that an infection will be fatal

These can again be passed as a dictionary of a single group:

```
group = dict(
    name='main', n=2, n_inf=1, mover='equal',
    tmf=1.25, susf=.75, ifr=0.005
)

sim = Sim(groups=group, **params)
sim.run()

chart = Chart(
    sim, figsize=(16,8), dotsize=2000, interval=200,
    layout='dots_only', show_intro=False
)
chart.to_html5_video()
```

alternative text

Or as an iterable of dictionaries. Each group is assigned a unique marker in the animation.

```
group1 = dict(name='1', n=1, n_inf=1, mover='local', tmf=1.25, susf=.75, ifr=0.005)
group2 = dict(name='2', n=1, n_inf=0, mover='equal', tmf=0.75, susf=0.95, ifr=0.05)
group3 = dict(name='3', n=1, n_inf=0, mover='equal', tmf=0.25, susf=0.5, ifr=0.4)
groups = [group1, group2, group3]

sim = Sim(groups=groups, **params)
sim.run()

chart = Chart(
    sim, figsize=(16,8), dotsize=2000, interval=200,
```

(continues on next page)

(continued from previous page)

```
    layout='dots_only', show_intro=False
)
chart.to_html5_video()
```

alternative text

## 3.3 The Grid

All interactions in an RKnot simulation take place inside the Grid. The grid is a `Grid` object, which in turn is a sub-classed numpy array with some additional features.

The Grid size can be determined by passing the `square` or `density` parameters. Each `density` accepts either a `str` or a `float`. The `float` value is a specific desired subject per location and a `str` must be on of the three categories below.

Available `str` values for `density` and their corresponding densities are:

low: 0.2

med: 1

high: 10

If we set `density=med`, the Grid will be set such that the density is 1 subject per location. For a group of 100 subjects, that will result in a 10x10 grid. We can see these attributes by passing `details=True`.

```
[9]: params = {'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365}
     group = dict(name='main', n=100, n_inf=1)

     sim = Sim(groups=group, density='med', details=True, **params)
```

```
HBox(children=(FloatProgress(value=0.0, layout=Layout(flex='2'), max=93.0),␣
↪HTML(value='')), layout=Layout(dis...
```

```
---------------------------------------------------------------------------------
|                                  SIM DETAILS                                  |
|-------------------------------------------------------------------------------|
|            Boundary|       [ 1 10   1 10]|           Locations|                 100|
|--------------------|--------------------|--------------------|--------------------|
|          Population|                 100|             Density|                 1.0|
|--------------------|--------------------|--------------------|--------------------|
|        Contact Rate|                1.01|                    |                    |
|--------------------|--------------------|--------------------|--------------------|
```

```
sim.run()

chart = Chart(
    sim, figsize=(16,8), layout='dots_only',
    show_intro=False, use_init_func=False
)
chart.to_html5_video()
```

alternative text

For smaller populations, the density level can only be approximated. RKnot defaults to rounding up to the nearest square value.

You can also pass a float value to `density` in order to create a specified density. Here, we set `density=3.5`

```
[13]: group1 = dict(name='1', n=1000, n_inf=1)
group2 = dict(name='2', n=20, n_inf=20)
groups = [group1, group2]

sim = Sim(groups=groups, density=3.5, details=True, **params)
```

```
HBox(children=(FloatProgress(value=0.0, layout=Layout(flex='2'), max=93.0),␣
↪HTML(value='')), layout=Layout(dis...
```

```
---------------------------------------------------------------------------------
|                                  SIM DETAILS                                  |
|-------------------------------------------------------------------------------|
|           Boundary|      [ 1 18   1 18]|           Locations|                324|
|-------------------|-------------------|-------------------|-------------------|
|         Population|              1,020|             Density|               3.15|
|-------------------|-------------------|-------------------|-------------------|
|       Contact Rate|               3.16|                   |                   |
|-------------------|-------------------|-------------------|-------------------|
```

```
sim.run()

chart = Chart(sim, figsize=(16,8), layout='dots_only', show_intro=False)
chart.to_html5_video()
```

alternative text

---

### 3.3.1 Mover Functions

When a subject changes locations, this is called a 'move'. A move is completed during a tick and the movement of a subject on any tick is governed by its mover function. Movers select a location according to a pre-defined probability distribution, so the general movement pattern of a dot can be pre-determined, but any one movement occurs randomly.

There are currently 5 mover functions. Their respective definitions, along with examples of their movement are provided below. A `float` value is also accepted which is used as the `p-value` in a geometric movement pattern.

Equal

The subject is equally likely to move to any location.

alternative text

Local

The subject has a strong bias towards dots only in its immediate vicinity.

alternative text

Traveller

The subject commonly moves to locations far across the Grid.

alternative text

Quarantine

The subject has a strong bias towards not moving, with only some movement occuring.

(continues on next page)

(continued from previous page)

alternative text

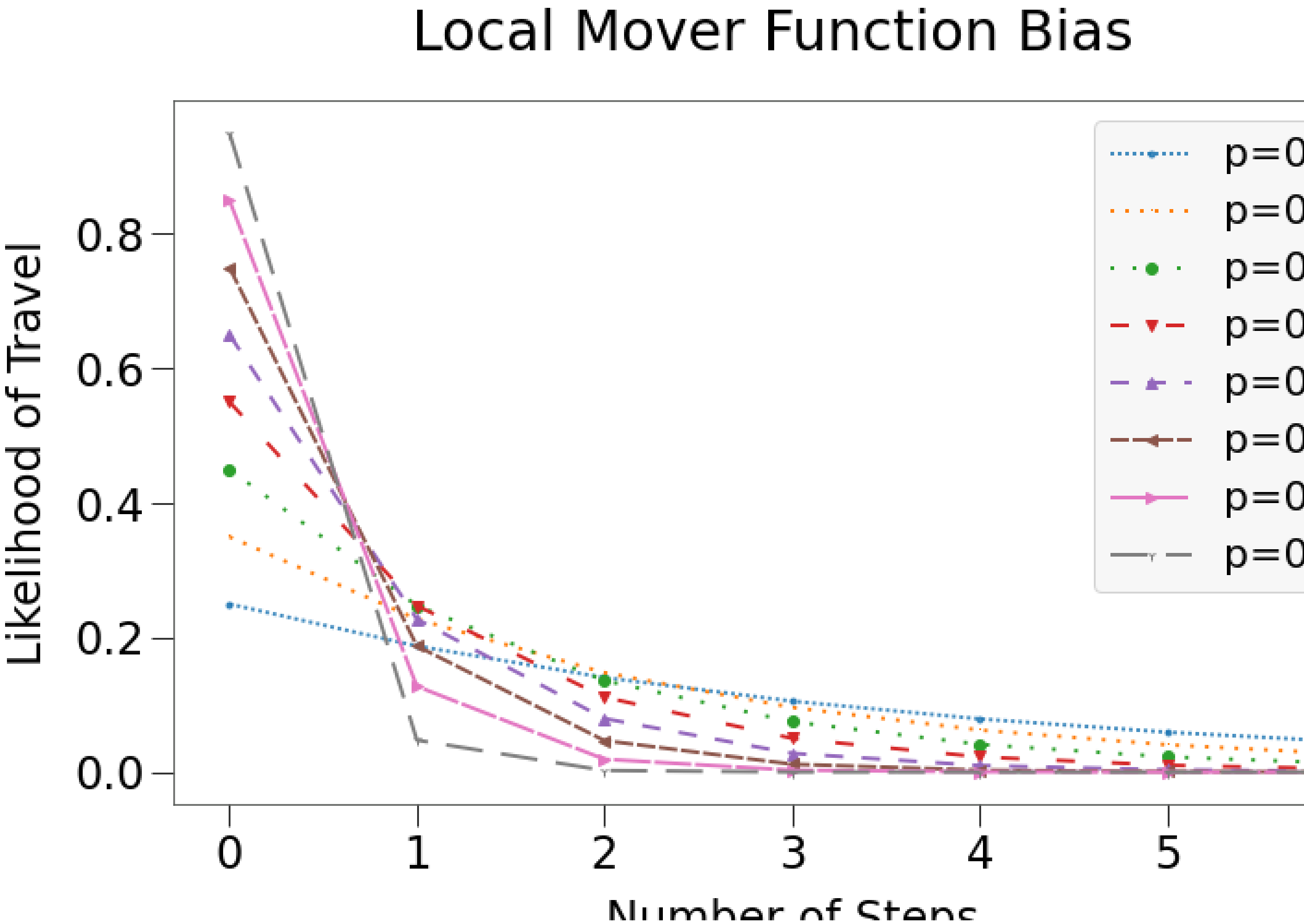
Social

The subject moves mostly within its vicinty but also to other more medium distance locations.

alternative text

In addition to specifying a mover function, the user can also simply specify a `float` value between 0 and 1.

This value corresponds to a `p-value` used in a geometric distrubtion. The relationship between `p-value` and movement is shown below.



The higher the `p-value`, the greater the bias towards shorter moves. Increasing `p-value`, all else equal, should decrease the number of contacts in a sim. This is explored further *here*.

Below we compare the movement patterns of two subjects with very different `p-value`.

```
params = {'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365}

group1 = dict(n=1, n_inf=1, mover=.25)
group2 = dict(n=1, n_inf=1, mover=.95)
groups = [group1, group2]
sim = Sim(groups=groups, square=10, **params)
```

(continues on next page)

(continued from previous page)

```
sim.run()

chart = Chart(sim, figsize=(16,8), layout='dots_only', show_intro=False)
chart.to_html5_video()
```

alternative text

## 3.4 Boxes and Gates

The movement of a subject across the Grid can be restricted by two concepts known as Boxes and Gates. These concepts are designed to mimick certain funcitonal or perceived boundaries between groups, such as international borders or closed-access communities like assisted-living facilities.

The distinction between boxes and gates is simple:

- Subjects *cannot* **exit** Boxes
- Subjects *cannot* **enter** Gates

### 3.4.1 Boxes

A box is a $m * n$ subset of locations within the Grid that a subject(s) cannot leave.

The locations are specified by passing a four element iterable that specifies the coordinates of the "four corners" of the box according to $[x_0, x_1, y_0, y_1]$

So passing:

```
box = [2,6,3,9]
```

creates a box with the four corners:

```
(2,3)   (2,9)    (6,3)   (6,9)
```

and a total of 35 locations.

Currently, a box *can only be specified* by

1. passing the `box` parameter as group keyword
2. by passing a ` vbox <#VBoxes>‘__.

Every dot in the group can only move within the `box`, regardless of the size of the Grid.

```
group1 = dict(name='1', n=2, n_inf=1, box=[1,3,2,4])

sim = Sim(groups=group1, square=10, **params)
sim.run()

chart = Chart(
    sim, figsize=(16,8), dotsize=2000,
    layout='dots_only', show_intro=False, use_init_func=False
)
chart.to_html5_video()
```

alternative text

A group can only have one box and each group can have its own box.

```
group1 = dict(name='1', n=2, n_inf=1, box=[1,3,2,4])
group2 = dict(name='2', n=2, n_inf=0, box=[6,9,6,10])
groups = [group1, group2]

sim = Sim(groups=groups, square=10, **params)
sim.run()

chart = Chart(
    sim, figsize=(16,8), dotsize=2000,
    layout='dots_only', show_intro=False, use_init_func=False
)
chart.to_html5_video()
```

alternative text

Remember that a box only restricts the subjects in that group from *leaving* the space. Other dots not assigned to that box can move into the space without restriction.

```
group1 = dict(name='1', n=5, n_inf=1, box=[1,3,1,3])
group2 = dict(name='2', n=5, n_inf=0)
groups = [group1, group2]

sim = Sim(groups=groups, square=10, **params)
sim.run()

chart = Chart(sim, figsize=(16,8), dotsize=2000, layout='dots_only', show_intro=False)
chart.to_html5_video()
```

alternative text

### 3.4.2 Gates

Gates are the inverse of boxes. A gate is an area that subjects cannot enter.

Gates are a Gate object, which are a subclass of the Box class (in turn a subclass of ndarray), and they are created via the same 4 element iterable. For now, a gate can only be created by passing the `box_is_gated=True` flag as a keyword in a group dictionary, or by specifying a `vbox`.

Using the previous example, we can see that `group2` dots can no longer enter the `group1` box.

```
group1 = dict(name='1', n=5, n_inf=1, box=[1,3,1,3], box_is_gated=True)
group2 = dict(name='2', n=5, n_inf=0)
groups = [group1, group2]

sim = Sim(groups=groups, square=10, **params)
sim.run()

chart = Chart(
    sim, figsize=(16,8), dotsize=2000, layout='dots_only', show_intro=False,
    use_init_func=False
)
chart.to_html5_video()
```

alternative text

This structure allows for intricate movement patterns. We show isolated groups below.

We will also provide the `show_restricted=True` flag, which will outline the boxes and gates for us. It will also the label the restricted area with the name of the group used to form the it.

```
group1 = dict(name='1', n=50, n_inf=5, box=[1,5,1,20], box_is_gated=True)
group2 = dict(name='2', n=50, n_inf=5, box=[6,25,3,10], box_is_gated=True)
group3 = dict(name='3', n=50, n_inf=5, box=[10,21,16,22], box_is_gated=True)
group4 = dict(name='4', n=50, n_inf=5, box=[2,15,23,25], box_is_gated=True)
groups = [group1, group2, group3, group4]

sim = Sim(groups=groups, square=25, **params)
sim.run()

chart = Chart(sim,
    figsize=(16,8), layout='dots_only', show_intro=False, use_init_func=False,
    show_restricted=True,
)
chart.to_html5_video()
```

alternative text

And here some isolated and some free moving.

```
group1 = dict(name='1', n=50, n_inf=5, box=[1,5,1,5], box_is_gated=True)
group2 = dict(name='2', n=50, n_inf=5, box=[14,19,14,19], box_is_gated=True)
group3 = dict(name='4', n=10, n_inf=5)
group4 = dict(name='4', n=10, n_inf=5)
groups = [group1, group2, group3, group4]

sim = Sim(groups=groups, square=25, **params)
sim.run()

chart = Chart(
    sim, figsize=(16,8), layout='dots_only',
      show_intro=False, use_init_func=False
)
chart.to_html5_video()
```

alternative text

### 3.4.3 VBoxes

A VBox is a vacant area of the Grid, meaning there are no subjects inside the box at the initiation of the Sim and that no subjects can enter the VBox except via Travel events.

VBoxes can be used to customize contact patterns, as done in the *Dynamic Transmission Risk* simulations. They can also be used to mimick areas that people typically only visit, rather than reside in, such as hospitals, sports arenas, office buildings, etc.

VBoxes are simply box objects and can be created by passing the `vboxes` parameter to `Sim`. VBoxes are always setup with a corresponding gate.

***IMPORANT***: A VBox is not included in the density calculation of the grid size.

If we pass an integer, `Sim` will create a VBox with the value corresponding to the number of locations in the VBox. The VBox will be placed in the top-left corner of the Grid.

```
from rknot import Sim

vbox = 4

groups = [
    dict(n=10, n_inf=1, mover='social'),
    dict(n=10, n_inf=1, mover='local')
]
params = {'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365, 'vboxes': vbox}
sim = Sim(groups=groups, **params)
sim.run(dotlog=True)

chart = Chart(
    sim, figsize=(16,8), layout='dots_only',
      show_intro=False, use_init_func=False
)
chart.to_html5_video()
```

alternative text

We can also pass a boundary as a 4 item iterable.

```
from rknot import Sim

vbox = [1,3,1,3]

groups = [
    dict(n=10, n_inf=1, mover='social'),
    dict(n=10, n_inf=1, mover='local')
]
params = {'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365, 'vboxes': vbox}
sim = Sim(groups=groups, **params)
sim.run(dotlog=True)

chart = Chart(
    sim, figsize=(16,8), layout='dots_only',
      show_intro=False, use_init_func=False
)
chart.to_html5_video()
```

alternative text

We can pass a dictionary and include the `label` keyword to indicate a name for the VBox.

We've included a couple `Travel` objects to show how the VBox can be accessed. Simply assign the index of the `vbox` as a parameter to `Travel` and the `Sim` will determine the location automatically.

```
from rknot import Sim
from rknot.events import Travel

vbox = {'box': [1,3,1,3], 'label': 'Hospital'}

groups = [
    dict(n=10, n_inf=1, mover='social'),
    dict(n=10, n_inf=1, mover='local')
]
params = {'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365, 'vboxes': vbox}
events = [
```

(continues on next page)

(continued from previous page)

```
    Travel(
        name='vbox_event', start_tick=3, recurring=3,
        groups=[0,1], capacity=1, vbox=0,
    )
]
sim = Sim(groups=groups, events=events, **params)
sim.run(dotlog=True)

chart = Chart(
    sim, figsize=(16,8), layout='dots_only',
      show_intro=False, use_init_func=False
)
chart.to_html5_video()
```

alternative text

Finally, we can pass multiple VBoxes as a list of dictionaries.

```
from rknot import Sim

vboxes = [
    {'box': [1,3,1,3], 'label': 'Hospital'},
    {'box': [5,7,1,3], 'label': 'Arena'}
]

groups = [
    dict(n=20, n_inf=1, mover='social'),
    dict(n=20, n_inf=1, mover='local')
]
params = {'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365, 'vboxes': vbox}
sim = Sim(groups=groups, **params)
sim.run(dotlog=True)

chart = Chart(
    sim, figsize=(16,8), layout='dots_only',
      show_intro=False, use_init_func=False
)
chart.to_html5_video()
```

alternative text

## 3.5 Events

Events impact the attributes of subjects over the course of the simulation.

Events are utilized to better simulate real-world behavior.

For instance, people do not move in consistent, prescribed ways. They move in regular ways most of the time with contacts that are well defined, but sometimes they attend events (perhaps periodically or uniquely) that are not governed by their regular movement patterns.

### 3.5.1 Event

An `Event` is an event that occurs at a particular location.

An Event accepts the following parameters:

- xy, the xy coordinates of the location
- start_tick, the tick when the event begins
- groups, an iterable of group ids that are eligible for the event
- capacity, the number of subjects that should attend
- recurring, how often the event recurs (i.e. every *nth* tick); if set to 0, the event does *not* recur

When an Event concludes, the subject returns to its home location as specified in the dot matrix.

To schedule an event, you must pass a list of event objects to the events parameter.

To begin with, we'll create a single Event object, called show, that occurs once on day 5.

```
from rknot.events import Event

params = {'square': 10, 'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365}
group1 = dict(name='1', n=10, n_inf=5)

show = Event(name='show', xy=(5,5), start_tick=5, groups=[0], capacity=10)
events = [show]

sim = Sim(groups=group1, events=events, **params)
sim.run(dotlog=True)

chart = Chart(sim, figsize=(16,8), layout='dots_only', show_intro=False)
chart.to_html5_video()
```

alternative text

If you watch closely, you'll see on Day 5 that all the dots seemingly disappear, save for one, at location (5,5).

In fact, all 10 dots are *actually at that location at the same time*.

We can confirm this by inspecting the *Dot Matrix* on that day via the dotlog attribute.

```
[22]: from rknot.dots import MATRIX_COL_LABELS as ML
      sim.dotlog[4][:, ML['x']:ML['y']+1]
```

```
[22]: array([[5,  5],
             [5,  5],
             [5,  5],
             [5,  5],
             [5,  5],
             [5,  5],
             [5,  5],
             [5,  5],
             [5,  5],
             [5,  5]])
```

We can see the event more clearly if we extend the duration to 10 days. We also significantly reduced the frame rate.

```
show = Event(name='show', xy=(5,5), start_tick=5, groups=[0], capacity=10,␣
↪duration=10)
events = [show]

sim = Sim(groups=group1, events=events, **params)
sim.run()
```

(continues on next page)

(continued from previous page)

```
chart = Chart(
    sim, figsize=(16,8), dotsize=1000, interval=300,
    layout='dots_only', show_intro=False, use_init_func=False
)
chart.to_html5.video()
```

alternative text

Many event objects can be specified at once, in various combinations of groups.

```
group1 = dict(name='1', n=10, n_inf=5)
group2 = dict(name='2', n=10, n_inf=0)

show = Event(name='show', xy=(5,5), start_tick=5, groups=[0,1], capacity=5,␣
↪recurring=30)
game = Event(name='game', xy=(1,1), start_tick=5, groups=[0], capacity=5,␣
↪recurring=14)
church = Event(name='church', xy=(1,1), start_tick=5, groups=[1], capacity=10,␣
↪recurring=7)

groups = [group1, group2]
events = [show, game, church]

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(
    sim, interval=300, dotsize=1000, layout='dots_only', show_intro=False,
    use_init_func=False
)
chart.to_html5_video()
```

alternative text

### 3.5.2 Travel

Travel is a special type of event that allows a subject to enter a gate.

When a dot enters a gate via a Travel object, its box and gate attributes are temporarily adjusted to match those of the groups within the gate. The attributes revert when the event ends (determined by `duration` parameter).

Once inside the gate, the dot(s) are free to interact with other dots normally.

```
from rknot.events import Travel

params = {'square': 10, 'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365}

group1 = dict(name='1', n=1, n_inf=1, box=[1,5,1,5], box_is_gated=True)
group2 = dict(name='2', n=10, n_inf=0, box=[6,10,6,10], box_is_gated=True)

visit = Travel(
    name='visit', xy=[1,1], start_tick=3, groups=[1], capacity=1, duration=5,␣
↪recurring=10
)
```

(continues on next page)

(continued from previous page)

```
groups = [group1, group2]
events = [visit]

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(
    sim, interval=200, dotsize=2000, layout='dots_only', show_intro=False
    use_init_func=False
)
chartto_html5_video()
```

alternative text

Many unique layouturations can be achieved with this structure. Below, the `group1` box will be vacated by the solitary `group1` dot (essentially switching places with a dot from `group2`).

```
from rknot.events import Travel

params = {'square': 10, 'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365}

group1 = dict(name='1', n=1, n_inf=1, box=[1,5,1,5], box_is_gated=True)
group2 = dict(name='2', n=10, n_inf=0, box=[6,10,6,10], box_is_gated=True)

visit2 = Travel(
    name='visit2', xy=[9,9], start_tick=3, groups=[0], capacity=1, duration=5,␣
↪recurring=10
)
visit1 = Travel(
    name='visit1', xy=[1,1], start_tick=3, groups=[1], capacity=1, duration=5,␣
↪recurring=10
)
groups = [group1, group2]
events = [visit2, visit1]

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(
    sim, interval=200, dotsize=2000, layout='dots_only', show_intro=False,
    use_init_func=False
)
chart.to_html5_video()
```

alternative text

### 3.5.3 Quarantine

A quarantine is an event object that makes several changes to a dots state in order to restrict its movement.

When a dot is quarantined,

1. it goes back to its home location (*see Dot Matrix*)
2. its boxes and gates are reset to match its group

3. its mover function is changed to ‘quarantine’

In addition, a Quarantine object will create a additional restriction objects that disallow events during the quarantine (*see Restrictions below*)

```
from rknot.events import Quarantine

params = {'square': 10, 'R0': 2.5, 'days': 50, 'imndur': 365, 'infdur': 365}

group1 = dict(name='1', n=2, n_inf=1, box=[1,5,1,5], box_is_gated=True)
group2 = dict(name='2', n=2, n_inf=0, box=[6,10,6,10], box_is_gated=True)

quar = Quarantine(name='all', start_tick=5, groups=[0,1], duration=30)

groups = [group1, group2]
events = [quar]

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(
    sim, interval=200, dotsize=2000, layout='dots_only', show_intro=False,
    use_init_func=False
)
chart.to_html5_video()
```

alternative text

We can see from above that once in quaratine, the subjects barely move. We can include events in our structure. The events will be restricted during the quarantine period, then will resume when the quarantine ends.

```
params = {'square': 10, 'R0': 2.5, 'days': 100, 'imndur': 365, 'infdur': 365}

group1 = dict(name='1', n=1, n_inf=1, box=[1,5,1,5], box_is_gated=True)
group2 = dict(name='2', n=10, n_inf=0, box=[6,10,6,10], box_is_gated=True)

show = Event(name='show', xy=(6,6), start_tick=5, groups=[1], capacity=5,␣
↪recurring=30)
visit2 = Travel(
    name='visit2', xy=[9,9], start_tick=3, groups=[0], capacity=1, duration=5,␣
↪recurring=10
)
visit1 = Travel(
    name='visit1', xy=[1,1], start_tick=3, groups=[1], capacity=1, duration=5,␣
↪recurring=10
)

groups = [group1, group2]
events = [show, visit2, visit1, quar]

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(
    sim, interval=200, dotsize=2000, layout='dots_only', show_intro=False,
    use_init_func=False,
)
chart.to_html5_video()
```

alternative text

### 3.5.4 Social Distancing

Event to mimick social distancing practices.

Social distancing is assumed to impact the Transmission Factor, $\tau$ of each dot. The core hypothesis is that practices such as maintaining 6-feet of distance or mask wearing don't reduce the number of contacts, but do reduce the likelihood that a contact will result in a new infection (ceterus paribus).

You can see use of this object *here*.

### 3.5.5 Vaccination

TBD

### 3.5.6 Restrictions

A restriction object restricts attendance to events that fall within the specified criteria. Each event has a `restricted` attribute that defaults to `False`. A restriction object filters out events from the event schedule by setting `restricted=True` for each event that satisfies the criteria.

To clarify, a Restriction is *not* an event. Events act on dots. Restrictions act on events.

Restrictions have potential as a versatile tool that can be used to investigate the impact of various government and business policy decisions that impact spread.

The Restriction object has a `criteria` parameter that accepts a `dict`, with keywords related to event object attributes. Acceptable `criteria` keys are currently:

capacity name ticks groups loc_id

The simplest way to restrict an event is by its name:

```
from rknot.events import Restriction

params = {'square': 10, 'R0': 2.5, 'days': 100, 'imndur': 365, 'infdur': 365}

group1 = dict(name='1', n=10, n_inf=1, box=[1,5,1,5], box_is_gated=True)
group2 = dict(name='2', n=10, n_inf=0, box=[6,10,6,10], box_is_gated=True)

show1 = Event(name='show1', xy=(1,1), start_tick=2, groups=[0], capacity=10,␣
↪recurring=2)
show2 = Event(name='show2', xy=(10,10), start_tick=2, groups=[1], capacity=10,␣
↪recurring=2)


criteria = {'name': 'show1'}
res1 = Restriction(name='no_show1', start_tick=10, duration=20, criteria=criteria)

groups = [group1, group2]
```

(continues on next page)

(continued from previous page)

```
events = [show1, show2, res1]

sim = Sim(groups=groups, events=events, **params)
sim.run(dotlog=True)

chart = Chart(
    sim, interval=300, dotsize=2000, layout='dots_only', show_intro=False,
    use_init_func=False,
)
chart.to_html5_video()
```

alternative text

In the above, we can see that every other day both group boxes have events that are attended by all dots in the group.

But on day 10, the `group1` dots no longer converge on location (1,1). Instead, they are spread throughout their box. So `res1` has successfully restricted attendance to `show1`.

Unlike `Quarantine` objects, however, the `group1` dots have not changed their standard movement patterns.

We can restrict multiple events via the other criteria. Next we will restrict events based on their capacity. Events with more than 5 subjects in attendance will be restricted.

```
criteria = {'capacity': 5}
res1 = Restriction(name='cap5', start_tick=10, duration=20, criteria=criteria)

groups = [group1, group2]
events = [show1, show2, res1]

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(
    sim, interval=300, dotsize=2000, layout='dots_only', show_intro=False,
    use_init_func=False,
)
chart.to_html5_video()
```

alternative text

In the above we see that neither of the groups had events from day 10 onward during the restriction period.

Restrictions can be chained together as desired to form a complex and tailored policy recipe for the population of the sim. *See this example*.

## 3.6 Dot Matrix

The dot matrix is essentially RKnot's canonical form of data structure. The matrix is simply a 2D `numpy` array of shape (*n*, 23) with each of the *n* rows representing a dot and each column representing an attribute.

More typical Python objects have been eschewed in favor the Dot Matrix because:

- **RKnot** relies heavily on Ray for parallel processing and Numba for just-in-time compilation and vectorization to improve processing speed.
- `Numpy` arrays have several advantages in **Ray** including rapid serialization and ease of batching.

- **Numba** also integrates well with `numpy`, supporting many of its features and leads to major performance improvements.

The dot matrix is created inside a **Ray** actor at instantiation and is only passed back to the main `Sim` object when the simulation is completed.

It can be accessed via the `dots` attribute. Below is a sample of 4 dots:

```
[33]: sim.dots[:4]
```

```
[33]: array([[  0,    0,    1,    0,    0,    1,    1,   65,    7,    6,   54,    6,    5,
              0,    0,   -1,    0,  100,  650,    0,   -1,  365,  730],
           [  1,    0,    1,    0,    0,    1,    1,   77,    8,    8,   22,    3,    3,
              0,    0,   -1,    0,  100,  650,    0,   -1,  365,  730],
           [  2,    0,    1,    0,    0,    1,    1,   11,    2,    2,   88,    9,    9,
              0,    0,   -1,    0,  100,  650,    0,   -1,  365,  730],
           [  3,    0,    1,    0,    0,    1,    1,   27,    3,    8,   96,   10,    7,
              0,    0,   -1,    0,  100,  650,   42,   -1,  407,  772]])
```

The column attributes have corresponding labels:

```
[11]: from rknot.dots import MATRIX_LABELS
      print (MATRIX_LABELS)
```

```
['id', 'group_id', 'is_alive', 'is_vaxxed', 'is_sus', 'is_inf', 'ever_inf', 'loc_id',
↪'x', 'y', 'home_id', 'homex', 'homey', 'go_home', 'box_id', 'event_id', 'mover',
↪'mover_p', 'tmf', 'ifr', 'inf_tick', 'depart', 'recover', 'relapse']
```

With these labels, the 4 dot matrix above can be shown in a table.

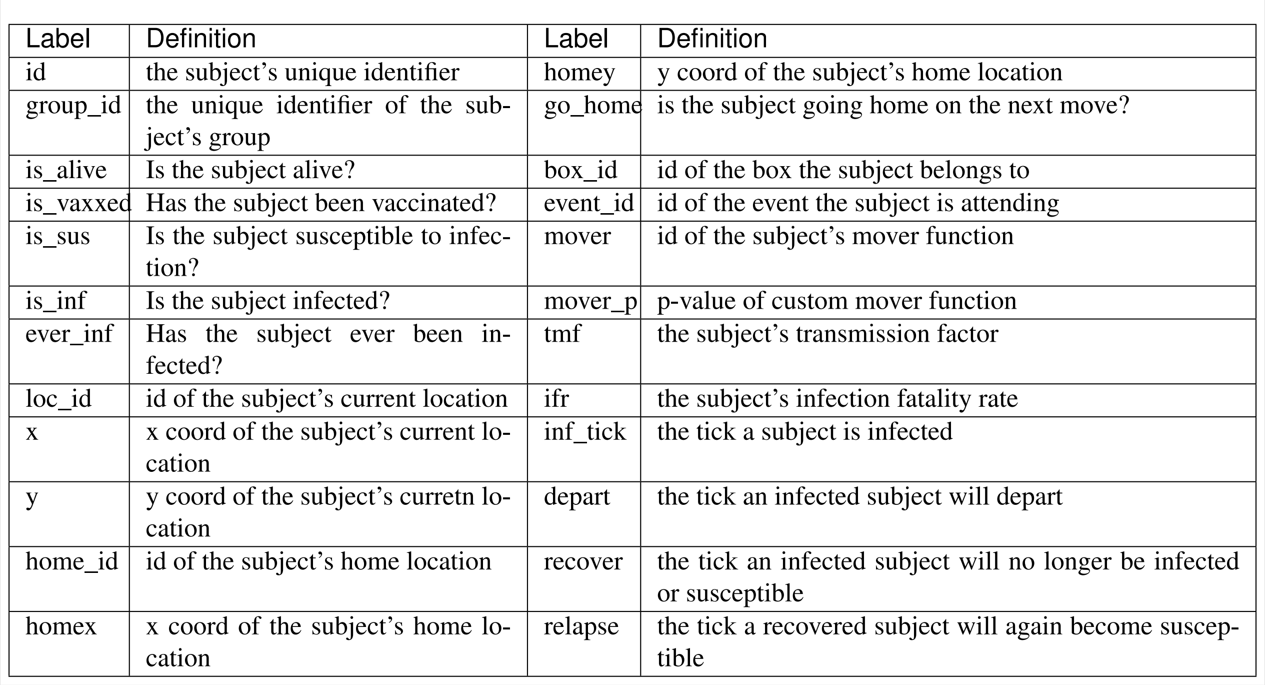
| id | group_id | is_alive | is_vaxxed | is_sus | is_inf | ever_inf | loc_id | x | y | home_id | homex | homey | go_home | box_id | event_id | mover | mover_p | tmf | ifr | inf_tick | depart | recover | relapse |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 46 | 6 | 7 | 36 | 5 | 5 | 0 | 0 | -1 | 4 | -999 | 100 | 0 | -1 | -1 | -1 | -1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 61 | 8 | 6 | 27 | 4 | 4 | 0 | 0 | -1 | 4 | -999 | 100 | 0 | -1 | -1 | -1 | -1 |
| 2 | 0 | 1 | 0 | 1 | 0 | 0 | 52 | 7 | 5 | 58 | 8 | 3 | 0 | 0 | -1 | 4 | -999 | 100 | 0 | -1 | -1 | -1 | -1 |
| 3 | 0 | 1 | 0 | 1 | 0 | 0 | 20 | 3 | 5 | 3 | 1 | 4 | 0 | 0 | -1 | 4 | -999 | 100 | 0 | -1 | -1 | -1 | -1 |
| 4 | 0 | 1 | 0 | 1 | 0 | 0 | 15 | 2 | 8 | 24 | 4 | 1 | 0 | 0 | -1 | 4 | -999 | 100 | 0 | -1 | -1 | -1 | -1 |

There are several data types at work:

- *categorical integers*; used to identify related objects
  - `id`, `group_id`, `loc_id`, `home_id`, `box_id`, `event_id`, `mover`
- *boolean integers*; used to set boolean flags
  - `0` means `False` and `1` means `True`
  - `is_alive`, `is_vaxxed`, `is_sus`, `is_inf`, `ever_inf`, `go_home`
- *coordinates*; used to identify locations
  - `x`, `y`, `homex`, `homey`
- *event ticks*; integers that trigger an event on the given tick
  - `depart`, `recover`, `relapse`

- *factors*; scaled integers that must be unscaled before being used in multiplicative formulas
  - `tmf`, `ifr`

The column attributes are defined as follows:



| Label | Definition | Label | Definition |
|---|---|---|---|
| id | the subject's unique identifier | homey | y coord of the subject's home location |
| group_id | the unique identifier of the subject's group | go_home | is the subject going home on the next move? |
| is_alive | Is the subject alive? | box_id | id of the box the subject belongs to |
| is_vaxxed | Has the subject been vaccinated? | event_id | id of the event the subject is attending |
| is_sus | Is the subject susceptible to infection? | mover | id of the subject's mover function |
| is_inf | Is the subject infected? | mover_p | p-value of custom mover function |
| ever_inf | Has the subject ever been infected? | tmf | the subject's transmission factor |
| loc_id | id of the subject's current location | ifr | the subject's infection fatality rate |
| x | x coord of the subject's current location | inf_tick | the tick a subject is infected |
| y | y coord of the subject's curretn location | depart | the tick an infected subject will depart |
| home_id | id of the subject's home location | recover | the tick an infected subject will no longer be infected or susceptible |
| homex | x coord of the subject's home location | relapse | the tick a recovered subject will again become susceptible |

CHAPTER 4

# Sizing

Appropriate sizing of a simulation environment is more art than science but we can use several tools in the **RKnot** package and the scientific literature to approximate appropriate attributes and initial conditions.

Our main guides for appropriate sizing:

1. Actual $R_0$ should be close to target $R_0$
2. Contact rate should follow a gamma distribution with parameters found in the Hutch model.
3. $R_0$ distribution should replicate the Endo distribution.

Our main tools for adjusting contact rates and $R_0$ distribution are:

- Density
- *Mover functions*
- *Events*
- *VBoxes*

## 4.1 Importance of Distributions

Both the distribution of contacts *and* individual $R_0$ are important considerations when formulating a realistic model of viral spread.

In terms of contacts, both frequency and timing are important. All else equal, an infected individual will cause more secondary infections if they have more contacts. BUT an infected individual contacting the same number of people at two different times is likely to cause more secondary infections at the time of greater viral load.

The SIR model makes many simplifying assumptions that fail to replicate real world spread. In particular, SIR is characterized by:

- Uniform, average contact rates
- Constant transmission risk

These two assumptions result in normally distributed contact rates among all individuals in an environment and normally distributed individual $R_0$.

Endo et al, however, have suggested that 70%+ of all sars-cov-2 infections *lead to :math:‘underline{text{NO}}‘ secondary infections.* And that the vast majority of secondary infections are due to a small minority of super-spreader carriers.

To account for this, the Hutch team modelled contacts with a gamma distribution, which (relative to the Gaussian) favors many more instances of 0 contacts, but also allows for a very long tail of large contacts depending on the dispersion parameter.

---

## 4.2 Constant Contact Rate and Transmission Risk

To replicate the Hutch approach in **RKnot**, we must first understand contact and transmission dynamics in more generalized scensarios.

