Infectious Disease Simulation

SDS 322 - ISP2017F - Final Project

Jasmin Lim - December 10, 2017

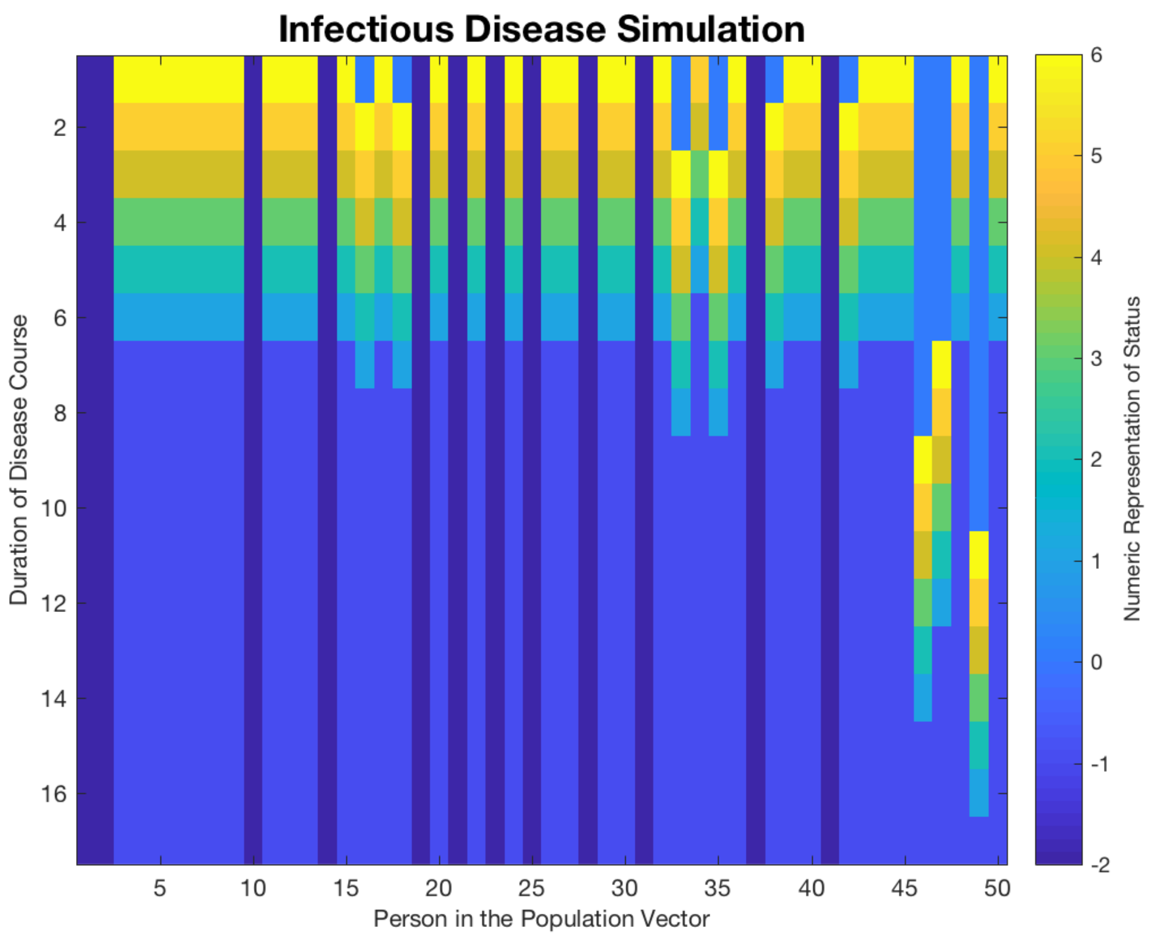


Table of Contents

Introduction

The objective of the Introduction to Scientific Programming final project is to create a stimulation modeling the spread of an infectious disease. Disease models are commonly created with statistics, however, for this project, we were asked to create an explicit stimulation. The model will keep track of the status each person in the population based off a straightforward description. To simplify this model, we will follow a few guidelines. The model will always start with the infection of one person and will run indefinitely until the number of people sick in the population is zero. Since each person can only be infected once, the model's run will always end. Finally, while a person is sick, they can infect another person during any point of the disease.

Each person can either be sick, susceptible, recovered or inoculated. If a person is sick, then they currently have the disease and are infectious to others throughout the duration of their illness. A susceptible person is one who has never been sick, however, they can still be infected by a person with a sick status. When a person in the population has perviously had the disease, then one's status is recovered. When a person has recovered from the disease, they are healthy and can no longer be infected. A person can also be inoculated, meaning they are healthy and cannot be infected.

