C++ Project

Infectious Disease Simulation

BY: Ethan Hoyt (esh774)

This project demonstrates a miniature scale model for the spread of diseases. Two models were coded and tested with various controlled conditions. The first, the Contagion Model, allows for the spread of disease in a population only to the neighbor (i.e. adjacent element in an array). Whereas, the second model, the SIR Model, is a more realistic representation and allows for the spread of disease based on contact with 'n' people per day/step. The infection duration was set to 6 days for every test, and the number of people an infected person could come in contact with per day/step was arbitrarily set to 6 days. Additionally, the probability an infected person could infect another person in population, the probability of contagion, and the percentage of the population that was already immune to the disease, the percentage of inoculation, were controlled variables. Lastly, population size was varied from 26 people to 12 people.

From the outset, I believed that the Contagion Model would take much longer for the disease to dissipate simply because the contact number is at most two people versus six for one person in the SIR Model. Many tests corroborated this too as the mean days till contagion was eliminated was far greater. The histograms below, created in MATLAB, in Figures 1 through 6 display all the results from the experiment. Thirty tests were conducted for a specified population size, probability of contagion and percentage of inoculation for both models.

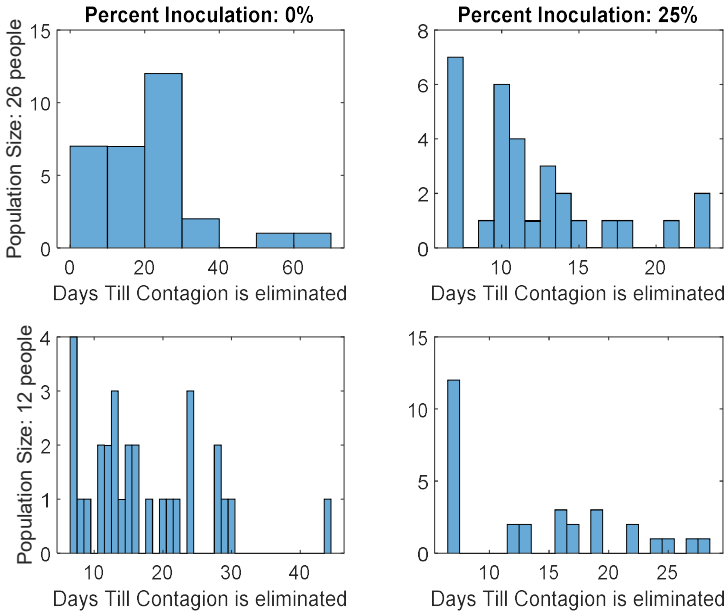

Figure 1: Contagion Model: Contagion probability of 25%

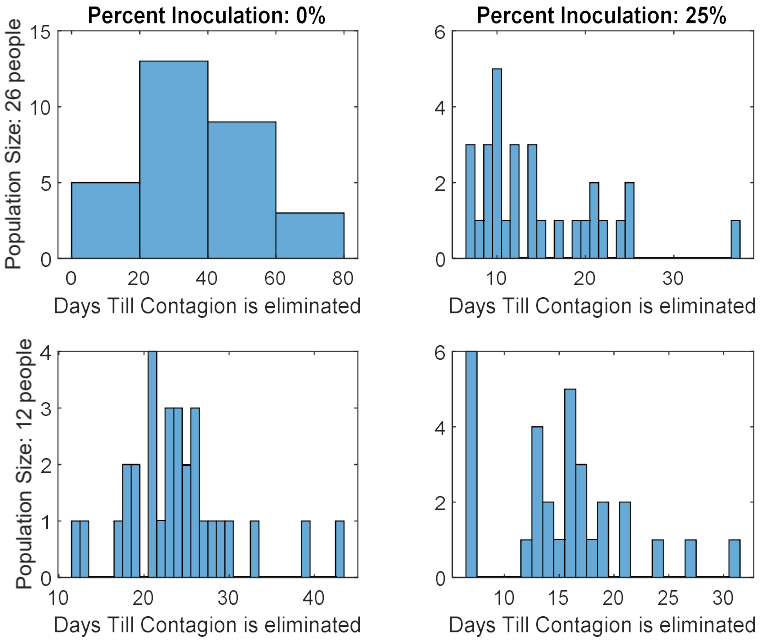

Figure 2: Contagion Model: Contagion probability of 50%

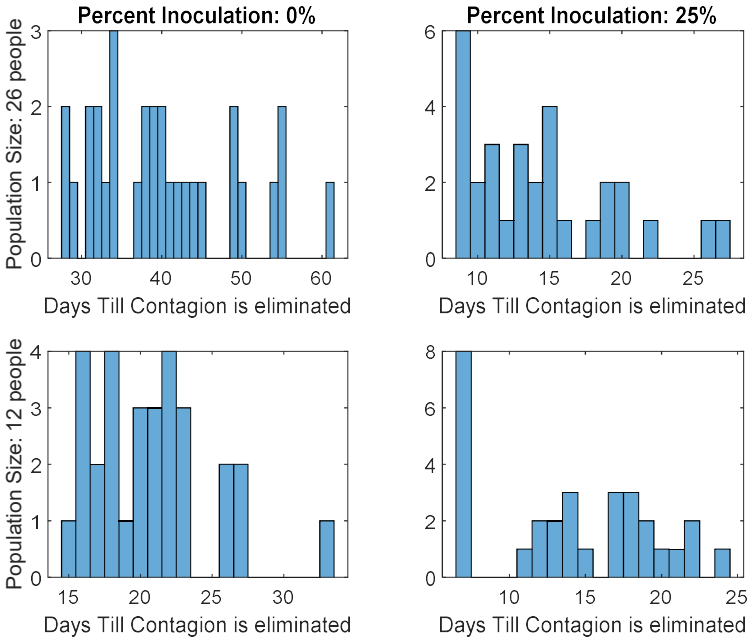

Figure 3: Contagion Model: Contagion probability of 75%

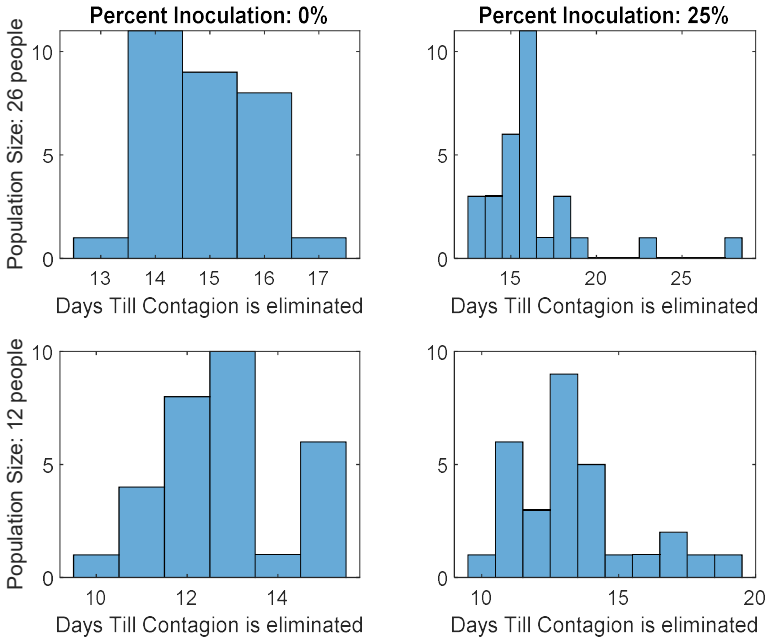

Figure 4: SIR Model: Contagion probability of 25%

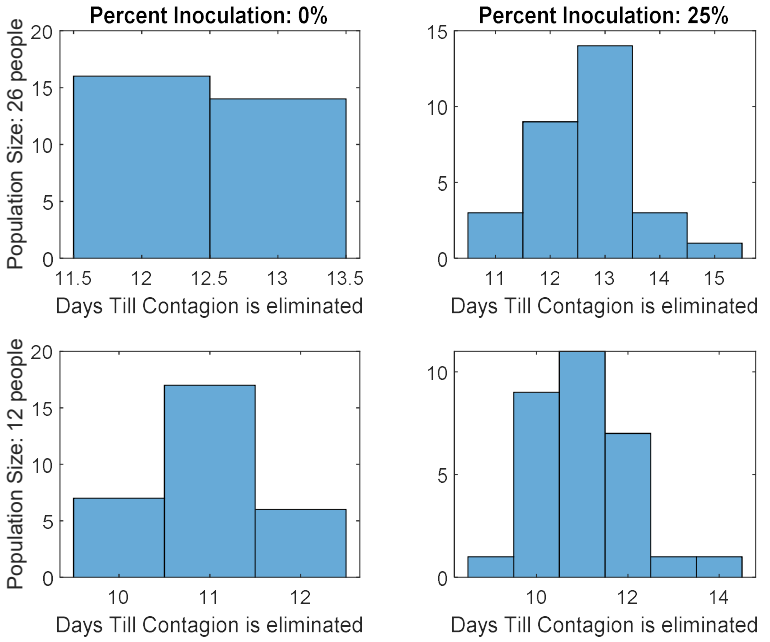

Figure 5: SIR Model: Contagion probability of 50%

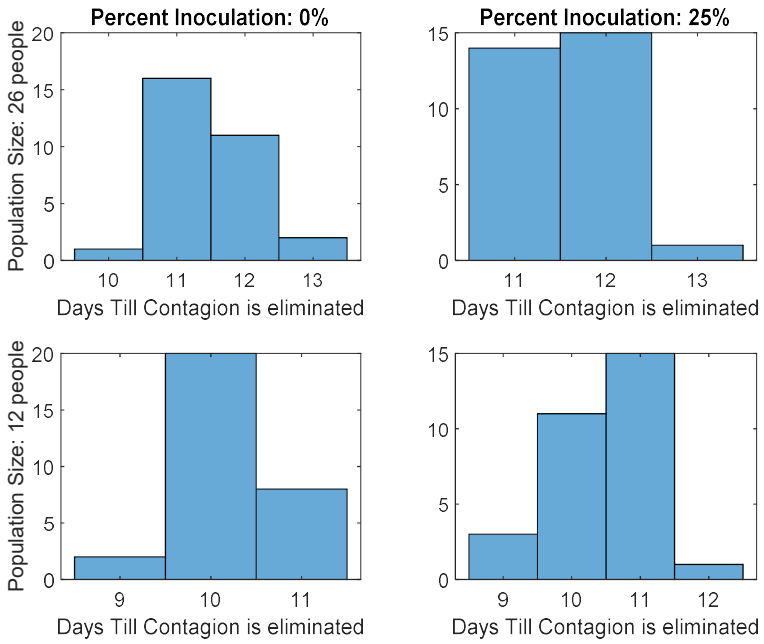

Figure 6: SIR Model: Contagion probability of 75%

There are many conclusions to make from this data.

Starting with the Contagion Model: for one, one can clearly see the high standard deviations for the simulations and higher degree of randomness. The graphs with 25% inoculation start high at 7 and slope downwards, a unique look compared to an expected bell curve. While, the 0% inoculation graphs are incredibly random and rarely repeating. This can be attributed to the dynamic of contagion; because the contagion is spread through neighbors, it tends to be "blocked" by immune neighbors. Refer to Figure 7 for this phenomenon.

```
[ehoyt@isp02 ~]$ ./Project_1
Please enter a number between 0 and 1 that represents the contagion probability:
.75
Please enter a number between 0 and 1 that represents the percentage of innoculation:
.25
In step  0: # of sick people:  1: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! + ? ?
In step  1: # of sick people:  1: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! + ? ?
In step  2: # of sick people:  2: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! + + ?
In step  3: # of sick people:  3: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! + + +
In step  4: # of sick people:  3: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! + + +
In step  5: # of sick people:  3: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! + + +
In step  6: # of sick people:  3: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! + + +
In step  7: # of sick people:  2: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! - + +
In step  8: # of sick people:  1: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! - - +
In step  9: # of sick people:  0: ? ! ? ? ? ? ? ? ? ? ! ! ! ? ? ? ! ? ? ? ? ! - - -
Disease ran its course by step 9
```

Figure 7: Contagion Model output

See how the pluses (the infected) are blocked in by the one immune person and can't infect anyone else in the population. This happened either because the first person infected was on the edge of the population or surrounded by immune people. Moreover, the population change created a noticeable effect on the duration of the contagion in the Contagion Model for 0% inoculation, not so much for 25% inoculation. Because the contact number of this model is so small, the time for the contagion to dissipate is longer since the contagion has to reach everyone by "neighbors." And lastly, the probability of contagion seemed to only affect the 0% inoculation simulations. A higher likelihood of infections results in the contagion dying out faster.