We will use the `looper` helper function to run 1,000 interations of simulations for two environments:

1. Equal Mover scenario with `density=1`.
2. Equal Mover with `density=5`

```
from rknot.notebook import looper

params = dict(density=1, R0=2.5, infdur=14, days=14, sterile=True,)
group = dict(name='all', n=10000, n_inf=1, ifr=0, mover='equal',)
contacts_1, nsecs_1, _, __ = looper(n=1000, groups=group, **params)

params['density'] = 5
contacts_5, nsecs_5, _, __ = looper(n=1000, groups=group, **params)
```

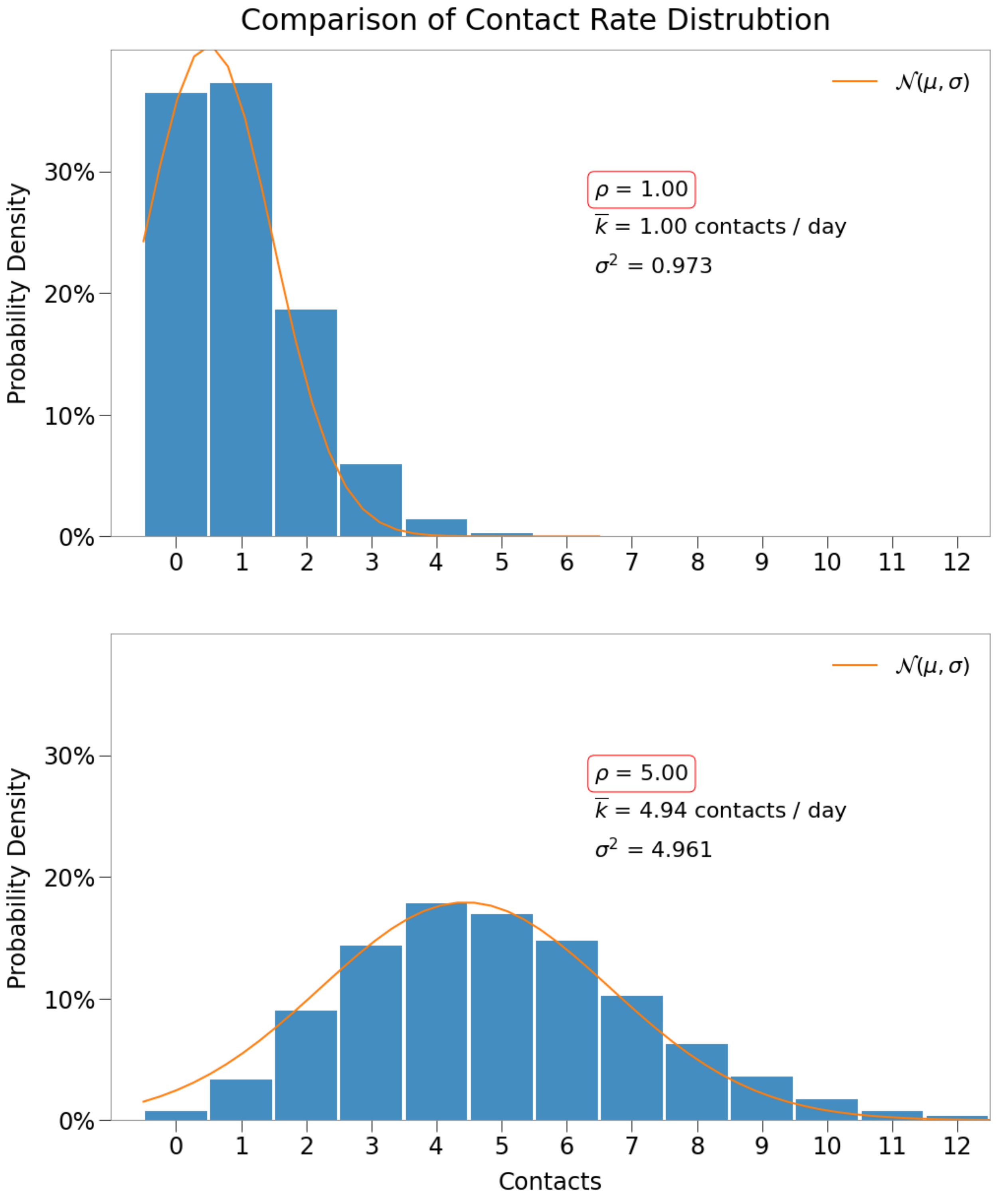
### 4.2.1 Daily Contact Distribution

The daily contact mean and variance are found simply as `contacts_1.mean()`, `contacts_1.var()`, etc. For each environment, the mean, variance, and density are about equal.

Seen as per table below:

```
<IPython.core.display.HTML object>
```

We can also see below that daily contacts are normally distributed in *both* environments (technically, *binomially distributed as discussed here*).



```
<IPython.core.display.HTML object>
```

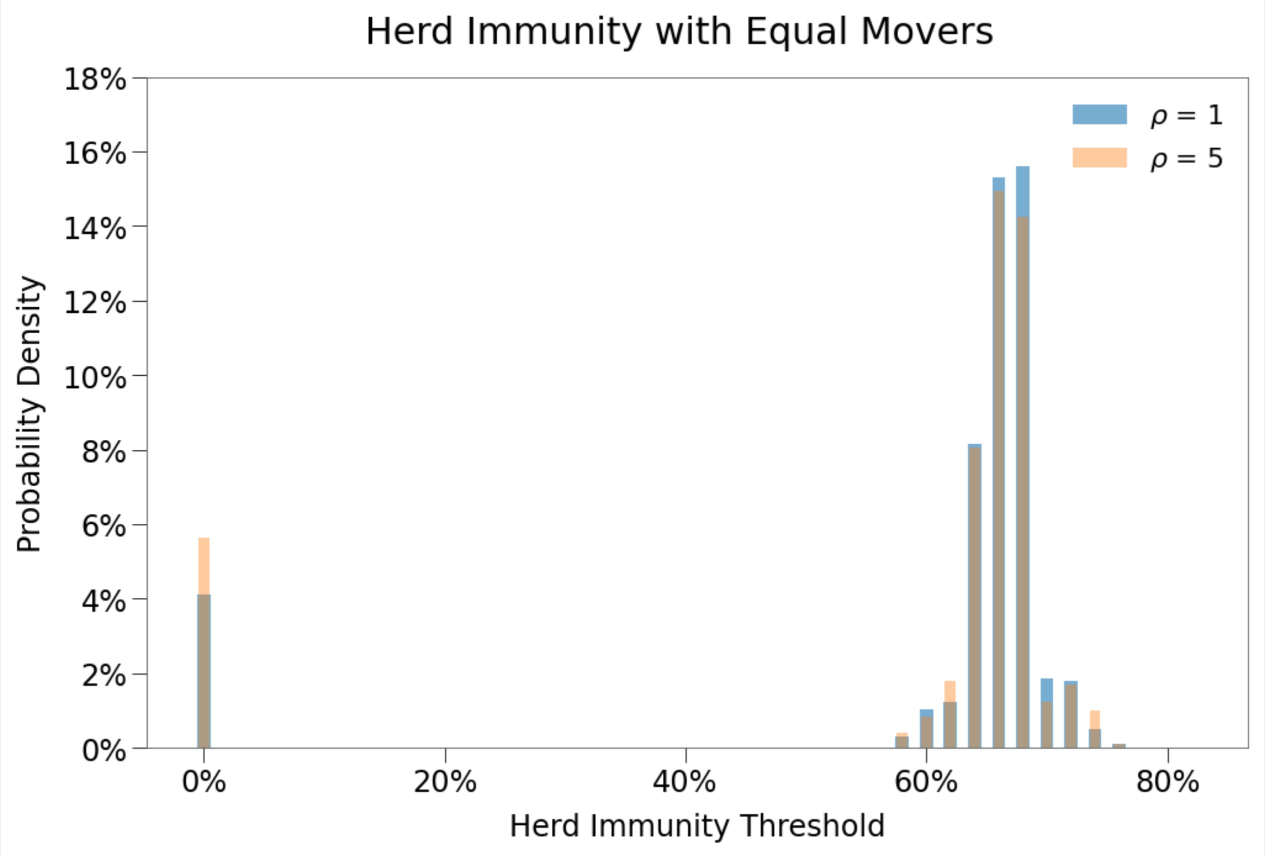
### 4.2.2 Curve Details

In the table below, we show the results of the simulations across the two environments.

```
<IPython.core.display.HTML object>
```

Both environments do a decent job of meeting the expected HIT. With $R_0$ of 2.5 (and assuming SIR conditions), HIT is generally understood to be $1 - 1/R_0 = 1 - 1/2.5 = 60\%$

That said, this is in part due to the fact that ~10% of all simulations ended with *no secondary infections*, so most simulations ended with between 65% and 75% HIT as per below:



### 4.2.3 Secondary Infection Distribution

Now, we compare the $R_0$ generated in each simulation of the two environments. Each chart below shows the actual number of secondary infections generated in each of 1,000 simulations with two different distributions overlayed.

- a Binomial distribution deterimined as follows:

$$\begin{aligned} B(n,p) = B(\text{infdur} * \overline{k}, \frac{R_0/\text{infdur}}{\overline{k}}) \\ where: n = \overline{c} = \text{infdur} * \overline{k} \\ p = R_0/\overline{k}/\text{infdur} \\ \overline{c} = \text{average total contacts} \\ \overline{k} = \text{average daily contact rate} \end{aligned} \quad (4.1)$$

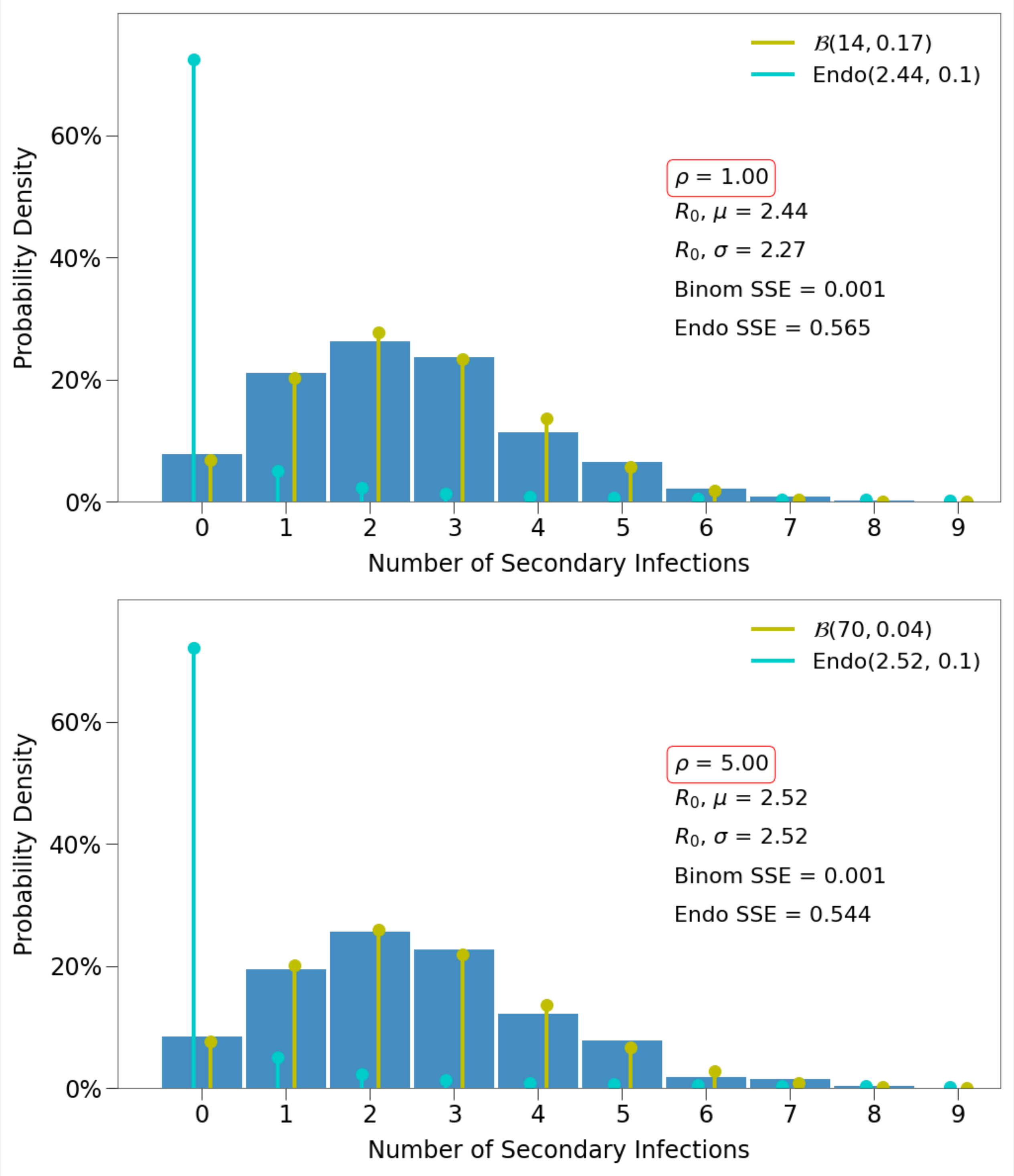
$\frac{R_0}{infdur}$ number of infections occuring during each day and $\frac{R_0/infdur}{\overline{k}}$ infections occuring per contact.

- the Endo distribution determined as follows:

$$Endo(R_0, \omega)$$

*where: $\omega$ is the overdispersion parameter*(4.2)

The Endo model is fairly complex so readers are encouraged to research it here. The paper determined an optimized value of $\omega$ of **0.1** sars-cov-2, which will be used for our purposes.



We can see both environments do a good job of generating ~2.5 secondary infections, on average.

We also see that both have $R_0$ distributions that fit the binomial distribution, not the Endo.

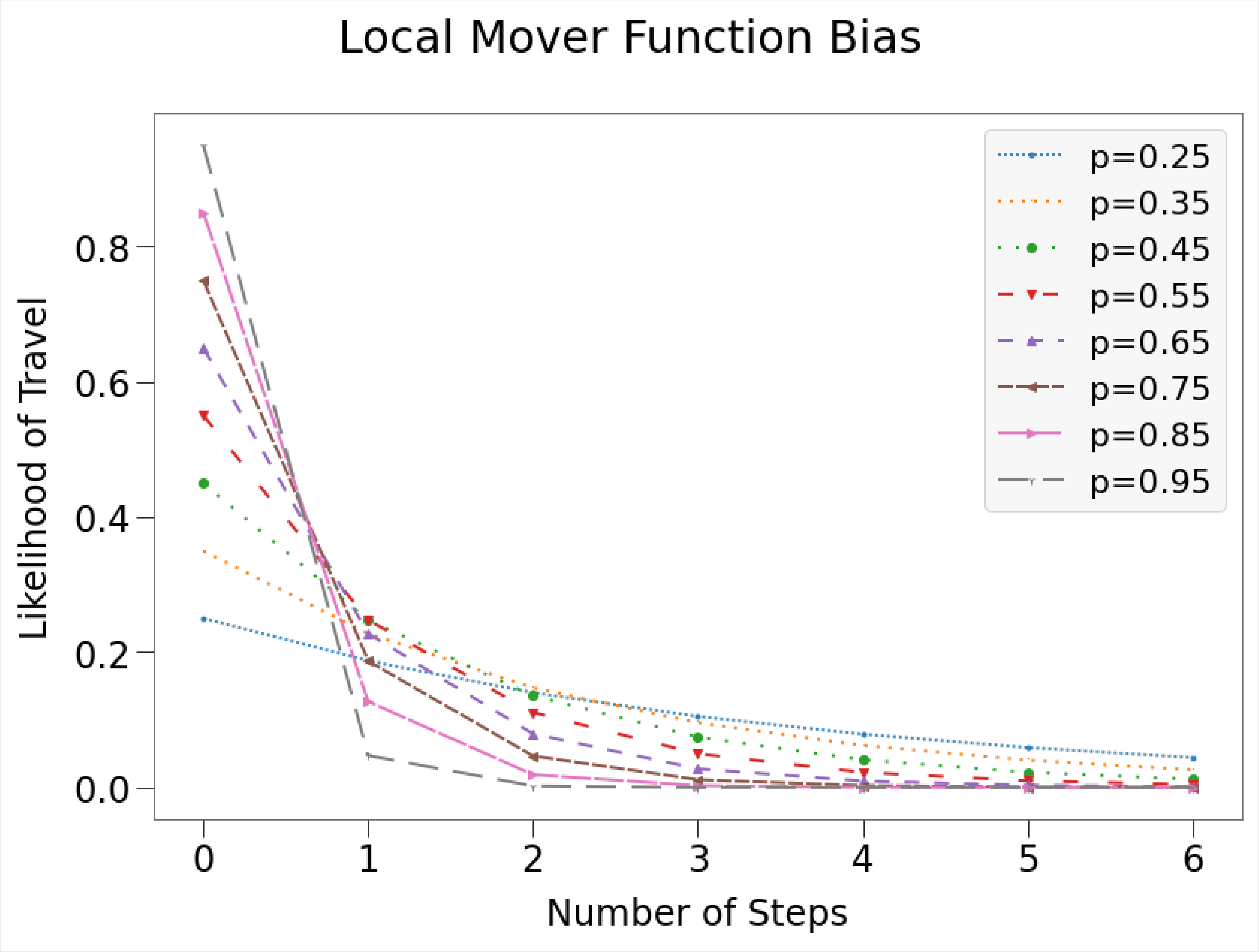
## 4.3 Movement Changes

We have seen the impact of density on the properties of the spread. Now we investigate the impact of movement patterns.

### 4.3.1 Local Bias

`local` patterns restrict the movement of a dot to only the nearest available locations. This is achieved using a geometric distribution with the `p` parameter governing the amount of restriction. The larger the `p`, the greater the restriction.

This is illustrated below:



There are *several pre-defined mover functions*, which can be utilized by providing a string or integer value to the `mover` attribute of a `Group`. The `local` mover function, for instance, utilizes a geometric distribution with p-value of 0.6.

The `mover` attribute *also* accepts a float value, which is then provided as the specified p-value to a geometric distribution.

With this feature, we can generate simulations across a range of p-values.

Then, we can use the helper function `find_all_contacts` to generate a 2d array of the number of the contacts on each tick for every dot in each sim. Then multiply the contacts the transmission risk curve to get the *expected* $R_0$ for every dot in the sim *as though they were infected initially*.

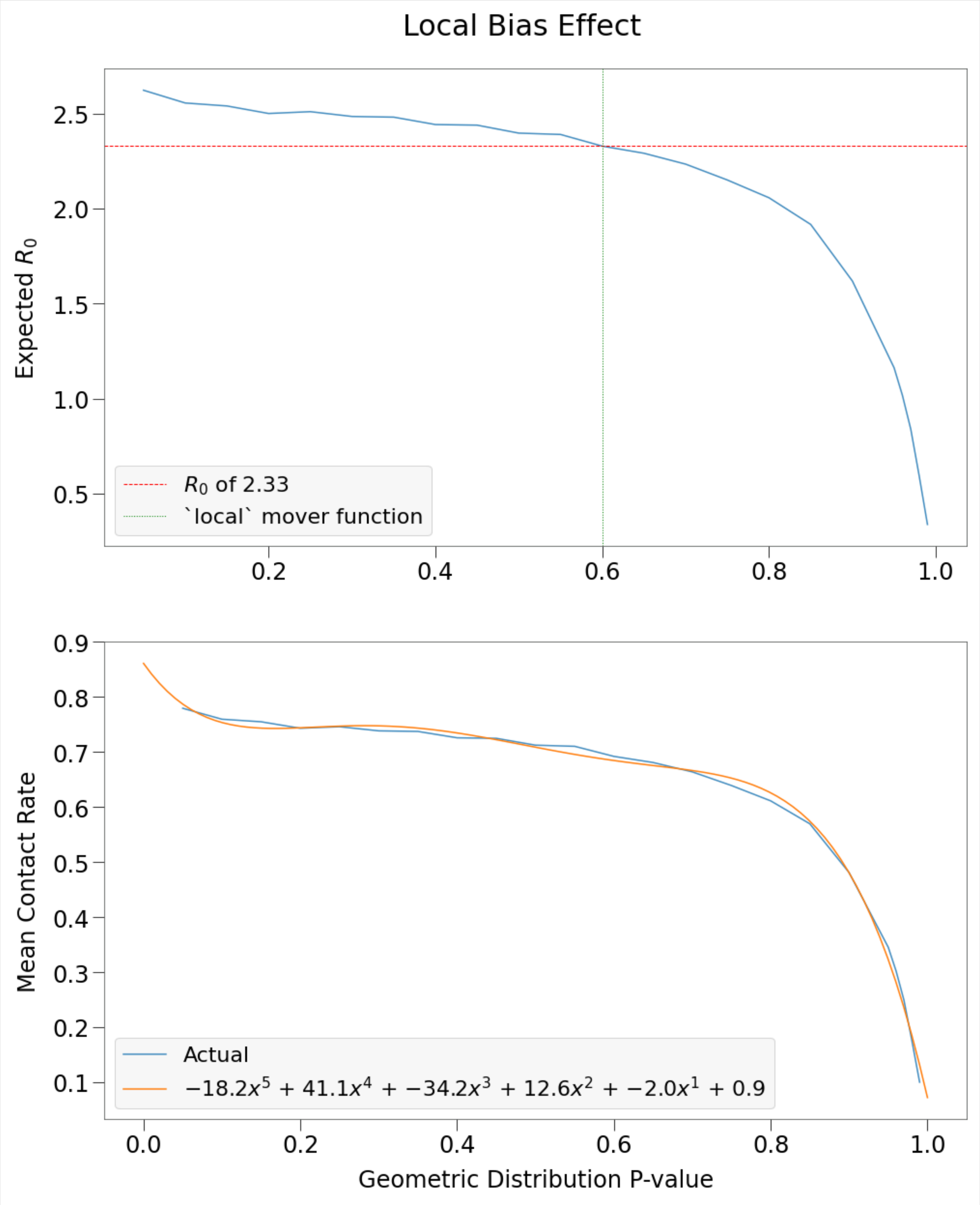
Repeat this for each `p-value`. Here, we cannot use `looper`, because the properties of each sim will change slightly, but we can use the `recycler` decorator to simplify the loop and pass existing `server` and `worker` actors if they are available.

```
from rknot import Sim
from rknot.sim import recycler
from rknot.notebook import find_all_contacts

params = dict(density=.75, R0=2.5, infdur=14, days=14, sterile=True, n_workers=1)
group = dict(name='all', n=10000, n_inf=1, ifr=0,)

p = np.concatenate((p, np.linspace(0.96, .99, 4)))

dotlogs = np.zeros(shape=(p.shape[0], params['days']+1, group['n'], len(ML)),␣
↪dtype=np.int32)
all_contacts = np.zeros(shape=(p.shape[0], group['n'], params['days']), dtype=np.
↪int32)
tmrs = np.zeros(shape=(p.shape[0], params['infdur']))

sim_cycler = recycler(p.shape[0])
for i in trange(p.shape[0]):
    group['mover'] = p[i]
    sim = sim_cycler(groups=group, pbar=False,  **params)
    sim.run(dotlog=True, pbar=False)
    dotlogs[i] = sim.dotlog
    all_contacts[i] = nbf.find_all_contacts(sim.dotlog, MLNB)
    tmrs[i] = sim.tmrs

contact_means = all_contacts.sum(axis=2).mean(axis=1) / params['days']
eR0s = np.zeros(all_contacts.shape[0])
for i in range(eR0s.shape[0]):
    eR0_by_dot = tmrs[i] * all_contacts[i]
    eR0_mu = eR0_by_dot.sum(axis=1).mean()
    eR0s[i] = eR0_mu
```

We can then chart $E(R_0)$ against `p-value` to see the impact of restricting movement on spread.

Further, we show the change in mean contacts against `p-value` (which is a similar curve) and fit that curve to a 5-degree polynomial. This will be helpful in building *Complex Environments*.

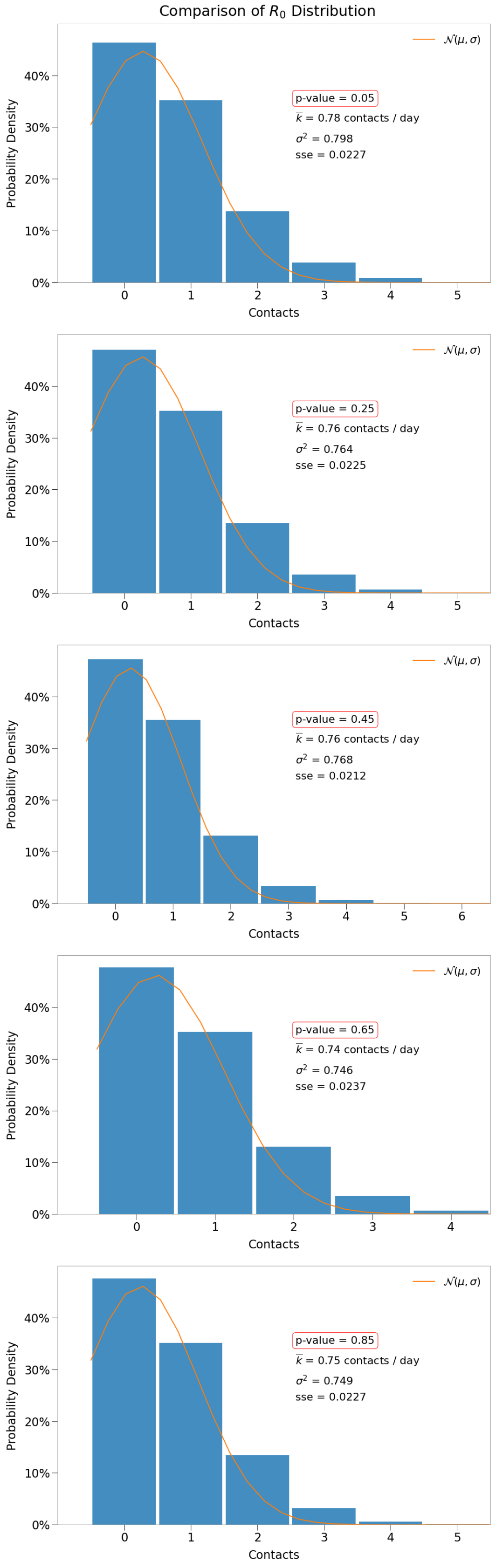
We can also easily accomodate densities less than one by simply multiplying the polyfit parameters by the density.



We see that $R_0$ has an inverse relationship to restricted movement (i.e. spread increases as populations become more well-mixed). This, of course, makes sense and is a fundamental premise of viral spread theory.

However what cannot be overlooked is how closely $R_0$ in a *moderately* restricted environment resembles a completely well-mixed environment. And yet we know that the `local` *mover function (with p-value of 0.6) results in a very* *different long-term spread curve*.

Further, each p-value results in normally-distributed contacts. We show this for a handful of p-values below.


Comparison of $R_0$ Distribution
$\mathcal{N}(\mu, \sigma)$
p-value = 0.05
$\bar{k}$ = 0.78 contacts / day
$\sigma^2$ = 0.798
sse = 0.0227
Probability Density
0%
10%
20%
30%
40%
0
1
2
3
4
5
Contacts
$\mathcal{N}(\mu, \sigma)$
p-value = 0.25
$\bar{k}$ = 0.76 contacts / day
$\sigma^2$ = 0.764
sse = 0.0225
Probability Density
Contacts
$\mathcal{N}(\mu, \sigma)$
p-value = 0.45
$\bar{k}$ = 0.76 contacts / day
$\sigma^2$ = 0.768
sse = 0.0212
Probability Density
6
Contacts
$\mathcal{N}(\mu, \sigma)$
p-value = 0.65
$\bar{k}$ = 0.74 contacts / day
$\sigma^2$ = 0.746
sse = 0.0237
Probability Density
Contacts
$\mathcal{N}(\mu, \sigma)$
p-value = 0.85
$\bar{k}$ = 0.75 contacts / day
$\sigma^2$ = 0.749
sse = 0.0227
Probability Density
Contacts

### 4.3.2 Interplay of Movement and Density

We can also run each p-value against a range of densities.

```
params = dict(
    R0=2.5,
    infdur=14,
    days=14,
    sterile=True,
    pbar=False,
    pbar_leave=False,
    reset_env=True,
)
group = dict(
    name='all',
    n=10000,
    n_inf=1,
    ifr=0,
)
densities = np.array([.5, .75, 1, 2.5, 5])

dotlogs = np.zeros(shape=(
        p.shape[0], densities.shape[0], params['days']+1, group['n'], len(ML)
    ), dtype=np.int32
)
all_contacts = np.zeros(shape=(
        p.shape[0], densities.shape[0], group['n'], params['days']
    ), dtype=np.int32
)
tmrs = np.zeros(shape=(densities.shape[0], params['infdur']))

n_iters = p.shape[0]*densities.shape[0]
pbar = tqdm(total=n_iters)
sim_cycler = recycler(n_iters)
for i in range(p.shape[0]):
    group['mover'] = p[i]
    for j in range(densities.shape[0]):
        sim = sim_cycler(groups=group, density=densities[j], **params)
        sim.run(dotlog=True, pbar=False)

        dotlogs[i, j] = sim.dotlog
        all_contacts[i, j] = find_all_contacts(sim.dotlog, MLNB)
        tmrs[j] = sim.tmrs

        pbar.update(1)
```

Then find the expected $R_0$ for each combination:

```
eR0s = np.zeros(shape=(all_contacts.shape[0], all_contacts.shape[1]))
for i in range(eR0s.shape[0]):
    for j in range(eR0s.shape[1]):
        eR0_by_dot = tmrs[j] * all_contacts[i, j]
        eR0_mu = eR0_by_dot.sum(axis=1).mean()
        eR0s[i, j] = eR0_mu
```

And below we show $R_0$, contact $\mu$, and contact variance for a subset of the `p-value/density` combinations.

| | | $\rho$ = 0.5 | | | $\rho$ = 0.75 | | | $\rho$ = 1.0 | | | $\rho$ = 2.5 | | | $\rho$ = 5.0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| p-value | $R_0$ | $\overline{k}$ | $\sigma^2$ | $R_0$ | $\overline{k}$ | $\sigma^2$ | $R_0$ | $\overline{k}$ | $\sigma^2$ | $R_0$ | $\overline{k}$ | $\sigma^2$ | $R_0$ | $\overline{k}$ | $\sigma^2$ |
| 0.05 | 2.60 | 0.52 | 0.52 | 2.61 | 0.78 | 0.80 | 2.60 | 1.04 | 1.08 | 2.54 | 2.48 | 2.55 | 2.52 | 4.98 | 5.12 |
| 0.20 | 2.51 | 0.50 | 0.50 | 2.52 | 0.75 | 0.75 | 2.51 | 1.00 | 1.00 | 2.55 | 2.50 | 2.60 | 2.57 | 5.09 | 5.56 |
| 0.35 | 2.45 | 0.49 | 0.49 | 2.48 | 0.74 | 0.73 | 2.48 | 0.99 | 1.01 | 2.52 | 2.46 | 2.46 | 2.53 | 5.00 | 5.20 |
| 0.50 | 2.41 | 0.48 | 0.48 | 2.41 | 0.72 | 0.70 | 2.43 | 0.97 | 0.96 | 2.50 | 2.44 | 2.41 | 2.51 | 4.96 | 4.99 |
| 0.65 | 2.30 | 0.46 | 0.45 | 2.32 | 0.69 | 0.67 | 2.29 | 0.92 | 0.88 | 2.50 | 2.44 | 2.46 | 2.51 | 4.96 | 5.06 |
| 0.80 | 2.04 | 0.41 | 0.40 | 2.05 | 0.61 | 0.58 | 2.03 | 0.81 | 0.75 | 2.54 | 2.48 | 2.58 | 2.50 | 4.94 | 4.95 |
| 0.95 | 1.17 | 0.23 | 0.22 | 1.14 | 0.34 | 0.33 | 1.17 | 0.47 | 0.43 | 2.50 | 2.44 | 2.45 | 2.48 | 4.90 | 4.66 |
| 0.98 | 0.64 | 0.13 | 0.12 | 0.60 | 0.18 | 0.17 | 0.61 | 0.25 | 0.23 | 2.49 | 2.43 | 2.40 | 2.47 | 4.89 | 4.74 |

The table above has some interesting properties:

1. As expected, mean contacts, $\overline{k}$, increases with density.
2. Also, for densities of 1 or less, $\overline{k}$ decreases with p-value.
3. Also, for densities of 1 or less, $R_0$ decreases with density.

HOWEVER, when density > 1, all of the values remain almost constant for each density regardless of p-value. When a grid is overloaded with dots, by definition the same number of dots are expected at each location regardless of the movement pattern (on average).

### 4.3.3 Location Clustering

We should also mention the phenomena of dot clustering. We can determine how many different locations are occupied by at least one dot at each tick as follows:

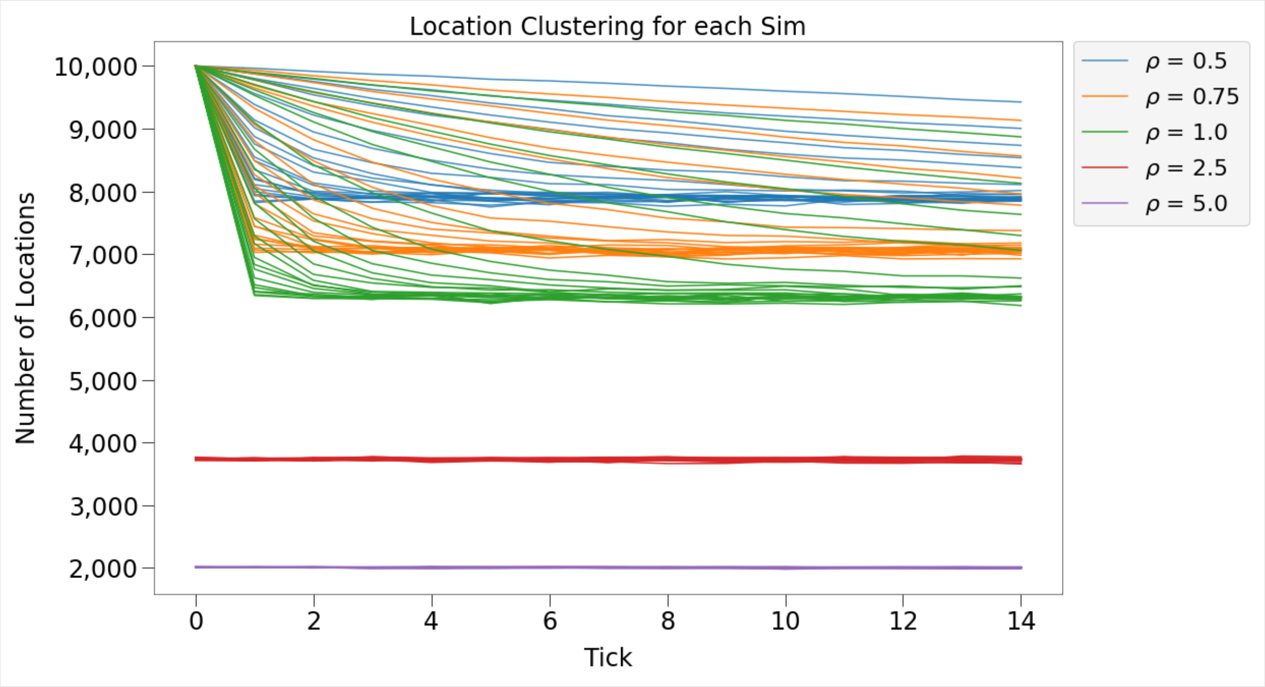
[27]:

```
unq_locs = np.zeros(shape=dotlogs.shape[:3])
for i in range(dotlogs.shape[0]):
    for j in range(dotlogs.shape[1]):
        for k in range(dotlogs.shape[2]):
            unq_locs[i,j,k] = np.unique(dotlogs[i,j,k,:,ML['loc_id']]).shape[0]
```

In the chart below, we can see that each sim with density < 1, starts with dots occupying 10,000 unique locations (so each dot gets its own location), but over time dots cluster to a smaller number of locations.

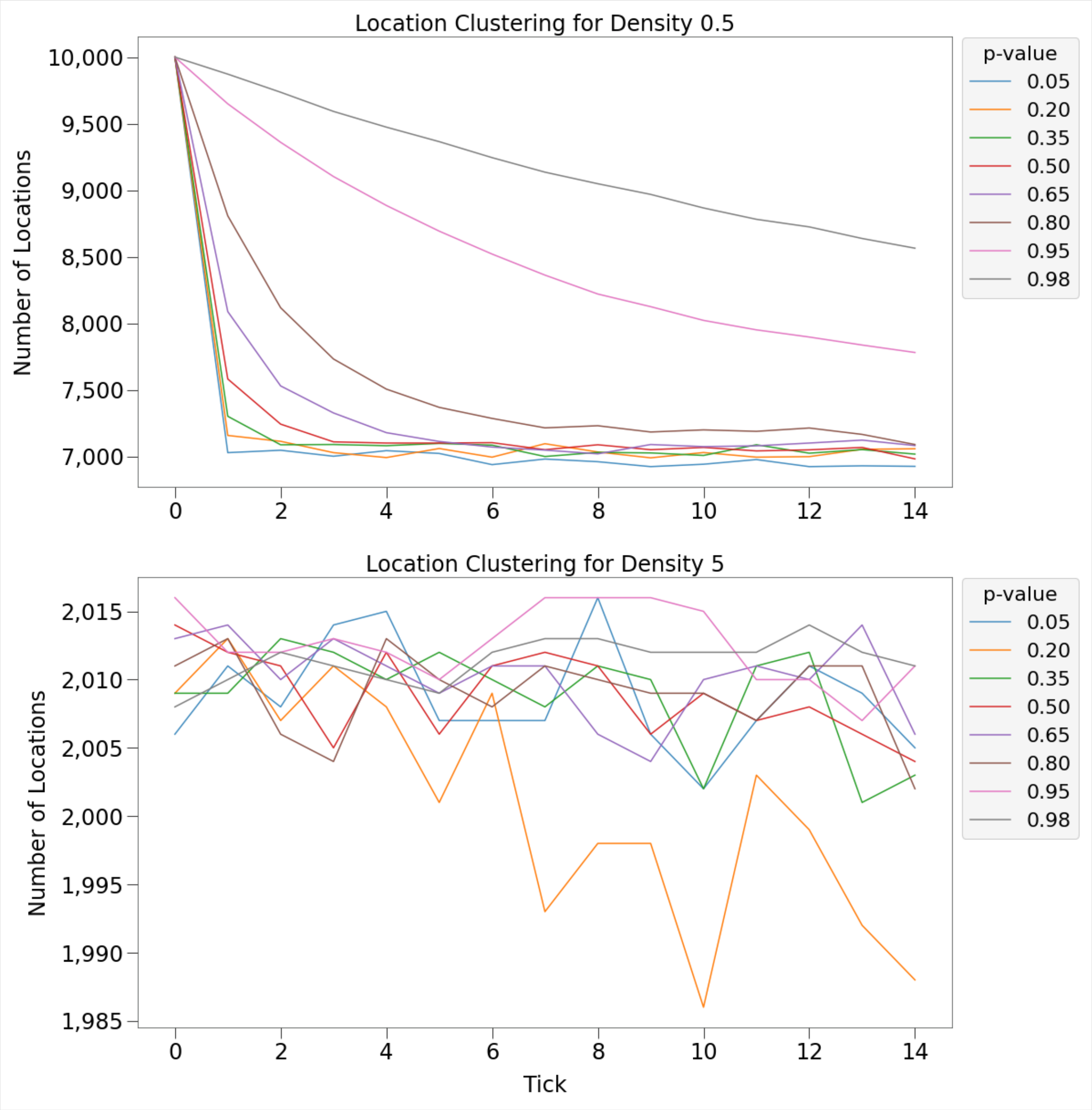
The clustering appears asymptotic to a long-term value. That value appears to be inversely related to the density (so lower density means more locations and less clustering). And the speed at which that value is reached is related to the p-value.

For sims with density > 1, there is no clustering.



Below we more clearly see the relationship between p-value and speed of clustering:

1. For density of 0.5:
   - p-value is inversely related to clustering speed. i.e. movement patterns with higher p-values cluster slower
2. For density of 5:
   - there is no clustering and number of unique locations varies randomly in a tight range.



### 4.3.4 $R_0$ Distribution

For $R_0$ distribution, we must loop through a large number of simulations. We will focus only on p-values [.25, .65, .95] and densities [.5, 1].