We will start out with the model of one person in the population and, through a sequence of exercises, gradually build our program to create a more realistic simulation. There is an introduction of different factors that will improve the model's authenticity of an actual disease spread. We will observe and take note of the effects of these changes in the course the disease's spread and adhere to the model's remaining deficiencies.

Model Design

In this model, each person can be one of the following:

Status	Symbolic Representation	Numeric Representation
Sick	+	1-6 (corresponding to days remaining)
Susceptible	?	0
Recovered	-	-1
Inoculated	i	-2

Person

person.h

```cpp
#include <iostream>
#include <string>
using namespace std;

class Person {
 public:
  Person();

  string status_string();
  void update();
  bool infect(int duration);
  bool is_stable();

  string status();
  bool is_sick();

  /// exercise 4
  void immune();

  /// exercise 5 & MATLAB modeling
  int get_status();

 private:
  int _status;
};
```

person.cc

```
#include "person.h"

using namespace std;

Person::Person() {
  _status = 0;
}

string Person::status_string() {
  if (_status == 0) {
    return  "susceptible";
  } else if (_status == -1) {
      return "recovered";
      _status--;
  } else {
    string state;
    state = "is sick (" + to_string(_status) + " to go)";
    return state;
  }
}

void Person::update() {
  if (_status > 0) {
    _status--;
    if (_status == 0) _status--;
  }
}

bool Person::infect(int duration) {
  _status = duration;
  return true;
}

bool Person::is_stable() {
  if (_status < 0) {
    return true;
  }
  return false;
}

string Person::status() {
    if ( _status > 0 ) {
        return string(" + ");
    }
    if (status_string() == "susceptible") {
        return string(" ? ");
    }
    if (status_string() == "recovered") {
        return string(" - ");
    }
    if (_status < -1) {
        return string(" i ");
    }
    return string();
}
```

```cpp
bool Person::is_sick() {
    return (_status > 0);
}

/// exercise 4
void Person::immune() {
    _status = -2;
}

/// exercise 5 & MATLAB modeling
int Person::get_status() {
    int status = _status;
    return status;
}
```

Method Explanations

`public:`

`Person()` - Default class constructor. Sets `_status` to zero, the numerical representation of susceptibility.

`string status_string()` - Translates a person's symbolic representation of their status and displays how many days until the person is no longer infected.

`void update()` - Updates a person's status throughout each day of the course of the disease. If a person's status is greater than or equal to zero, it will subtract one from the person's `_status`. However, it will only subtract one when the `_status` is equal to zero after a person has been infected. Therefore representing that a person can only become recovered after they've been infected.

`bool infect(int duration)` - Sets the person's `_status` equal to the integer input duration of the disease. It also returns the method as `true`, initializing the course of the disease on a person.

`bool is_stable()` - Defines if a person is healthy or not. If `_status` is less than zero, the person has either had the disease and recovered or is vaccinated and immune to the disease. In this case, the method will return `true`. The method will return `false` if the person is sick or susceptible.

`string status()` - Returns the symbolic representation of the person's status. Dependent on `_status` and `status_string()`.

`bool is_sick()` - Returns `true` when a person is sick, when `_status` is greater than zero.

`void immune()` - Sets a person's `_status` equal to -2. The person therefore becomes inoculated.

`int get_status()` - Returns the numeric representation of the person's status. Dependent on `_status`.

`private:`
`int _status` - stores the numeric representation of the person's status.

Population

population.h

```
#include <vector>
#include <string>
#include "person.h"

using namespace std;

class Population {
    public:
        Population(int npeople);

        bool random_infection(int how_many);
        int count_infected();
        void update();
        string status_string();

        /// exercise 3
        void set_probability_of_transfer(float probability);
        void update_probability();

        /// exercise 4
        void vaccinated(float percent);

        /// exercise 5
        void update_random_interaction();

        /// MATLAB modeling
        string num_string();
    private:
        vector<Person> _population;

        /// exercise 3
        float _transfer_prob;
};
```

