Report 1: A study of herd immunity using SIRS model

Akshay Shankar, MS18117

This report records an attempt to create and probe a minimal disease spreading model using a cellular automaton (SIRS) with a slight variation being the introduction of a new state, namely "permanently immune". The goal of this analysis is to understand the viability of herd immunity, i.e, the containment of an epidemic via immunization of a percentage of the population. The basic SIRS model is described below.

1 The SIRS model

This well known model is a cellular automaton with 3 states; Susceptible, Infected and Recovering. We use a 2-dimensional lattice where each node represents an individual as they progress through the disease cycle. Time is a discrete quantity and each time-step the entire grid is simultaneously updated according to the local rule.

1.1 States and Parameters

The disease cycle is characterized by two integers; the length of infection period, τ_I and the length of recovery period, τ_R. We then have $\tau_0 = \tau_I + \tau_R$ such that the total disease cycle is $\tau_0 + 1$ time-steps long. At time t, the node (i, j) contains an integral value $\tau_{i,j}(t)$ such that $0 \leq \tau_{i,j}(t) \leq \tau_0$. The 3 automaton states are characterized in the following way:

1. **Susceptible (S):** $\tau_{i,j}(t) = 0$
2. **Infected (I):** $1 \leq \tau_{i,j}(t) \leq \tau_I$
3. **Recovering (R):** $\tau_I < \tau_{i,j}(t) \leq \tau_R$

1.2 Local rule

The systems evolves with a combination of stochastic and deterministic steps. These are succinctly displayed below:

$$\tau_{i,j}(t+1) = \begin{cases} 1 & \text{with probability q} \\ 0 & \text{with probability 1-q} \end{cases} \qquad \text{if } \tau_{i,j}(t) = 0$$

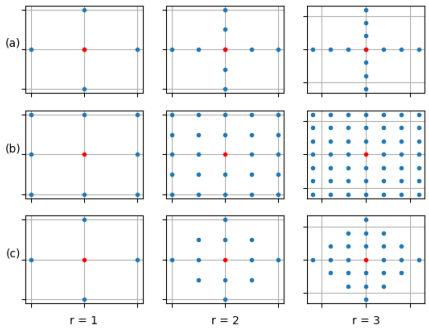
$$\tau_{i,j}(t+1) = \tau_{i,j}(t) + 1 \qquad \text{if } 1 \leq \tau_{i,j}(t) \leq \tau_I$$

$$\tau_{i,j}(t+1) = 0 \qquad \text{if } \tau_{i,j}(t) = \tau_0$$

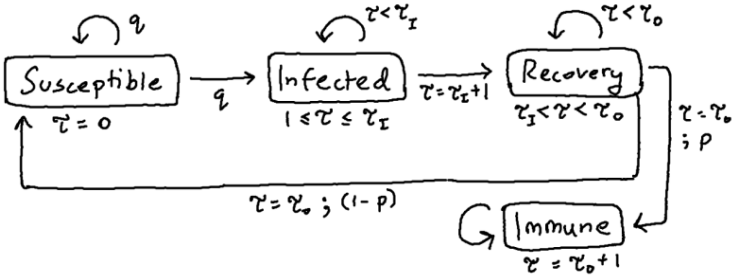
The probability $q = n_{infected}/n_{total}$ where n is the number of neighbouring nodes. So, we see the $S \to I$ transition is stochastic, and the rest of the cycle progresses in a deterministic way. As we shall see, this produces interesting dynamics similar to actual spreading of infectious diseases.

1.3 Boundary conditions and neighbourhood

A fixed boundary condition is used for this analysis, where a padding of static nodes around the grid are stuck in the Recovery state ($\tau_I + 1$) such that it forms an infection-absorbing boundary. The neighbourhoods used are displayed below.



(a) Neighbourhood types with varying radius: (a) Plus (b) Moore (c) Von Neumann



(b) State space and evolution of the automaton

1.4 Modification to the model

A new state, "permanently immune" is added where $\tau = \tau_0 + 1$ and once the state is reached it remains there, effectively removing the node from the susceptible pool. This immunity is granted based on a probabilistic parameter p_I and the third rule is altered as follows:

$$\tau_{i,j}(t+1) = \begin{cases} 0 & \text{with probability } 1 - p_I \\ \tau_0 + 1 & \text{with probability } p_I \end{cases} \qquad \text{if } \tau_{i,j}(t) = \tau_0$$

$$\tau_{i,j}(t+1) = \tau_{i,j}(t) \qquad \text{if } \tau_{i,j}(t) = \tau_0 + 1$$

This means that everyone has a finite immunity period (τ_R), but a few lucky individuals obtain 'permanent' immunity (relative to the timescale of the simulation at the very least). As a result, every simulation run will terminate. When the infection dies out, the remaining population is either susceptible or completely immune to the disease.