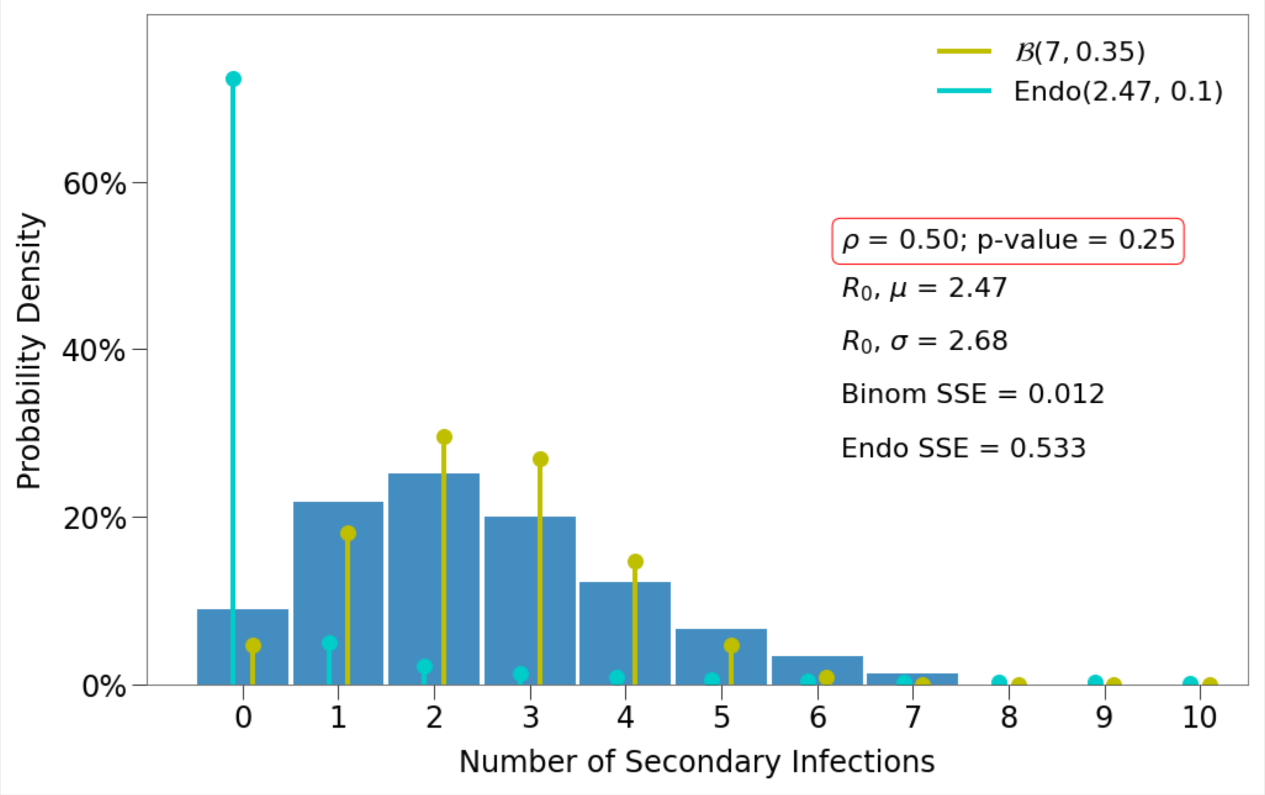
Again, we utilize the `looper` function.

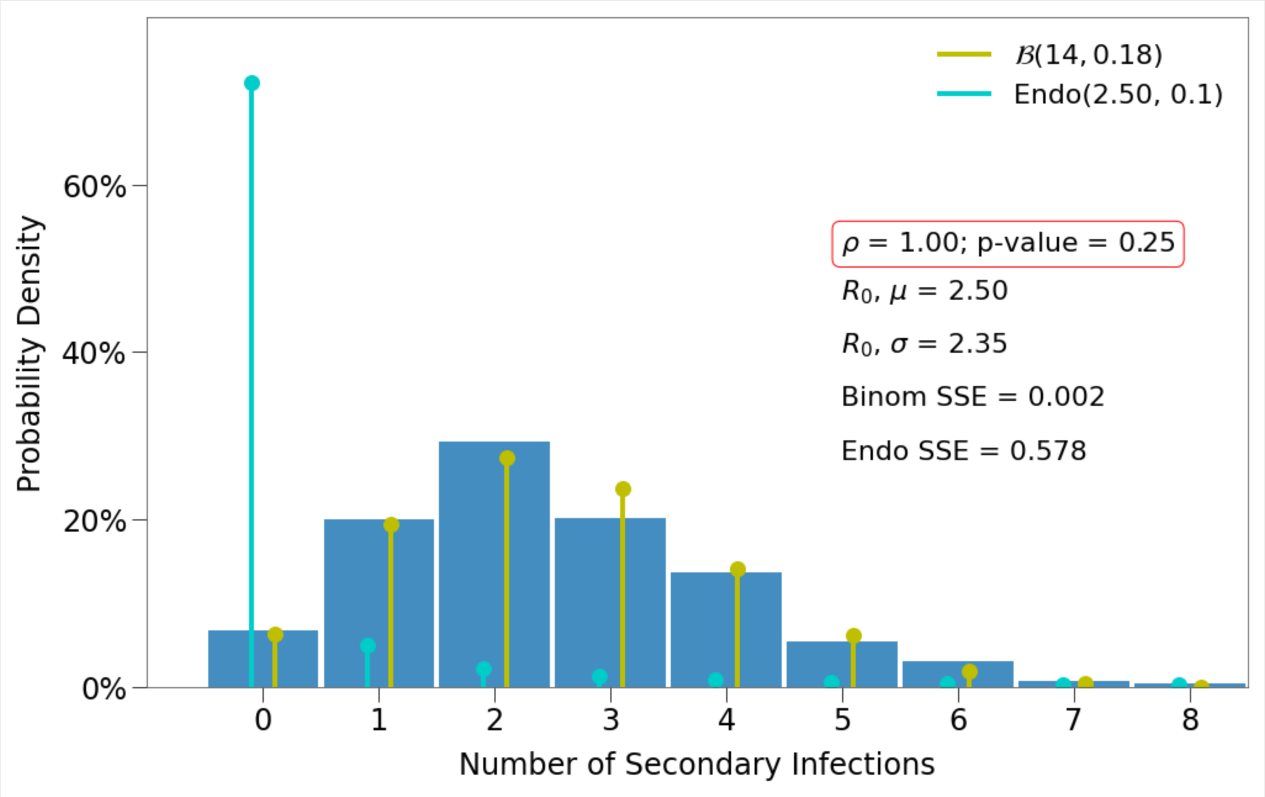
```python
from rknot.notebook import looper

params = dict(
    R0=2.5,
    infdur=14,
```

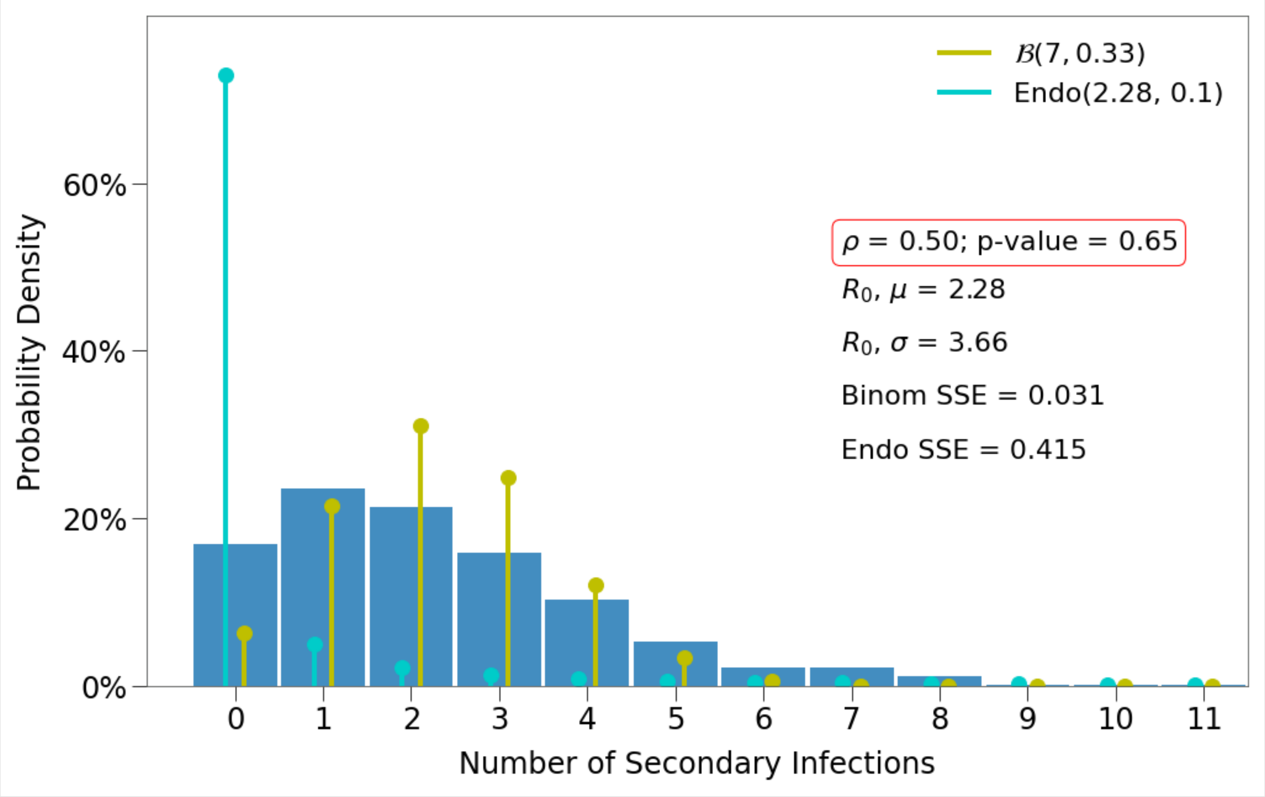
(continues on next page)

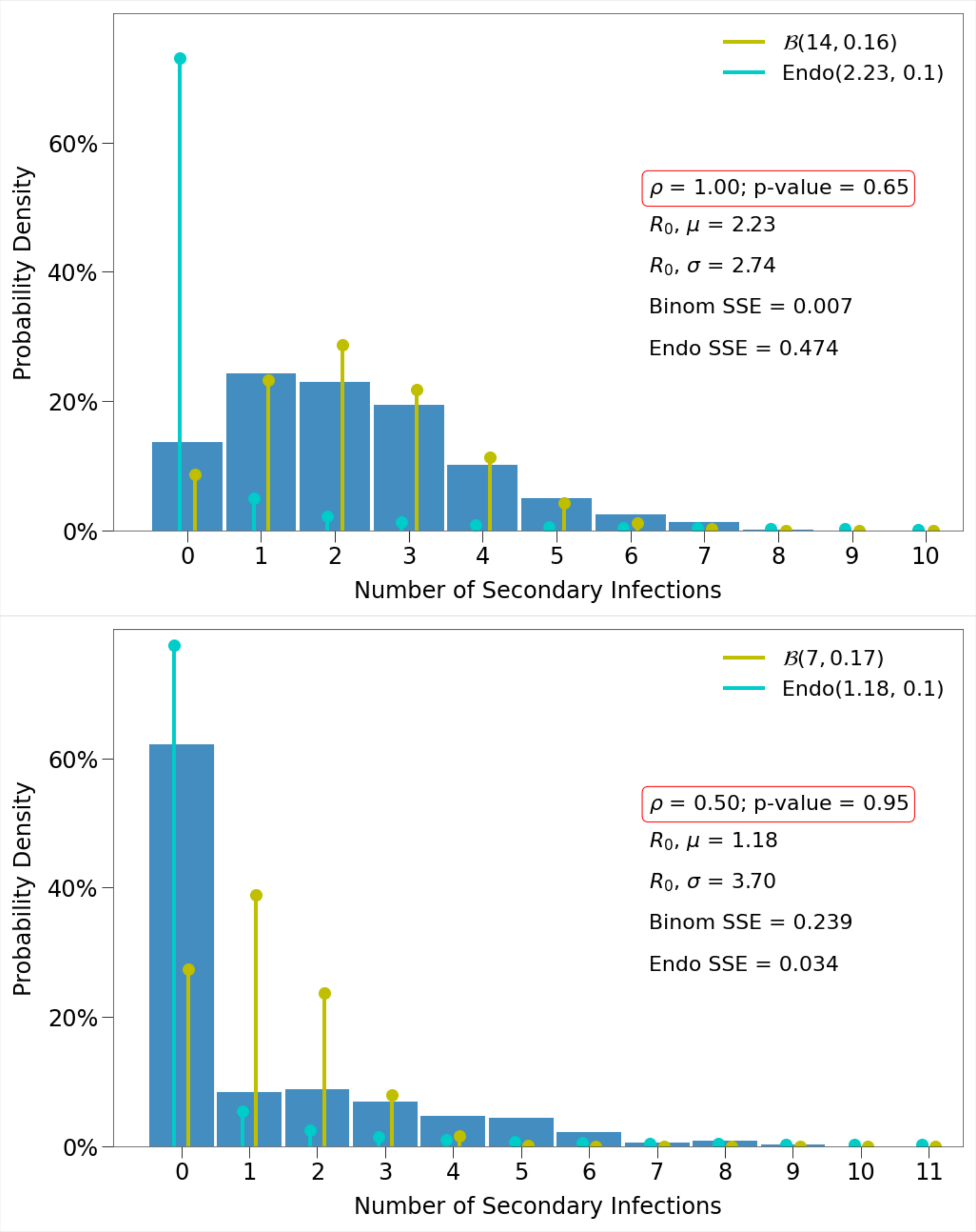
(continued from previous page)

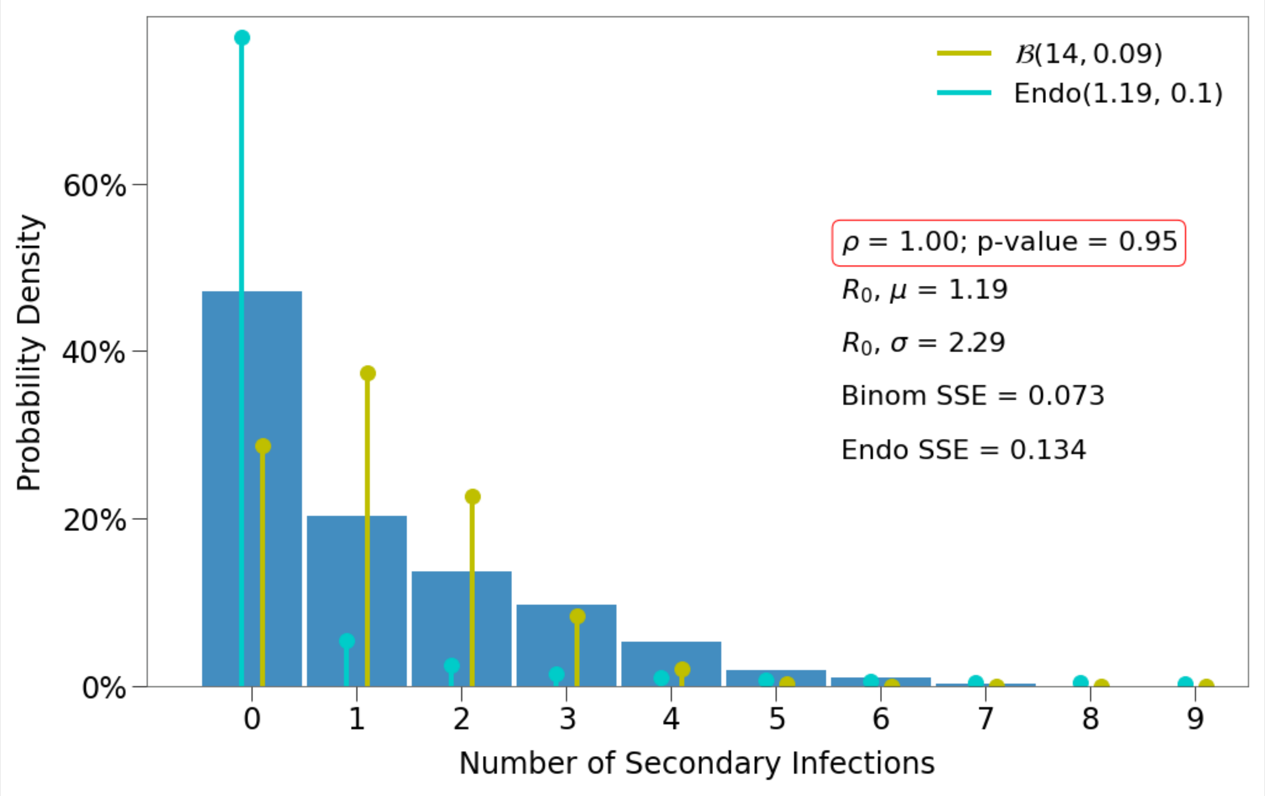
```
    days=14,
)
group = dict(
    name='all',
    n=10000,
    n_inf=1,
    ifr=0,
)

p = np.array([.25, .65, .95])
d = np.array([.5, 1])
n = 1000

nsecs = np.zeros(shape=(3, 2, n, params['days']))
pbar = tqdm(total=p.shape[0]*d.shape[0])
for i in range(p.shape[0]):
    group['mover'] = p[i]
    for j in range(d.shape[0]):
        params['density'] = d[j]
        _, loop_nsecs, _ = looper(n=n, groups=group, **params)

        nsecs[i, j] = loop_nsecs

        pbar.update(1)
```




$\mathcal{B}(14, 0.18)$
Endo(2.50, 0.1)
$\rho$ = 1.00; p-value = 0.25
$R_0$, $\mu$ = 2.50
$R_0$, $\sigma$ = 2.35
Binom SSE = 0.002
Endo SSE = 0.578
Probability Density
0%
20%
40%
60%
0
1
2
3
4
5
6
7
8
Number of Secondary Infections


$\mathcal{B}(7, 0.33)$
Endo(2.28, 0.1)
$\rho$ = 0.50; p-value = 0.65
$R_0$, $\mu$ = 2.28
$R_0$, $\sigma$ = 3.66
Binom SSE = 0.031
Endo SSE = 0.415
Probability Density
0%
20%
40%
60%
0
1
2
3
4
5
6
7
8
9
10
11
Number of Secondary Infections


$\mathcal{B}(14, 0.16)$
Endo(2.23, 0.1)
$\rho$ = 1.00; p-value = 0.65
$R_0$, $\mu$ = 2.23
$R_0$, $\sigma$ = 2.74
Binom SSE = 0.007
Endo SSE = 0.474
Probability Density
0%
20%
40%
60%
0
1
2
3
4
5
6
7
8
9
10
Number of Secondary Infections
$\mathcal{B}(7, 0.17)$
Endo(1.18, 0.1)
$\rho$ = 0.50; p-value = 0.95
$R_0$, $\mu$ = 1.18
$R_0$, $\sigma$ = 3.70
Binom SSE = 0.239
Endo SSE = 0.034
Probability Density
0%
20%
40%
60%
0
1
2
3
4
5
6
7
8
9
10
11
Number of Secondary Infections



Here we have a few findings:

1. There is one decent fit for the Endo model, which comes at the lowest density and the highest p-value.
2. The closest fit to the Binomial distribution comes at `density=1` and the lowest p-value.

## 4.4 Complex Environments

We can use our findings to structure more complex environments that more appropriately mock real world interactions. Here, we will incorporate a dynamic transmission curve with $R_0$ dispersion that fits the Endo model.

Hutch

As noted previously, with respect to sars-cov-2, the Hutch model uses a Gamma distribution to govern real world interactions as per below:

$$\begin{aligned} c_t \ \approx \ \Gamma(\alpha, \theta) \ &\approx \ \Gamma(\frac{\overline{k}}{\theta}, \theta) \\ &\alpha \textit{ is the shape parameter,} \\ \mu = \overline{k}&\textit{, or the average daily contact rate} \\ &\theta \textit{ is the dispersion parameter} \end{aligned} \quad (4.3)$$

The scaling parameters are related to mean and variance as follows:

$$\overline{k} = \alpha\theta$$
$$\sigma^2 = \alpha\theta^2 (4.4)$$

The Hutch model combines this approach to contact distributions with a *model for dynamic viral load* to estimate contact level parameters for viral transmission.

They detail two sets of parameters for two $R_0$ outcomes.

$R_0 = 1.8$

- mean $R_0$ of 1.8
- 4 mean contacts per day, $\overline{k}$
- 40 dispersion, $\theta$
- $\lambda$ of 10**7 (a parameter influencing the relative infectiousness of an infected subjected)

$R_0 = 2.8$

- mean $R_0$ of 2.8
- 20 mean contacts per day, $\overline{k}$
- 30 dispersion, $\theta$
- $\lambda$ of 10**7.5

The parameter best fit for both outcomes was determined on a number of external inputs including adherence to the findings of the Endo model that >70%+ primary infections caused no secondary infections.

```
<IPython.core.display.HTML object>
```

Process

Our goal is to replicate both the Hutch model gamma distribution (both $\overline{k}$ and $\theta$) and the Endo model $R_0$ distribution.

**RKnot** currently has 4 tools for increasing contact rate:

- changing density
- changing movement patterns
- events
- vboxes

We have seen that increasing density does increase $\overline{k}$, but that both contacts and $R_0$ are normally/binomially distributed. So increasing density is not appropriate.

We have also seen that changing movement patterns only has an impact where density < 1 and can only increase it up to the density. So, if our density must be <= 1, our movement pattern can only ever increase density to <= 1.

So we are left with events and vboxes.

Thus, our process for building complex environments is as follows:

```
<IPython.core.display.HTML object>
```

### 4.4.1 Hutch Model: $R_0$ of 1.8

We will demonstrate the building of a complex environment using $R_0$ 1.8 Hutch model.

Our environment will have with following initial conditions:

- Population of 10,000
- Initial infected of 1
- Density of 1
- Length of 30 days
- 0.98 p-value for mover

Our model targets: + $R_0 = 1.8$ + $\overline{k} = 4$ + $\theta = 40$

Exploring the Contact Distribution

We can visualize the expected contact rate distribution by generating a random set of contacts on the gamma distribution.

We know that there should be $n_{dots}$ * $n_{days}$ number of records of contacts (300,000) and so can show the number of occurences of each amount of contacts from a gamma sample:

The initial parameters are as follows:

[39]:

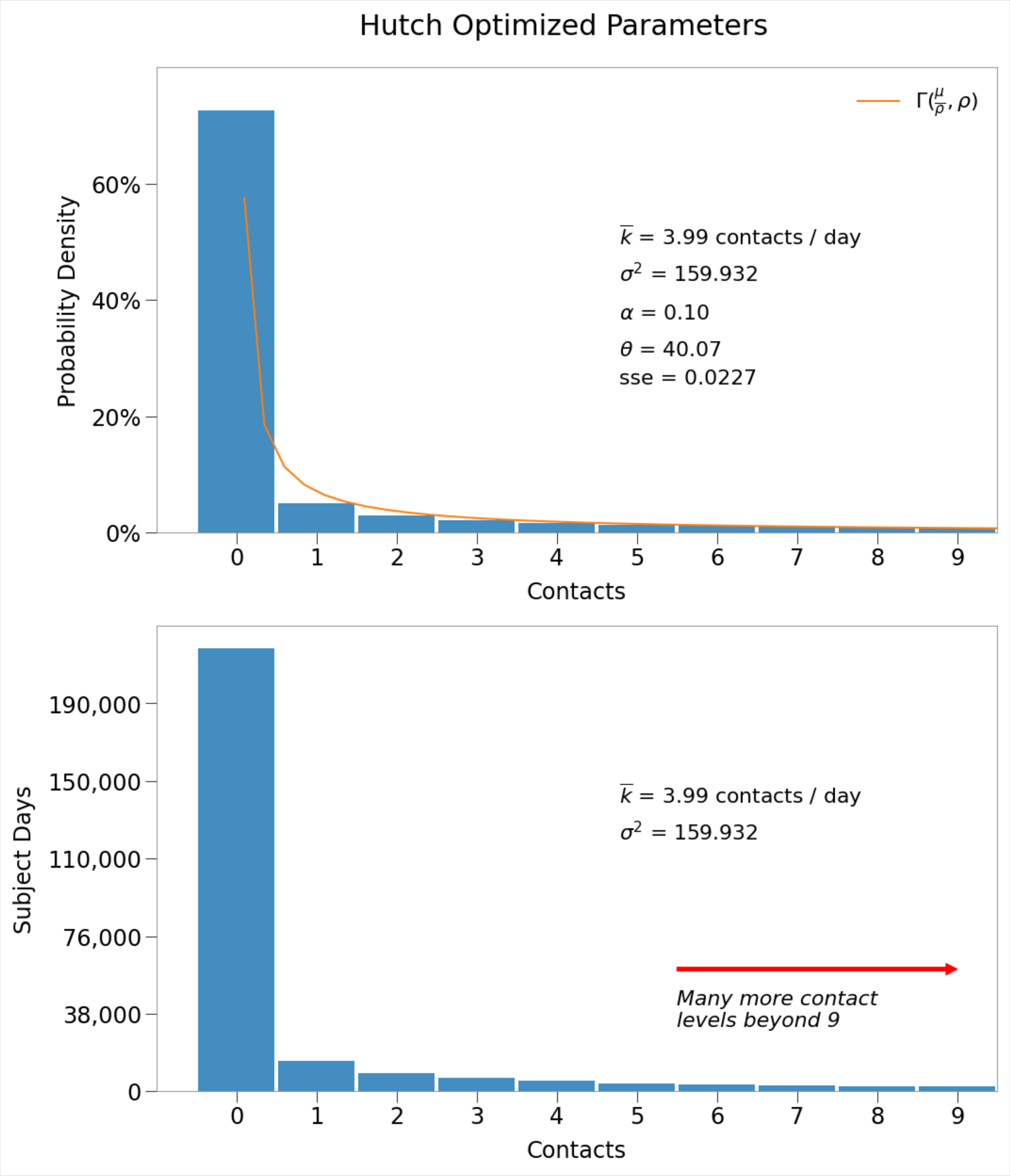
```
n = 10000
days = 30
size = n*days
density = 1
gloc = 2
sim_tmr = tmr

mu = 4
theta = 40
tgt_R0 = 1.8
```

From these, we can generate a sample gamma distribution as follows:

[40]:

```
alpha = mu / theta
k_gam = np.random.gamma(alpha, size=size, scale=theta)
```

`k_gam` can be passed to a histogram and results in the folllowing:

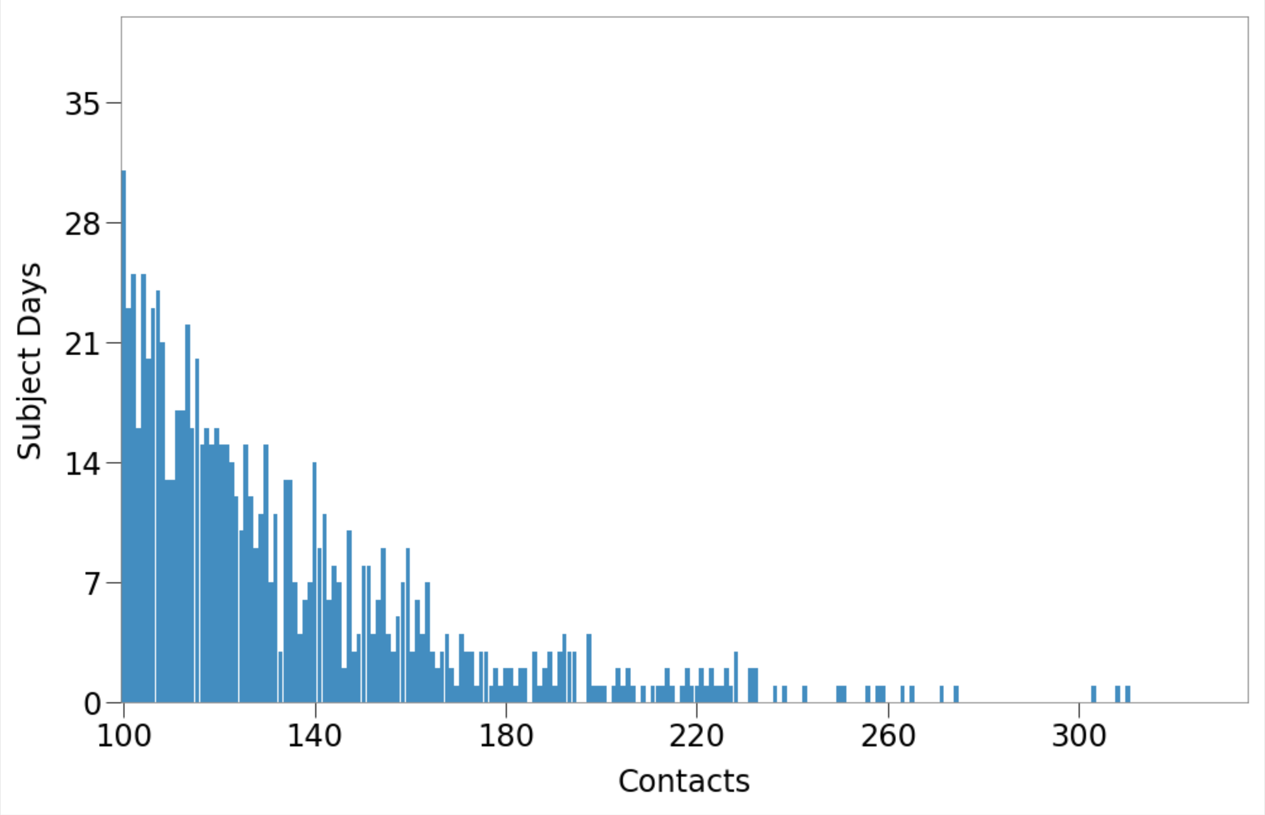


By far, most dot-days must have 0 contacts. Above we zoomed in on the first 10 contact levels to give a sense for the most pertinent values.

Below we show the distribution tail. Remember that while the number of occurences is relatively low, given the contact level is so high they still produce a large number of contacts.

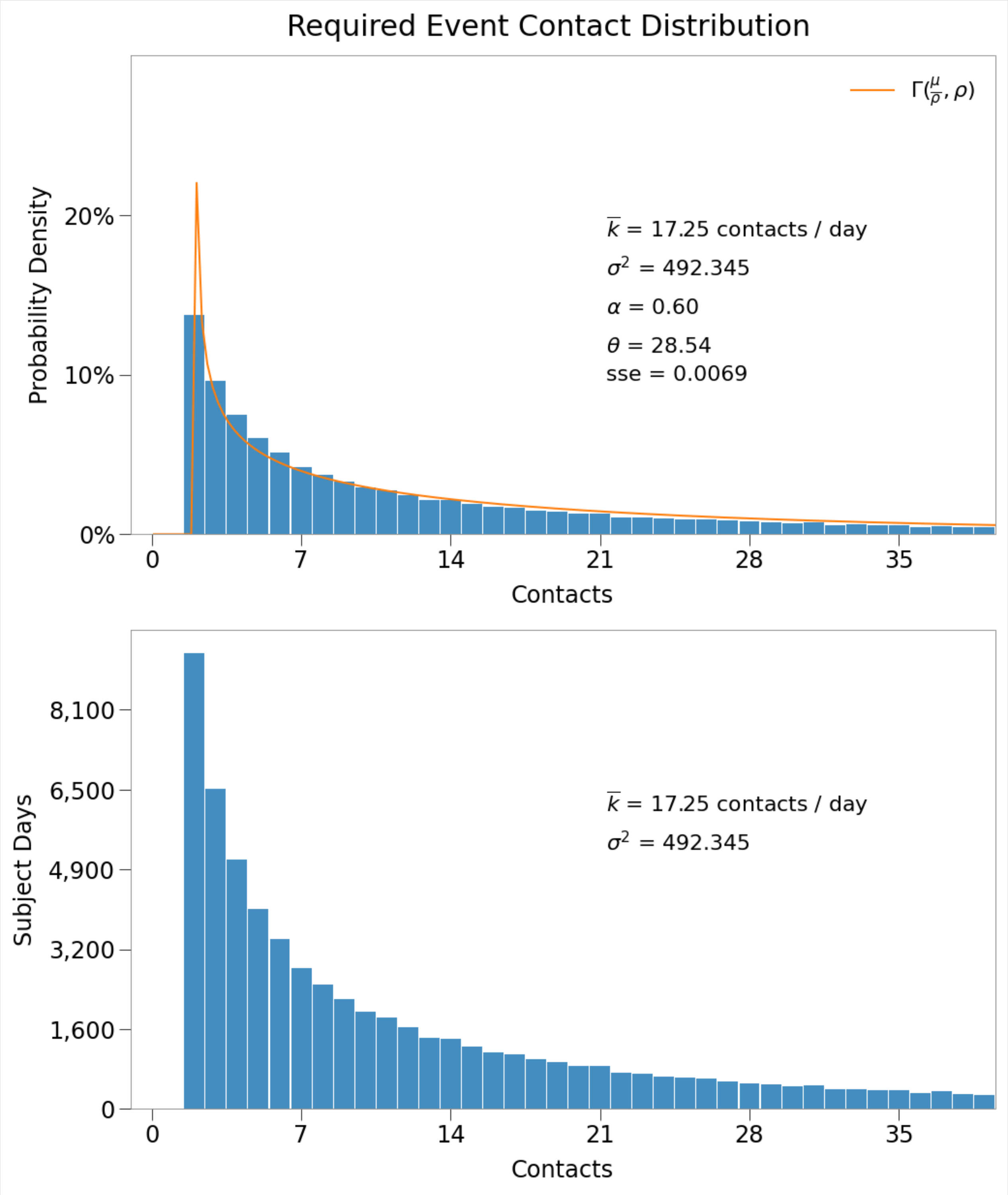
For instance:

- 2,000 dot-days of 2 contacts produces 4,000 contacts
- 40 dot-days of 100 contacts produces 40,000 contacts



From the distribution above, we can focus on the contacts that must be event-driven. For our sim, we have chosen contacts of 2 or more (equal to event apacity of 3) to result from events.

Thus, events must replicate the distribution below, which is still a gamma distribution, slightly modified from the original.



Creating Events

The number of contacts for a single dot at an event is simply the capacity minus 1, since a dot cannot contact itself.

$$\begin{aligned} k_e &= \text{cap}_e - 1 \\ \textit{where: } k_e &= \textit{contact level for event e} \\ \textit{cap}_e &= \textit{capacity for event e} \end{aligned} \tag{4.5}$$

Multiplying again for every dot at the event gives us the total contacts for all dots:

$$\begin{aligned} c_e &= (cap_e - 1) * cap_e \\ \textit{where: } c_e &= \textit{total contacts for event e} \end{aligned} \tag{4.6}$$

Remember that our target $\overline{k}$ is the average contact rate *per dot*, so the number of events should be structured on a per dot basis. The number of events required for a particular contact level, $k$, then, is:

$$\begin{aligned} n_{e,j} &= \frac{k_{tgt,i}}{cap_{e,j}} \\ \textit{i = contact level} \\ \textit{j = i + 1 = event capacity} \end{aligned} \tag{4.7}$$

```
[44]: c_eve = c_eve.astype(np.int32)
      caps = np.arange(c_eve.shape[0]) + 1

      n_events = c_eve[gloc:] / caps[gloc:]
```

**RKnot's** ability to replicate the Gamma distribution fails on the long tail of the distribution when the number of occurences of a specified contact level is fewer than the contacts that would be generated by a single event.

For instance, the Hutch gamma distribution results in ~40 dot-days with 100 contacts. **RKnot** can only generate 100 contacts through a 100 contact event and a 100 contact event generates $(100 - 1) * 100 = 9,900$ contacts, far more than required.

Our current hack-fix is to regroup and redistribute all contact occurences above that threshold into events. This results in a tail that is less evenly distributed.

A better solution would be to increase the number of ticks per day, which would allow for many more independent streams of contact distributions among dots. This feature will be available in future versions.

```
[45]: i_events = np.argwhere(n_events < 1 ).ravel()

      i_large = i_events[0]
```

(continues on next page)

(continued from previous page)

```
k_to_replicate = n_events[i_large:]*caps[gloc + i_large:]*caps[gloc + i_large:]

total = k_to_replicate.sum()
skip = 70
add_caps = []
for cap in range(i_large + skip + 1, i_large + k_to_replicate.shape[0] + 1, skip):
    total = total - ((cap - 1)*cap)
    if total < 0:
        break
    else:
        add_caps.append(cap)
        n_events[cap - 1] = 1

n_events = np.where(n_events >= 1, n_events, 0).astype(np.int32)
```

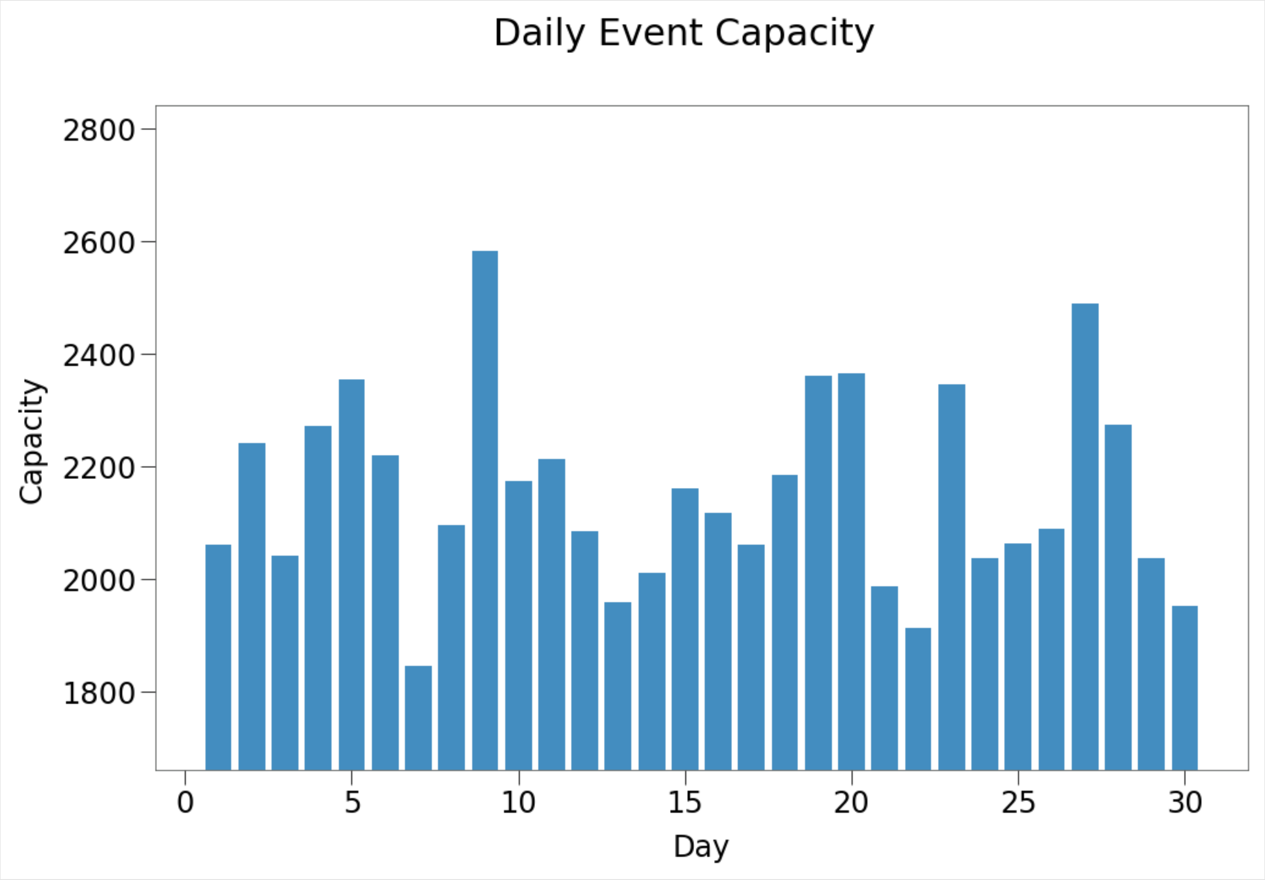
With the number of events for each capacity determined, we can assign event objects.