population.cc

```
#include "population.h"

Population::Population( int npeople) {
    _population.resize(npeople);

    for (int i = 0; i < npeople ; i++) {
        _population[i] = Person();
    }

    /// exercise 3
    _transfer_prob = 0.0;
}

bool Population::random_infection( int how_many) {
    for (int ntries = 0 ;; ntries++) {
        int r = rand() % _population.size();
        if (_population[r].infect(how_many)) {
            return true;
        }
    }
    return false;
}


int Population::count_infected() {
    int n_sick = 0;
    for ( int i = 0; i < _population.size(); i++) {
      if (_population[i].is_sick()) {
        n_sick++;
      }
    }
    return n_sick;
}

void Population::update() {

    for (int i = 0; i < _population.size(); i++){
        _population[i].update();
        }
}

string Population::status_string() {
    string status;
    for ( int i = 0; i < _population.size() ; i++) {
        status += _population[i].status();
    }
    return status;
}

/// exercise 3
void Population::set_probability_of_transfer(float probability) {
    _transfer_prob = probability;
}

void Population::update_probability() {
    for (int i = 0; i < _population.size(); i++) {
        _population[i].update();

        float bad_luck = (float) rand() / (float) RAND_MAX;
        if (bad_luck > _transfer_prob && _population[i].status() == " ? " ) {
            if (i > 0 && _population[i-1].status() == " + " ||
                i < _population.size()-1 && _population[i+1].status() == " + ") {
                _population[i].infect(6);
            }
        }
    }
}

/// exercise 4
void Population::vaccinated(float percent) {
    if (percent == 0) {
        return;
    }
    int n_vaccinated = int( percent * _population.size());
    int n = 0;
    do {
        srand( time(NULL) );
        int index = rand()%_population.size();
        if ( !_population[index].is_stable() ) {
            _population[index].immune();
            n++;
        }
    }
    while ( n < n_vaccinated);
}

/// exercise 5
void Population::update_random_interaction() {

    for (int i = 0; i < _population.size(); i++) {
        _population[i].update();
        int n_interactions = 0;

        do {

            float bad_luck = (float) rand()/(float) RAND_MAX;
            int index = rand()%_population.size();
            if ( bad_luck > _transfer_prob && _population[i].status() == " ? ") {
                if (_population[index].status() == " ? " ) {
                    _population[i].infect(6);
                }
            }
            n_interactions++;
        } while ( n_interactions < 6 );
    }
}

/// MATLAB modeling
string Population::num_string() {
    string status;
    for ( int i = 0; i<_population.size(); i++) {
      char tostring[1024];
      sprintf(tostring, "%4d", _population[i].get_status());
      status += std::string(tostring);
    }
    return status;
}
```

Method Explanations

`public:`

`Population(int npeople)` - Default class constructor. Resizes the population vector to the user's input of the size of the population. Defines each component in the vector to the class `Person`. It also sets the initial probability of disease transfer equal to zero.

`bool random_infection(int how_many)` - Randomly infects one person in the population. A random number is generated between one and the size of the population. Once the first person is infected, the method will return `true`. The method has an indefinite amount of tries to infect one person in the population, otherwise the method will return `false`.

`int count_infected()` - Returns the number of people sick in the population. The method initializes the number of people sick to zero. It then checks whether or not each person in the population is infected. If the person's `is_sick()` is `true`, it adds one to the number of people sick.

`void update()` - Updates each person's status in the population throughout the course of the disease using the `update()` method in the `Person` class.

`string status_string()` - Return a string of the representation for the status of each person in the population. It acquires each person's symbolic representation through the `Person` class method `status()` and concatenates the string to the size of the population.

`void set_probability_of_transfer(float probability)` - Sets the user's input for the probability of disease transmission equal to `_transfer_prob`.

`void update_probability()` - Updates each person's status in the population throughout the course of the disease using the `update()` method in the `Person` class. It also determines whether or not a person will spread the disease to their direct neighbor. First, the method generates a random number between zero and one. If `_transfer_prob` is less than the random number and the person's status is susceptible, the person will become infected. This method also infects a susceptible person if their direct neighbors are sick.

`void vaccinated(float percent)` - Determines which people in the population are vaccinated and immune to the disease. It first determines the number of people that will be inoculated by multiplying the user input of the percent that is vaccinated to the size of the population. It then generates a random number between one and the size of the population and assigns the people in the population who are immune.

void update_random_interaction() - Updates each person's status in the population throughout the course of the disease using the `update()` method in the `Person` class. Working under the condition that a person will come into contact with six random people a day. It will determine if a susceptible person will become infected from a series of interactions. The method creates a random number between one and the size of the population that will determine the person of contact. If the random person is sick, then susceptible person will become infected.

`string num_string()` - Returns the numeric representation for the status of each person in the population. It acquires each person's numeric representation of their status through the `Person` class method `get_status()` and concatenates the string to the size of the population.

`private:`
`vector<Person> _population -` creates the vector population.