2 Analysis

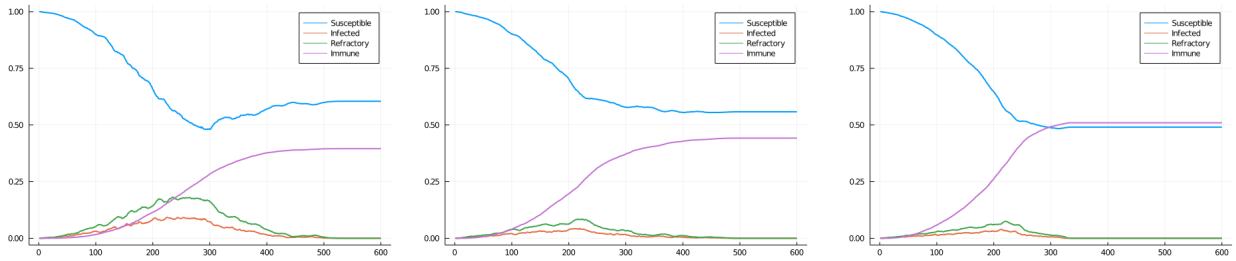
Since every simulation run will terminate, we record the fraction of individuals immune (**%I**) at the end of each run. The goal is to study how this varies with the model parameters in order to understand how effective herd immunity can be. Our fixed control parameters will be a **100x100 grid**, running for **600 time steps** (or till infection dies out, whichever comes first), $\tau_0 = 15$; this let us vary the neighbourhood and τ_I, τ_R, p_I values. The colors chosen for the plotting are; **(black)-(red)-(orange/yellow)-(grey) for (S-I-R-Im)** and are used as gradients to indicate the stage within the state. Every point plotted in a graph is obtained by conducting 5 simulation runs and averaging them.

Initial Conditions: A single infected seed ($\tau = 1$) is placed at (50, 50), while others are Susceptible.

Note: The analysis performed involves recording only **%I** and not the number of time-steps t_{net} taken for the infection to die out since t_{net} was found to vary with huge error bounds which provides us minimal insight to our question.

2.1 Variation with probability p_I

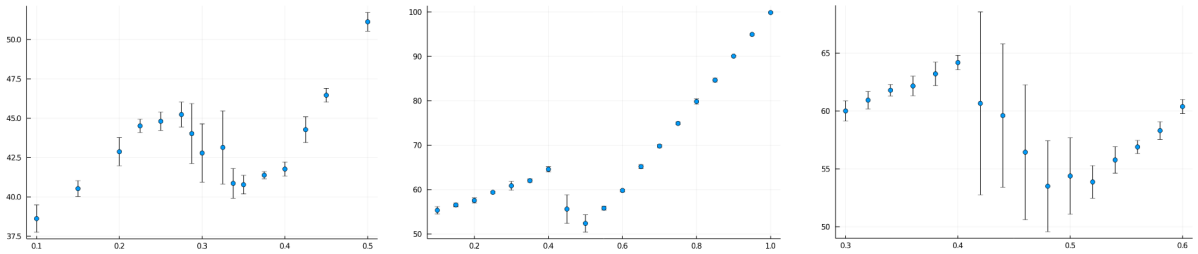
Naively we might expect the $\%I$ to increase linearly with p_I, but analysis performed unveils something more. We plot the SIR curves for three cases (Fig. 2) and notice a single infection peak, after which it dies out and the Susceptible and Immune reach a static equilibrium as expected. We also notice increasing $\%I$ with higher p_I. To further investigate, we plot the % Immune (**%I**) vs. probability $\mathbf{p_I}$ holding other parameters constant in (Fig. 3).



(a) p = 0.1 (b) p = 0.3 (c) p = 0.5

Figure 2: **%I vs. $\mathbf{p_I}$:** $\tau_I = 5$, $\tau_R = 10$, Plus neighbour; r=1

We have plotted the curve for two neighbourhood radii (Fig. 3) and initially there is a linear increase of %I with p_I. However, with increasing p_I there is a kind of phase transition during which the behaviour is erratic. After this point, the linear increase continues with a larger slope. Fig 3. (c) shows the zoomed area of the transition where the error shoots up, indicating wildly fluctuating results in that region.