To do so, we:

- utilize the *“baseus” groups used in the SIR analysis*.
- assign a p-value of .98 to the mover function of group.
    - *determined through a trial and error process that best fit the Hutch distribution.*
- randomly assign each event a start tick between 1 and 30, which is the length of the simulation.
- assign each event to `vbox=0`
- as all events will occur inside a vbox, a `Travel` event must be used to transport them to the event location.

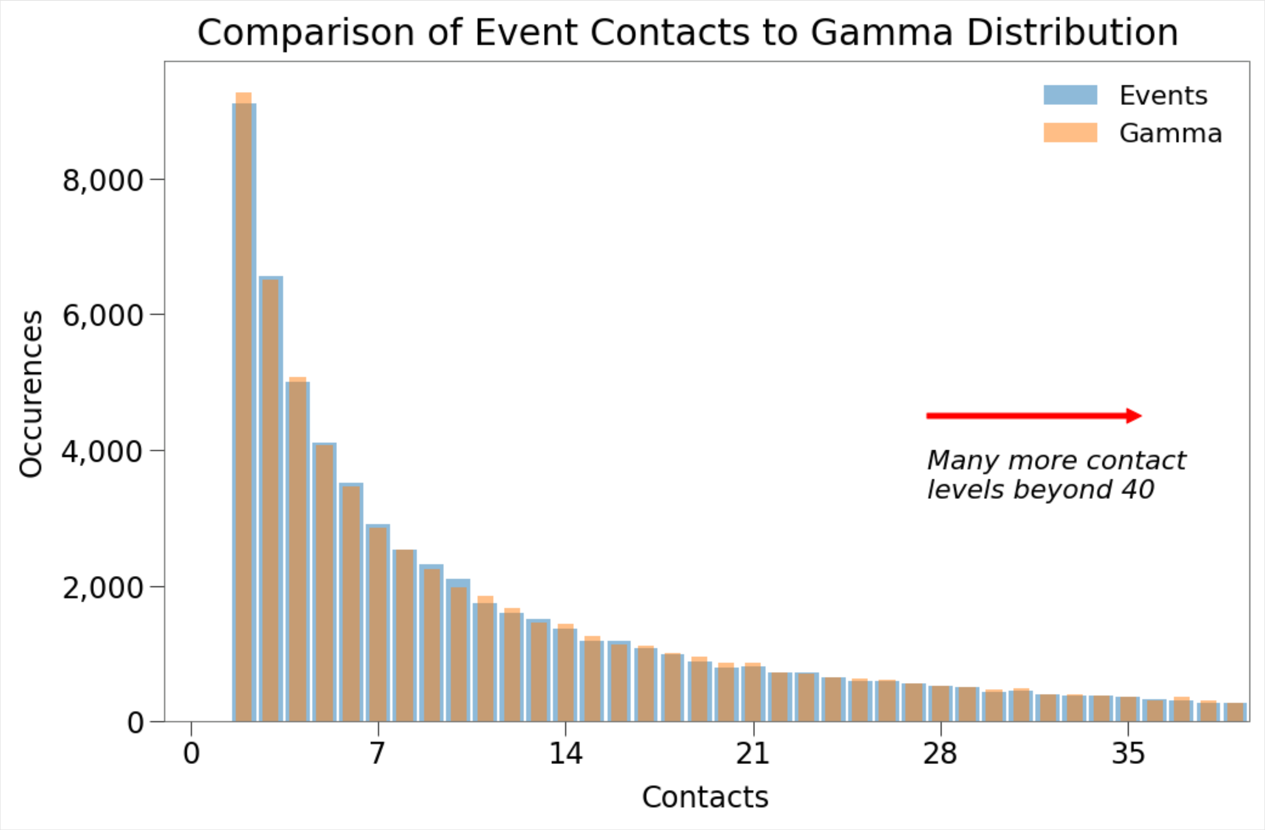
```
from rknot.sims import baseus
from rknot.events import Travel

groups = baseus.groups
groups[2]['n_inf'] = 0
for group in groups:
    group['mover'] = .98

params = {'days': days, 'tmr_curve': sim_tmr, 'density': density, 'sterile': True}

event_groups = []
for i in np.arange(n_events.shape[0]):
    if n_events[i] >= 1:
        event_group = {'name': f'{i+gloc+1}n', 'n': n_events[i], 'groups': [0,1,2,3],
↪'capacity': i+gloc+1}
        event_groups.append(event_group)

events = []
for i, e in enumerate(event_groups):
    for j in range(e['n']):
        start_tick = np.random.randint(1, 31, 1, dtype=np.int32)
        events.append(
            Travel(name='{}_{}'.format(e['name'], j), start_tick=start_tick[0],␣
↪groups=e['groups'], capacity=e['capacity'], vbox=0)
        )
```

In the chart below, we can see the distribution of the capacity of the events generated by this process over the life of the simulation.

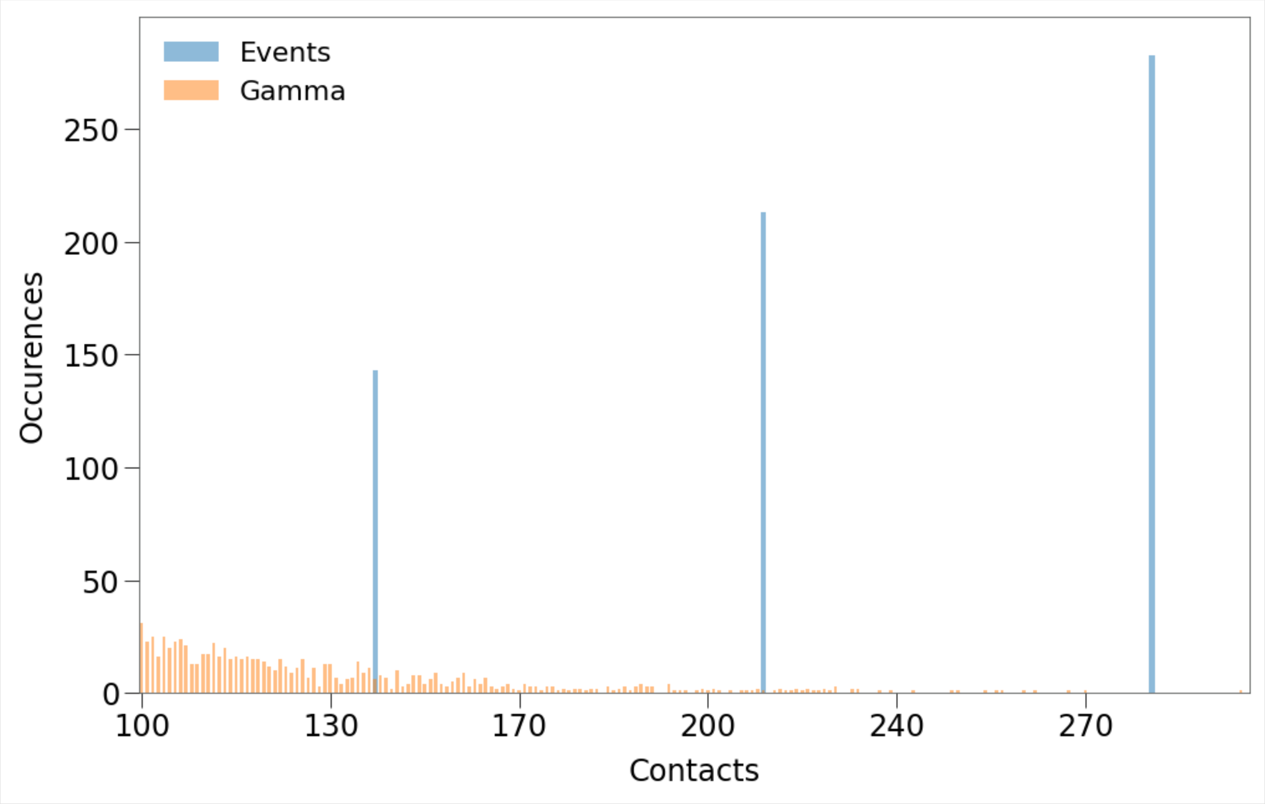


So, for the length of the simulation between 18 to 26% of the population will be attending events *each day of the simulation*.

Below, we can see that for the lower contact levels, this approach generates a near perfect replica of the required Gamma distribution.



When we focus on contact levels of 100+, however, you can see that we only generate contacts at 3 levels.



### Forming the Remaining Environment

With contact levels 2+ settled, we must set the non-event environment such that it generates mainly 0 contacts.

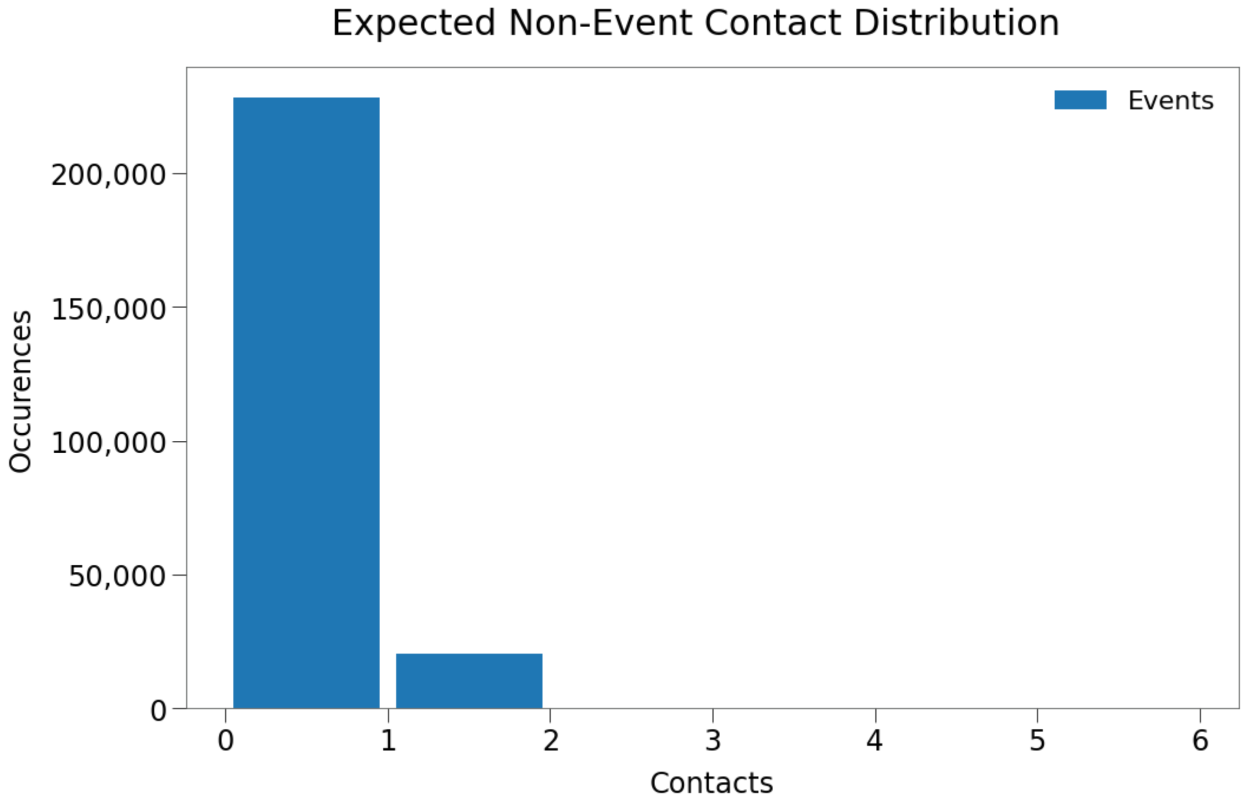
First, we find the average density of the grid (excluding the vbox) during the sim. Only dots not attending events will be in the main grid, so density is as follows:

```
[51]: n_dots_at_events = np.array(list(c.values())).mean()
      n_dots_not_events = n - n_dots_at_events

      d = n_dots_not_events / n # given density of 1

      print (np.around(d, 2))
```

```
0.78
```

The density of the grid excluding dots on events will be 0.78.So we can find the expected $\overline{k}$ of that space using the *polyfit equation we found above* to find the corresponding variance and providing the `p-value` for group movement. And then apply $\overline{k}$ to a normal distribution to find the expected contacts for the non-event space.

```
[54]: f = np.poly1d(d*z)
      mu_space = f(.98)
      var_space = mu_space
      x = np.linspace(0, 5, 6)
      c_non = st.norm.pdf(x, mu_space, np.sqrt(var_space))
      c_non = d*n*30*c_non
```



```
[57]: text = 'And so the expect 0 contacts occurences are right in line with the target: '
      text += f'\n\n {c_non[0]:,.0f} vs. {c_gam[0]:,.0f}'
      text += '\n\nThere is greater delta in the 1 contact occurences but still fairly␣
       ↪close: '
```

(continues on next page)

(continued from previous page)

```
text += f'\n\n {c_non[1]:,.0f} vs. {c_gam[1]:,.0f}'

md(text)
```

And so the expect 0 contacts occurences are right in line with the target:

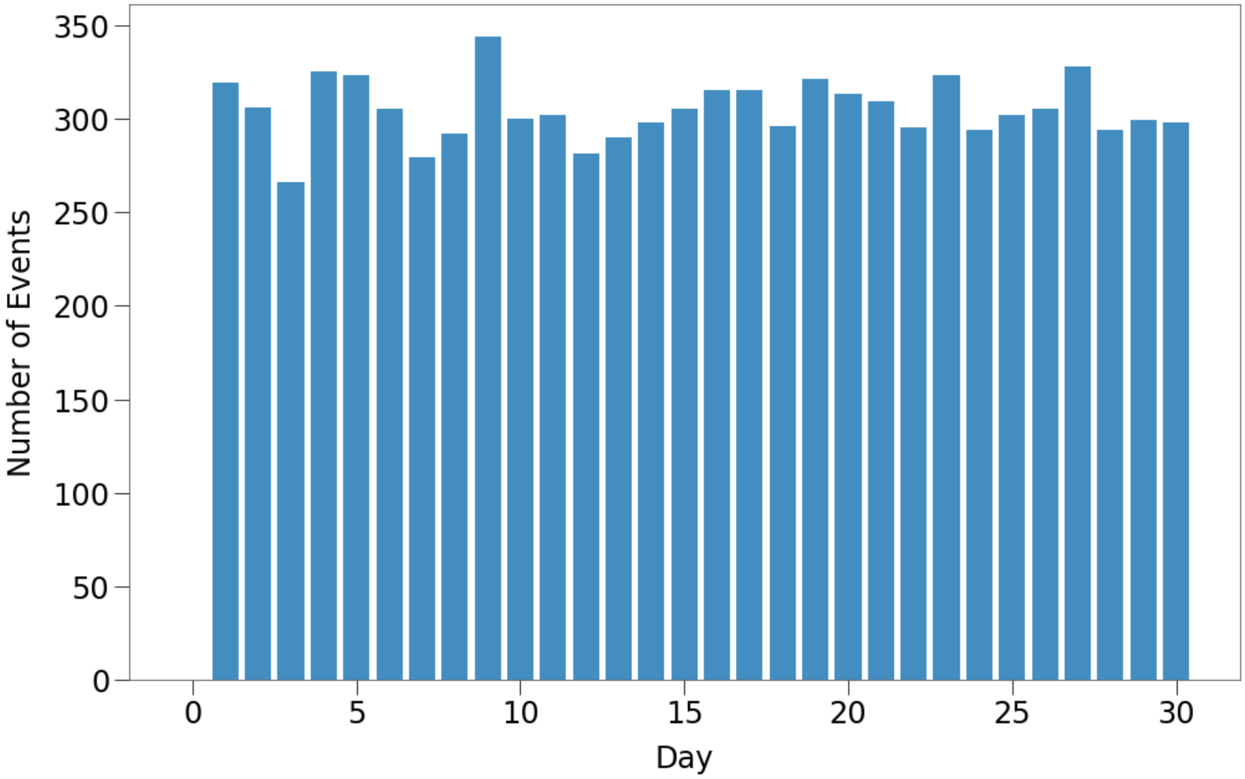
228,464 vs. 217,740

There is greater delta in the 1 contact occurences but still fairly close:

20,230 vs. 15,090

Thus, we expect the non-event space will result in 200,000+ dot days with no contacts, right in line with Hutch gamma.

The final step in preparing the simulation environment is determing the size of the VBox. The chart below shows the number of events occuring on each tick in the Sim.



Above we can see that there are typically ~300 events occuring on a given tick.

The vbox must allow for each event to have its own location, so the vbox should have enough locations to support the maximum number of events occuring on the same tick, which is:

[59]:

```
start_ticks = np.array([e.start_tick for e in events])
n_events_by_tick = np.bincount(start_ticks)
n_events_by_tick.max()
```

[59]:

```
344
```

The vbox must have 344 locations.

Run the Sim

We will run a single Sim to use for `all_contacts` comparison:

```
vbox = n_events_by_tick.max()
sim = Sim(groups=groups, events=events, vboxes=vbox, **params)
sim.run(dotlog=True)

all_contacts = find_all_contacts(sim.dotlog, MLNB)
eR0_all = np.sum(all_contacts[:,:sim_tmr.shape[0]]*sim_tmr, axis=1).mean()
```

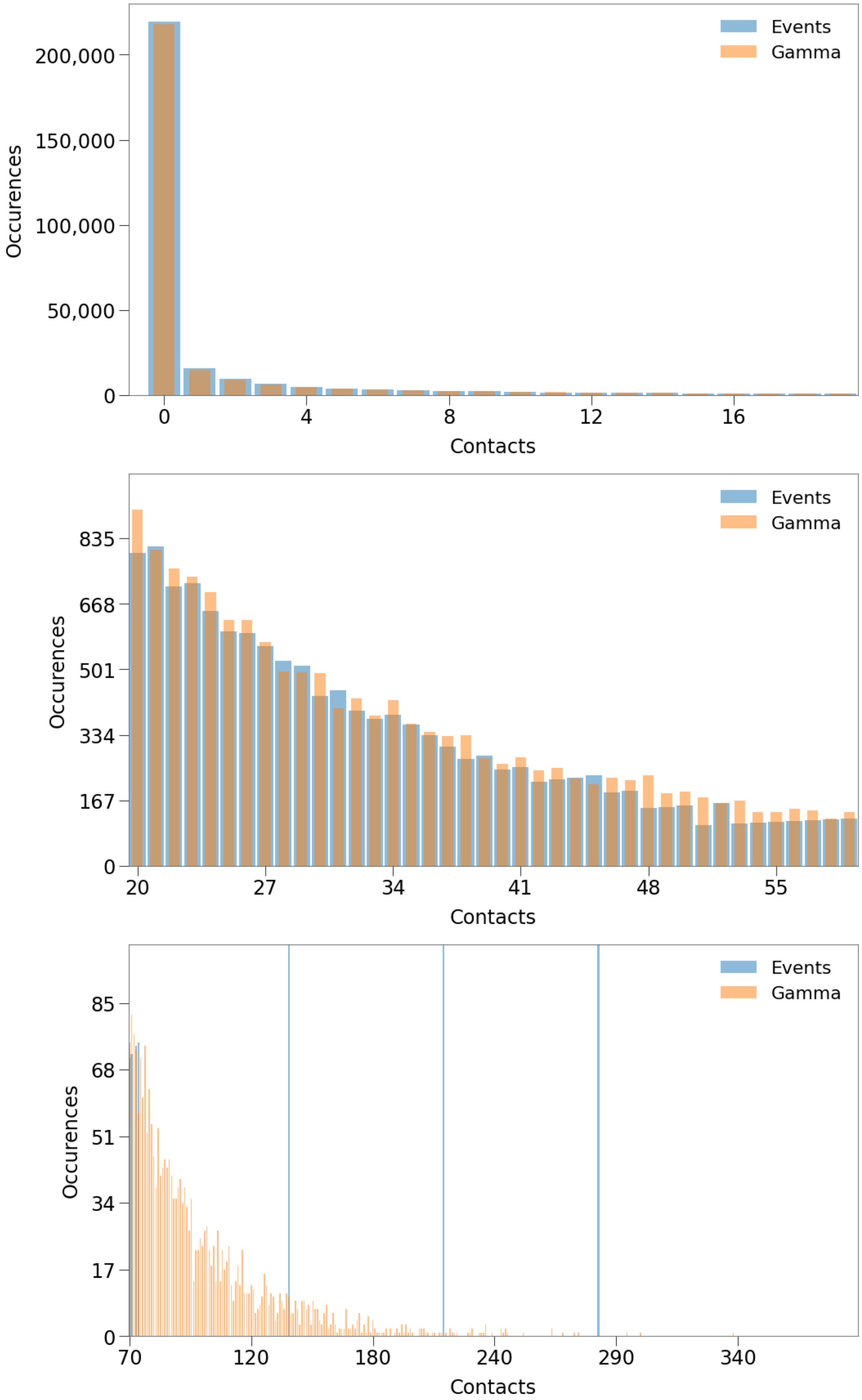
We can see from the outputs that the sim produces fairly similar outcomes to those desired.

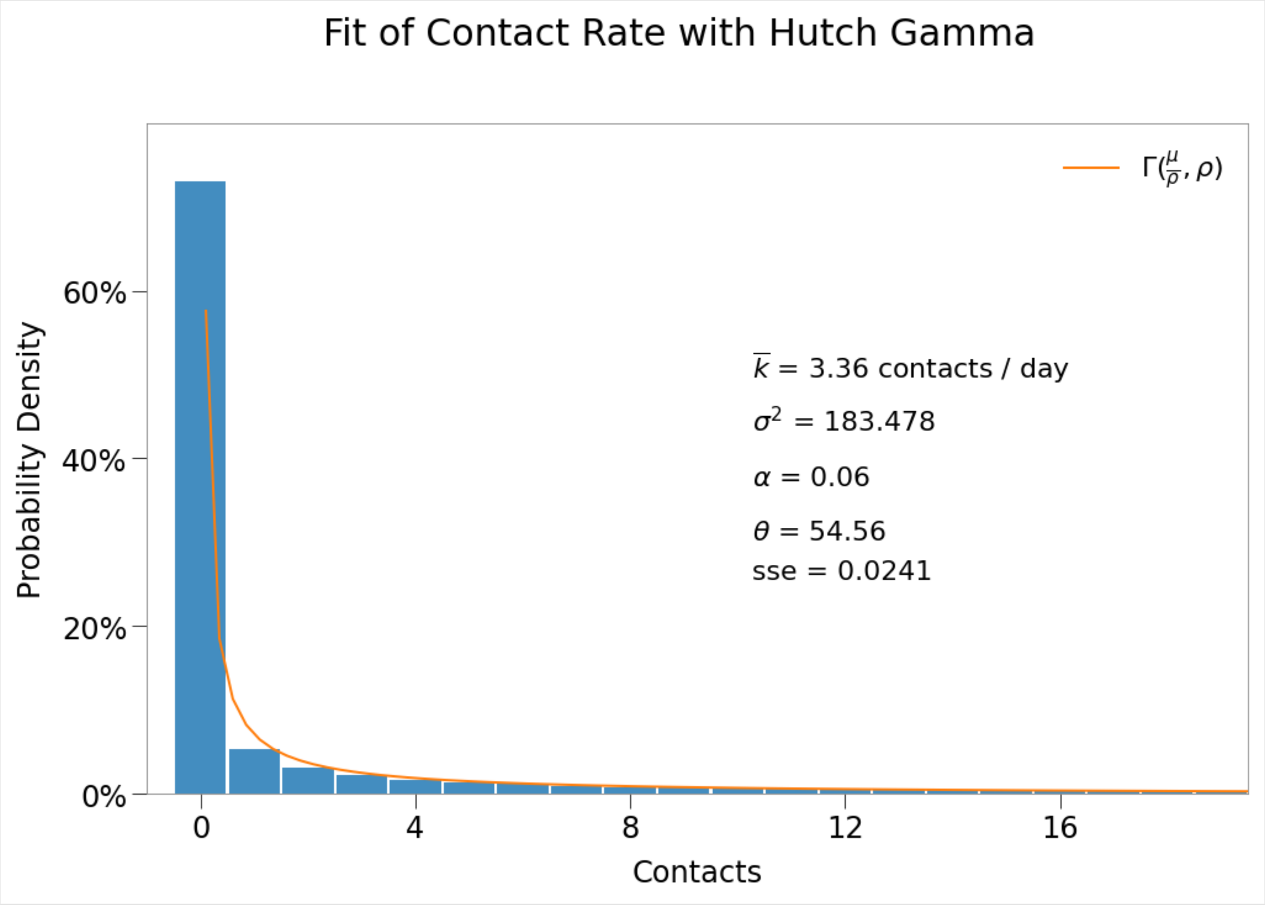
Results

The sim reproduced $R_0$ as desired, however, contact rate is slightly below target and theta slightly over. This is likely the result of the long-tail issues described earlier.

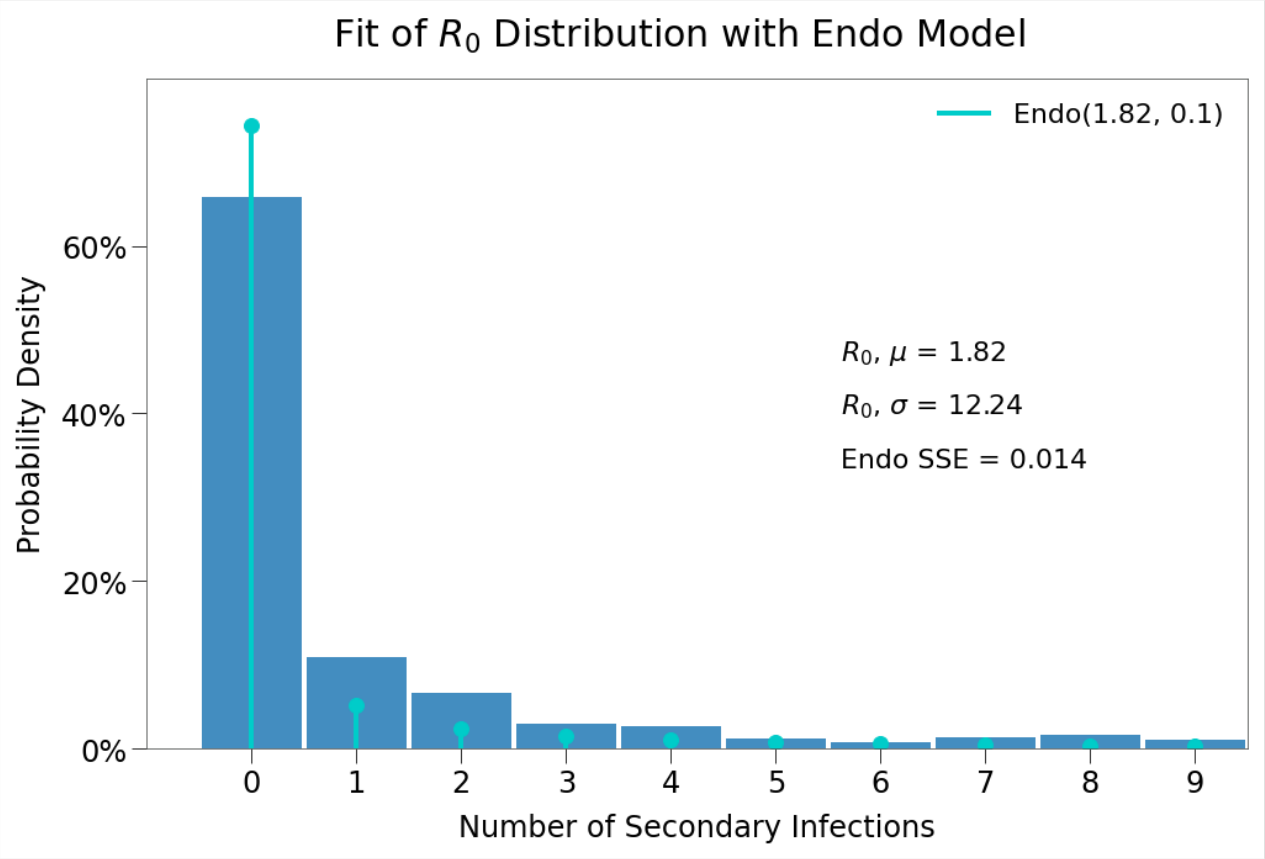
```
<IPython.core.display.HTML object>
```

And the contact distribution visually fits the desired Gamma almost perfectly (except for contact levels > 100 as previously noted).


Events
Gamma
Occurences
200,000
150,000
100,000
50,000
0
0
4
8
12
16
Contacts
Events
Gamma
Occurences
835
668
501
334
167
0
20
27
34
41
48
55
Contacts
Events
Gamma
Occurences
85
68
51
34
17
0
70
120
180
240
290
340
Contacts



And that the expected $R_0$ distribtion is a much stronger fit for the Endo model (although still underweight slightly to 0 infection outcomes).



Run a Larger Dataset

The `all_contacts` method is good way to get a quick read on the viability of an environment, however, it is still only a rough estimate. Truly reliable statistics can only come from running the simulation many times.

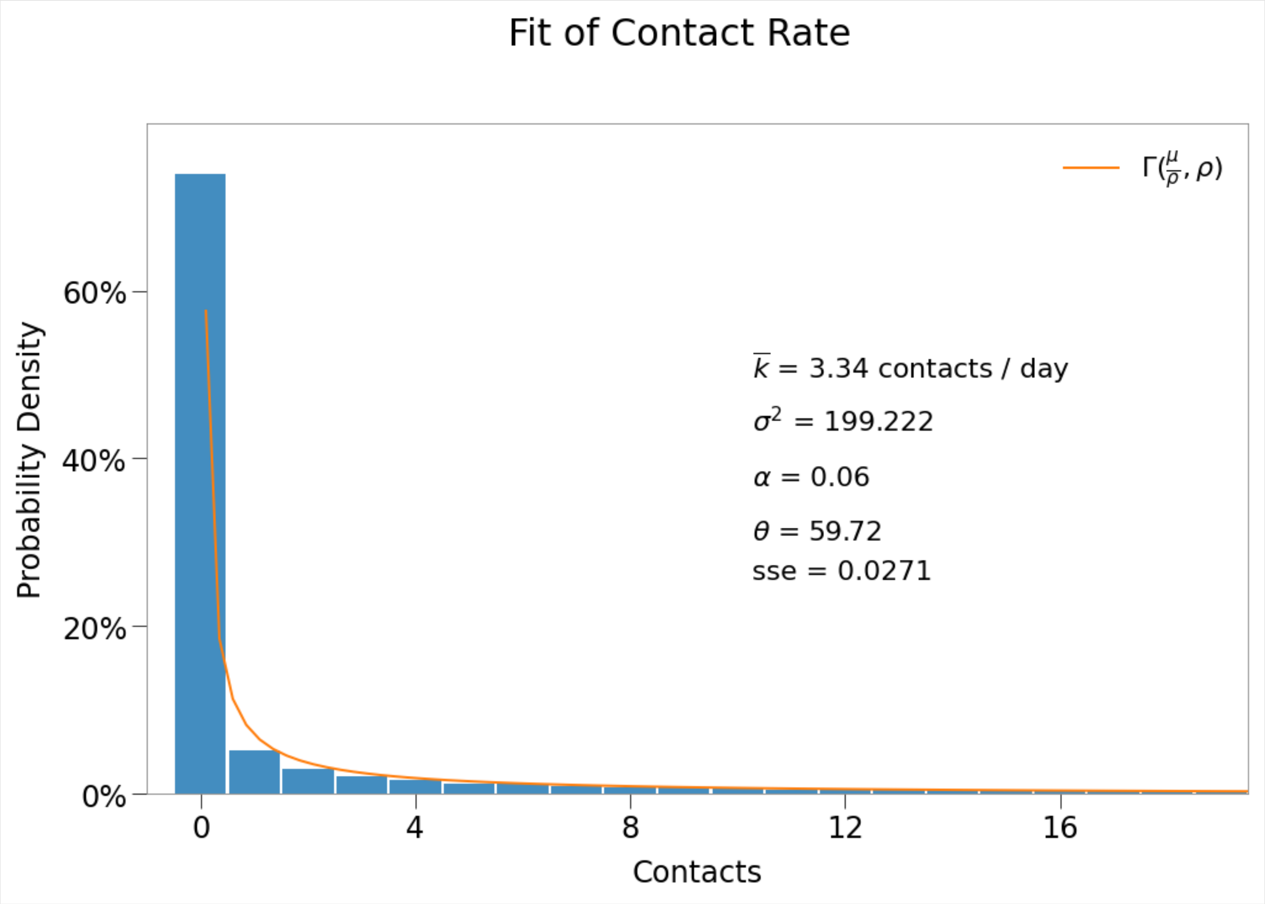
We will loop through 1,000 sims:

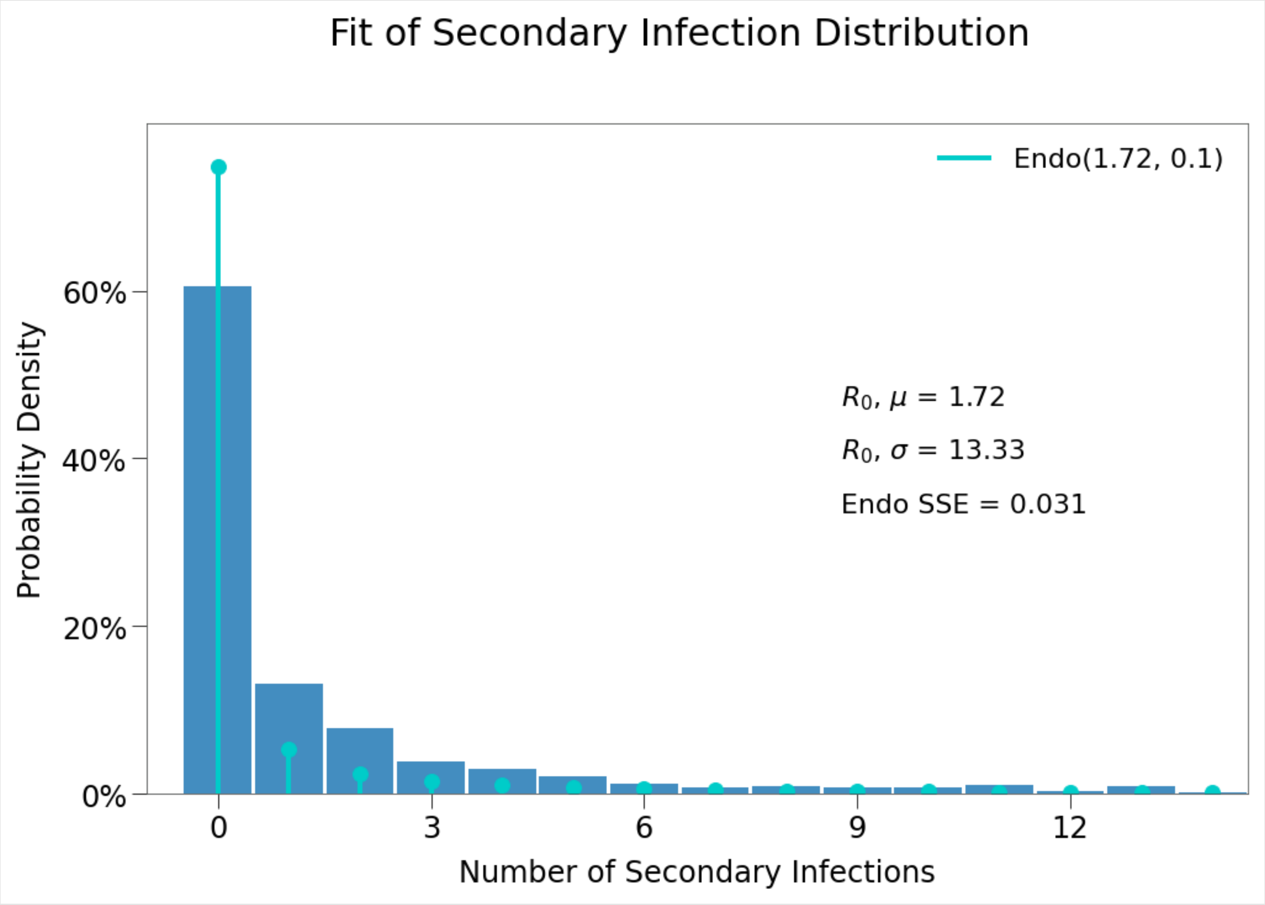
```
n = 1000
contacts, nsecs, results, exceptions = looper(groups, n, params['density'], params[
↪'days'], tmr=sim_tmr, events=events, vboxes=vbox)
```

Results

```
[68]: eR0 = np.mean(contacts.reshape(-1,sim_tmr.shape[0])*sim_tmr, axis=0).sum()
      R0 = nsecs.sum(axis=1).mean()
```

```
<IPython.core.display.HTML object>
```


Fit of Contact Rate
$\Gamma(\frac{\mu}{\rho}, \rho)$
$\overline{k}$ = 3.34 contacts / day
$\sigma^2$ = 199.222
$\alpha$ = 0.06
$\theta$ = 59.72
sse = 0.0271
Probability Density
60%
40%
20%
0%
0
4
8
12
16
Contacts

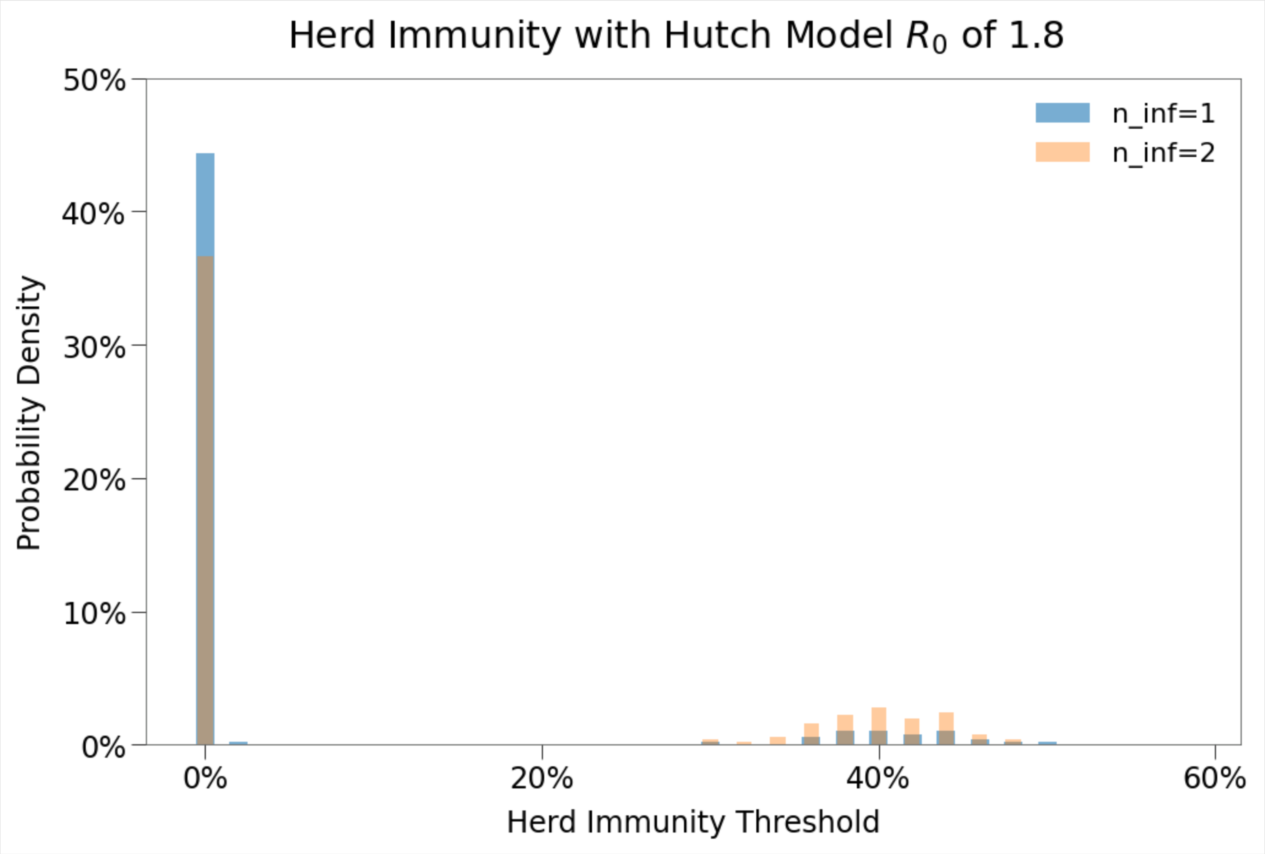


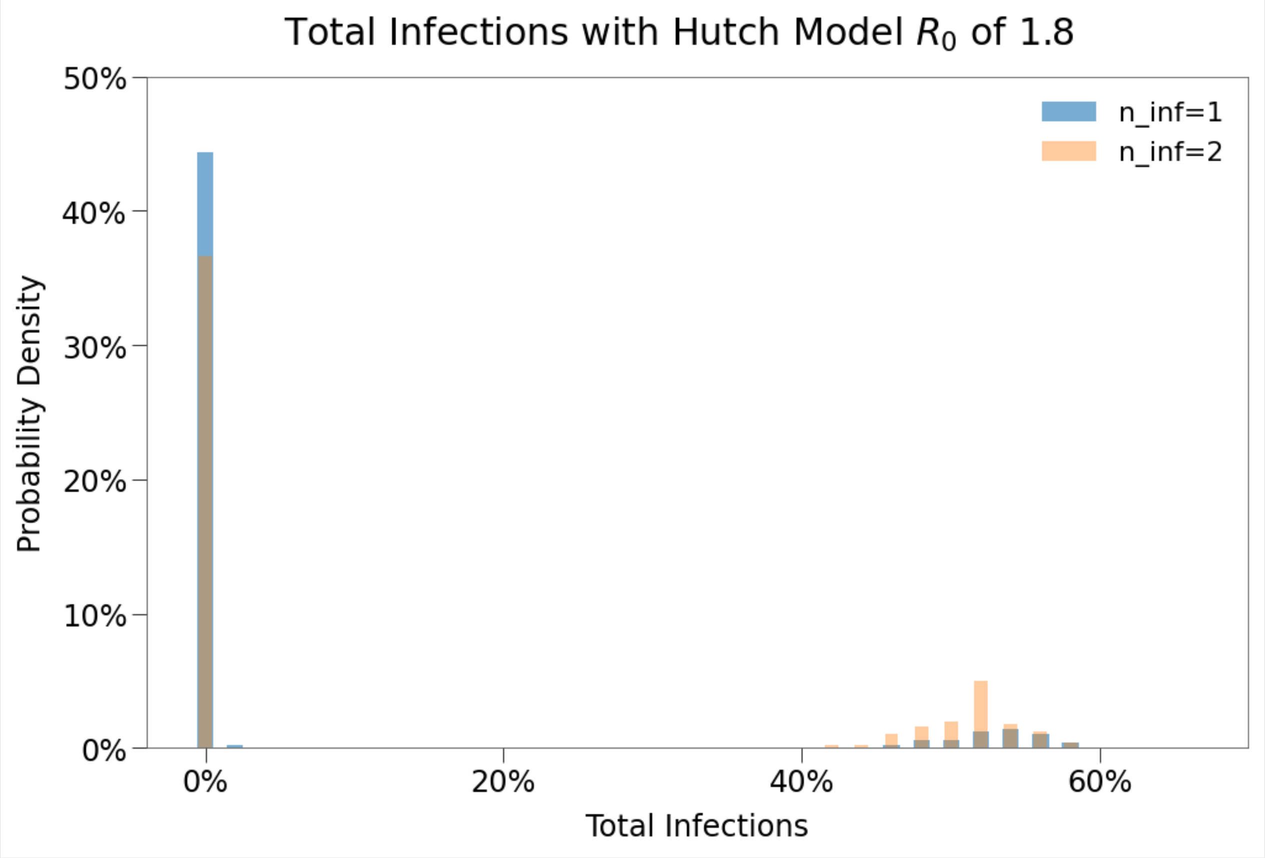
The above iterations set `sterile=True`, in order to isolate the contacts and secondary infections of *just the initial infected*.