`float _transfer_prob` - stores user input for the probability that the disease will be transferred upon contact.

Make File

makefile

```
.cc.o:
    icpc -std=c++11 -c $*.cc

infection: person.o infection.o
    icpc -o infection person.o infection.o

run_population: run_population.o person.o population.o
    icpc -o run_population run_population.o person.o population.o

probability: probability.o person.o population.o
    icpc  -o probability probability.o person.o population.o

vaccinated: vaccinated.o person.o population.o
    icpc -o vaccinated vaccinated.o person.o population.o

random_interactions: random_interactions.o person.o population.o
    icpc -o random_interactions random_interactions.o person.o population.o

number_output: number_output.o person.o population.o
    icpc -o number_output number_output.o person.o population.o

clean:
    /bin/rm -f *.o infection population
```

Coding Up the Basics

Exercise 1

The code, `infection.cc`, models a single person. It uses the class `Person` to infect the person with the disease and track their progress throughout the infection. To infect a person, a random number, `bad_luck`, between zero and one, is generated and compared to 0.95. If `bad_luck` is greater than 0.95, the person becomes infected with a disease over a course of five days. The person's status is updated until it becomes recovered. If `bad_luck` is less than zero, the program will generate another random number. The program will continue to create a random number until the person becomes infected.

infection.cc

```cpp
#include <iostream>
#include <stdlib.h>
#include "person.h"
using namespace std;

int main() {
    Person Joe;

    for (int day = 0 ;; day++) {
        Joe.update();
        float bad_luck = (float) rand() / (float)RAND_MAX;
        if (bad_luck > .95){
            Joe.infect(5);
        }
        cout << "On day " << day << ", Joe is " << Joe.status_string() << endl;

        if (Joe.is_stable()) break;
    }
}
```

Output

The output displays a timeline of Joe's status throughout the disease's duration.

```
[jasminyl@isp02 finalproject]$ ./infection
On day 0, Joe is susceptible
On day 1, Joe is susceptible
On day 2, Joe is susceptible
On day 3, Joe is susceptible
On day 4, Joe is susceptible
On day 5, Joe is susceptible
On day 6, Joe is susceptible
On day 7, Joe is susceptible
On day 8, Joe is susceptible
On day 9, Joe is susceptible
On day 10, Joe is susceptible
On day 11, Joe is susceptible
On day 12, Joe is susceptible
On day 13, Joe is susceptible
On day 14, Joe is is sick (5 to go)
On day 15, Joe is is sick (4 to go)
On day 16, Joe is is sick (3 to go)
On day 17, Joe is is sick (2 to go)
On day 18, Joe is is sick (1 to go)
On day 19, Joe is recovered
```

Population

Exercise 2

To improve our model of the disease, we will create a `Population` class that models a vector of people in a population. The code asks the user to input the size of the population. It then initializes the infection of one random person for a duration of six days using the `Population` method `random_infection(6)`. In this exercise, we work under the condition that the disease is non-transmittable. Once the number of people sick in the population becomes zero, the loop breaks and the code's run ends.

run_population.cc

```cpp
#include <iostream>
#include "population.h"
using namespace std;

int main() {
    cout << "Enter the size of the population: " << endl;
    int n;
    cin >> n;

  Population small_town(n);
  small_town.random_infection(6);

  for (int day = 0;; day++) {
    small_town.update();

    int nsick = small_town.count_infected();
    cout << "In step    " << day+1 << " #sick: " << nsick << " : " <<
small_town.status_string() << endl;
```

```
    if (nsick == 0) {
      cout << "Disease ran its course by step " << day+1 << endl;
      break;
    }
  }
}
```

Output

The output of `run_population.cc` displays the symbolic representation of the each person in the population, the timeline of the infection and the number of people sick each day. Finally, it outputs the number of days it took the disease to run its entire course on the population.

```
[jasminyl@isp02 finalproject]$ ./run_population
Enter the size of the population:
20
In step    1 #sick: 1 :  ?  ?  ?  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
In step    2 #sick: 1 :  ?  ?  ?  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
In step    3 #sick: 1 :  ?  ?  ?  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
In step    4 #sick: 1 :  ?  ?  ?  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
In step    5 #sick: 1 :  ?  ?  ?  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
In step    6 #sick: 0 :  ?  ?  ?  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
Disease ran its course by step 6
```