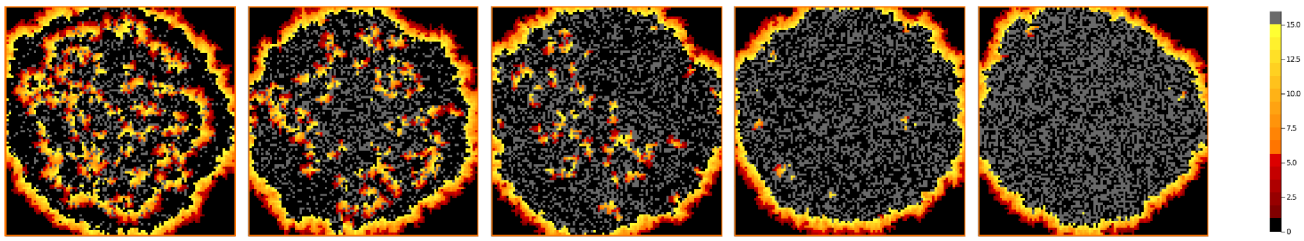


(a) r=1 (b) r=2 (c) Zoomed transition in (b)

Figure 3: **%I vs. $\mathbf{p_I}$:** $\tau_I = 5$, $\tau_R = 10$, Plus neighbourhood

This raises the question of why such a transition exists, and we put forth a possible explanation by examining the actual grids (Fig. 4) corresponding to the plots in Fig. 2. There are two processes that occur here;

1. Increased infection spread allows more chance to become immune.
2. More number of immune nodes create a percolation network which hinders the spread of infection and hence reduces further immunization of the population.



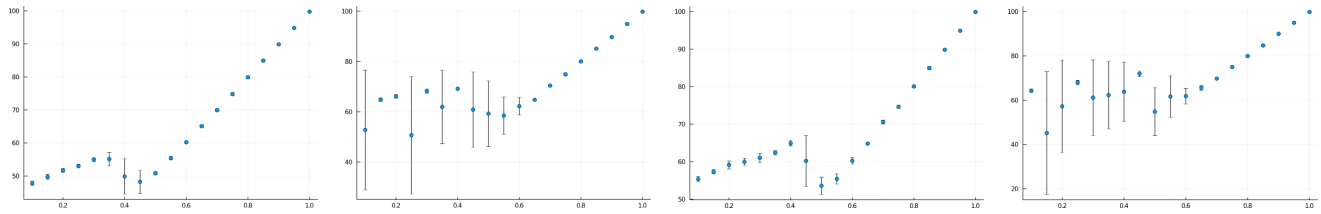
(a) p = 0.1 (b) p = 0.2 (c) p = 0.3 (d) p = 0.4 (e) p = 0.5

Figure 4: $\tau_I = 5$, $\tau_R = 10$, Plus neighbour w/ r=1, time-step = 130

The grid plots in (Fig. 4) support this fact; as p_I is increased, the distribution of immune nodes is more dense thereby creating a barrier for further spreading of the disease. So these two processes are competitive and we would expect some kind of resulting trend, possibly a local minimum in the graph or a region of uncertain behaviour, after which one of the processes dominate.

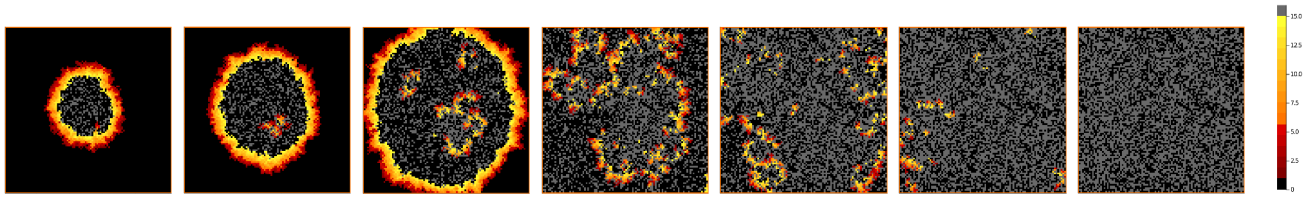
2.2 Variation with Neighbourhoods

In (Fig. 5), the smaller radius neighbourhoods exhibit similar trends as previously observed. However, for larger radius neighbourhoods, $\%I$ for low p_I is erratic whereas for higher p_I it is the expected linear increase. This could be explained by looking at the grid images (Fig. 6, 7, 8) for a specific neighbourhood.