We can learn more about the outcomes in the simulation by again using `looper` and setting `sterile=False`, `days=365`. We will complete just 250 iterations, because each sim will run longer.

```
n = 250
days = 365
contacts, nsecs, results, exceptions = looper(groups, n, density, days, tmr=sim_tmr,␣
↪events=events, vboxes=vbox, sterile=False)
```

```
<IPython.core.display.HTML object>
```


Herd Immunity with Hutch Model $R_0$ of 1.8
50%
40%
30%
20%
10%
0%
Probability Density
0%
20%
40%
60%
Herd Immunity Threshold
🟦 n_inf=1
🟧 n_inf=2


Total Infections with Hutch Model $R_0$ of 1.8
50%
40%
30%
20%
10%
0%
Probability Density
0%
20%
40%
60%
Total Infections
🟦 n_inf=1
🟧 n_inf=2

CHAPTER 5

# SIR: Factors Influencing Spread

**REPLICABILITY**

*Randomness is in an important factor in RKnot's approach to simulation (and frankly in real-word viral transmission), so the results of the sims below will not be repeatable with each iteration. The below examples are meant to show general differences based on state; further analysis should run the same simulation multiple times to see the mean impact.*

## 5.1 Base US Simulation

To explore the various concepts of RKnot and viral spread, we'll use a simulation design based on CDC Best Planning Scenario guidelines for COVID-19 characteristics including:

- $R_0$ 2.5
- IFR for each of 4 age groups
  - 0-19: 0.003%
  - 20-49: 0.02%
  - 50-69: 0.5%
  - 70+: 5.4%

Other assumptions:

- Population of $10,000$[1]
- Initial Infected of 2
- Duration of Infectiousness 14 days[2]
- Duration of Immunity 365 days
- Density of ~1 subject per location (`dlevel='med'`)

[1] *proportionately split among the 4 age groups to match US census data.*

[2] *equal likelihood of transmission on any day (i.e. no viral load curve)*

US Census Data

## 5.2 Natural

### 5.2.1 1. Equal

The first simulation makes the most homogeneous assumptions.

- No group is restricted in terms of movement.
- All dots are able to interact with one another.
- All dots are susceptible at initiation.
- All dots are equally likely to move to any dot on the grid (`mover='equal'`)

The basic layout is below. These parameters can also be imported from `rknot.sims.baseus` for convenienve.

```
group1 = dict(
    name='0-19',
    n=2700,
    n_inf=0,
    ifr=0.00003,
    mover='equal',
)
group2 = dict(
    name='20-49',
    n=4100,
    n_inf=1,
    ifr=0.0002,
    mover='equal',
)
group3 = dict(
    name='50-69',
    n=2300,
    n_inf=1,
    ifr=0.005,
    mover='equal',
)
group4 = dict(
    name='70+',
    n=900,
    n_inf=0,
    ifr=0.054,
    mover='equal',
)
groups = [group1, group2, group3, group4]
params = {'dlevel': 'med', 'Ro':2.5, 'days': 365, 'imndur': 365, 'infdur': 14}
```

We instantiate a new sim by passing `groups` and `params`. We can also flag `details` to get some information about the Sim structure.

```python
from rknot import Sim, Chart
sim = Sim(groups=groups, details=True, **params)
sim.run()
```

```
-------------------------------------------------------------------------------------
|                                    SIM DETAILS                                    |
|-----------------------------------------------------------------------------------|
|            Boundary|      [ 1 45   1 45]|           Locations|                2,025|
|--------------------|--------------------|--------------------|--------------------|
|          Population|              10,000|             Density|                 4.94|
|--------------------|--------------------|--------------------|--------------------|
|        Contact Rate|                4.94|                    |                     |
|--------------------|--------------------|--------------------|--------------------|
```

Running the animation will result in a video as per below:

```python
chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

alternative text

*Note embedded videos are used for convenience purposes. Given the random processes involved, running the same code will produce slightly different results each time.*

Results:

| | |
|---|---|
| Peak | 37% |
| HIT | 67% |
| Total | 87% |
| Fatalities | 0.55% |
| % > 70 | 42% |
| IFR | 0.63% |
| Days to Peak | 72 |

In this scenario, **RKnot** fairly closely replicates the curve of a standard SIR model, which expects HIT of 60% for $R_0$ of 2.5 (HIT = 1 - 1 / $R_0$).

Variation from the standard SIR model will always result given:

1. In the simulation, movement and transmission are stochastic processes.
2. This sim does not have an entirely homogeneous population, with different IFRs and varying numbers of contacts between subjects.

### 5.2.2 2. Local

In remaining simulations, we begin to introduce ever increasing homogeneity.

Our first change is to adjust the subjects mover functions to `local`. The `local` mover has a strong bias towards locations only in its immediate vicinity, which is a better approximation of real world processes (though certainly not a perfect one).

We will also extend the simulation an extra year.

```python
group1['mover'] = 'local'
group2['mover'] = 'local'
group3['mover'] = 'local'
group4['mover'] = 'local'
```

(continues on next page)

(continued from previous page)

```
groups = [group1, group2, group3, group4]
params['days'] = int(365*2)

sim = Sim(groups=groups, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed To Load

Results:

| | |
|---|---|
| Peak | 4% |
| HIT | 56% |
| Total | 69% |
| Fatalities | 0.42% |
| % > 70 | 36% |
| IFR | 0.60% |
| Days to Peak | 355 |

Compared to *Example 1*, we can see that restricting movement has a major impact on spread. The curve is flattened and extended with total infected, peak, HIT, and fatalities all reduce.

But the virus is never completely eradicated and instead moves in a progressive wave across the grid space. Note that the infection and fatalitiy levels are somewhat artificially reduced as the virus still has a large susceptible population in the upper portion of the grid that it has not yet reached.

Local movement does *not* satisfy the "well-mixed" condition of the SIR model. Particularly interesting is that the *initial conditions of this scenario *DO* very closely resemble a SIR model*. So, an observer measuring $R_0$ in a local environment might see spread consistent with a well-mixed population *but the population may not be well-mixed at a larger scale*.

### 5.2.3 3. Social

In this simulation, we set `mover=social` for just the 20-49 age group. This is a rough approximation of that group's real-world propensity to travel more frequently (or go to more events).

```
group1['mover'] = 'local'
group2['mover'] = 'social'
group3['mover'] = 'local'
group4['mover'] = 'local'
groups = [group1, group2, group3, group4]

sim = Sim(groups=groups, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed To Load

Results:

(continues on next page)

(continued from previous page)

| | |
|---|---|
| Peak | 34% |
| HIT | 62% |
| Total | 85% |
| Fatalities | 0.54% |
| % > 70 | 49% |
| IFR | 0.63% |
| Days to Peak | 71 |

Comparing again to *Example 1*, we can see how powerful mixing is within a population. Even with the majority of subjects moving only locally, a small group of subjects is moving more broadly across the space will significantly increase the amount of spread.

## 5.2.4 4. Pre-Existing Immunity

In this scenario, we adjust the susceptibility factor for just two groups by relatively small amounts as follows:

- 20-49: 80%
- 50-69: 65%

This means, in the inverse, that 10% and 25% of the subjects in the respective groups are already immune to the virus (whether through pre-existing T-cell immunity, anti-bodies, or otherwise).

The older group is assumed to have a lower susceptibility factor as it is more likely that older people will have had more exposure to similar viruses over their lifetime.

T-cell immunity to sars-cov-2 remains a controversial subject, but many studies have found prevalance of T-cells between 20% - 50% in *people unexposed to sars-cov-2*. It is suggested that exposure to "common cold" coronaviruses (or more dangerous ones) may convey this immunity.

```
group2['susf'] = .8
group3['susf'] = .65
groups = [group1, group2, group3, group4]

sim = Sim(groups=groups, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

| | |
|---|---|
| Peak | 19% |
| HIT | 44% |
| Total | 63% |
| Fatalities | 0.35% |
| % > 70 | 29% |
| IFR | 0.56% |
| Days to Peak | 93 |

Relative to *Example 3* above, pre-existing immunity would reduce all aspects of the spread curve signifcantly.

Note the reduction in fatalities results *even though the most susceptible group is assumed to NOT have pre-existing immunity*.

### 5.2.5 5. Events

Certainly, the vast majority of people in the US do not move in such pre-defined ways as set out by the `mover` function. In reality, people tend to move with a local bias with a small number of interactions, supplemented by larger movements to locations with a large number of interactions in a small amount of time.

In RKnot, we can simulate this with *Event objects*. And we will incorporate a number of them in this simulation.

First, we will reset our group parameters by importing from `sims.baseus`.

Then we will instantiate a handful of events recurring periodically over the duration of the sim.

```
from rknot.sims.baseus import params, groups

from rknot.events import Event

school1 = Event(
    name='school1', xy=[25,42], start_tick=2,
    groups=[0], capacity=10, recurring=2
)
school2 = Event(
    name='school2', xy=[78,82], start_tick=3,
    groups=[0], capacity=10, recurring=2
)
school3 = Event(
    name='school3', xy=[92,32], start_tick=4,
    groups=[0], capacity=7, recurring=2
)
game1 = Event(
    name='game1', xy=[50,84], start_tick=6,
    groups=[0,1,2,3], capacity=100, recurring=14
)
game2 = Event(
    name='game2', xy=[45,52], start_tick=5,
    groups=[0,1,2], capacity=76, recurring=14
)
game3 = Event(
    name='game3', xy=[12,87], start_tick=1,
    groups=[0,1,2], capacity=56, recurring=14
)
game4 = Event(
    name='game4', xy=[52,98], start_tick=3,
    groups=[1,2], capacity=113, recurring=28
)
concert1 = Event(
    name='concert1', xy=[20,20], start_tick=7,
    groups=[0,1], capacity=50, recurring=14
)
concert2 = Event(
    name='concert2', xy=[91,92], start_tick=28,
    groups=[1], capacity=50, recurring=14
)
concert3 = Event(
    name='concert3', xy=[62,38], start_tick=21,
    groups=[2,3], capacity=25, recurring=14
)
concert4 = Event(
    name='concert4', xy=[38,42], start_tick=14,
    groups=[1,2], capacity=50, recurring=14
```

(continues on next page)

(continued from previous page)

```
)
bar1 = Event(
    name='bar1', xy=[17,24], start_tick=4,
    groups=[1], capacity=5, recurring=3
)
bar2 = Event(
    name='bar2', xy=[87,13], start_tick=5,
    groups=[1], capacity=5, recurring=4
)
bar3 = Event(
    name='bar3', xy=[52,89], start_tick=6,
    groups=[1,2], capacity=5, recurring=3
)
bar4 = Event(
    name='bar4', xy=[16,27], start_tick=7,
    groups=[1,2,3], capacity=4, recurring=7
)
bar5 = Event(
    name='bar5', xy=[89,46], start_tick=6,
    groups=[1,2], capacity=7, recurring=7
)
church = Event(
    name='church', xy=[2,91], start_tick=7,
    groups=[2,3], capacity=20, recurring=7
)

events = [
    school1, school2, school3, game1, game2, game3, game4,
    concert1, concert2, concert3, concert4,
    bar1, bar2, bar3, bar4, bar5, church
]

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

| | |
|---|---|
| Peak | 29% |
| HIT | 56% |
| Total | 82% |
| Fatalities | 0.50% |
| % > 70 | 39% |
| IFR | 0.61% |
| Days to Peak | 77 |

We've attempted to tailor the event setup so that the infection curve resembles that of *Example 1*. The theory is that the $R_0$ measured in the early stages of a pandemic should be reflective of actual contacts, not the theoretical contacts of the model.

This scenario is not perfectly substitutable with Example 1, however.

One measure we can use to compare scenarios is the total number of interactions. We find that under the Event and

Gated approaches it takes *more* contacts to result in the same level of spread.

Avg Number of Contacts per Subject:

Example 1: 50.3

Example 2: 52.3

Example 3: 53.4

Example 5: 65.5

Example 6: 75.5

*Average of first 100 days across 5 sims for each scenario.*

Other issues with substitutability may exist and are being explored. SIR models determine R0 in the idealized environment of Example 1 and so may not be suitable for customized environments such as this one.

### 5.2.6 6. Gates

We can further improve the real world relevance of interactions by introducing *gates*. Subjects are not always freely able to interact with all other people in a population. Often their movement is restricted to within certain areas. Furthermore, other people's access into those areas is restricted.

Elderly people living in retirement homes or assisted living centers is an example. To simulate this, we will split `group4` into two separate groups.

- `group4a`
  - population of 600 (2/3s of `group4`)
  - IFR of 4.2%
  - move freely throughout the entire grid as previously
- `group4b`
  - population of 300 (1/3rd of `group4`)
  - IFR of 7.8%
  - movement restricted to 6x6 box

We have also adjusted the IFR on the basis that `group4b` is likely older and also probably more frail than `group4a`. IFRs approximate those found here.

In addition, we will add an event specifically for the new group inside the gate.

```
group4a = dict(
    name='70+',
    n=600,
    n_inf=0,
    ifr=0.042,
    mover='local',
)
group4b = dict(
    name='70+G',
    n=300,
    n_inf=0,
    ifr=0.0683,
    mover='local',
    box=[1,6,1,6],
```

(continues on next page)

(continued from previous page)

```
    box_is_gated=True,
)
groups = [group1, group2, group3, group4a, group4b]

church2 = Event(
    name='church2', xy=[2,3], start_tick=7,
    groups=[4], capacity=5, recurring=7
)

events_gated = [
    school1, school2, school3, game1, game2, game3, game4,
    concert1, concert2, concert3, concert4,
    bar1, bar2, bar3, bar4, bar5,
    church, church2,
]
```

Now, such elderly populations are entirely sealed of from the rest of the world. In fact, they are often visited by family members or friends. We can mimick this with the use of a Travel object.

In this sim, at least one person will enter into the `group4b` gate for a day only. And this will repeat every day of the sim.

```
from rknot.events import Travel

visit = Travel(
    name='visit', xy=[1,1], start_tick=3,
    groups=[1,2], capacity=1, duration=1, recurring=1
)
events.append(visit)

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

| | |
|---|---|
| Peak | 30% |
| HIT | 62% |
| Total | 83% |
| Fatalities | 0.52% |
| % > 70 | 40% |
| IFR | 0.62% |
| Days to Peak | 66 |

Again, the gated structure is intended to mimic *Example 1*, *Example 3*, and *Example 5*.

The use for Events and Gates will become clear when we explore the impact of *Policy Decisions*.

### 5.2.7 7. Pre-Immunity with Events and Gates

Now we can see how pre-immunity might impact viral spread in a population with more heterogeneous interactions.

```
group2['susf'] = .8
group3['susf'] = .65
groups = [group1, group2, group3, group4a, group4b]

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

| | |
|---|---|
| Peak | 20% |
| HIT | 42% |
| Total | 62% |
| Fatalities | 0.47% |
| % > 70 | 37% |
| IFR | 0.76% |
| Days to Peak | 86 |

Similar to *Example 4*, pre-existing immunity flattens the curve somewhat and results in a modest decrease in fatalities (even with an abnormally high IFR in this case).

### 5.2.8 8. Self Aware Social Distancing

In a self-aware population, we can also incorporate an assumption that certain members of the population will implement social distancing practices even in the absence of prescribed government policy. For example, individuals might wear masks or face shields while in public.

This is implemented via a SocialDistancing object, which reduces the transmission factor of the subjects in the applicable group.

```
from rknot.events import SocialDistancing as SD

sd = SD(name='6-feet', tmfs=[.975,.95,.75,.5], groups=[1,2,3,4], start_tick=5,␣
↪duration=90)
events.append(sd)

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

(continues on next page)

(continued from previous page)

| | |
|---|---|
| Peak | 28% |
| HIT | 55% |
| Total | 82% |
| Fatalities | 0.51% |
| % > 70 | 42% |
| IFR | 0.62% |
| Days to Peak | 81 |

Social distancing does flatten the infection curve, resulting in a modest decrease in peak infections and HIT and delaying the peak slightly. Fatalities are also reduced.

### 5.2.9 9. Self Aware Social Distancing + Pre-Immunity

In a self-aware population, we can also incorporate an assumption that certain members of the population will implement social distancing practices (even in the absence of prescribed government policy). For example, individuals might wear masks or face shields while in public.

This is implemented via a SocialDistancing object, which reduces the transmission factor of the subjects in the applicable group.

```
from rknot.events import SocialDistancing as SD

sd = SD(name='6-feet', tmfs=[.975,.95,.75,.5], groups=[1,2,3,4], start_tick=5,␣
↪duration=90)
events.append(sd)

sim = Sim(groups=groups, events=events, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

| | |
|---|---|
| Peak | 14% |
| HIT | 36% |
| Total | 60% |
| Fatalities | 0.33% |
| % > 70 | 27% |
| IFR | 0.55% |
| Days to Peak | 92 |

Here we see that just the combination of pre-immunity and a modest amount of social distancing reduces all aspects of the infection curve.

Note that, despite the continued visits from outside, an outbreak in the 70+G group did *not* occur until *after* the social distancing practices were ceased. This outbreak resulted in a small surge in cases (visible in a rebound in the curve at ~100 days) and a tripling in fatalities in a short period.

So social distancing practices were helpful, but only so long as they were maintained.

### 5.2.10 10. Isolation

TBD

## 5.3 Policy

With a more realistic model of subject interaction, we can begin to experiment with the impact of different policy measures.

*RKnot* can simulate policy measures via `Restriction`, `SocialDistancing`, and `Quarantine` objects. Further details *here*.

The simulations are based on *this scenario* and so the impact of the policy measures contemplated should be considered relative to that scenario.

The structure can be imported as follows:

```
from rknot.sims.baseus import params, events_gated, groups_gated
```

### 5.3.1 1. Restrict Large Gatherings

We'll start by simply restricting large gathers, which for this sim is any event with 10+ capacity (0.1% of the entire population). The restrictions will last for 120 days.

When considering capacity, remember that a 100,000-seat stadium in a 10 million person catchment represents 1% of the population.

We assume this policy are implemented on day 30, when the population finally realizes there is a pandemic and government has had time to implement prevention measures.

The restriction will last for 120 days.

```
from rknot.events import Restriction

large_gatherings = Restriction(
    name='large', start_tick=30, duration=120, criteria={'capacity': 10}
)
events_w_res = events_gated + [large_gatherings]

sim = Sim(groups=groups_gated, events=events_w_res, details=True, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

| | |
|---|---|
| Peak | 15% |
| HIT | 28% |
| Total | 73% |
| Fatalities | 0.45% |
| % > 70 | 38% |
| IFR | 0.62% |
| Days to Peak | 69 |

We see that the curve is flattened significantly during the restriction period. Infections have a much lower peak, BUT also a much longer tail (evident by the still high relatively high level of total infections).

This results as the event restriction is lifted, which leads to a slower rate of decline of the virus.

### 5.3.2 2. Social Distancing

We can mimick the implementation of Social Distancing policies in certain settings via the `SocialDistancing` object. Mask wearing, hand sanitizing, and 6-foot perimeters all provide varying levels of protection.

It is hard to estimate the degree of protection from each, and even harder in combination. For instance, this study found anywhere between a 1.1- and 55-fold reduction in exposure to influenza with varying mask designs.

So we provide `tmfs` here for illustration purposes only, attempting to catch all social distancing practices. We also provide different `tmfs` for the different age groups, meant to simulate adherence to policy.

The policy measure is implemented on day 30 and maintained for 120 days.

```python
from rknot.events import SocialDistancing as SD

sd = SD(
    name='all', tmfs=[.8,.8,.7,.65,.5],
    groups=[0,1,2,3,4], start_tick=30, duration=120
)
events_w_res = events_gated + [sd]

sim = Sim(groups=groups_gated, events=events_w_res, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

| | |
|---|---|
| Peak | 8% |
| HIT | 24% |
| Total | 68% |
| Fatalities | 0.39% |
| % > 70 | 33% |
| IFR | 0.57% |
| Days to Peak | 96 |

We can see that social distancing is certainly the most effective approach in terms of “flattening the curve” with the lowest HIT and peak infections seen thus far.

The social distancing methods are successful in preventing an outbreak among the 70+G group until ~120 days, well after peak infections are reach.

However, consistent with the *restriction on large gatherings*, the infection curve has protracted tail coinciding with the restriction being lifted. Over an extended time frame, the virus still infects a fairly large portion of the population.

### 5.3.3 3. Restrict Elderly Visits

We can restrict events by name by passing `name` key to `criteria`. We can do this to restrict the travel event to the `70+G` area.

The policy measure is implemented on day 30 and maintained for another 120 days. There will be no other restrictions.

```
no_visits = Restriction(
    name='no_visits', start_tick=30,
    duration=120, criteria={'name': 'visit'}
)
events_w_res = events_gated + [no_visits]

sim = Sim(groups=groups_gated, events=events_w_res, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load

Results:

| | |
|---|---|
| Peak | 27% |
| HIT | 54% |
| Total | 80% |
| Fatalities | 0.29% |
| % > 70 | 17% |
| IFR | 0.36% |
| Days to Peak | 77 |

By simply quarantining the elderly, we see the most dramatic reduction in fatalities of any scenario thus far. An outbreak NEVER occurs in the 70+G gated area, even AFTER the policy restrictions are lifted, because the virus has already been eradicated by that time.

We can see here that allowing a *high level of infection* among the **least susceptible** has resulted in a *low level of fatalities* among the **most susceptible**.

This is a controversial approach and does have difficult ethical implications, but its power to reduce death cannot be ignored.

### 5.3.4 4. Social Distancing and Restrict Elderly Visits and Large Gatherings

What happens if we combine the three approaches above?

```
large_gatherings = Restriction(
    name='large', start_tick=30, duration=120, criteria={'capacity': 10}
)
no_visits = Restriction(
    name='no_visits', start_tick=30,
    duration=120, criteria={'name': 'visit'}
)
sd = SD(
    name='all', tmfs=[.8,.8,.7,.65,.5],
    groups=[0,1,2,3,4], start_tick=30, duration=120
)
events_w_res = events_gated + [large_gatherings, no_visits, sd]

sim = Sim(groups=groups_gated, events=events_w_res, **params)
sim.run()
```

(continues on next page)

(continued from previous page)

```
chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load
Results:

| | |
|---|---|
| Peak | 24% |
| HIT | 55% |
| Total | 82% |
| Fatalities | 0.42% |
| % > 70 | 36% |
| IFR | 0.51% |
| Days to Peak | 199 |

Incredibly, the combination of restrictions leads to worse outcomes than any of the individual restrictions on their own.

While in place, the restrictions do tightly contain infections and signficantly delay the onset of the pandemic. But once they are lifted, the virus spreads unabated through a still highly susceptible population.

i.e. Not enough subjects have achieved immunity by the time the restrictions are lifted.

### 5.3.5 5. Quarantine

We can even mimick the impact of quarantines via the Quarantine object.

Here we show the impact of a 30-day quarantine for all groups in the Sim. We include a restriction on visits to the elderly during the quarantine.

```
from rknot.events import Quarantine

quarantine = Quarantine(
    name='all', start_tick=30,
    groups=[0,1,2,3,4], duration=30
)
no_visits = Restriction(
    name='no_visits', start_tick=30,
    duration=30, criteria={'name': 'visit'}
)

events_w_res = events_gated + [quarantine, no_visits]

sim = Sim(groups=groups_gated, events=events_w_res, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Video Failed to Load
Results:

(continues on next page)

(continued from previous page)

| | |
|---|---|
| Peak | 1% |
| HIT | 1% |
| Total | 1% |
| Fatalities | 0.00% |
| % > 70 | 0% |
| IFR | 0.00% |
| Days to Peak | 32 |

The pandemic never materializes in this simulation. The quarantine eradicates the virus swiftly upon implementation.

This is an interesting result. This obvioulsy did not occur in many places in the real world that took this approach. There are several possible reasons worth considering:

- Random Variation: if you run this simulation multiple times, you will note some sims where spread does occur.
- Scale: a larger simulation would increase the possibility that a very small number of subjects can continue to pass around the virus while it is muted in the broader population
- Adherence: real-world adherence to the implemented policies was much lower than this simulation suggests.
- Inconsistency: some areas implemented strict quarantines while others did not, and there was mixing among those populations.

### 5.3.6 6. Quarantine with Adherence

We can investigate the impact of low adherence to quarantine requirements by setting the `adherence` property of the `Quarantine` object.

```
from rknot.events import Quarantine

quarantine = Quarantine(
    name='all', start_tick=30,
    groups=[0,1,2,3,4], duration=30
)
no_visits = Restriction(
    name='no_visits', start_tick=30,
    duration=30, criteria={'name': 'visit'}
)

events_w_res = events_gated + [quarantine, no_visits]

sim = Sim(groups=groups_gated, events=events_w_res, **params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

```
[16]: chart_params
```

```
[16]: {'use_init_func': True,
       'show_intro': True,
       'dotsize': 0.1,
       'h_base': 10,
       'interval': 100.0}
```

Video Failed to Load

With just 10% of the population ignoring the quarantine requirements, the virus spreads almost as though there was no quarantine at all.

Video Failed to Load

The virus barely survives for the first 6-months of the outbreak, then as per other scenarios, once restrictions are lifted an outbreak occurs. Still, when the outbreak does occur, it is characterized by one of the lowest peaks and HITs in our analysis (due in part to the pre-immmunity of some of the groups).

And it achieves the lowest fatality rate of the group, mainly by restricting access to the elderly population for the duration of the pandemic and ensuring an outbreak never occurs in that region.

CHAPTER 6

# Dynamic Transmission Risk

## 6.1 Structure

In *SIR: Factors Influencing Spread*, we assumed a constant daily rate of transmission risk during the infectious period. This is in keeping with the SIR model, however, in reality we know that transmission risk is influenced by an infected person's viral load, which is a dynamic property.

Here we will demonstrate how to incorporate dynamic transmission risk and compare its impact to the outcomes of the SIR simulations.

The model and parameters used in this scenario are derived from a paper from the Fred Hutchinson Cancer Research Center, henceforth known as the Hutch model.

In particular, the Hutch model requires the following parameters in a Gamma-distributed contact regime:

- average of 4 contacts per day per subject, $\overline{k}$
- dispersion of 40, $\omega$
- These parameters should result in $R_0$ 1.8.

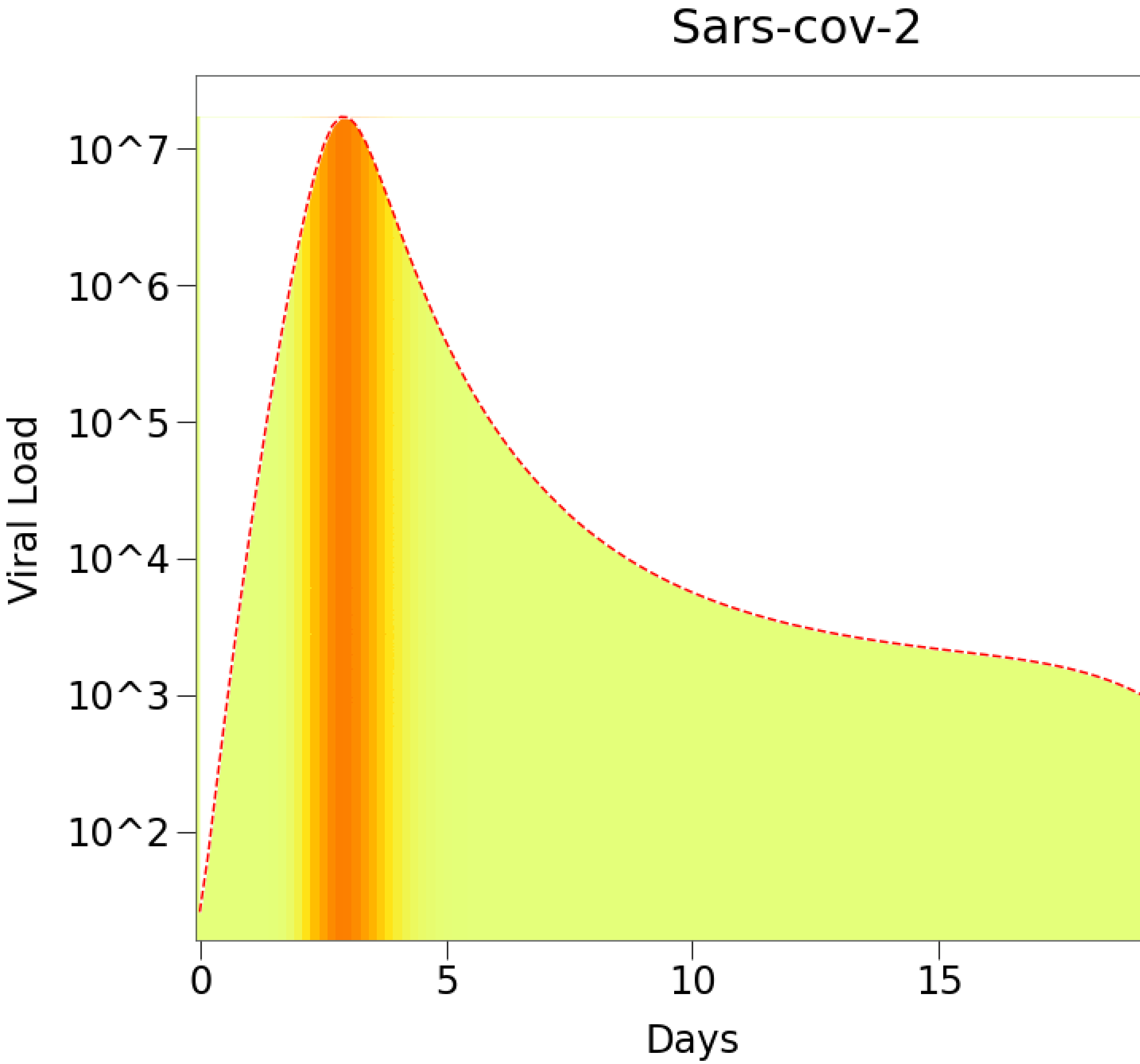
This is achieved using *Events*. See *Sizing* for a detailed demonstration.

The group structure continues to be informed by CDC Best Planning Scenario guidelines for IFR.

Other assumptions:

- Population of 10,000
  - proportionately split among the 4 age groups to match US Census data.
- Initial Infected of 2
- Duration of Immunity 365 days
- Density of 1 dot / location (excluding the vbox)
- a collection of 9,142 separate events, each recurring every 30 days.

The transmission risk curve used in these simulations is visualized below. See *Hutch Model* for the derivation.



## 6.2 Events

Here we show the basic outcome of dynamic viral load in a gamma-distributed contacts environment.

The transmission curve has bee calculated separately and is available as `tmr` from the `rknot.dots.fhutch` module.

All of the 9,000+ events occur in the “Events” vbox. All events have capacities of 3 subjects or more.

Again, details on the event structure are found *here*.