Contagion

Exercise 3

The previous exercises will be used as a basic model of our disease's infection. To make the model more realistic, we will introduce contagion and model how the disease spreads from a single person. The programs asked the user to input the size of the population and the probability that the disease will be spread upon contact with direct neighbors. The status and probability for disease infection transmission for each person in the population is updated and determined by the `Population` class method `update_probability()`. Once the number of people sick in the population becomes zero, the code's run will end.

probability.cc

```
#include "population.h"

int main() {
    cout << "Enter the size of the population: " << endl;
    int n;
    cin >> n;

    cout << "Enter the probabilty of disease transmission :" << endl;
    float probability;
    cin >> probability;

    Population small_town(n);
    small_town.random_infection(6);
    small_town.set_probability_of_transfer(probability);

    for (int day = 0 ;; day++) {
        small_town.update_probability();

        int nsick = small_town.count_infected();
        cout << small_town.status_string() << endl;

        if (nsick == 0) {
            cout << "Disease ran its course by day " << day+1 << endl;
            break;
        }
    }
}
```

Output

The output of probability.cc shows the spread of the disease between direct neighbors. As shown by the results, the probability of disease transmission impacts the duration of the disease's course and the number of people who are infected. As the probability is increased, the amount of days it takes for the disease to run its course is increased. However, once a high enough probability is entered, we can see from the results that people in the population will escape being infected. When people escape from being infected by the disease, it stiffens the course of the infection.

```
[jasminyl@isp02 finalproject]$ ./probability
Enter the size of the population:
20
Enter the probabilty of disease transmission :
.3
 ?  ?  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  ?  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  +  +  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  +  +  +  +  +  +  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +
 +  +  +  -  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +
 +  +  -  -  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +
 +  +  -  -  -  +  +  +  +  +  +  +  +  +  +  +  +  +  +  +
 +  -  -  -  -  -  -  +  +  +  +  +  +  +  +  +  +  +  +  +
 -  -  -  -  -  -  -  -  -  -  +  +  +  +  +  +  +  +  +  +
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
Disease ran its course by day 11
```

```
[jasminyl@isp02 finalproject]$ ./probability
Enter the size of the population:
20
Enter the probabilty of disease transmission :
.5
 ?  ?  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  ?  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  ?  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  +  -  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  -  -  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  +  -  -  +  +  +  +  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  +  -  -  +  +  +  +  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  -  -  -  -  +  +  +  +  +  +  +  +  ?  ?  ?  ?  ?  ?  ?
 +  -  -  -  -  +  +  +  +  +  +  +  +  +  ?  ?  ?  ?  ?  ?
 +  -  -  -  -  -  +  +  +  +  +  +  +  +  ?  ?  ?  ?  ?  ?
 +  -  -  -  -  -  +  +  +  +  +  +  +  +  +  +  ?  ?  ?  ?
 -  -  -  -  -  -  -  -  -  -  -  +  +  +  +  +  ?  ?  ?  ?
 -  -  -  -  -  -  -  -  -  -  -  +  +  +  +  +  ?  ?  ?  ?
 -  -  -  -  -  -  -  -  -  -  -  -  -  +  +  +  ?  ?  ?  ?
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  +  +  +  +  ?  ?
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  +  +  +  +  ?  ?
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +  +  +  ?
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +  +  +  ?
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +  +  +  ?
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +  +  +  +
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +  +
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +  +
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  +
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
Disease ran its course by day 28

[jasminyl@isp02 finalproject]$ ./probability
Enter the size of the population:
20
Enter the probabilty of disease transmission :
.8
 ?  ?  ?  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  ?  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  ?  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  +  +  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  +  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 ?  +  +  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  +  -  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  +  -  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  -  -  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  -  -  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  -  -  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 +  -  -  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
 -  -  -  -  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?  ?
Disease ran its course by day 14
```

Exercise 4

Taking into account real-world protections against diseases, we will introduce vaccinations that will cause a person to become immune to a disease. To improve our infectious disease stimulation, we will establish the status of inoculated. If a person is inoculated, they cannot become infected nor spread the infection to others. The program `vaccinated.cc`, takes the user's input for the size of the population, the probability that the disease is transmitted and the percent of the population that is vaccinated. It initializes the `Population` class method `vaccinated(percent)`, which will applies the user input for percent of immunity to the population vector. The status of each person is updated each day and the disease is passed on to direct neighbors if the probability of contagion is met.