For the SIR Model, the histograms show much more orderly graphs with common means and low standard deviations. Hence, this model is more predictable and produces less outliers. In fact, the variables in this experiment hardly changed the results for this model. Of course, a lower probability of contagion will result in a higher duration of days till the contagion is eliminated. But the population size and percentage of inoculation didn't drastically change the durations of contagion, usually by one or two days. Also, this model doesn't suffer from the contagion being "blocked" since it can spread to anyone in the population.

One last comparison between the two models: the contagion duration is much shorter for the SIR Model than the Contagion Model. Once the contagion spreads to more people in the population, the number of people contacted effectively doubles per infected person. Thus, the infection will die out because everyone will get infected quicker and subsequently get immune quicker.

The main difference in the codes for these models lies in the update method for each Population class. Figures 8 and 9 display these, respectively. Both loop and update the population until no one in the population is infected anymore. The Contagion model update method will first update each person in the population, i.e. decreasing the days till immunity for the infected people. Then, the code will iterate through the population looking for infected people, ignoring stable people and people just infected in that step. Then, if a random number is in the percentage of the probability of contagion, a user input, then the neighbors of the infected person will be themselves be infected. To avoid errors, if a person is at the edge of the population, only their one neighbor will be infected. The SIR model update method starts also by updating the population. Next, it initializes an empty vector and contact counter. For each infected person, it loops until the contact number is met. The code will also loop to make sure a person is not randomly selected twice to be contacted with by one person. I.e. each person will contact six unique people each step. Lastly, like the other model, a person will be infected if a random number is within the probability of contagion.

```
void update(){
  bool one_is_infected = true;
  while (one_is_infected){
    for (int j=0; j<npeople; j++){
      population[j].update();

      //Infections of neighbors
      if (!population[j].is_stable() and !population[j].get_infection_status()){
        //population[j].status_string();
        if (rand()%101 <= probability_infection){
          if (j == 0){
            if (population[j+1].is_stable())
              population[j+1].infect();
          }
          else if (j == (npeople-1))
            population[j-1].infect();
          else{
            population[j-1].infect();
            population[j+1].infect();
          }
        }
      }
    }
```

Figure 8: Contagion update method

```
void update(){
  bool one_is_infected = true;
  int contact_count = 0, i, tmp;
  vector<int> contacted(contact_num);
  while (one_is_infected){
    for (int j=0; j<npeople; j++){
      population[j].update();
      contact_count = 0;
      for (auto &a : contacted)
        a = 0;

      //Infections of others in population based on contact_num
      if (!population[j].is_stable() and !population[j].get_infection_status()){
        //population[j].status_string();
        while (contact_count < contact_num){
          tmp = rand()%npeople;
          while (tmp == j or already_contacted(tmp, contacted))
            tmp = rand()%npeople;
          if (rand()%101 <= probability_infection)
            population[tmp].infect();
          contacted[contact_count] = tmp;
          contact_count += 1;
        }
      }
    }
```

Figure 9: SIR update method

The rest of the code is virtually identical and can be seen in the files. I created various Boolean and void functions to optimize the process, but otherwise the code didn't stray far from what was instructed in the textbook. The main() part simply initializes the population, asks

for input on the probability of contagion and the percentage of inoculation, prints the 0^{th} step, then calls the Population update method till the contagion is eliminated. The brunt of the code lies in the Population class. This is where a random person is infected, the amount of people infected is counted, the probability of contagion and the percentage of inoculation are initialized, and the output is printed.

Overall, I thought it was interesting how this project combined many topics we learned over the semester: classes, functions, loops, conditionals, setw(), and user input, for instance. I ran into a problem I'd never seen before where the code compiled but set off a "Segmentation Fault (Core Dumped)" error. This was due to my update method in the Contagion Model not accounting for the ends of the population. In essence, it wouldn't discriminate the 0^{th} and array.size()th elements and infect neighbors they didn't have.