```
from rknot import Sim, Chart

from rknot.dots.fhutch import tmr
from rknot.sims.us_w_load_18 import events

group1 = dict(name='0-19', n=2700, n_inf=0, ifr=0.00003, mover=.98)
group2 = dict(name='20-49', n=4100, n_inf=1, ifr=0.0002, mover=.98)
group3 = dict(name='50-69', n=2300, n_inf=1, ifr=0.005, mover=.98)
group4 = dict(name='70+', n=900, n_inf=0, ifr=0.054, mover=.98)

vbox = {'label': 344, 'box': 344}
params = {'groups': groups, 'density': 1, 'days': 365, 'tmr_curve': tmr, 'vboxes':␣
↪vbox, 'events': events}

sim = Sim(**params)
sim.run()

chart = Chart(sim).to_html5_video()
```

```
<IPython.core.display.HTML object>
```

The results are:

```
<IPython.core.display.HTML object>
```

The results relative to the *SIR Events* simulation are lower across the board, which is mainly attributable to the lower $R_0$ utilized. With $R_0$ of 1.8, we would expect HIT of ~44%, which is close to the result here.

There are two other important differences:

- the peak occurs earlier
- there is a narrower range between peak/hit/total infections

These differences result for a couple reasons:

1. in the Hutch model used here, the infection duration is 30 days, double that used in SIR. Thus, as new infections occur, there are fewer recoveries.
2. while the infection duration is longer, the likelihood of infection is highly concentrated in the 3 to 5 day period of peak viral load. Thus, to achieve the same $R_0$, there must be more infections sooner. By the same token, as herd immunity is reached and more infections reach later life cycle, there are fewer infections to extend the tail.

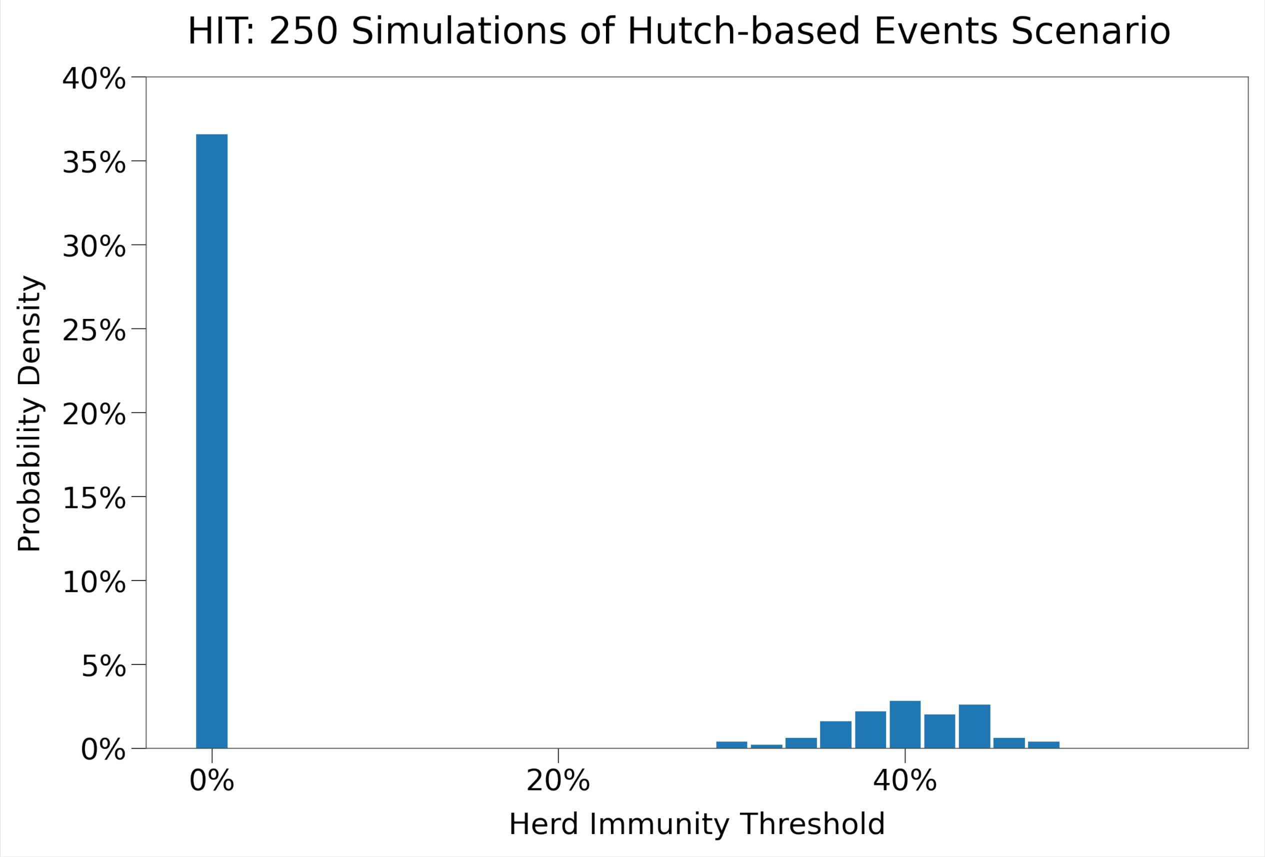
Fatality measures are inline with expectations from SIR model.

Using the `looper` function demonstrated in *Sizing*, we can quickly generate a sample of simulations to determine patterns.

The table below shows the results of 250 iterations of this scenario:

```
<IPython.core.display.HTML object>
```

The chart below shows the distribution of HIT for each of the 250 simulations. The vast majority of simulations resulted in no secondary infections (i.e. an outbreak never occured). Where an outbreak did take hold, outcomes centered around ~40% HIT.



## 6.3 Care Homes

We replicate the *SIR Gates scenario* by creating a separate group of elderly isolated within a gate, intended to simulate care homes or assisted living centers.

One augmentation is made: a small group of care home workers are added that will reqularly enter the gate to service residents. The care home workers will be drawn from the `20-49` age group.

The new groups are:

- `group2b`
  - population of 66
  - Assuming 2.2MM care home workers in the United States out of a population of 330MM.
  - remaining attributes similar to `20-49`
  - events individual travel events for half the group, recurring every day
- `group4a`
  - population of 600 (2/3s of `group4`)
  - IFR of 4.2%
  - remaining attributes mathcing prior `70+` group

- group4b
  - population of 300 (1/3rd of group4)
  - 25 locations
  - IFR of 7.8%
  - 'local' mover function
  - not eligible for any events

group4b used the local mover and its gated area has increased density. This is perhaps counter-intuitive. Care home residents are likely more well-mixed than the broader population and more closely follow normally-distributed contacts. So we approximate increased mixing with a higher p-value. Ideally, contact distribution in such environments would be researched for guidance.

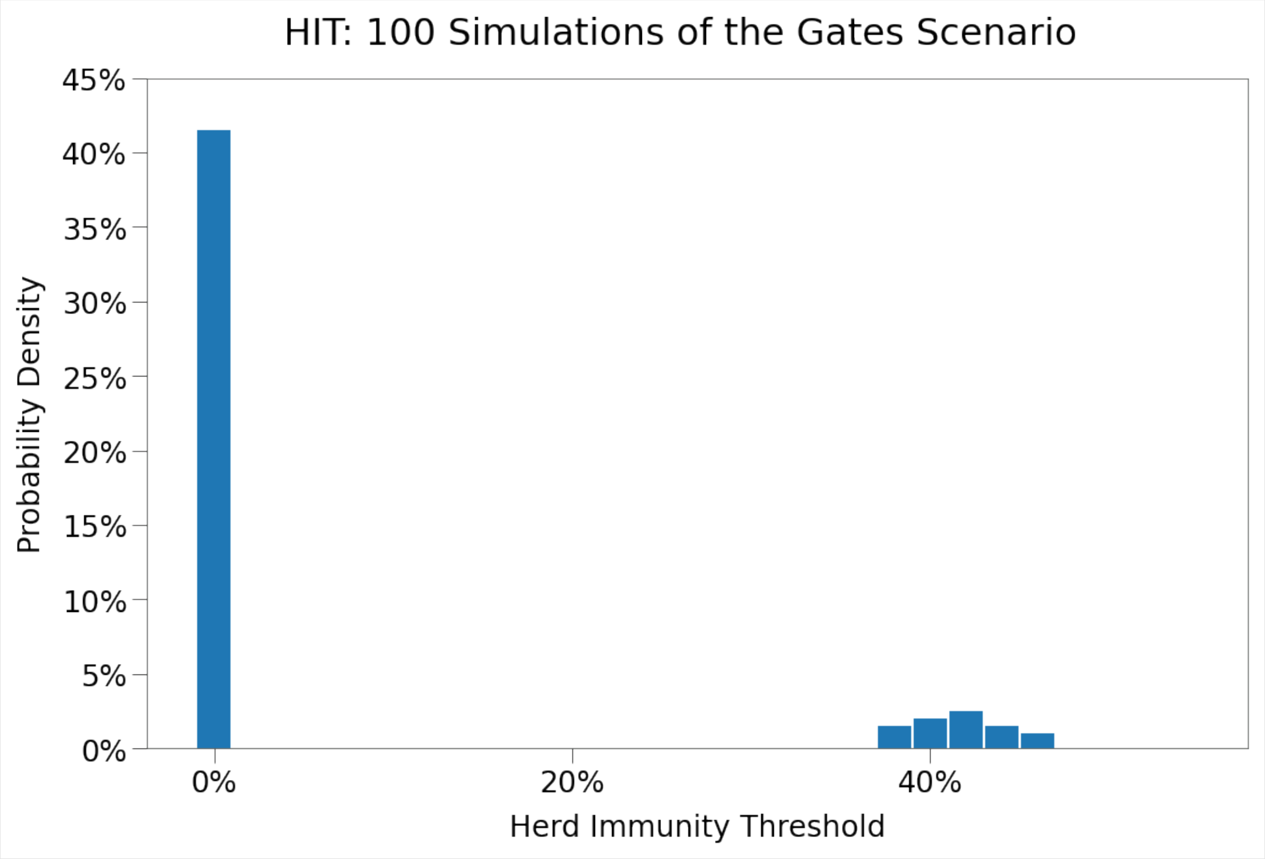
The event vbox has been adjusted to include the 5 groups that are not gated.

```
from rknot import Sim, Chart
from rknot.events import Travel
from rknot.dots.fhutch import tmr

from rknot.sims import us_w_load_18

group1 = dict(name='0-19', n=2700, n_inf=0, ifr=0.00003, mover=.98)
group2a = dict(name='20-49', n=4034, n_inf=1, ifr=0.0002, mover=.98)
group2b = dict(name='HCW', n=66, n_inf=0, ifr=0.0002, mover=.98)
group3 = dict(name='50-69', n=2300, n_inf=1, ifr=0.005, mover=.98)
group4a = dict(name='70+', n=600, n_inf=0, ifr=0.042, mover=.98)
group4b = dict(name='70+G', n=300, n_inf=0, ifr=0.0683, mover='local', box=[1,5,1,5],␣
↪box_is_gated=True)

groups = [group1, group2a, group2b, group3, group4a, group4b]

for e in us_w_load_18.events:
    e.groups = [0,1,2,3,4]

visit = Travel(name='visit', xy=[1,1], start_tick=3, groups=[1,3,4], capacity=1,␣
↪duration=1, recurring=1)

works = []
for i in range(group2b['n'] // 2):
    loc  = np.random.randint(1, 11, size=(2,))
    work = Travel(name=f'hcw-work-{i}', xy=loc, start_tick=1, groups=[2], capacity=1,␣
↪duration=1, recurring=1)
    works.append(work)

for e in events_gated:
    e.groups = [0,1,2,3,4]
events_gated = events_gated + [visit] + works

vbox = {'label': 'Events', 'box': 344}
params = {'groups': groups, 'density': 1, 'days': 365, 'tmr_curve': tmr, 'vboxes':␣
↪vbox, 'events': us_w_load_18.events}

sim = Sim(**params)
sim.run(dotlog=True)

chart = Chart(sim).to_html5_video()
```

This time, we will first run 100 simulations of the scenario to find the average outcomes. The average results are shown in the table below:

```
<IPython.core.display.HTML object>
```

Below we can see the distribution of HIT, which is very similar to the *Events*.



And now we generate a representative simulation:

```
<IPython.core.display.HTML object>
```

The results are shown below compared to SIR scenario:

```
<IPython.core.display.HTML object>
```

Again, relative to SIR, the simulation has a lower peak. In this instance the peak is also significantly later as well. Consistent with the Events scenario, a larger proportion of the fatalities are experienced among the elderly.

---

## 6.4 Capacity Restriction

As with the *SIR Model simulations*, we can explore the impact of policy restrictions on spread in our more sophisticated environment. First, we will restrict large gatherings.

We'll again restrict gatherings with 10+ capacity. We assume this policy is implemented on day 30. The restriction will last for 120 days.

With a gamma distributed contact distribution, a very large number of contacts can be eliminated by eliminating just a small fraction of events.

Below we see that events with capacity 10+ represent just 17.0% of all events:

```
import numpy as np

caps = np.array([e.capacity for e in us_w_load_18.events_gated])
n_events = caps.shape[0]
n_events_gt10 = np.bincount(caps)[10:].sum()
per = n_events_gt10 / n_events
```

```
<IPython.core.display.HTML object>
```

And that these events represent 84.4% of all contacts generated by those events:

```
counts, bins = np.histogram(caps-1, bins=np.arange(max(caps) + 1))
contacts = counts*(np.arange(counts.shape[0])+1)
c_gt10 = np.sum(contacts[10:]*np.arange(10, contacts[10:].shape[0] + 10))
c = np.sum(contacts*np.arange(contacts.shape[0]))
c_per = c_gt10 / c
```

```
<IPython.core.display.HTML object>
```

The updated structure is as follows:

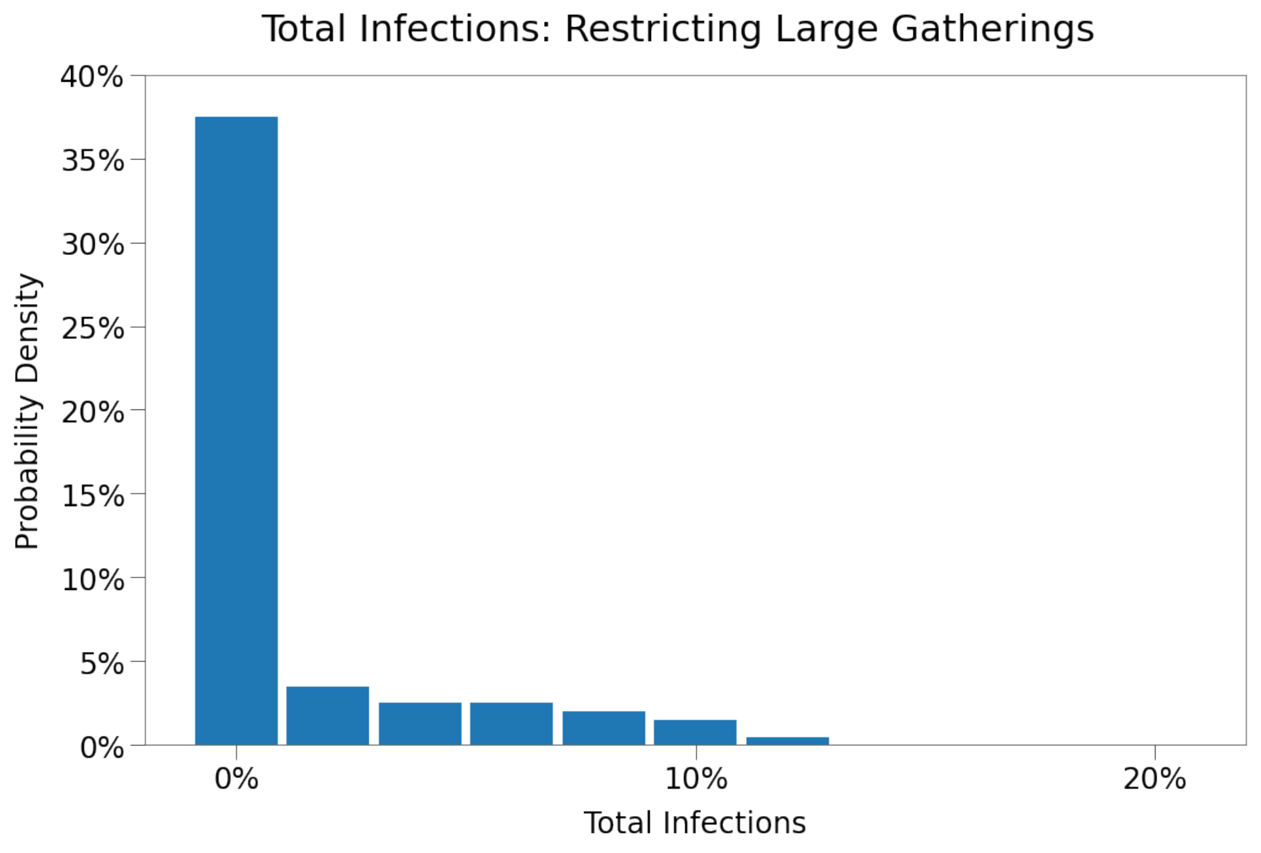
```
from rknot.events import Restriction

lg = Restriction(name='large', start_tick=30, duration=120, criteria={'capacity': 10})

events_w_res = events_gated + [lg]

params = {'groups': groups, 'density': 1, 'days': 365, 'tmr_curve': tmr, 'vboxes':␣
↪vbox, 'events': events_w_res}

sim = Sim(**params)
sim.run(dotlog=True)

chart = Chart(sim).to_html5_video()
```

We again show the average results of 250 simulations in the table below:

```
<IPython.core.display.HTML object>
```

We can see that restricting contacts to a maximum of 10 per day has a dramatic impact on spread, reducing the total amount of infections and significantly shortening the duration.

We can see below that the vast majority (almost 80%) of simulations resulted in no outbreak at all. Note, however, that some simulations resulted in moderate outbreaks of 10% total infections or more. So given enough iterations (say among different municipalities, provinces, states or countries), a significant outbreak is sure to occur despite the best efforts of policy.



```
<IPython.core.display.HTML object>
```

The results of the simulation are shown in the table below, compared with the same scenario from the SIR model simulations:

```
<IPython.core.display.HTML object>
```

The Hutch model results in a far more muted curve. This aligns with research that indicates super-spreader events (where a single individual is responsible for a large number of secondary infections) are responsible for the vast majority of secondary infections.

When those superspreader events are eliminated, spread is curtailed.

This result is somewhat confounding, however, given that many jurisidictions across the global have implemented a similar policy without such a dramatic impact.

Some reasons for this deviation may include:

1. few policies have truly limited all people to less than 10 contacts per day for 120 days. Often there have been exemptions for essential services. For example, during the second wave in North America, many children continued to go to school.
2. adherence to such policies is likely not 100%.
3. mixing amongst jurisdictions with different policies

### 6.4.1 Capacity Restrictions with Adherence

To investigate the impact of adherence (or lack thereof), simulations were ran for a dozen scenarios of different maximum event capacities and adherence factors (and some combinations). The results are shown in the tables below:

```
<IPython.core.display.HTML object>
```

And below the results of different adherence factors for a policy of maximum capacity 10. Adherence of 100 means full compliance and matches the first scenario above. Aherence of 0 means no compliance and outcomes match that of the *Gates scenario* above.

We have split the table in two sections, one with all simulations and the second showing only those with secondary infections.

```
<IPython.core.display.HTML object>
```

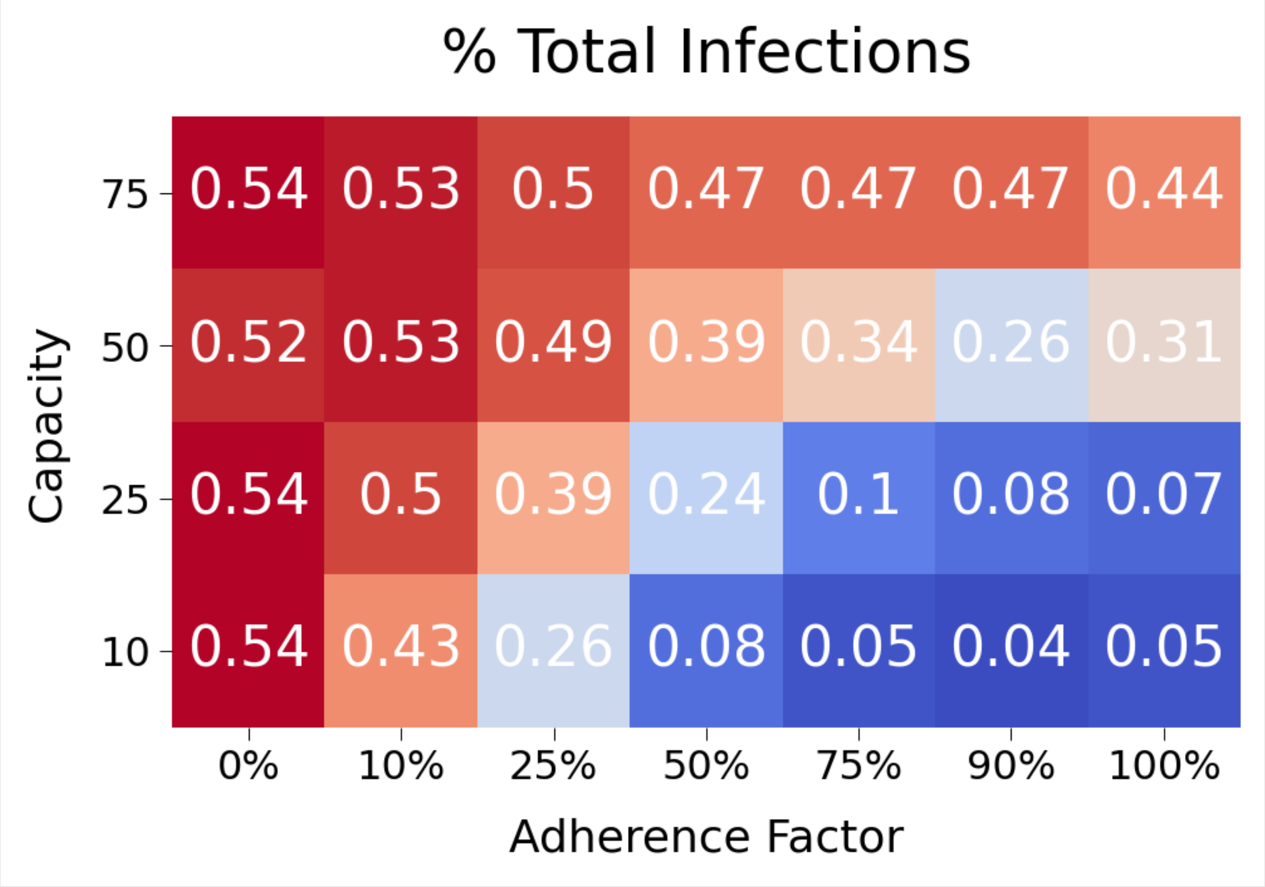
```
<IPython.core.display.HTML object>
```

Finally, we combined limited adherence with the other maximum capacity policies, as per the table below:

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

The above tables are a bit busy, so we prepared a heat map across the two dimensions of capacity and adherence with color corresponding to total infections.



Some findings from the data above:

1. restricting capacity is very impactful right up to 75 max capacity. So restricting only just the largest events *should* still have a moderating effect on spread.
2. adherence factor has only a very modest impact on spread at the higher factors. If fully 20% of subjects are not observing the prescribed policies, this shouldn't lead to dramatic increase in spread. Even at 50% adherence, spread is significantly muted relative to no policy at all.

3. A 25 capacity restriction at ~75% adherence would result in just 10% total infections across the population. This appears to be a good target that allows for some flexibility.

For comparison purposes, we will show a sample simulation of the 25 Max, 50% adherence scenario.

```
from rknot.events import Restriction

lg = Restriction(name='large', start_tick=30, duration=120, criteria={'capacity': 25},
↪ adherence=.75)

events_w_res = events_gated + [lg]

params = {'groups': groups, 'density': 1, 'days': 365, 'tmr_curve': tmr, 'vboxes':␣
↪vbox, 'events': events_w_res}

sim = Sim(**params)
sim.run(dotlog=True)

chart = Chart(sim).to_html5_video()
```

```
<IPython.core.display.HTML object>
```

The results of th sim are shown below in comparison to our base 10 max capacity scenario:

```
<IPython.core.display.HTML object>
```

So restricting contacts to maximum 25 per day has a similar impact on spread even with only 75% adherence to the policy.

---

## 6.5 Social Distancing

We will now investigate the impact of social distancing measures *as outlined here*.

We provice `tmfs` to each age group, representing the adherence to and impact of various tactics including 6-feet of distance, masks, hand sanitizer, etc.

The policy measure is implemented on day 30 and maintained for 120 days.

Note we set `group2b` to `tmf=.5`, indicating much striter adherence to social distancing practices than its age cohort (which is likely consistent with care home workers in the real world).

```
from rknot.events import SocialDistancing as SD

sd = SD(name='all', tmfs=[.8, .8, .5, .7,.65,.5], groups=[0,1,2,3,4,5], start_tick=30,
↪ duration=120)

events_w_res = events_gated + [sd]

params = {
    'groups': groups, 'density': 1, 'days': 365, 'tmr_curve': tmr,
    'vboxes': {'label': 'Events', 'box': 344}, 'events': us_w_load_18.events
}

sim = Sim(**params)
sim.run()
```

(continues on next page)

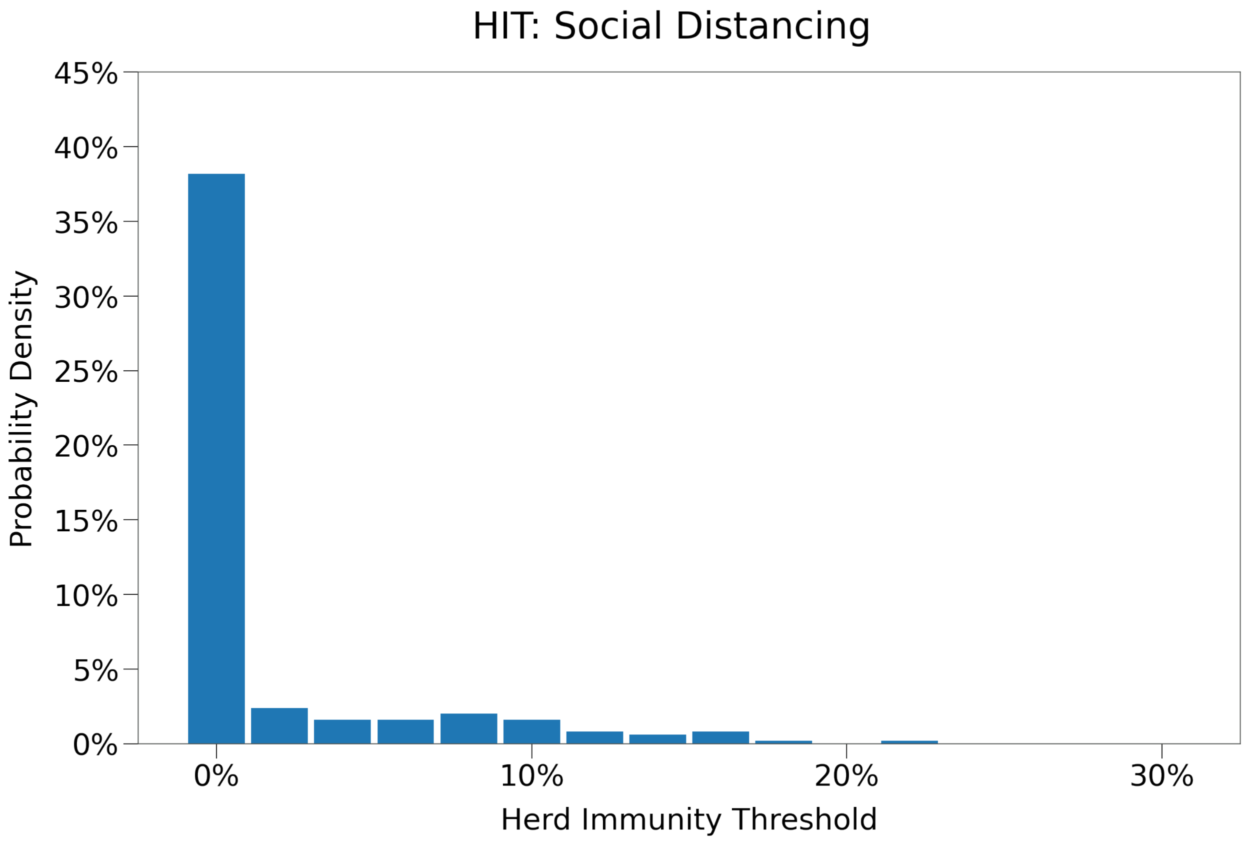
(continued from previous page)

```
chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

The average results of 250 iterations of the scenario are shown in the table below:

```
<IPython.core.display.HTML object>
```

The distribution of infections is heavily skewed towards zero with fairly volatile deviations at the margin.



```
<IPython.core.display.HTML object>
```

The results of the simulation are compared to the SIR model below:

```
<IPython.core.display.HTML object>
```

Here we see that social distancing has an even more pronounced impact on spread. The peak and hit are flattened significantly, but the tail is also much shorter.

A lower $R_0$ is in part responsible, but so too is the viral load curve in Hutch, which has a longer *infection* duration but a much shorter truly *infectious* period.

Additionally, fatalities are MUCH lower by a factor of 10. This is due to the lower overall infection level, but note that an outbreak was prevented in the care home (70+G area). With the shorter infectious period, the virus has fewer opportunities to enter the gate.

### 6.5.1 Social Distancing with Adherence

As with *Capacity Restrictions* above, we can investigate the impact that sub-100% adherence might have on spread.

Adherence with respect to `SocialDistancing` requires one additional consideration. A single event is used to cover all the groups in the sim, however, each group is likely to have a different adherence factor given their differing perceived risk factors and motivations.

For instance, `70+G` subjects are likely to have very high adherence given the known risks invovled and the supervision provide by home care professionals. `20-49` subjects (particularly the youngest in the group) exhibit greater risk-taking behavior in general and covid-19 has provded no different; their adherence would be expected to be lower.

We ran 5 different scenarios, each with differing adherence factors for each group. The table below shows the adherence factors used for each group in each scenario.

```
<IPython.core.display.HTML object>
```

The results of 100 iterations of each scenario are shown below:

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

Again we see that the policy restriction is resilient in the face of non-compliance, although social distancing appears less resilient than capacity restrictions.

Below we show a representative simulation for the Medium adherence scenario.

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

## 6.6 Restrict Elderly Visits

Again, as with the *SIR simulations*, we will restrict visits to the elderly, HOWEVER, care home workers will continue to have access.

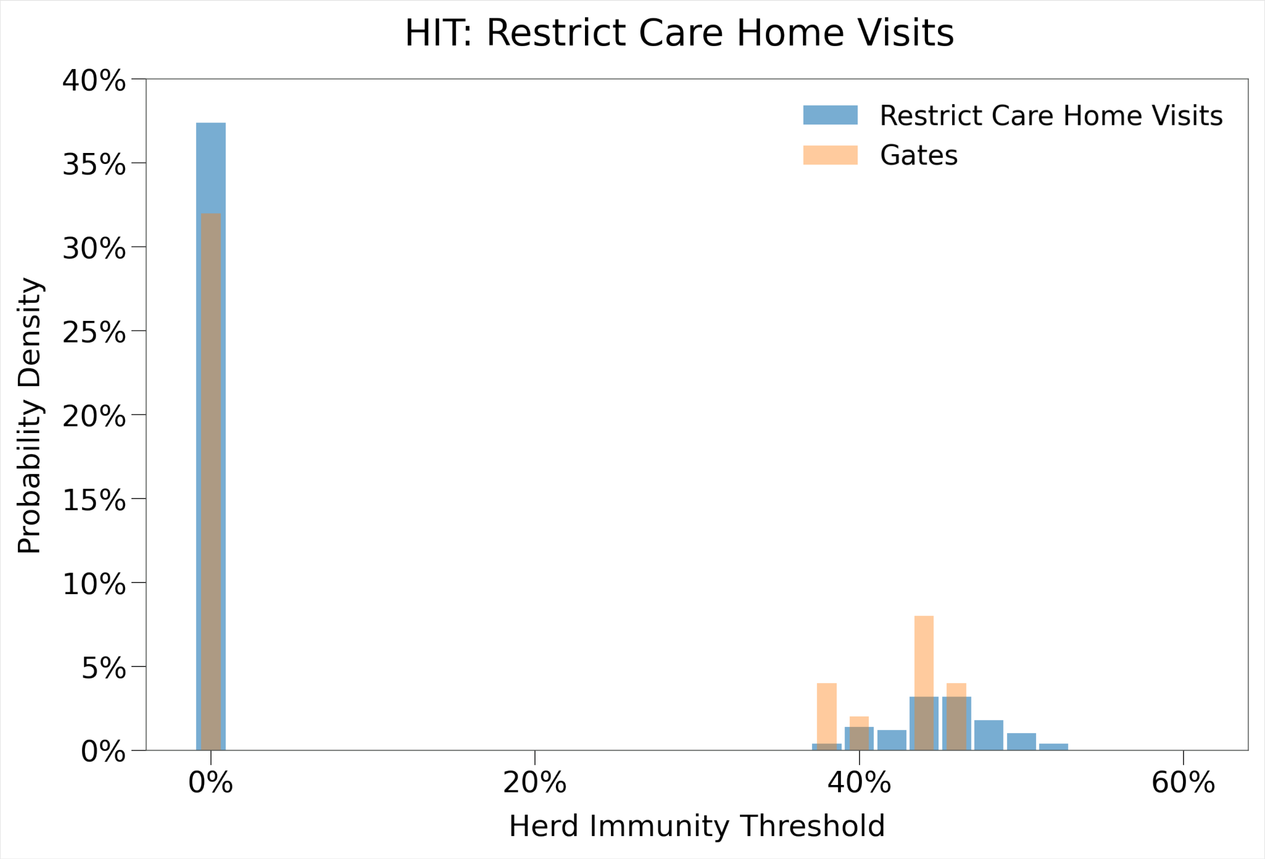
The policy measure is implemented on day 30 and maintained for another 120 days. There will be no other restrictions.

```
no_visits = Restriction(name='no_visits', start_tick=30, duration=120, criteria={'name
↪': 'visit'})

events_w_res = events_gated + [no_visits]

params = {'groups': groups, 'density': 1, 'days': 365, 'tmr_curve': tmr, 'vboxes':␣
↪vbox, 'events': us_w_load_18.events}

sim = Sim(**params)
sim.run()

chart = Chart(sim, use_init_func=True)
chart.animate.to_html5_video()
```

Below we see the results of 250 simulations of the scenario:

```
<IPython.core.display.HTML object>
```

Below we can see the distribution of HIT among the different iterations. Given the restrictions on contacts are very limited in the general population, HIT is very similar to that of the *Gates scenario*.



Below we show a representative simulation:

```
<IPython.core.display.HTML object>
```

The results of the simulation are shown below:

```
<IPython.core.display.HTML object>
```

So the outbreak in the sample simulation is on par with that in SIR, with the usual steeper slope and shorter duration.

Fatalities in this particular sim are also on par with SIR, however, the average values were much higher than SIR and relative to the total infections, fatalities were much higher in Hutch. Fatalities were skewed much higher towards the >70 age group as well.

We can see from the animation that around tick 50 an outbreak occurs in the 70+G gated area. This did not occur in the SIR simulation. We can investigate this by analyzing the `dotlog`.

Below we can see that the restrictions were successful in limiting infections inside the gate up to tick 40. On tick 40, however, 4 infections occured within the group and that expanded to 19 infections by tick 44.

[64]:

```
from rknot import Sim
from rknot.dots import MATRIX_COL_LABELS as ML
infs_in_70g = []
for i in range(30, 45):
    dots = sim.dotlog[i]
    g5s = dots[dots[:, ML['group_id']] == 5]
    inf5 = g5s[g5s[:, ML['is_inf']] == 1]
    infs_in_70g.append((i, inf5.shape[0]))
print (infs_in_70g)
```

```
[(30, 0), (31, 0), (32, 0), (33, 0), (34, 0), (35, 0), (36, 0), (37, 0), (38, 0), (39,
↪ 0), (40, 4), (41, 4), (42, 4), (43, 4), (44, 19)]
```

We can focus in on tick 39 to see how the virus entered the gate.

First, we find the subjects inside the gate on tick 39.

[66]:

```
dots = sim.dotlog[39]
gate_locs = sim.gates[0]['locs']
dots_in_gate = dots[np.isin(dots[:, ML['loc_id']], gate_locs[:,0])]
print (dots_in_gate.shape[0])
```

```
307
```

There were 307 subjects inside the gate on tick 39.

[68]:

```
not5_in_gate = dots_in_gate[dots_in_gate[:, ML['group_id']] != 5]
print (not5_in_gate.shape[0])
```

```
7
```

Of those 307 subjects, 7 were not in the 70+G group. In fact, all 7 are home care workers (`group_id=2`). Below we show the slice of the dot matrix showing the home care workers inside the gate on tick 39.

[70]:

```
print (not5_in_gate)
```

```
[[   6734      2      1      0      1      0      0    412      5      5
     9305     92     24      0   9168 119684      6     98    100     20
       -1     -1     -1     -1      0      1]
 [   6737      2      1      0      1      0      0    207      3      4
     1647     17     16      0   9167 119320      6     98    100     20
       -1     -1     -1     -1      0      1]
 [   6750      2      1      0      1      0      0    309      4      4
    10124    100     27      0   9155 114952      6     98    100     20
       -1     -1     -1     -1      0      1]
 [   6770      2      1      0      1      0      0    104      2      3
     9027     89     52      0   9160 116772      6     98    100     20
       -1     -1     -1     -1      0      1]
 [   6780      2      1      0      0      1      1    408      5      1
     7166     71     27      0   9147 112040      6     98    100     20
       35     -1     66    432      0      1]
 [   6785      2      1      0      1      0      0    206      3      3
     8392     83     29      0   9158 116044      6     98    100     20
       -1     -1     -1     -1      0      1]
 [   6791      2      1      0      1      0      0    205      3      2
     5369     53     66      0   9149 112768      6     98    100     20
       -1     -1     -1     -1      0      1]]
```

If we inspect the matrix above closely, we can see that only one subject is infected, subject `id=6780`.

We can isolate this subject below:

[71]:

```
not5_in_gate[not5_in_gate[:,ML['is_inf']] == 1]
```

[71]:

```
array([[  6780,      2,      1,      0,      0,      1,      1,    408,
             5,      1,   7166,     71,     27,      0,   9147, 112040,
             6,     98,    100,     20,     35,     -1,     66,    432,
             0,      1]], dtype=int32)
```

So restricting eldery visits is ultimately a meaningless and ineffectual policy if home care workers enter the gate without obstruction. Obviously, home care workers must be allowed to enter the gate to care for the patrons of those residenices. So, how then to approach the issue?

One possibility is to implement a testing regime.

---

## 6.7 Schools

As noted in *Capacity Restrictions*, settting a maximum capacity of 10 for events, or the equivalent of limiting contacts to 10 per day per person, *should* significantly reduce spread and kill the virus very quickly.

This, of course, has not happened in the case of sars-cov-2, despite the widespread use of similar policies. This is likely in part due to the many *exceptions* that have been allowed to those policies.

For example, leading up to the second/third wave in North America, many jurisdictions maintained open schools. This approach is understandable given the importance of education during developmental years and the extremely low fatality risk for the age group, however, this approach will inevitably lead to greater spread.

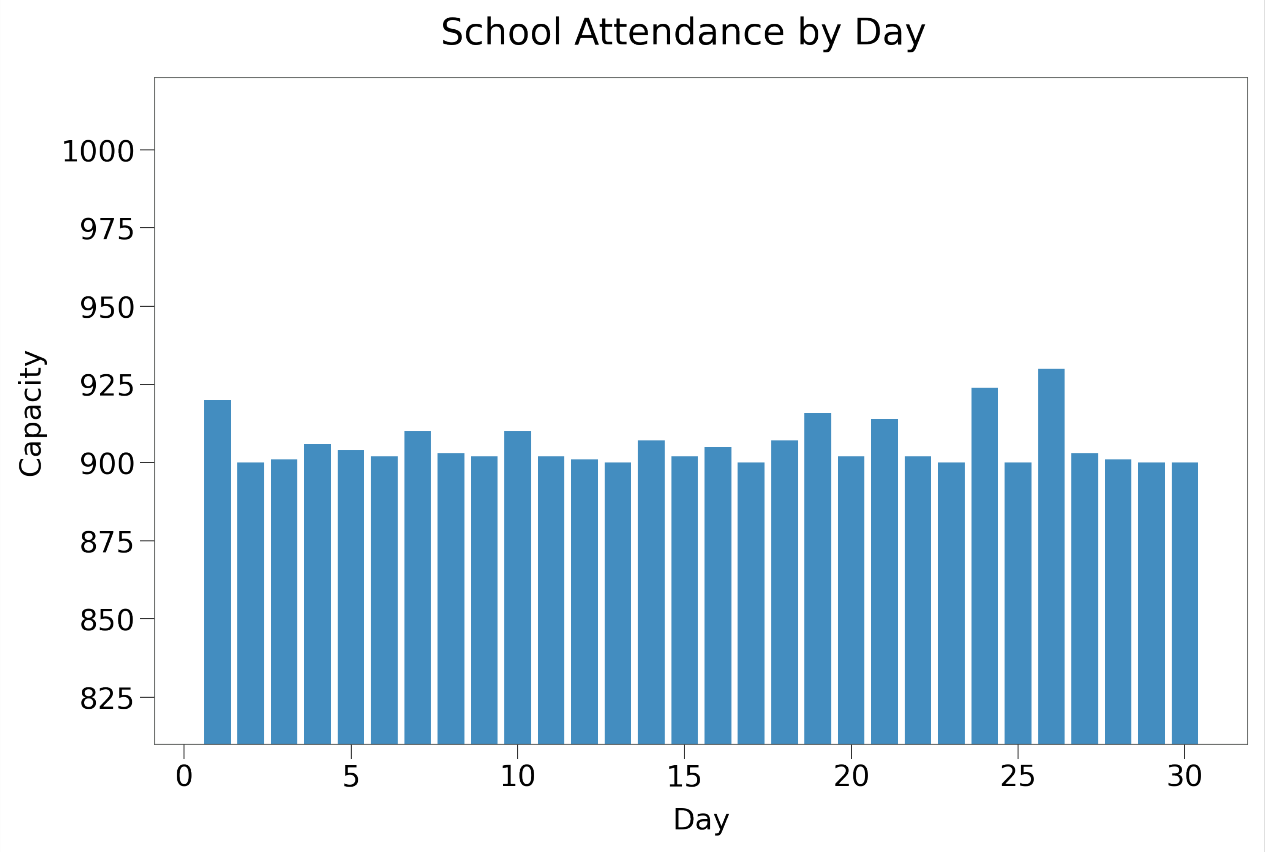
To demonstrate, we implement an augmentation to the *Home Care* environment. Changes include:

- create a new *VBox* for events exclusive to the 0-19 age group.
- reassign a specific number of events to that age group
- add a new group, `Teachers`, that will also attend events in the `Schools` VBox.
  - the `Teachers` group will have `n=97`, proportioned based on 3.2MM teachers in the United States out of a population of 330MM.
  - remaining attributes similar to `20-49` group
  - there will be 49 travel events into the `Schools` VBox, recurring every day

Full details on the environment can be found here. The adjusted events are available in the `us_w_load_18` module.