vaccinated.cc

```
#include "population.h"
#include <time.h>

int main() {

    cout << "Enter the size of the population: " << endl;
    int n;
    cin >> n;

    cout << "Enter the probability of disease transmission: " << endl;
    float probability;
    cin >> probability;

    cout << "Enter the percent of the population that is vaccinated: " << endl;
    float percent;
    cin >> percent;

    Population small_town(n);
    small_town.random_infection(6);
    small_town.vaccinated(percent);
    small_town.set_probability_of_transfer(probability);

    for (int day = 0 ;; day ++) {
        small_town.update_probability();

        int nsick = small_town.count_infected();
        cout << small_town.status_string() << endl;

        if (nsick == 0) {
            cout << "Disease ran its course by day " << day+1 << endl;
            break;
        }
    }
}
```

Output

```
[jasminyl@isp02 finalproject]$ ./vaccinated
Enter the size of the population:
20
Enter the probability of disease transmission:
.35
Enter the percent of the population that is vaccinated:
.2
```

```
 ?  ?  i  +  +  ?  ?  ?  ?  ?  ?  ?  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  +  +  +  +  ?  ?  ?  ?  ?  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  +  +  +  +  +  +  +  ?  ?  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  +  +  +  +  +  +  +  +  ?  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  +  +  +  +  +  +  +  +  ?  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  -  +  +  +  +  +  +  +  +  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  -  -  +  +  +  +  +  +  +  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  -  -  -  -  +  +  +  +  +  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  -  -  -  -  -  -  -  +  +  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  -  -  -  -  -  -  -  -  +  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  -  -  -  -  -  -  -  -  +  i  i  ?  ?  i  ?  ?  ?
 ?  ?  i  -  -  -  -  -  -  -  -  -  i  i  ?  ?  i  ?  ?  ?
Disease ran its course by day 12
```

The output of vaccinated.cc displays the symbolic representation of each person's status in the population. One can observe how the disease is spread through the contact with direct neighbors if the probability for disease transmission is met. The vaccinated people shorten the course of the disease and stop the spreading of the infection, increasing the amount of people who will escape being infected. This however, is an unrealistic model because once a direct neighbor with an inoculated status is reached, the disease's spread is stopped. Realistically, the spread of a disease will not be solely dependent on the status of one person.

Spreading

Exercise 5

To make the spread of the disease more realistic, we will have each person come into contact with a fixed number of random people a day. We will apply the already programmed probability the disease will be transmitted upon contact over a course of six interactions. The program will ask the user to input the size of the population, the probability that the disease is transmitted and the percent of the population that is vaccinated. We will start with the infection of one random person and use the `Population` method `update_random_interactions()` to update the status of each person during every step of the disease's course and determine if another person is transmitted the disease upon six interactions. Once the number of people sick in the population is zero, the code's run will end.

random_interactions.cc

```
#include "population.h"
#include <time.h>

int main() {
    cout << "Enter the size of the population: " << endl;
    int n;
    cin >> n;

    cout << "Enter the probability of disease transmission: " << endl;
    float probability;
    cin >> probability;

    cout << "Enter the percent of the population that is vaccinated: " << endl;
    float percent;
    cin >> percent;

    Population small_town(n);
    small_town.random_infection(6);
    small_town.vaccinated(percent);
    small_town.set_probability_of_transfer(probability);

    for (int day = 0 ;; day++) {
        small_town.update_random_interaction();


        int nsick = small_town.count_infected();
        cout << small_town.status_string()  << endl;


        if (nsick == 0) {
            cout << "Disease ran its course by day " << day+1 << endl;
            break;
        }
    }
}
```