(a) Moore; r = 1 (b) Moore; r = 2 (c) Von Neumann; r = 2 (d) Von Neumann; r = 3

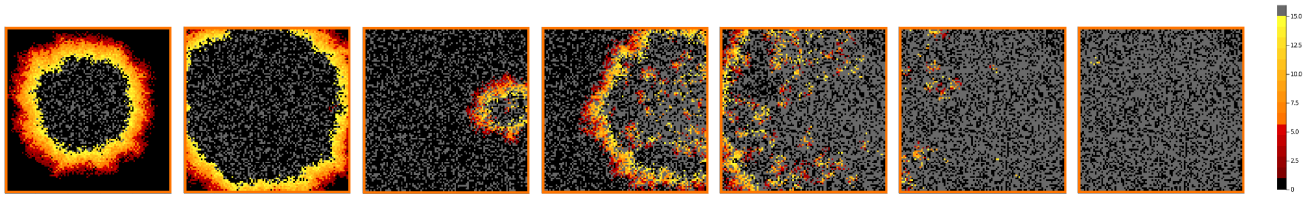
Figure 5: **%I vs. p_I:** $\tau_I = 5$, $\tau_R = 10$



(a) t = 50 (b) t = 75 (c) t = 100 (d) t = 150 (e) t = 200 (f) t = 250 (g) t = 300

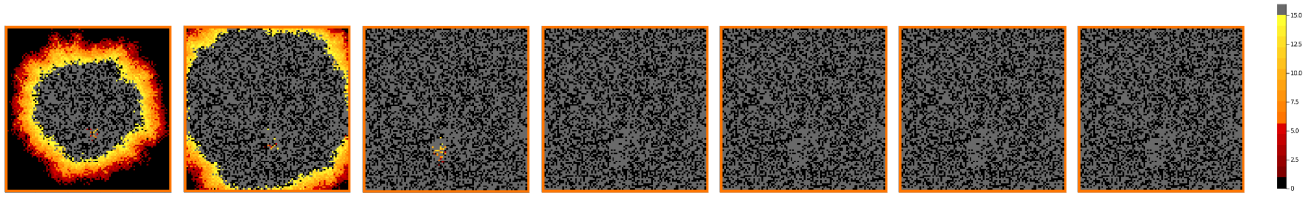
Figure 6: $\tau_I = 5$, $\tau_R = 10$, $p_I = 0.3$, Moore neighbour; r=1

When p_I is low but neighbourhood is large, the wave has larger extent and inhibits back propagation, but we see in (Fig. 7), the second infection wave is more potent and introduces denser immune nodes due to the larger influence of each node. As opposed to (Fig. 6) where the radius = 1 and secondary waves fizzle out quickly.



(a) t = 50 (b) t = 75 (c) t = 100 (d) t = 150 (e) t = 200 (f) t = 250 (g) t = 300

Figure 7: $\tau_I = 5$, $\tau_R = 10$, $p_I = 0.3$, Moore neighbour; r=2



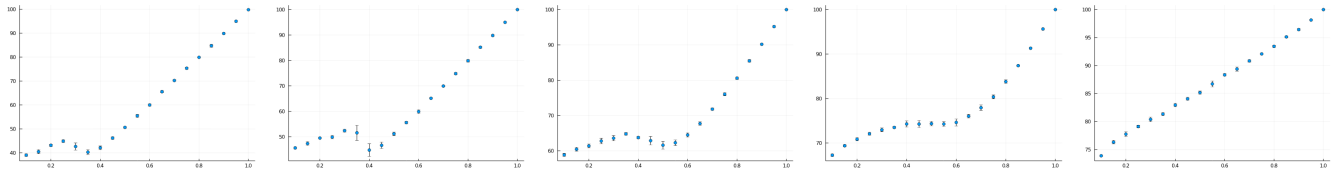
(a) t = 50 (b) t = 75 (c) t = 100 (d) t = 150 (e) t = 200 (f) t = 250 (g) t = 300

Figure 8: $\tau_I = 5$, $\tau_R = 10$, $p_I = 0.6$, Moore neighbour; r=2

The terminal $\%I$ value strongly depends on where/when the secondary infection waves begin to propagate which is fairly random, resulting in large error bars. On the other hand, for higher p_I (Fig. 8), the larger neighbourhood radius is countered by the immediately denser formation of immune nodes, thereby allowing only a single infection wave to propagate. This is in agreement with the linear increase in (Fig. 5 (b), (d)) after a threshold p_I.