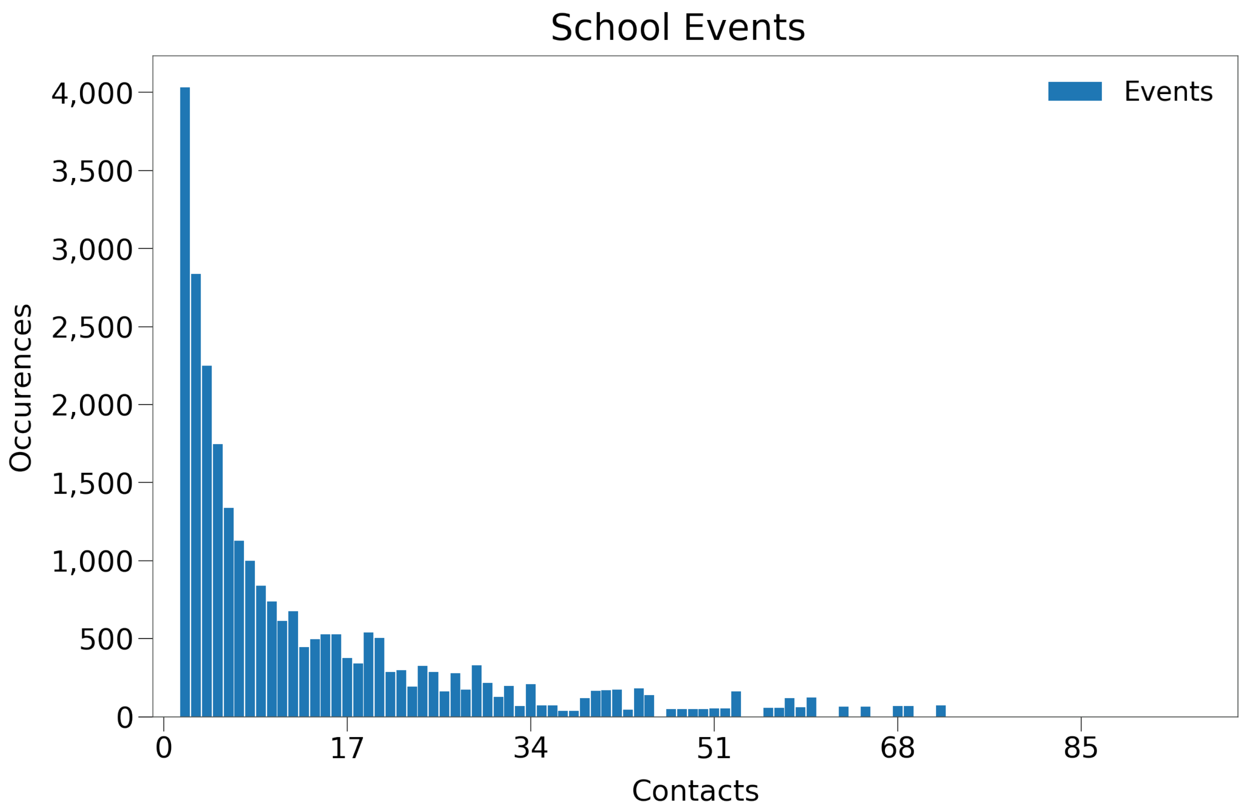
```
from rknot.sims import us_w_load_18
from rknot.events import Restriction

group1 = dict(name='0-19', n=2700, n_inf=0, ifr=0.00003, mover=.982)
group2a = dict(name='20-49', n=3937, n_inf=1, ifr=0.0002, mover=.982)
group2b = dict(name='HCW', n=66, n_inf=0, ifr=0.0002, mover=.982)
group2c = dict(name='Teachers', n=97, n_inf=0, ifr=0.0002, mover=.982)
group3 = dict(name='50-69', n=2300, n_inf=1, ifr=0.005, mover=.982)
group4a = dict(name='70+', n=600, n_inf=0, ifr=0.042, mover=.982)
group4b = dict(name='70+G', n=300, n_inf=0, ifr=0.0683, mover='local', box=[1,6,1,6],␣
↪mover=.982)

groups = [group1, group2a, group2b, group2c, group3, group4a, group4b]

events_schools = us_w_load_18.params_schools['events']
vboxes = [{'label': 'Events', 'box': 231}, {'label': 'Schools', 'box': 152}]
params = {'groups': groups, 'density': 1, 'days': 365, 'tmr_curve': tmr, 'vboxes':␣
↪vboxes, 'events': events_schools}

sim = Sim(**params)
sim.run(dotlog=True)

chart = Chart(sim).to_html5_video()
```

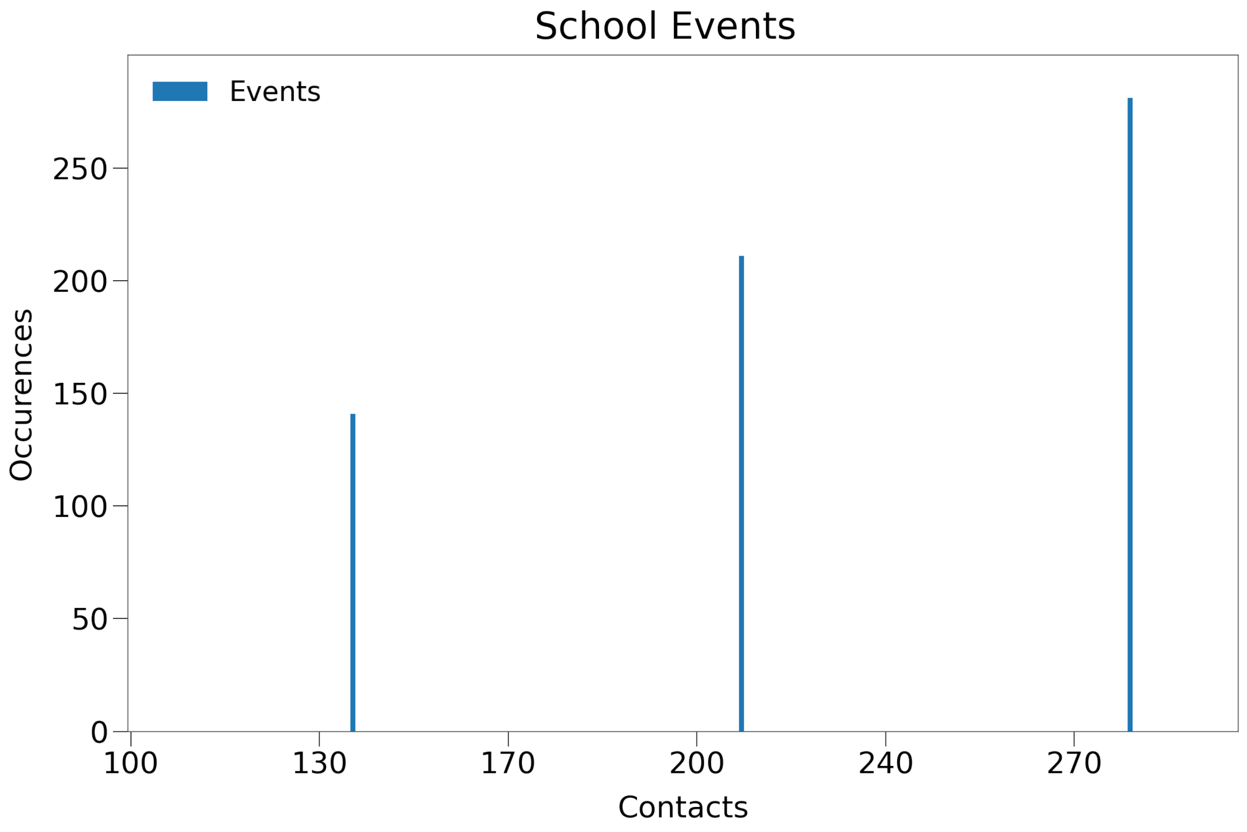
The `Schools` vbox will maintain ~900 children at each tick, at events of various capacities. Below we confirm the number of 0-19 subjects participating in events in the School box on each tick.



Below we show the contact distribution created by the School box events. These events are randomly reassigned from the existing event structure in the Home Care scenario. This approach should maintain the required contact distribution across the entire environment.

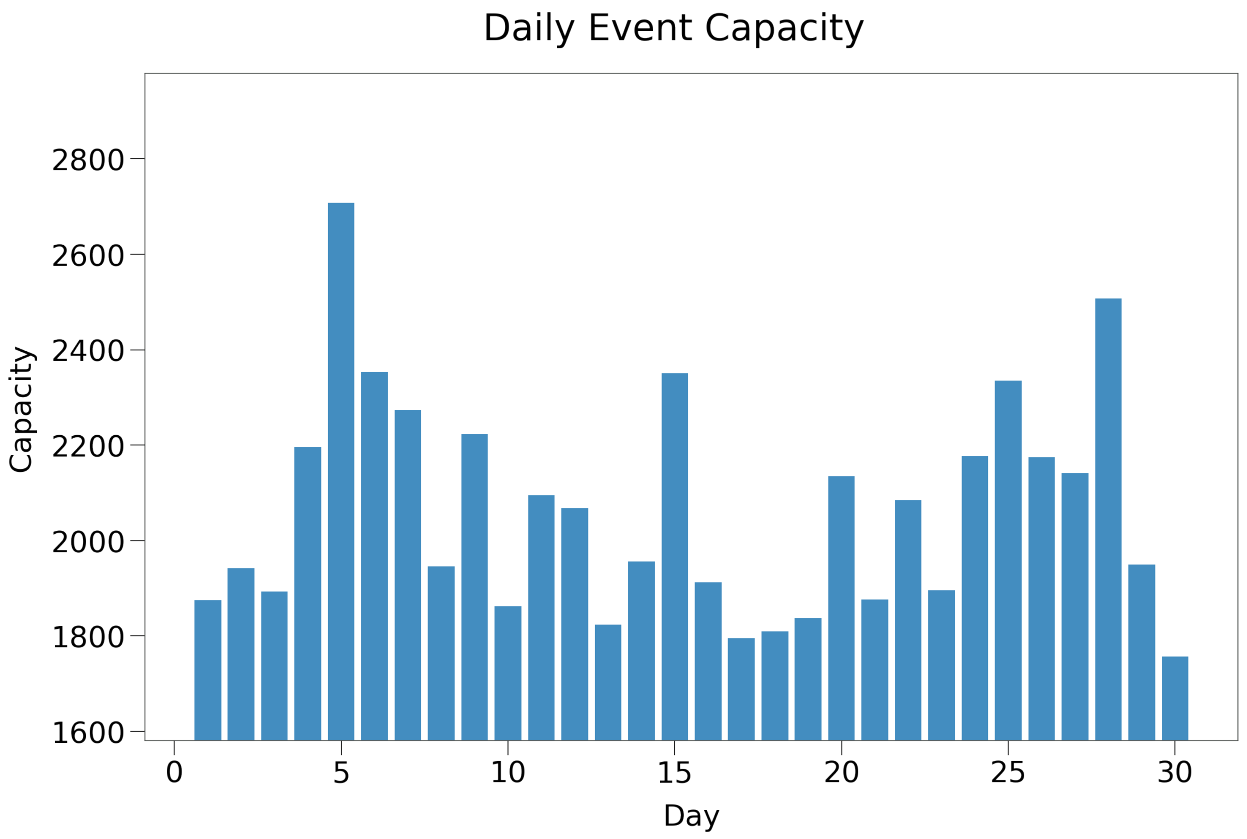
Note the three largest contact events (all 100+ capacity) will occur inside the School box.


School Events
Events
Occurences
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
0
17
34
51
68
85
Contacts


School Events
Events
Occurences
250
200
150
100
50
0
100
130
170
200
240
270
Contacts

```
<IPython.core.display.HTML object>
```

Below, we see the total daily event capacity for the School Box scenario, which is identitcal to the Home Care scenario.



The base scenario was run through 100 simulations, with the results compared to the original Gates scenario and shown below:

```
<IPython.core.display.HTML object>
```

We see the curves between the new `Schools` scenario and the `Home Care` scenario are very similar, which is the expected result. Note, however, the much narrower spread between Peak/HIT/Total infections. This is indicative of steeper and more abrupt tail to the spread curve.

A sample simulation is shown below:

```
<IPython.core.display.HTML object>
```

The results of the simulation are shown below:

Again, the addition of schools results in a similar curve. We can see the steeper curve on both sides of the peak, which in this scenario actually leads to fewer total infections the care home scenario.

The number of fatalities and age distribution are similar as well.

### 6.7.1 Capacity Restriction w School Exception

We will now incorporate a capacity restriction that does *not* apply to the newly created School events using the `exclude` parameter.

```
from rknot.sims import us_w_load_18
from rknot.events import Restriction

lg = Restriction(name='large', start_tick=30, duration=120, criteria={'capacity': 10},
↪ exclude={'name': 'school'})
params['events'] += [lg]

sim = Sim(**params)
sim.run(dotlog=True)

chart = Chart(sim).to_html5_video()
```

Below are the results of 100 simulations of the above structure, compared with *original max-10 policy restriction*.

```
<IPython.core.display.HTML object>
```

From the above, we can see that the approach of leaving schools open leads to a dramatic increase in spread. Below we compare to some other capacity restriction scenarios:

```
<IPython.core.display.HTML object>
```

We can see that leaving schools open can result in a spread curve similar to some of the more permissive capacity restrictions that were inspected. Below we show a representative simulation:

```
<IPython.core.display.HTML object>
```

The results of the simulation are shown below:

```
<IPython.core.display.HTML object>
```

We see that providing an exclusion for in-person school attendance makes a 10 max capacity restriction in the broader population much less effective in reducing spread.

### 6.7.2 Capacity Restriction w School Exception and Non-Compliance

The capacity restriction in the prior sims assumed 100% adherence factor to the 10 max capacity restriction. We will show the results of 75% adherence.

```
from rknot.sims import us_w_load_18
from rknot.events import Restriction

lg = Restriction(name='large', start_tick=30, duration=120, criteria={'capacity': 10},
     exclude={'name': 'school'}, adherence=.75,
)
params['events'] += [lg]

sim = Sim(**params)
sim.run(dotlog=True)

chart = Chart(sim).to_html5_video()
```

The results of 100 iterations are shown below for only the outbreak iterations:

```
<IPython.core.display.HTML object>
```

We can see above that additional infringement of the policy amongst the broader population can negatively influence spread, but the impact is muted.

An example simulation is presented below.

[13]:
```
from IPython.core.display import display, HTML
from rknot.notebook import animHTML
display(HTML(animHTML('us_w_load/' + slug)))
```

```
<IPython.core.display.HTML object>
```

The results are as follows, compared with the 100% adherence sim:

```
<IPython.core.display.HTML object>
```

We can see that 75% adherence results in a slightly larger outbreak with a more abrupt tail (evidenced by the narrower spread between Peak/HIT/Total infections). Fatalities were also inline and confined to only the oldest age groups.

### 6.7.3 Separate Restrictions for Schools

The scenarios up to now have assumed exactly zero policy restrictions on School box interactions. In reality, schools have implemented soccial distancing policies designed to limit spread, which we will attempt to mimic.

In determining the potential restrictions, we have considered some real world examples including the fulsome details provided by the Ontario provincial government (in Canada).

These restrictions include:

- usual social distancing measures of masks, hand sanitizer, separate entrance and exits, etc.
- expected 70% of maximum attendance during the year
- no class size maximums for elementary schools (although students are expected to interact only with student in their class)
- class size restrictions of 15 students for secondary schools

In practice, it is very hard to limit interactions. Videos of schools under these restrictions show long lineups with students in close quarters, using lockers, interacting in hallways etc.

We simulated three different evironments:

- 100Max restriction with 100% adherence
- 50Max restriction with 75% adherence factor
- 25Max restriction with 75% adherence factor

*One possibility we have not considered is that the 0-19 age group is somehow more resilient and incurs fewer infections per contact than the broader population. This simply does not appear to be the case based on the prevailing testing data. In Ontario, for example, not only have children *not* been shown to incur fewer infections, ages 0-24 were all shown to have the highest level of positive tests during the second wave and were still increasing through late Dec 2020 even when all other age groups had flattened out.* See positive results by age group.

```
<IPython.core.display.HTML object>
```

First, we show the implementation of the 100 capacity event restriction in the School box, via the `schoolmax` restriction. The only difference in the other scenarios include the `adherence` parameter in the `schoolmax` object.

```
from rknot.sims import us_w_load_18
from rknot.events import Restriction

lg = Restriction(
    name='large', start_tick=30, duration=120, criteria={'capacity': 10},
    exclude={'name': 'school'}, adherence=.75,
)
```

(continues on next page)

(continued from previous page)

```
schoolmax = Restriction(
    name='large', start_tick=30, duration=120,
    criteria={'capacity': 100, 'name': 'school'},
)
us_w_load_18.params_schools['events'] += [lg, schoolmax]

sim = Sim(**us_w_load_18.params)
sim.run(dotlog=True)

chart = Chart(sim).to_html5_video()
```

```
<IPython.core.display.HTML object>
```

We can see the progressively suppressed spread curve as we reduce the capacity maximum in the School box. Despite having all 3 of the largest events (all >100 capacity), eliminating those events has any only minor impact on spread. There are simply not enough contacts in generated by those 3 events to spur broader infection.

Spread is only materially impact with a lower maximum that encompasses many more events.

Still, relative to the original *10max 75% adherence restriction*, the outbreak in 25max 75% adherence is 4 - 5x larger.

It is difficult to imagine how a school could limit contacts to fewer than 25 per day for students and staff given class sizes and other interactions. And so we see that open schools, even with their own restrictions, can undo much of the work being done by more rigorous restrictions in the broader population.

Below we show a representation simulation:

```
<IPython.core.display.HTML object>
```

The results are shown below compared to the original *Home Care scenario*, which did not have a seperate school vbox or any exceptions for schools.

```
<IPython.core.display.HTML object>
```

We can see from the above that providing exclusions for schools, despite including some restrictions without those schools, significantly impairs the impact of restrictions in the broader economy.

We must consider if an outbreak can ever be prevented if we open schools (and we cannot reduce daily contacts at schools to less than maximum 25 per day).

# CHAPTER 7

# Multiple Jurisdictions

With our credible model for dynamic transmission at the contact level and our ability to size environments with approriate contact distributions, we can show the true power of **RKnot** in modelling complex scenarios.

We *have previously shown* that contact restrictions and social distancing in any one community *should* be effective in eliminating `sars-cov-2` relatively quickly. Across the world, however, many states, provinces, and countries, have experienced second and third waves spaced out months apart despite various forms of strict quarantines and lockdowns.

So why haven't they been effective?

In part, this has resulted from the many exceptions made to these restrictions, in particular, *schools remaining open* in many jurisdictions.

We must also consider the impact of multiple jurisdictions that have unique characteristics, set policy independently, and, importantly, allow travel between them.

We will focus on two jurisdictions to start, building to four, and building in complexity along the way.

## 7.1 Mixed

To start, we will simply double the size of the environment used in *our dynamic transmission risk models*. Thus,

- n = 20,000 and 4 initial infections
- 8 groups, split evenly into two states: "East State" and "West State"
- the states will be exact reflections of each other:
  - same size, n = 10,000
  - 2 initial infections each
  - same demographic mix
  - same number of events
- the event structrue for each state is detailed in *Sizing* (and imported from a pickled object as per below).

- each state will have its own vbox. The `East State` vbox must be reassigned and the events adjusted to point to the correct groups.

```
from copy import deepcopy

from rknot import Sim, Chart
from rknot.events import Restriction, Quarantine
from rknot.dots.fhutch import tmr
from rknot.sims import us_w_load_18.events

group1 = dict(name='W0-19', n=2700, n_inf=0, ifr=0.00003, mover=.98)
group2 = dict(name='W20-49', n=4100, n_inf=1, ifr=0.0002, mover=.98)
group3 = dict(name='W50-69', n=2300, n_inf=1, ifr=0.005, mover=.98)
group4 = dict(name='W70+', n=900, n_inf=0, ifr=0.054, mover=.98)
wstate = [group1, group2, group3, group4]

wbox = {'label': 'W Main', 'box': 344}
wevents = deepcopy(us_w_load_18.events)
wrsxns = deepcopy(us_w_load_18.rsxns)

group5 = dict(name='E0-19', n=2700, n_inf=0, ifr=0.00003, mover=.98)
group6 = dict(name='E20-49', n=4100, n_inf=1, ifr=0.0002, mover=.98)
group7 = dict(name='E50-69', n=2300, n_inf=1, ifr=0.005, mover=.98)
group8 = dict(name='E70+', n=900, n_inf=0, ifr=0.054, mover=.98)
estate =  [group5, group6, group7, group8]

ebox = {'label': 'E Main', 'box': 344}
eevents = deepcopy(us_w_load_18.events)

for e in eevents:
    e.groups = [4,5,6,7]
    e.vbox = 1

groups = wstate + estate
vboxes = [wbox, ebox]
events = wevents + eevents

params = {
    'groups': groups, 'density': 1, 'days': 365, 'tmr_curve': tmr,
    'vboxes': vboxes, 'events': events
}

sim = Sim(**params)
sim.run()
```

Below, we show the arrangement of all of the dots mixed throughout the space. The only difference from the single jurisdiction structure is the presence of two vboxes (instead of one).

```
<IPython.core.display.HTML object>
```

On a proportionate basis, this structure is indistinguishable from the n=10,000, 4 group structure used in the viral load simulations. And it should result in the same outbreak curve, on average.

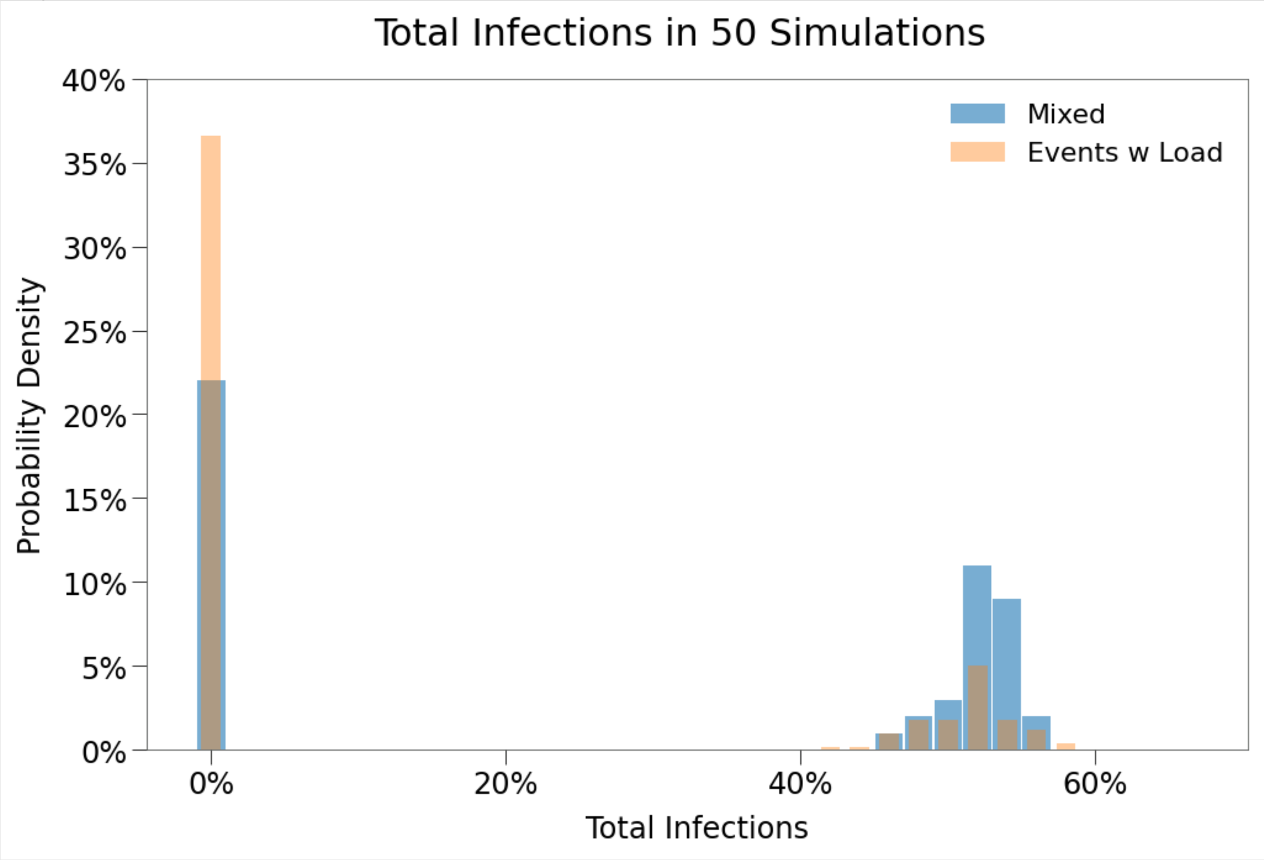
To test, we ran 50 sims for a quick comparison to the base *Events* structure that we are building from.

```
<IPython.core.display.HTML object>
```

As expected, we can see above that, on average, the spread results generated in this structure match those of the *Events structure* from the viral load analysis. Notice, however, that an outbreak ($R_0 > 0$) occured much more frequently: 56% vs. 27%. This is likely a result of the greater number of initial infections (4 vs. 2).

Also notice that the average Peak is lower and Days to Peak higher in our new scenario despite all other metrics being the same.

In the distribution below, we can see that, in this scenario, more simulations result in outbreaks, but when they do, they concentrated around the same level of total infection.



We can see this in the sample simulation below:

```
<IPython.core.display.HTML object>
```

Above, we can see a "wave" effect occuring with two peaks forming, the first around 70 days and second around 115 days. We can seperate the curve among the constituent states to see how each state's curve might have added to the aggregate.

```
<IPython.core.display.HTML object>
```

From above, we can see that what looked like a single curve and a single outbreak was, in fact, two outbreaks among the two "states", West and East.

Remarkably, the dots from both states are free to mix in the main grid. The separate waves result simply from isolated event spaces where only the largest events (3+ capacity) occur within the state groups.

Next we will show the impact of further isolating the two states.

---

## 7.2 Borders

Now we split the environment into two halves, separating the subjects of the West and East states by a border.

They cannot interact in any way.

This can be accomplished by simply assign a separate box to the groups in each state, as per below:

```
for w in wstate:
    w['box'] = {'bounds': [1, 72, 1, 144], 'label': 'West State'}

for e in estate:
    e['box'] = {'bounds': [73, 144, 1, 144], 'label': 'East State'}

groups = wstate + estate
params['groups'] = groups

sim = Sim(**params)
sim.run(dotlog=True)
```

This results in the environment shown below:

```
<IPython.core.display.HTML object>
```

We ran 50 simulations to compare with the Mixed environment:

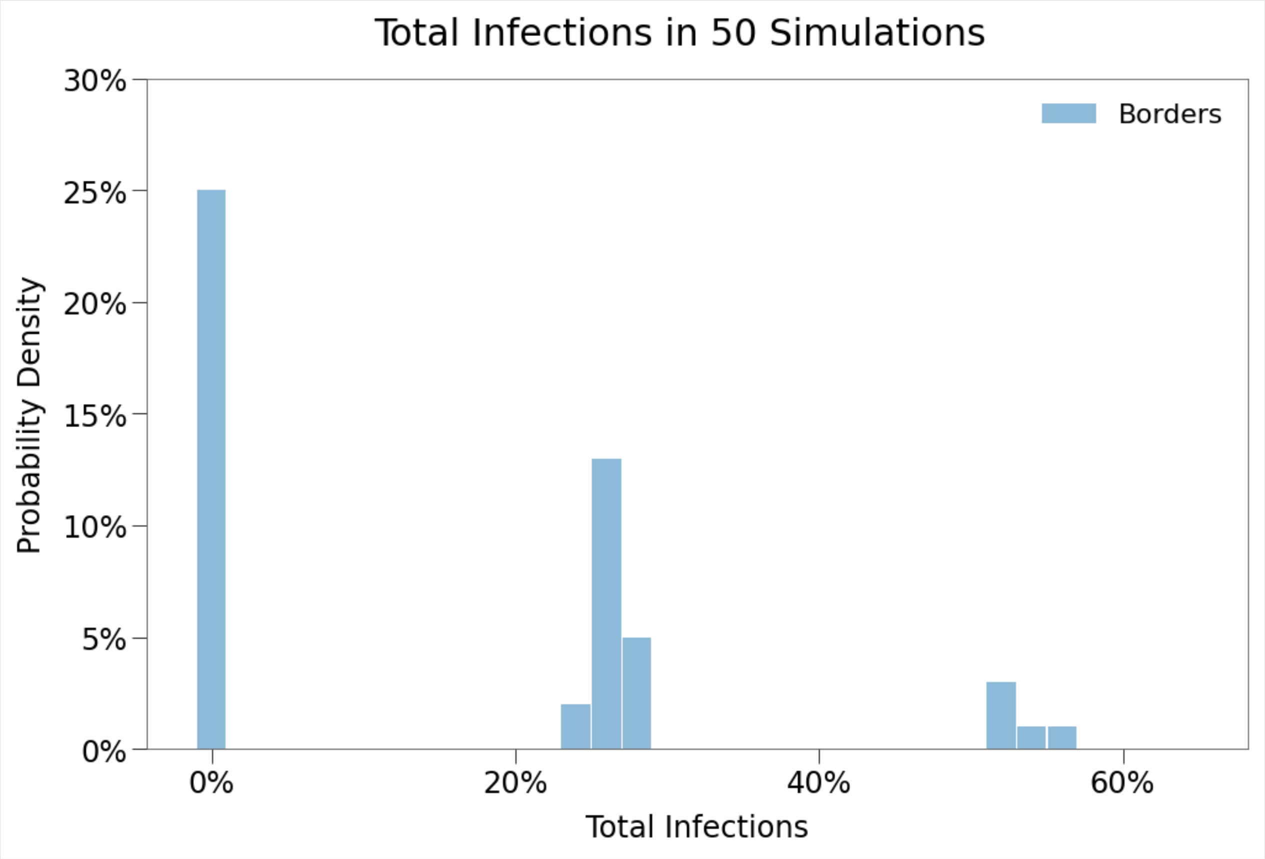
```
<IPython.core.display.HTML object>
```

We can see:

- an outbreak occurs with about the same frequency; likely b/c both environments have the same number of initial infections
- the size of the outbreaks is significantly lower in the split environment
- the peaks are significantly lower in the split environment

```
<IPython.core.display.HTML object>
```

Again, we see below that the contact distribution of the Borders environment is quite different from the Mixed environment.



Here, the same number of outbreaks occur, but the virus cannot cross the border so at worst it can only spread amongst half the population. As a result, we see two local peaks form, representing when an outbreak occurs in only one state or in both.

In a uniform environment, there are only two outcomes:

1. Peak
2. No Peak

When we split the space, there are now 4 distinct possible outcomes:

1. No Peaks
2. One Peak: West
3. One Peak: East
4. Two Peaks

We can even get a quick sense for how often these different outcomes might occur, as per below.

```
<IPython.core.display.HTML object>
```

As per the piechart above,only 10% of outbreaks resulted in a Double Peak, while more than 40% resulted in an outbreak in either state. Both states were equally likely to have an outbreak on their own.

Below we see a representative simulation that evidences the border separating the two states. We can also see in the grid on the left that the outbreak was entirely isolated to the West state and did *not* seep into the East state at all. This resulted in the shorter, more muted peak.

```
<IPython.core.display.HTML object>
```

Finally, we also show a typical Double Peak scenario:

```
<IPython.core.display.HTML object>
```

Here we see both states develop outbreaks *independently* and simultaneously, both with very similar spread curves, peaking just after 60 days and reaching ~17% peaks. The aggregate curve is inline with expectations for a well-mixed environment.

---

## 7.3 Border Crossings

Now that we have controlled borders between isolated states, we can explore the impact of border crossings on spread.

To do so, we will create Travel events allowing subjects from each state to cross over into the other state, as follows:

- total of 20 crossings per day
  - 10 crossings per day West to East
  - 10 crossings per day East to West
- each group will have equal likelihood to participate in the crossings

The crossing amount was roughly determined based on international travel statistics out of the US:

- ~100MM travellers annually *from* the US to other countries
- ~90MM visits *from* other countries annually
- we have assumed 100MM for reach version.

$$\frac{\text{100MM visitors annually}}{\text{330MM actual population}} * \frac{\text{10,000 pop}}{\text{state}} = \frac{\text{3,031 sim visitors annually}}{\text{state}}$$

$$\frac{\text{3,031 sim visitors}}{\text{365 days}} = 9\ visitors \text{ per day per state}$$

To do this, we will create 8 random locations for each event (within each state's box), then assign them to our 8 travel events.

```python
from rknot.events import Travel

xs = np.random.randint(19, 120, size=4)
ys = np.random.randint(20, 120, size=4)
w_locs = np.vstack((xs, ys)).T

xs = np.random.randint(74, 120, size=4)
ys = np.random.randint(10, 120, size=4)
e_locs = np.vstack((xs, ys)).T

crosses = []
for i in range(9):
    crosses += [Travel(name=f'WtoE_{i}', xy=w_locs[i], start_tick=1,
        groups=[0,1,2,3], capacity=1, duration=1, recurring=1
    )]
    crosses += [Travel(
        name=f'EtoW_{i}', xy=e_locs[i], start_tick=1,
        groups=[4,5,6,7], capacity=1, duration=1, recurring=1
```

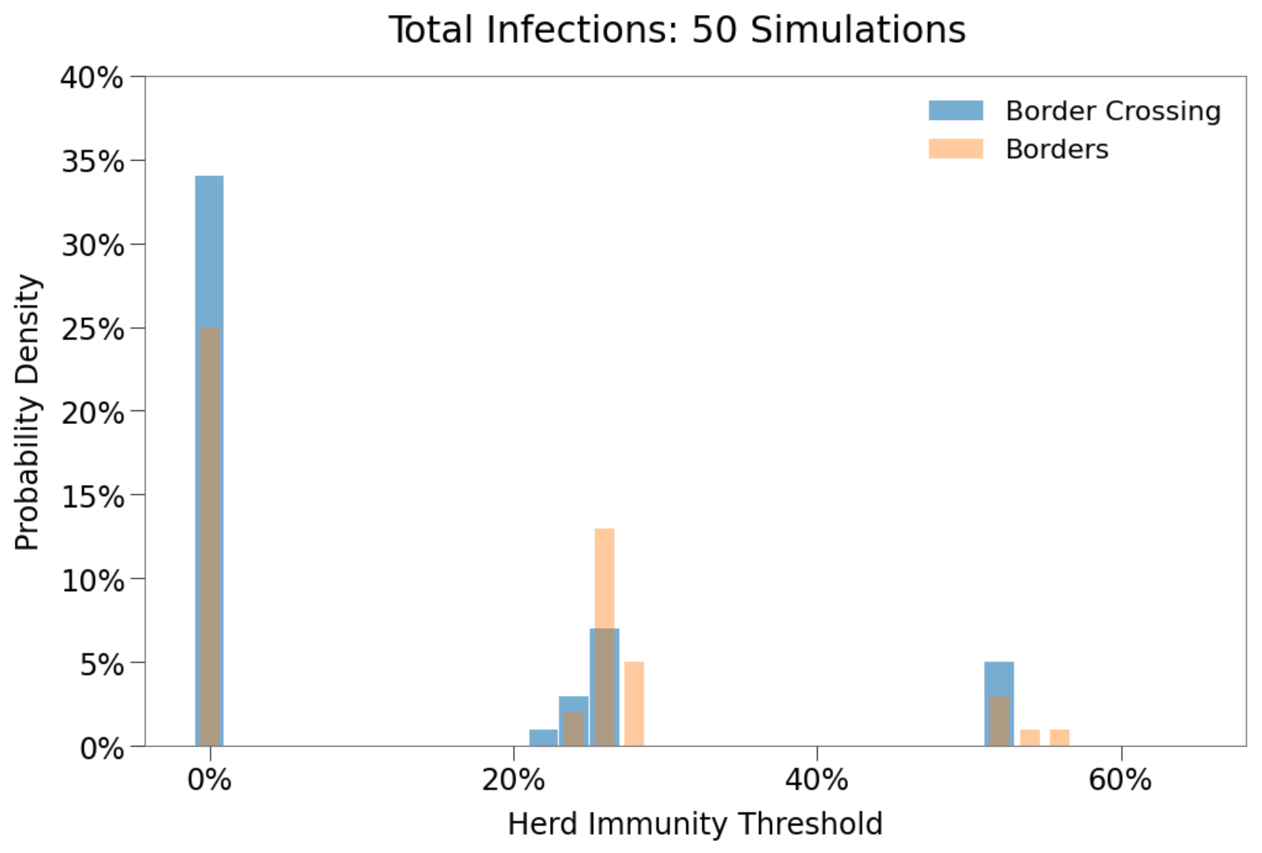
(continues on next page)

(continued from previous page)

```
    )]
params['events'] += crosses
```

The results of 50 simulations are shown below:

```
<IPython.core.display.HTML object>
```



We see only a limited impact from border travel as constructed. The structure actually resulted in fewer outbreaks and resulted in similar single and double peak occurences.

```
<IPython.core.display.HTML object>
```

Below we show a typical Double Peak outcome. The outbreak is almost entirely extinguished and in decline in the East state, but then spread takes hold *independently* in the West State and creates a second peak.

```
<IPython.core.display.HTML object>
```

## 7.4 Crossings with Event Access

The above simulation has major omission. Each traveller only visits the other state for a single tick, only ever land on a single location in the other state's main grid, and so have almost no contact with anyone while travelling.