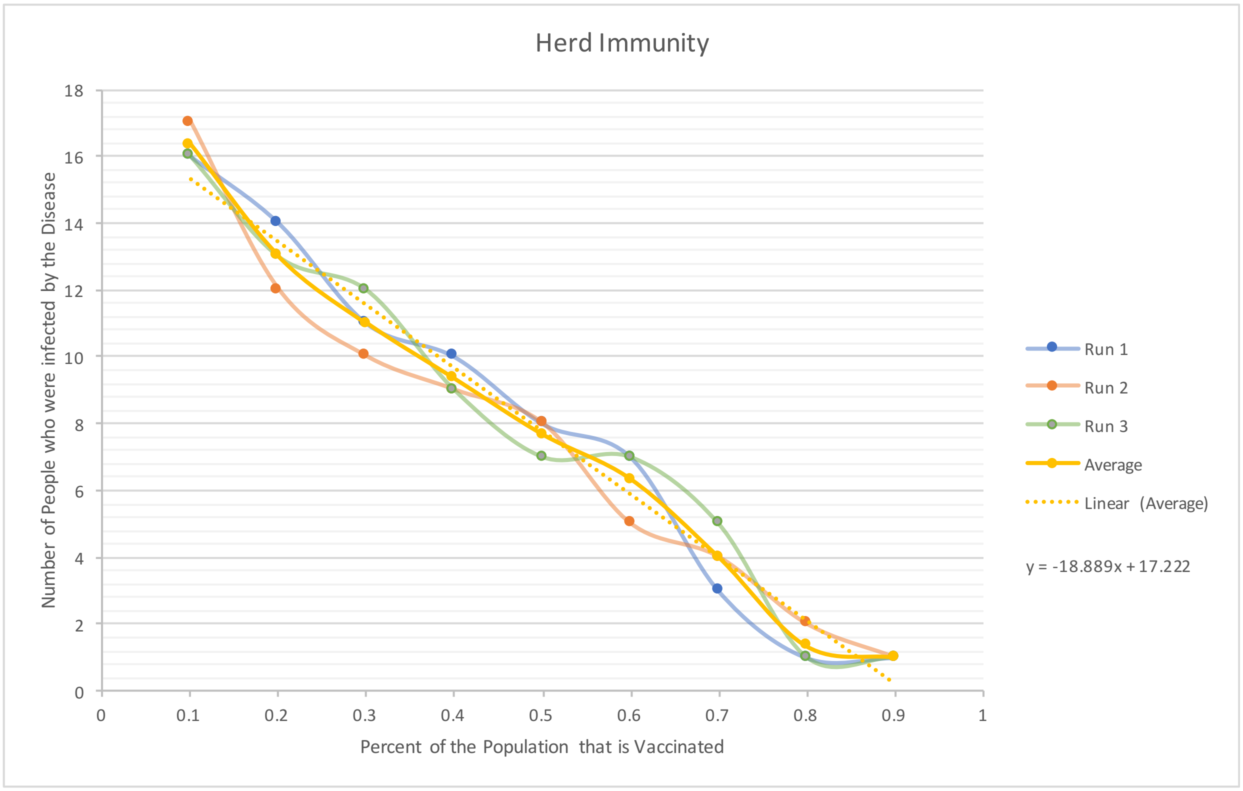
Output

```
[jasminyl@isp02 finalproject]$ ./random_interactions
Enter the size of the population:
20
Enter the probability of disease transmission:
.95
Enter the percent of the population that is vaccinated:
.4
 +   i   i   +   i   ?   i   +   ?   i   ?   i   i   ?   ?   ?   ?   ?   i   ?
 +   i   i   +   i   ?   i   +   ?   i   ?   i   i   ?   ?   ?   ?   ?   i   ?
 +   i   i   +   i   ?   i   +   ?   i   ?   i   i   ?   ?   ?   ?   +   i   ?
 +   i   i   +   i   ?   i   +   ?   i   +   i   i   ?   ?   ?   ?   +   i   ?
 +   i   i   +   i   ?   i   +   ?   i   +   i   i   ?   ?   ?   ?   +   i   +
 +   i   i   -   i   +   i   +   ?   i   +   i   i   ?   ?   ?   ?   +   i   +
 -   i   i   -   i   +   i   -   ?   i   +   i   i   ?   ?   ?   ?   +   i   +
 -   i   i   -   i   +   i   -   ?   i   +   i   i   ?   ?   ?   ?   +   i   +
 -   i   i   -   i   +   i   -   ?   i   +   i   i   ?   ?   ?   ?   -   i   +
 -   i   i   -   i   +   i   -   ?   i   -   i   i   ?   ?   ?   ?   -   i   +
 -   i   i   -   i   +   i   -   ?   i   -   i   i   ?   ?   ?   ?   -   i   -
 -   i   i   -   i   -   i   -   ?   i   -   i   i   ?   ?   ?   ?   -   i   -
Disease ran its course by day 12
```

The code output for `random_interactions.cc` displays the spread of the disease on a population based on the probability that the disease is transmitted upon contact and the percent of the population that is immune. With a fixed number of six contacts per day and a constant probability of disease transmission, the percent vaccinated and the how long the disease's course runs is somewhat modeled by a bell curve. Initially, as one increases the number vaccinated, it takes longer for the disease to spread throughout the population. As a random person is generated to be the person of contact, it is a higher probability that the person is inoculated. Due to the increasing amount of people that the disease cannot be transmitted to, it takes more random interactions for the disease to run its course. However, when the number vaccinated is high enough, we see a number of people to actually escape the infection, stiffening the duration of the disease's course. Due to a majority of the population's status being inoculated, it is less likely for a sick person to come into contact with a susceptible person. If a person cannot infect another person during their course of the disease, it will decrease the number of people the disease infects, allowing some to never catch it. This is called herd immunity.