2.3 Variation with τ_I and τ_R

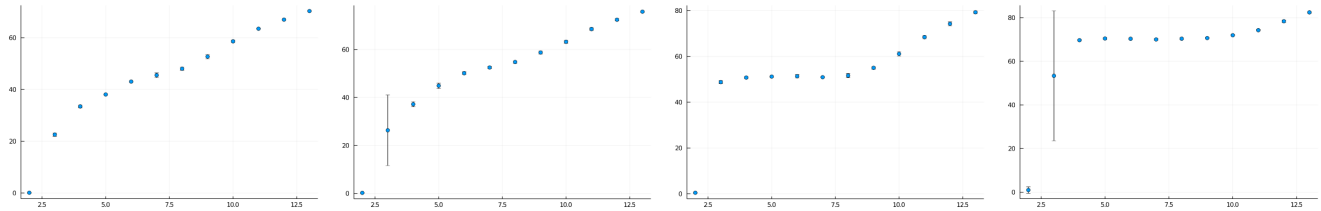
We now vary the cycle parameters τ_I and τ_R while fixing $\tau_0 = 15$, and plot $\%I$ against p_I (Fig. 9). We see that as τ_I is increased, the transition point shifts further to the right and flattens till it becomes nearly linear. As τ_I is increased, each wave tends to produce more immune nodes, on the other hand τ_R is decreased and this would make creation of secondary waves more likely. Somehow these effects work together to shift and flatten the transition in the curve, but we are unable to come up with a complete explanation.



(a) $\tau_I = 5$, $\tau_R = 10$ (b) $\tau_I = 7$, $\tau_R = 8$ (c) $\tau_I = 10$, $\tau_R = 5$ (d) $\tau_I = 13$, $\tau_R = 2$ (e) $\tau_I = 15$, $\tau_R = 0$

Figure 9: **%I vs. p_I:** Von Neumann neighbour; r=1

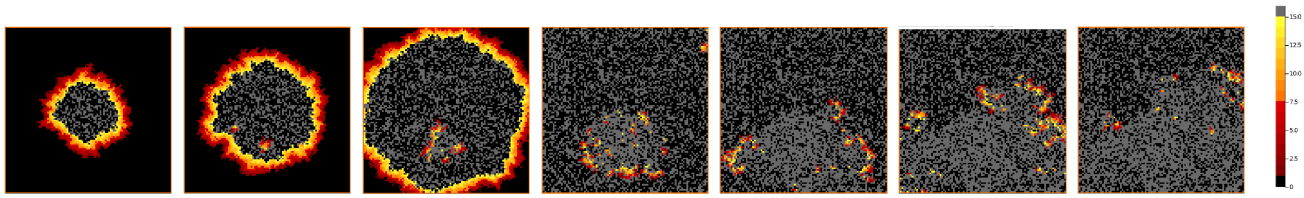
In (Fig. 10) we fix p_I and plot $\%I$ against τ_I (or equivalently $15 - \tau_R$). We see that for small p_I we obtain a somewhat linear increase with increasing τ_I which is as expected. As probability of gaining immunity increases, all the cases seem to converge to a specific value of $\%I$ eventually tending to 100% ($p_I = 1.0$). This is obvious from the fact that a single wave of infection is enough to immunize most of the population when $p_I \to 1.0$.



(a) $p_I = 0.1$ (b) $p_I = 0.3$ (c) $p_I = 0.5$ (d) $p_I = 0.7$

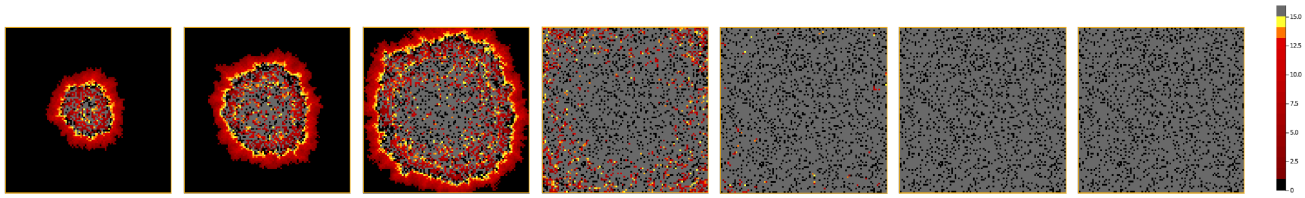
Figure 10: **%I vs. τ_I:** $2 \leq \tau_I \leq 13$, $\tau_R = 15 - \tau_I$, Von Neumann neighbour; r=1