In reality, travellers often have extended stays of 3/5/7/15/30+ days and, while in the other jurisidiction, they often attend events / have contacts in that jurisdiction like any other local person.

We can mimick this more realistic mixing approach by utilizing the `MultiTravel` event object. This object also a travel event greater than a single tick and allows any participants to attend events in their new region.

We must also implementing a few changes to the existing events. Each event will now pull participants from its own main grid as well as any box created to support cross border travel into the state.

We assume:

- 9 visits per day per state, consistent with the prior scenario
- Respective durations of the 9 visits:
  - 4 visits are for 3 days
  - 3 visits are for 5 days
  - 1 visits is for 7 days
  - 1 visit is for 11 days

The code looks like this:

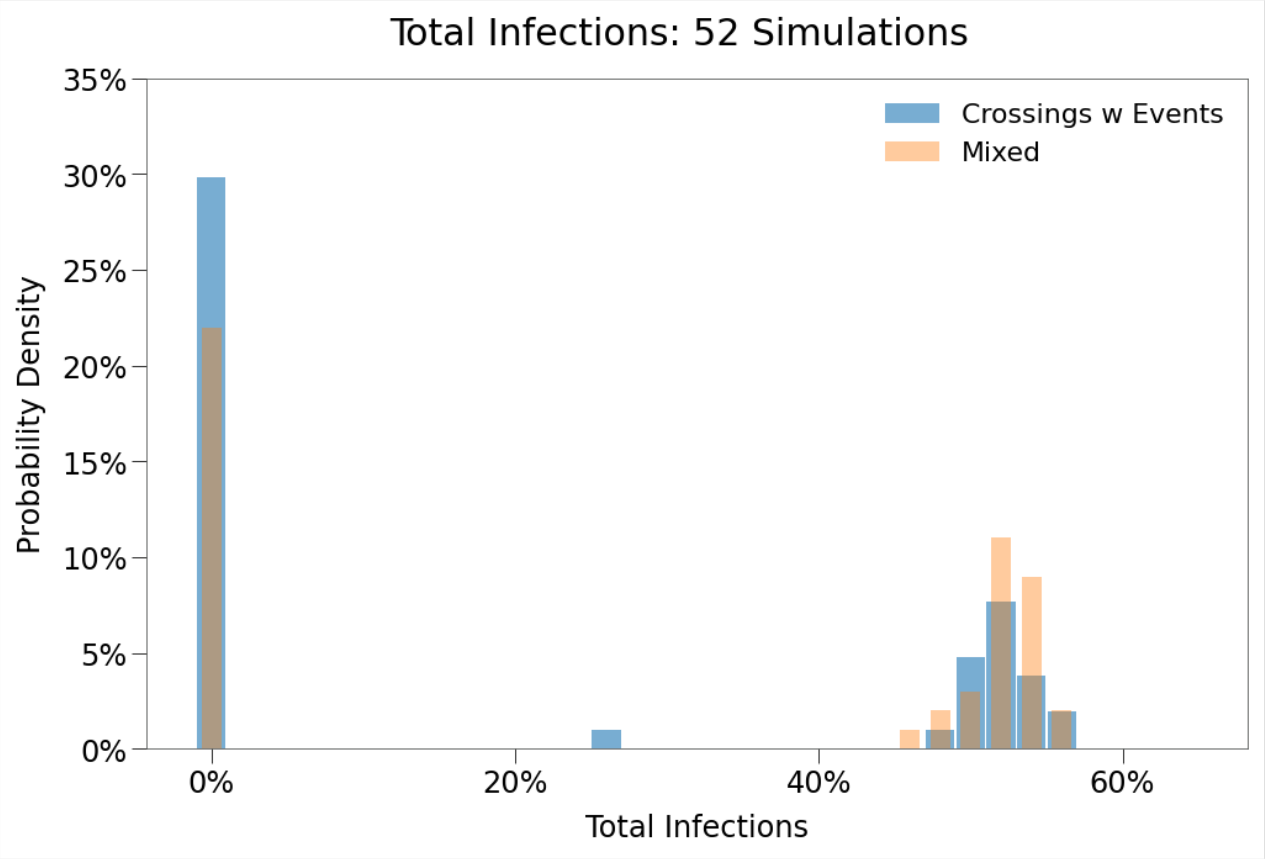
```python
from rknot.events import MultiTravel

for e in params['events']:
    e.groups = np.arange(8)
    if 'W' in e.name:
        e.from_boxes = ['West State', 'EtoW']
        e.no_events = False
    if 'E' in e.name:
        e.from_boxes = ['East State', 'WtoE']
        e.no_events = False
```

With the existing events restructured, we assign each event to locations in their box at random.

```python
n_cross = 9
xs = np.random.randint(19, 120, size=n_cross)
ys = np.random.randint(20, 120, size=n_cross)
w_locs = np.vstack((xs, ys)).T

xs = np.random.randint(74, 120, size=n_cross)
ys = np.random.randint(10, 120, size=n_cross)
e_locs = np.vstack((xs, ys)).T

crosses = []
for i, dur in zip(range(n_cross), (3,3,3,3,5,5,5,7,11)):
    crosses += [MultiTravel(name=f'WtoE_{dur}', xy=e_locs[i], start_tick=1,
        groups=[0,1,2,3], capacity=1, recurring=1, duration=dur
    )]
    crosses += [MultiTravel(
        name=f'EtoW_{dur}', xy=w_locs[i], start_tick=1,
        groups=[4,5,6,7], capacity=1, recurring=1, duration=dur
    )]

params['events'] += crosses
```

50 iterations of the above scenario lead to the following:

```
<IPython.core.display.HTML object>
```



Incredibly, we see that just 10 border crossings per day with travellers mixing normally leads to (almost) the same the results as though there were no border at all.

We can also see that the peak distribution has inverted almost completely. Now a Double Peak event is far more likely than a single peak in either state.

```
<IPython.core.display.HTML object>
```

We can use some additional tools to confirm that mixing between the states was in fact the culprit in increased prevalence of double peaks. The easiest way is to find simulations where a state reaches *zero* infections, effectively eradicating the virus in prior scenarios, only to have the virus re-emerge.

This can *ONLY* occur if the virus was transported in from the other state.

```
HBox(children=(FloatProgress(value=0.0, max=20.0), HTML(value='')))
```

```
{0: 'East', 17: 'East', 21: 'West', 43: 'West', 45: 'East'}
```

```
<IPython.core.display.HTML object>
```

It also tells us the index position of those states in our grouping of simulations. We will isolate simulation # 21, where West State inherited an outbreak.

We can isolate the specific subject that carried the virus.

```
[26]: sim = xesims[21]
id_groups = [westids, eastids]
west_inf, east_inf = find_group_infs(sim, id_groups, westids, eastids)
westtrim = np.trim_zeros(west_inf, 'fb')
```

Above is the number of current infections in the West State at each tick during the simulation. We can clearly see an *entire 14-day period* during that simulation where the West State had *zero* infections.

Then, on Tick 44, a single new infection emerges. We can easily isolate which subject was infected.

[27]:
```
from rknot.dots import MATRIX_COL_LABELS as ML
erads = np.argwhere(westtrim == 0)
i_erad = erads[-1, 0]

dots = sim.dotlog[i_erad + 1]
westmask = np.isin(dots[:, ML['group_id']], westids)
west = dots[westmask]
west[west[:, ML['is_inf']] == 1]
```

[27]:
```
array([[  5210,       1,       1,       0,       0,       1,       1,   15569,
           109,      18,   15569,     109,      18,       1,       2,       0,
        224729,       5,      98,     100,      20,      44,      -1,      75,
           441,       1,       1]], dtype=int32)
```

The array above shows the state of subject 5210 as of Tick 44. We can expand the dot matrix to ticks both before and after to get a sense of this subjects movement.

[28]:
```
id_new = west[west[:, ML['is_inf']] == 1][0,0]
dottrace = sim.dotlog[i_erad - 4 : i_erad + 4, id_new]
```

We can take the array above and put it in a table for easier viewing:

[29]:
```
ticks = np.arange(i_erad - 4, i_erad + 4)
dtrace = np.zeros(shape=(8,28), dtype=np.int32)
dtrace[:, 0] = ticks
dtrace[:, 1:] = dottrace
```

```
<IPython.core.display.HTML object>
```

From the table above we can see the following:

- Subject 5210 was inside the West State box and *not infected* in the days prior to Tick 44.
- On Tick 42, the subject travelled to East State via 3-day `MultiTravel` event.
- On Tick 43, the subject attended an event at location (128, 139), where it was infected.
- On Tick 45, the subject returned to West State and is at least partly responsible for the re-emergence of the virus in that state.

We can even confirm this visually by isolating that specific dot in the animation by passing `highlight=[5210]` at the Chart instantiation. We also slowed down the frame rate. You can see at Tick 44, the subject jumps from West to East and back very quickly.

```
chart = Chart(sim, highlight=[5210], show_intro=True, show_restricted=True)
chart.to_html5_video()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

Finally above, we show the full outcome of the sim, which pesents almost as a single prolonged peak.

## 7.5 Care Homes

With a few modifications, we can incorporate features of the *Care Home* to further increase realism.

Again, we simply double the number of groups, double the number of events, and assign a set of cross border events. In this sim, the `70+G` group in each state is restricted from cross border travel.

```
import numpy as np
from copy import deepcopy

from rknot import Sim, Chart
from rknot.events import Restriction, Quarantine, Travel, MultiTravel
from rknot.dots.fhutch import tmr
from rknot.sims import us_w_load_18

group1 = dict(name='W0-19', n=2700, n_inf=0, ifr=0.00003, mover=0.982)
group2a = dict(name='W20-49', n=4034, n_inf=1, ifr=0.0002, mover=0.982)
group2b = dict(name='WCHW', n=66, n_inf=0, ifr=0.0002, mover=0.982)
group3 = dict(name='W50-69', n=2300, n_inf=1, ifr=0.005, mover=0.982)
group4a = dict(name='W70+', n=600, n_inf=0, ifr=0.042, mover=0.982)
group4b = dict(name='W70+G', n=300, n_inf=0, ifr=0.0683, mover='local')

wstate = [group1, group2a, group2b, group3, group4a, group4b]

group5 = dict(name='E0-19', n=2700, n_inf=0, ifr=0.00003, mover=0.982)
group6a = dict(name='E20-49', n=4034, n_inf=1, ifr=0.0002, mover=0.982)
group6b = dict(name='ECHW', n=66, n_inf=0, ifr=0.0002, mover=0.982)
group7 = dict(name='E50-69', n=2300, n_inf=1, ifr=0.005, mover=0.982)
group8a = dict(name='E70+', n=600, n_inf=0, ifr=0.042, mover=0.982)
group8b = dict(name='E70+G', n=300, n_inf=0, ifr=0.0683, mover='local')
estate =  [group5, group6a, group6b, group7, group8a, group8b]

for w in wstate:
    w['box'] = {'bounds': [1, 72, 1, 144], 'label': 'West State'}
group4b['box'] = [1,6,1,6]

for e in estate:
    e['box'] = {'bounds': [73, 144, 1, 144], 'label': 'East State'}
group8b['box'] = [139,144,1,6]

groups = wstate + estate

wbox = {'label': 'W Main', 'box': 344}
ebox = {'label': 'E Main', 'box': 344}

wevents = deepcopy(us_w_load_18.events_gated)
eevents = deepcopy(us_w_load_18.events_gated)

# Care Home Events
i_ch = -34
for e in wevents[i_ch:]:
    w.name = f'W_{w.name}'

for e in eevents[i_ch:]:
    e.name = f'E_{e.name}'

# Main Events
```

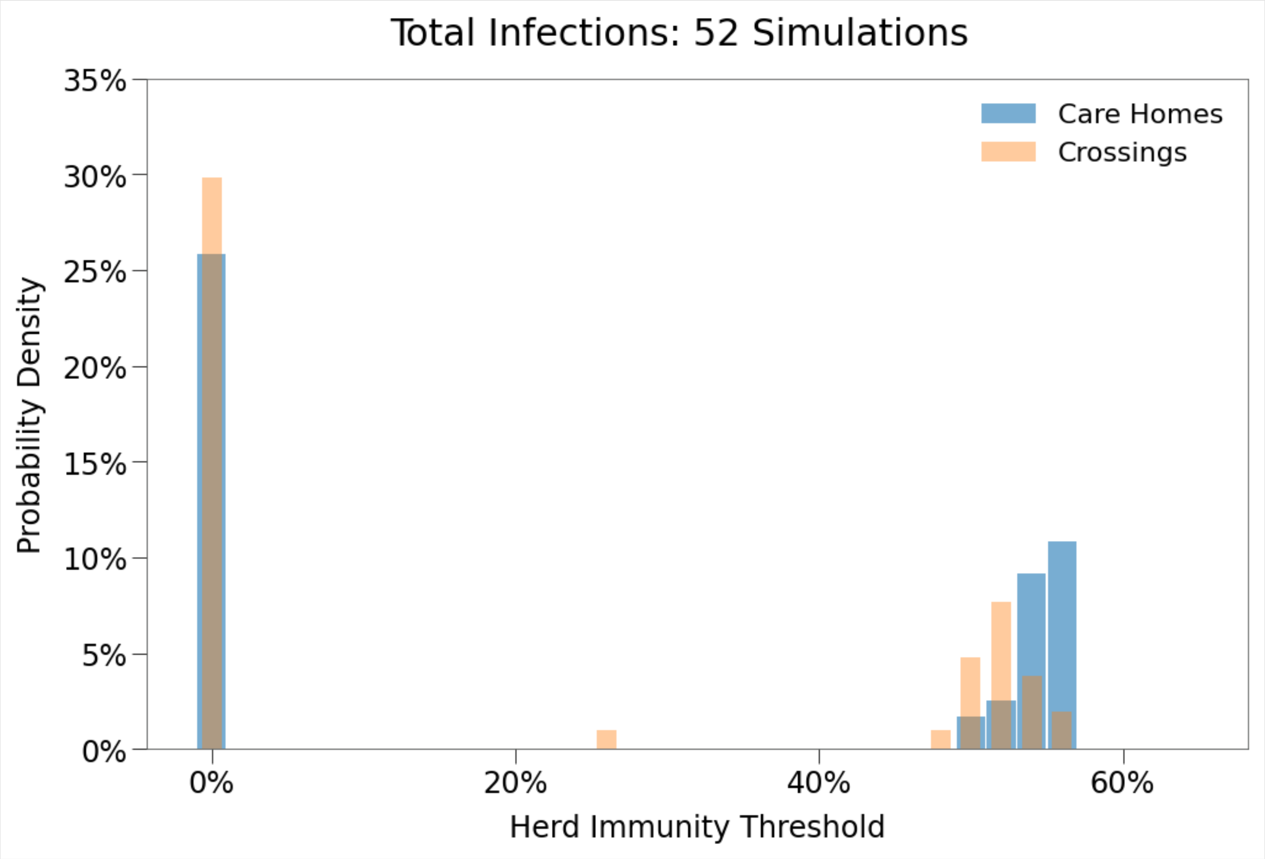
(continues on next page)

(continued from previous page)

```
for w in wevents[:i_ch]:
    w.name = f'W{e.name}'
    w.from_boxes = ['West State', 'EtoW']
    e.groups = np.arange(8)
    e.no_events = False

for e in eevents[:i_ch]:
    e.name = f'W{e.name}'
    e.from_boxes = ['East State', 'WtoE']
    e.vbox = 1
    e.groups = np.arange(8)
    e.no_events = False

groups = wstate + estate
vboxes = [wbox, ebox]
events = wevents + eevents

# Add border crossings
xs = np.random.randint(19, 120, size=4)
ys = np.random.randint(20, 120, size=4)
w_locs = np.vstack((xs, ys)).T

xs = np.random.randint(74, 120, size=4)
ys = np.random.randint(10, 120, size=4)
e_locs = np.vstack((xs, ys)).T

crosses = []
for i, rec in zip(range(4), (3,3,5,5)):
    crosses += [MultiTravel(name=f'WtoE_{i}', xy=e_locs[i], start_tick=1,
        groups=[0,1,2,3,4], capacity=1, recurring=rec
    )]
    crosses += [MultiTravel(
        name=f'EtoW_{i}', xy=w_locs[i], start_tick=1,
        groups=[6,7,8,9,10], capacity=1, recurring=rec
    )]

events_ch += crosses
rsxns = []
quars = []

params_ch = {
    'groups': groups, 'density': 1, 'days': 365,
    'tmr_curve': tmr, 'vboxes': vboxes,
    'events': events_ch, 'rsxns': rsxns, 'quars': quars,
}
sim = Sim(**params_ch)
sim.run(dotlog=True)
```

The results of 60 simulation show outcomes very similar to the *Crossing scenario*.

```
<IPython.core.display.HTML object>
```



We can also per below that in these 60 simulations, not a single outbreak was isolated to only one state.

```
<IPython.core.display.HTML object>
```

Finally, we show a sample simulation.

```
<IPython.core.display.HTML object>
```

### 7.5.1 Care Homes - 10Max

From the Care Home environment, we will begin to investigate the impact of policy restrictions in the two jurisdictions on spread. The restrictions are derived from those used previously in the *SIR* and *Dynamic Transmission*.

In our first scenario, both West State and East State will simultaneously impose a restriction as follows:

- no events with more than 10 subjects
- commences on Day 30
- lasting 120 days
- restriction covers all groups in the respective state
- there is 90% adherence to the restriction

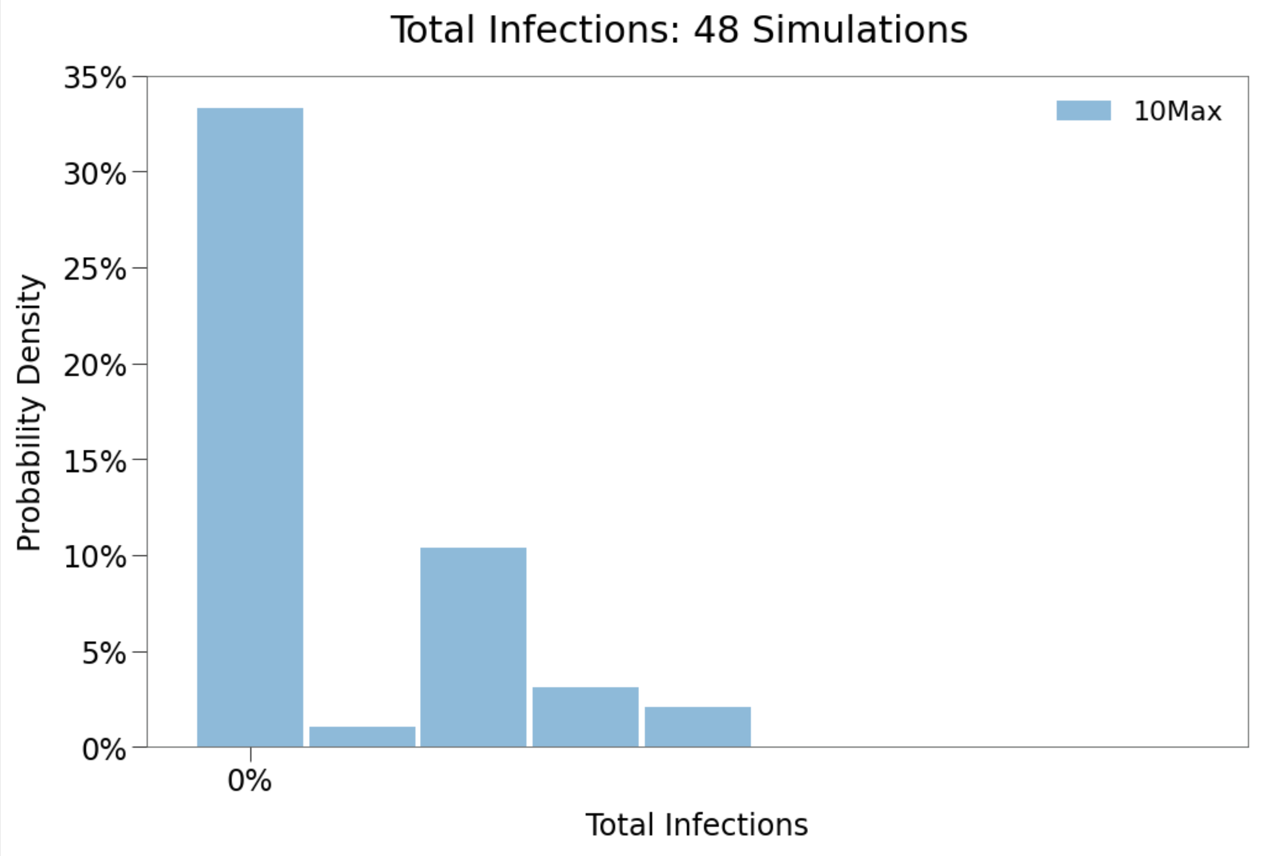
*Reminder that there are 10 subjects per day travelling between states.*

From our prior implementation, we simply add the following code to the `rsxns` keyword of our parameters.

```
wcap = Restriction(
    name='large', start_tick=30, duration=120,
    criteria={'capacity': 10, 'groups': [0,1,2,3,4,5]}, adherence=.9
)
ecap = Restriction(
    name='large', start_tick=30, duration=120,
    criteria={'capacity': 10, 'groups': [6,7,8,9,10,11]}, adherence=.9
)
params_ch['rsxns'] = [wcap, ecap]

sim = Sim(**params_ch)
sim.run(dotlog=True)
```

The results of 48 simulations are shown below:

```
<IPython.core.display.HTML object>
```



We can see from the above that a 10 max daily contact restriction remains highly effective at suppressing spread and eradicating the virus, this despite no restrictions on cross-border travel.

This is because, even if subjects travel cross border, their ability to mix with the population in the visiting state is severly limited.

We can see in the peak distribution below that isolated outbreaks in either state re-emerge somewhat, however, the level of total infections in all scenarios is substantially lower.

```
<IPython.core.display.HTML object>
```

Below we show a sample simulation of a single peak outbreak. Notice that spread is materially curtailed and that fatalities are isolated almost exclusively to the 70+G group in the care homes.

```
<IPython.core.display.HTML object>
```

### 7.5.2 Care Homes - 25Max

We will augment the prior scenario only slightly, replacing the 10 max capacity restriction with a 25 max capacity restrction in each state and tuning the adherence down to 75%.

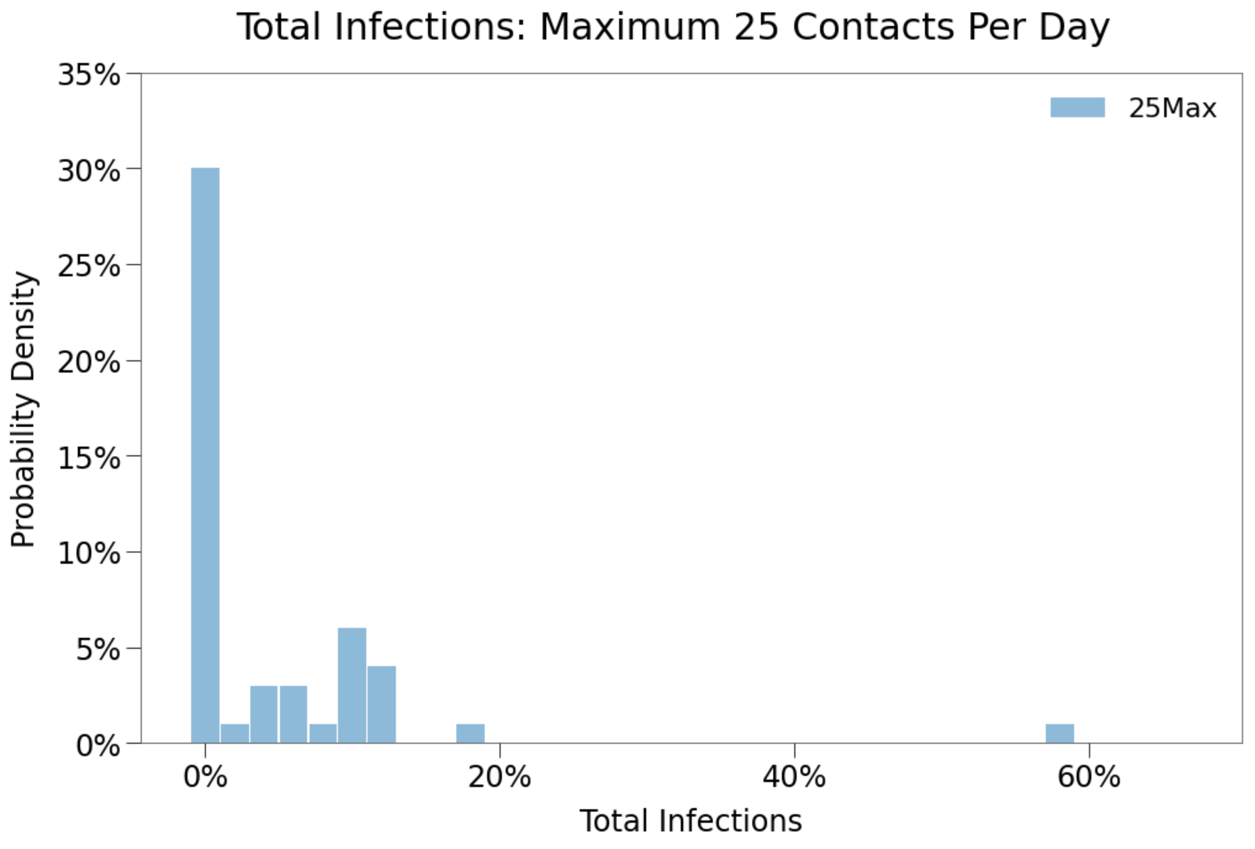
The code is as follows:

‘‘‘python wcap = Restriction( name=’large’, start_tick=30, duration=120, criteria={‘capacity’: 25, ‘groups’: [0,1,2,3,4,5]}, adherence=.75 ) ecap = Restriction( name=’large’, start_tick=30, duration=120, criteria={‘capacity’: 25, ‘groups’: [6,7,8,9,10,11]}, adherence=.75 ) params_ch[‘rsxns’] = [wcap, ecap]

sim = Sim(**params_ch) sim.run(dotlog=True) ‘‘‘

The results of 50 simulations are below:

```
<IPython.core.display.HTML object>
```



As expected, and inline with the same restriction in previous environments, spread impact is substantial, though less than a 10max restrction.

Also, note in the distribution above that there was a single simulation with total infections of ~58%. So, despite the best efforts of the populace under this policy, the virus was able to survive the lockdown and being spread unabated thereafter.

We can see below that peaks in both regions was the prevailing outcome when there was spread, although isolated spread did occur.

Our sample simulations will first focus on that single outcome where total infections reached 59%.

```
<IPython.core.display.HTML object>
```

We can see in the spread curve that both regions had infections essentially driven down to zero during the restriction period. Even as restrictions are lifted, spread remains very limited, only begins to rip higher in both states almost simultaeously at around 210 days. This is a full *60 days after* the restrictions were lifted.

We can see just how close the virus was to be eradicated in the sim logs.

Scipy can help us quickly find local minimums in the `curr_inf` log via `argrelextrema`.

[50]:

```
from scipy.signal import argrelextrema

sim = ch25sims[6]
mins, = argrelextrema(sim.log['curr_inf'], np.less)

lowest = sim.log['curr_inf'][mins].min()
i_lowest = np.argwhere(sim.log["curr_inf"] == 44).ravel()[0]
print (f'Low: {lowest}')
print (f'Day of Low: {i_lowest}')
```

```
Low: 44.0
Day of Low: 203
```

So we know the lowest number of infections reached during the midpoint of the outbreak was 44 on Day 203. We can prove this by isolating them in the dotlog.

[52]:

```
from rknot.dots import MATRIX_COL_LABELS as ML
infmask = sim.dotlog[i_lowest][:, ML['is_inf']] == 1
infdots = sim.dotlog[i_lowest][infmask]
print (infdots.shape)
```

```
(44, 27)
```

And we can show that all infected subjects were in the West State:

[53]:

```
np.all(infdots[:, ML['group_id']] <= 5)
```

[53]:

```
True
```

Finally, below we show the more typical outcome from this restriction, which is a quick run up in infections prior to and just after restrictions are put in place. Then, as the restrictions take hold, spread deflates and is eliminated very quickly.

```
<IPython.core.display.HTML object>
```

## 7.6 Independent Policy

Now, we will explore some more complex structures.

In the *dynamic transmission model*, we have seen policy restrictions can be successful at supressing spread and ultimately eradicating the virus (see *here* and *here*). In fact, these policy measures have proven fairly resilient even in the face of low adherence.

But what happens when there are is mixing between multiple jurisdictions that take independent approaches to policy?

We will build from the *Care Homes* scenario above.

In this sim, West State will be the more aggressive actor, implementing faster and more imposing policy. The East State will act more slowly and with less imposition. Accordingly, the resrictions will be:

**West State**:

- Maximum 10 contacts, from Day 30 to Day 60
- Maximum 25 contacts, from Day 60 to Day 105
- Maximum 75 contacts, from Day 105 to Day 150

***This was a common approach for many of the northern US states and Canadian provinces in the Spring of 2020.***

**East State**:

- Maximum 25 contacts, from Day 75 to Day 105
- Maximum 75 contacts, from Day 105 to Day 135

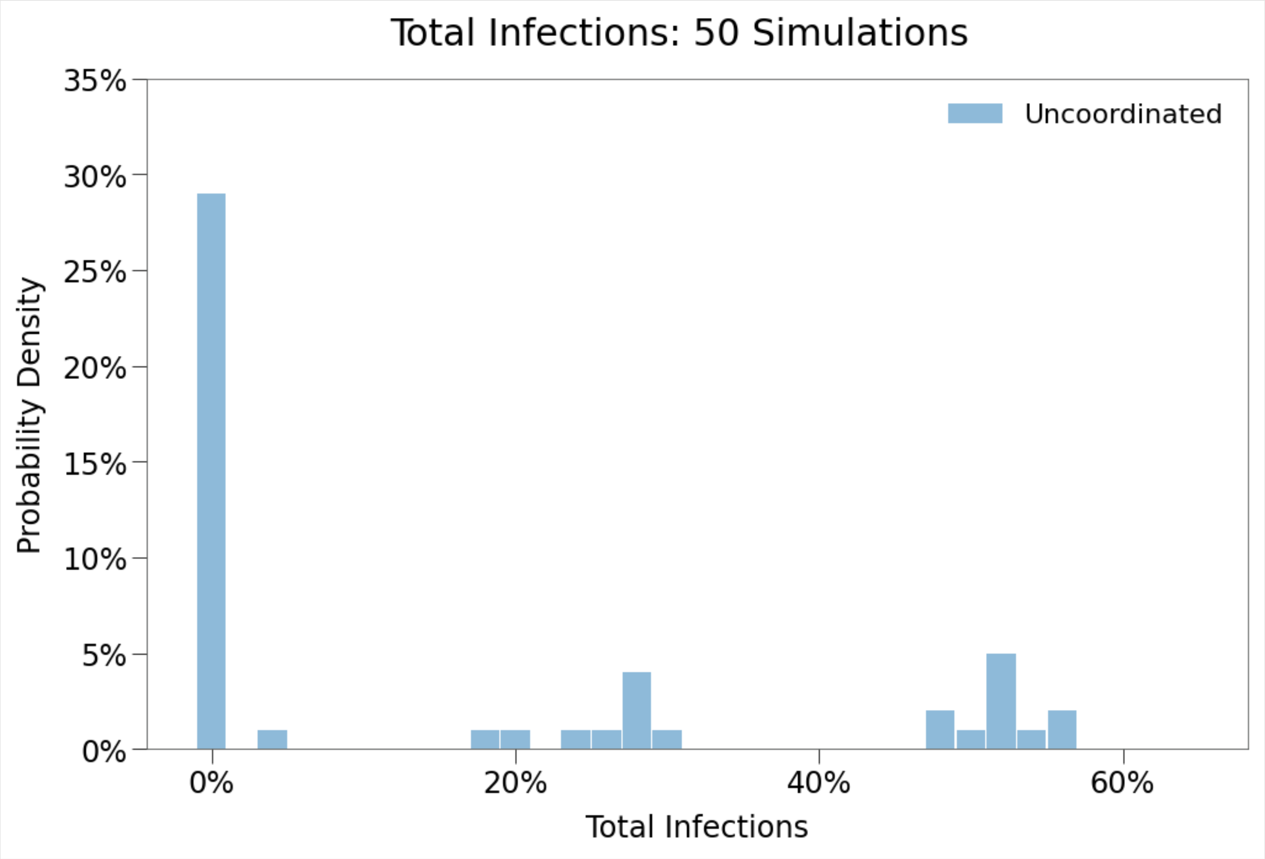
Under isolated circumstances, the West State would most often experience no outbreak and at worst only a limited one. When bordering with a less rigourous neighbor, the results prove to be very different.

Below is the code adding these restrictions.

```
wcap1 = Restriction(
    name='wcap1', start_tick=30, duration=30,
    criteria={'capacity': 10, 'from_boxes': ['West State', 'EtoW']}, adherence=.9
)
wcap2 = Restriction(
    name='wcap2', start_tick=30, duration=45,
    criteria={'capacity': 25, 'from_boxes': ['West State', 'EtoW']}, adherence=.75
)
wcap3 = Restriction(
    name='wcap3', start_tick=75, duration=45,
    criteria={'capacity': 75, 'from_boxes': ['West State', 'EtoW']}, adherence=.75
)


ecap1 = Restriction(
    name='ecap1', start_tick=75, duration=30,
    criteria={'capacity': 25, 'from_boxes': ['East State', 'WtoE']}, adherence=.5
)
ecap2 = Restriction(
    name='ecap2', start_tick=105, duration=30,
    criteria={'capacity': 75, 'from_boxes': ['East State', 'WtoE']}, adherence=.5
)

params['rsxns'] = [wcap1, wcap2, wcap3, ecap1, ecap2]
sim = Sim(**params)
sim.run(dotlog=True)
```

The results of 50 simulations are shown below:

```
<IPython.core.display.HTML object>
```



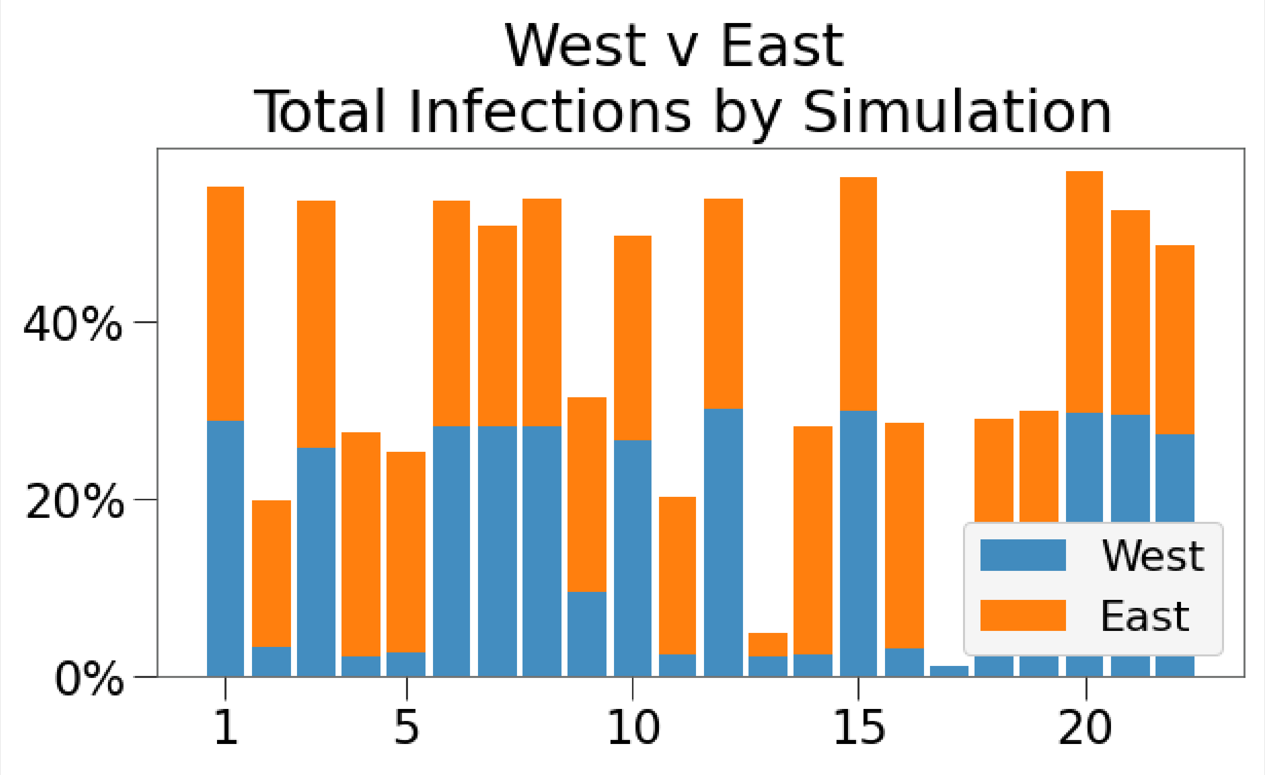
Spread comes back with a vengenance in this scenario, showing very limited impact from the implementations, although the time to peak does increase for reasons we'll see below.

In the histrogram above, note that two local maxima have once again formed, suggesting outcomes with both single peaks and double peaks, including a maxima around ~55% total infections, which is consistent with full outbreaks across both states (despite the strict implementations int eh West!).

We can the prevalence of double peaks below.

```
<IPython.core.display.HTML object>
```

We can also so the composition of spread amongst the two regions in the bar chart below:



Of the 22 simulations where spread was greater than 1%, the West State accounted for about half of all infections about half the time. So, where its implementations in isolation would be expected to reduce the risk of an outbreak to only 1%, here its risk remains 25%.

Below we show an example of such a double peak.

```
<IPython.core.display.HTML object>
```

Spread begins in the East and is eventually carried into the West. The East State's policies have essentially zero impact on spread, as the peak appears to occur *before* its 25 max restriction is implemented.

We can see a very modest peak in the West early in the sim and, essentially on queue with the loosening of restrictions, around Day 110 spread takes hold and rips through the West State.

We could map this onto a real world scenario where:

- A new virus is discovered. Both states are unsure how widespread it may be.
- West State locks down as a preventative measure, driven by a focus on public health. East State does not, driven by a focus on personal freedoms / responsibility / etc.
- West State derides East State for its lack of action.
- West State, believeing its outbreak is totally under control *and* seeing the outbreak on the downside, beings to reopen.
- Because the virus is not fully eradicated, West State inherits the virus from East State and, because West State does not have herd immunity, the virus spreads unabated.
- West State's policy actions are for not.

Finally, we also show the example of a single peak where the West avoids the 2nd wave of spread.

```
<IPython.core.display.HTML object>
```