As we can see from (Fig. 11, 12), higher infectivity (longer τ_I) creates more dense networks of immune nodes, but since transmission can happen for longer periods, we notice in (Fig 12(d)), the small gaps between the nodes are filled with infection seeds that infect more Susceptibles. This is an undesirable scenario since it involves more people getting infected before the end of the epidemic.



(a) t = 50 (b) t = 75 (c) t = 100 (d) t = 150 (e) t = 200 (f) t = 250 (g) t = 300

Figure 11: $\tau_I = 7$, $\tau_R = 8$, $p_I = 0.4$, Moore neighbour; r=1



(a) t = 50 (b) t = 75 (c) t = 100 (d) t = 150 (e) t = 200 (f) t = 250 (g) t = 300

Figure 12: $\tau_I = 13$, $\tau_R = 2$, $p_I = 0.4$, Moore neighbour; r=1

3 Physical Interpretation of Results

3.1 Average no. of infections per person

The value of %I at the end of the epidemic tells us how much of the population has to gain permanent immunity to contain an epidemic. Using this information, we can also deduce a lower bound on $\langle n_{inf} \rangle$ the average number of times an individual needs to be infected to attain this state.

Let the probability of gaining immunity every cycle be p_I and we note the terminal ratio of Immune individuals r_{imm} in a total population of N. If the total number of infections till the epidemic ends is N_{inf}, we have:

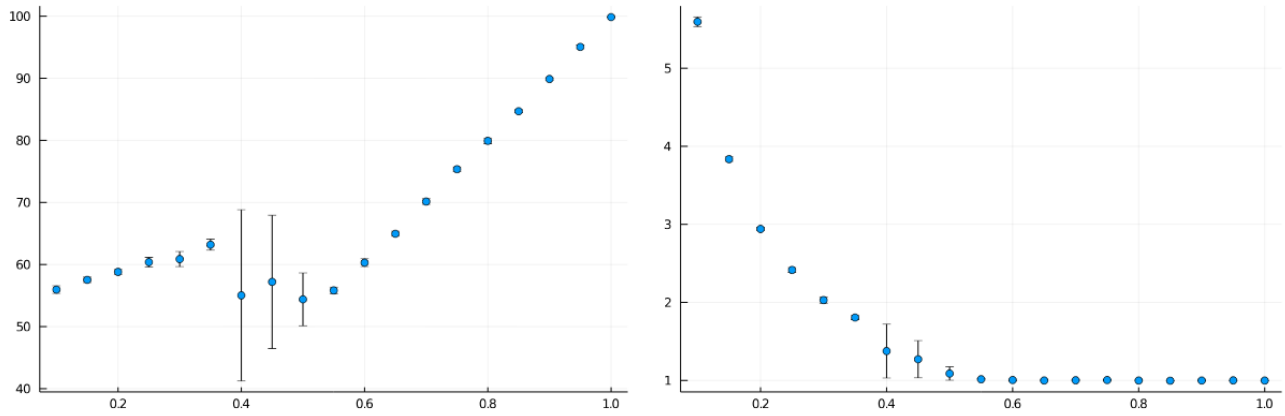
$$p_I \cdot N_{inf} \geq N \cdot r_{imm}$$

$$\implies N_{inf} \geq N \cdot \frac{r_{imm}}{p_I}$$

.

$$\implies \langle n_{inf} \rangle = \frac{N_{inf}}{N} \geq \frac{r_{imm}}{p_I}$$

As shown in (Fig. 9), we can see that $\langle n_{inf} \rangle$ sharply falls off with p_I, but in reality we may not have the option of setting $p_I \to 1$ so we must choose the best alternative. In this specific case, we see that the epidemic dies out even if $p_I = 0.3$ just by infecting each person twice on average. But clearly, for small p_I the outcomes is extremely undesirable, since the cost of gaining immunity is getting infected 5 times on average!



(a) %I vs. p_I (b) $\langle n_{inf} \rangle_{min}$ vs. p_I

Figure 13: $\tau_I = 5$, $\tau_R = 10$, Von Neumann neighbour; r=2

3.2 Conclusions

Ideally, we would want to keep $\langle n_{inf} \rangle$ minimum which then becomes a balancing act between r_{imm} and p_I. Now, p_I which is the probability of gaining immunity can either happen naturally if the disease is weak, or it may due to limited supply of vaccines. We shall focus on the second possibility which poses a constraint on p_I.

- We see that for any level of infectivity of the disease, there exists a sweet spot of minimum $\%I$ (the transition point) and we can set p_I accordingly to minimize $\langle n_{inf} \rangle$ at the end of the epidemic. However, if the disease is more infectious and we have less chance of gaining temporary immunity ($\tau_R \to 0$), the transition point shifts to higher p_I thus making it difficult to minimize $\langle n_{inf} \rangle$.
- Another facet is the negligence of the population. If people insist on maintaining social interactions (larger neighbourhoods), the offset of $\%I$ increases by default meaning more people must gain immunity to stop the disease spread. Furthermore, for low p_I we also lose predictability of $\%I$ (Fig. 5(b)(d)) which is also undesirable. In that case, setting a larger p_I seems to be a better option.
- In the situation where no vaccine exists and thereby we lose the ability to fix p_I, we may have some minimal p_I for permanent immunity by virtue of how the disease interacts with the immune system. In this case, the clear option is to reduce infectivity (Fig. 10), which can only be done by reducing neighbourhood sizes.

3.3 Possible extension to other scenarios

By changing the interpretation of the "permanently immune" state, our analysis still holds true and may provide further insight into other questions.

1. Immune could be replaced with 'Death' in which case p_I (mortality rate) is fixed by the disease and we must minimize $\%I$ (number of deaths) and $\langle n_{inf} \rangle$ (av. number of infections per person).
2. Immune could be replaced by 'Quarantine' where the infected nodes are randomly removed from the population to prevent disease spread. In this case, one might fix p_I based on resources needed for quarantining a single person and minimizing p_I will reduce costs. This is now a trade-off between $\%I$ and $\langle n_{inf} \rangle$.

In this way, the trends we have observed for the $\%I$ vs. p_I curves can be utilized in situations beyond the one discussed here.

4 Summary

We have seen that once we introduce permanent immunity, the epidemic will usually come to an end without needing the entire population to become immune, thus herd immunity in a technical sense always will kick in. Through our analysis we note that the % of the population required to gain immunity depends on the probability of gaining immunity in a non linear fashion. This allows us to set the free parameters (neighbourhood shapes, p_I) so as to reduce number of infections per person required for the epidemic to end.

4.1 Implementation

The model was implemented and analysed in Python 3.8.5 and later in Julia 1.4.2 for better performance. The graphs were plotted using Plots.jl and Images.jl. The Jupyter notebook used to perform the analysis can be found here:
`https://github.com/20akshay00/ModellingComplexSystems`.

```
begin
@time using Plots
@time using Images
@time using ColorSchemes
end
```

Subroutine to calculate neighbour indices

nType:

Plus = 0

Moore = 1

Von Neumann = 2

```
neighbourIndices (generic function with 1 method)
```

```
function neighbourIndices(nType::Int64, rad::Int64)
    indices = []
    if(nType == 0)
        indices = vcat([[i, 0] for i in -rad:rad], [[0, j] for j in -rad:rad])
    elseif(nType == 1)
        indices = [[i, j] for i in -rad:rad for j in -rad:rad]
    elseif(nType == 2)
        indices = [[i, j] for j in -rad:rad for i in (abs(j) - rad):(rad - abs(j))]
    end

    indices = [CartesianIndex(ind...) for ind in setdiff(indices, [[0,0]])]
end
```

Subroutine to set initial conditions

```
setInitConditions (generic function with 1 method)
```

```
function setInitConditions(grid, init, tauI, tauR, indices)
    if(init[1] == "random")
        for index in indices
            if(rand() < init[2])
                grid[index] = tauI+1
            end
        end
        grid[init[3], init[4]] = 1

    elseif(init[1] == "assorted")
        for index in indices
            if(rand() < init[2])
                grid[index] = 1
            elseif(rand() < init[3])
                grid[index] = tauI+1
            end
        end

    elseif(init[1] == "single")
        grid[init[2], init[3]] = 1

    elseif(init[1] == "corners")
```

```
        grid[rad + 2, rad + 2] = 1
        grid[rad + 2, n - rad - 2] = 1
        grid[n - rad - 2, rad + 2] = 1
        grid[n - rad - 2, n - rad - 2]  = 1

    elseif(init[1] == "custom")
        grid[init[2], init[3]] = 1
        grid[init[4], init[5]] = tauI + 1
    end

    return
end
```

SIRS Simulation code

```
SIRSmodel (generic function with 1 method)
```

```
function SIRSmodel(n::Int64, tauI::Int64, tauR::Int64, pImmune::Float64,
nType::Int64, rad::Int64, nsteps::Int64, anim::Bool, colbar::Bool, fname, init)
    grid = zeros(Int64, (n + 2*rad, n + 2*rad)) #2D grid
    nData = zeros(Int64, (nsteps, 4)) #S,I,R count
    frames = [] #to store grid for display
    nTime = 1 #notes number of time steps passed

    infp = 0 #infection probability
    tau0 = tauR + tauI #complete disease cycle
    isInfected(x) = 1 <= x <= tauI #infected check

    relNeighbour = neighbourIndices(nType, rad) #relative index of neighbours
    indices = [CartesianIndex(i,j) for i in (rad+1):(n+rad) for j in (rad+1):
(n+rad)]

    #initial conditions
    setInitConditions(grid, init, tauI, tauR, indices)

    #boundary conditions; fixed
    grid[1:rad, :] .= tauI + 1
    grid[(n+rad+1):(n+2*rad), :] .= tauI + 1
    grid[:, 1:rad] .= tauI + 1
    grid[:, (n+rad+1):(n+2*rad)] .= tauI + 1

    prev = copy(grid) #copy of the grid
    push!(frames, copy(grid))

    #initial count
    nData[1, 1] = count(x -> x==0, grid)
    nData[1, 2] = count(x -> 1 <= x <= tauI, grid[indices])
    nData[1, 3] = count(x -> tauI < x <= tau0, grid[indices])
    nData[1, 4] = count(x -> tau0 < x, grid[indices])

    for k in 2:nsteps
        nData[k, :] = nData[k-1, :]

        for index in indices
            if(prev[index] == 0) #susceptible
                infp = count(isInfected.(prev[relNeighbour .|> x->
index+x]))/length(relNeighbour)

                if(rand()<infp) #infection
                    grid[index] += 1
                    nData[k, 1] -= 1
                    nData[k, 2] += 1
                end

            elseif(prev[index] < tau0) #infected/refractory
```

```
                grid[index] += 1

                if(grid[index] == tauI + 1) #transition to R
                    nData[k, 2] -= 1
                    nData[k, 3] += 1
                end

            elseif(prev[index] == tau0) #transition to S
                nData[k, 3] -= 1

                if(rand() < pImmune)
                    grid[index] += 1
                    nData[k, 4] += 1
                else
                    grid[index] = 0
                    nData[k, 1] += 1
                end

            end
        end

        nTime += 1
        prev = copy(grid)
        push!(frames, copy(grid))

        if(nData[k, 2] == 0 && nData[k, 3] == 0) break end

    end

    #color gradient for plotting
    colors = vcat([RGB{N0f8}(0, 0, 0)], range(RGB{N0f8}(0.5,0.0,0.0), stop=RGB{N0f8}
(0.862,0,0), length= tauI),range(RGB{N0f8}(1,0.458,0.0), stop=RGB{N0f8}(1, 1,
0.196), length=tauR), [RGB{N0f8}(0.411,0.411,0.411)])

    if(colbar && anim) #with colorbar
        animGIF = @animate for i = 1:nTime
            heatmap(frames[i], c = palette(colors, tau0+1), clim = (0, tau0))
        end
        gif(animGIF, fname, fps = 15)

    elseif(anim) #without colorbar
        cmap = Dict(zip(0:tau0+1, colors))
        animGIF = @animate for i = 1:nTime
            plot([cmap[p] for p in frames[i]])
        end
        gif(animGIF, fname, fps = 15)
    end

    nData[nTime+1:end, 1] .= nData[nTime, 1]
    nData[nTime+1:end, 4] .= nData[nTime, 4]

    return [nData, nTime]
end
```

```
Enter cell code...
```