To further investigate how the percentage of inoculation effects herd immunity, we will used a fixed probability of transmission of 0.95. We will then observe how the number of people who become infected is a function of the percent of the population that is vaccinated.

Population Size	Probability of Disease Transmission	Percent of Population Vaccinated	# of people who become sick 1	# of people who become sick 2	# of people who become sick 3	Average
20	0.95	0.10	16	17	16	16.33
20	0.95	0.20	14	12	13	13
20	0.95	0.30	11	10	12	11
20	0.95	0.40	10	9	9	9.33
20	0.95	0.50	8	8	7	7.67
20	0.95	0.60	7	5	7	6.33
20	0.95	0.70	3	4	5	4
20	0.95	0.80	1	2	1	1.33
20	0.95	0.90	1	1	1	1



The contagiousness of the disease has an inverse linear relationship with the number of people who are vaccinated. As the number of people who are vaccinated is increased, the percentage of people who can become sick by infection is reduced. Therefore, it is less likely for a person to come into contact with a sick person during the six random interactions. This also causes the number of people who escape the disease's infections to increase.

MATLAB Modeling

Creation of MATLAB Graphs

1. Creation of `num_ouput.cc`

The code `num_ouput.cc` will take the same input and execute the same output as `random_interation.cc`. However, instead of displaying the symbolic representation of each person's status, it will display the numeric representation of each person's status. Input `cout` statements and the final report of the disease course are taken out to create a table of integer values.

num_ouput.cc

```cpp
#include "population.h"
#include <time.h>

int main() {
    int n;
    cin >> n;

    float prob;
    cin >> prob;

    float per;
    cin >> per;

    Population town(n);
    town.random_infection(6);
    town.vaccinated(per);
    town.set_probability_of_transfer(prob);

    for ( int day = 0 ;; day++) {
        town.update_random_interaction();

        int nsick = town.count_infected();
        cout << town.num_string() << endl;

        if (nsick == 0) {
            break;
        }
    }
}
```

Output

```
[jasminyl@isp02 finalproject]$ ./number_output
20
.4
.25
   6  -2   6  -2   6   6   6  -2  -2   0   6   0   6   0  -2   0   6   6   6   0
   5  -2   5  -2   5   5   5  -2  -2   0   5   6   5   6  -2   0   5   5   5   6
   4  -2   4  -2   4   4   4  -2  -2   6   4   5   4   5  -2   0   4   4   4   5
   3  -2   3  -2   3   3   3  -2  -2   5   3   4   3   4  -2   0   3   3   3   4
   2  -2   2  -2   2   2   2  -2  -2   4   2   3   2   3  -2   0   2   2   2   3
   1  -2   1  -2   1   1   1  -2  -2   3   1   2   1   2  -2   0   1   1   1   2
  -1  -2  -1  -2  -1  -1  -1  -2  -2   2  -1   1  -1   1  -2   0  -1  -1  -1   1
  -1  -2  -1  -2  -1  -1  -1  -2  -2   1  -1  -1  -1  -1  -2   6  -1  -1  -1  -1
  -1  -2  -1  -2  -1  -1  -1  -2  -2  -1  -1  -1  -1  -1  -2   5  -1  -1  -1  -1
  -1  -2  -1  -2  -1  -1  -1  -2  -2  -1  -1  -1  -1  -1  -2   4  -1  -1  -1  -1
  -1  -2  -1  -2  -1  -1  -1  -2  -2  -1  -1  -1  -1  -1  -2   3  -1  -1  -1  -1
  -1  -2  -1  -2  -1  -1  -1  -2  -2  -1  -1  -1  -1  -1  -2   2  -1  -1  -1  -1
  -1  -2  -1  -2  -1  -1  -1  -2  -2  -1  -1  -1  -1  -1  -2   1  -1  -1  -1  -1
  -1  -2  -1  -2  -1  -1  -1  -2  -2  -1  -1  -1  -1  -1  -2  -1  -1  -1  -1  -1
```

2. Save the results of `num_ouput.cc` as a .txt file

```
[jasminyl@isp02 finalproject]$ ./number_output > run1.txt
```

```
20
.4
.25
```

3. Create a MATLAB code that loads the .txt file and graphs the results as a color map.

The MATLAB code `graphdata.m` loads the .txt file and creates a cell of the data. It then creates a color map of the status of each person in the population and its update throughout the run of the disease's course. Finally, `graphdata.m` outputs a .png file of the figure created.

graphdata.m

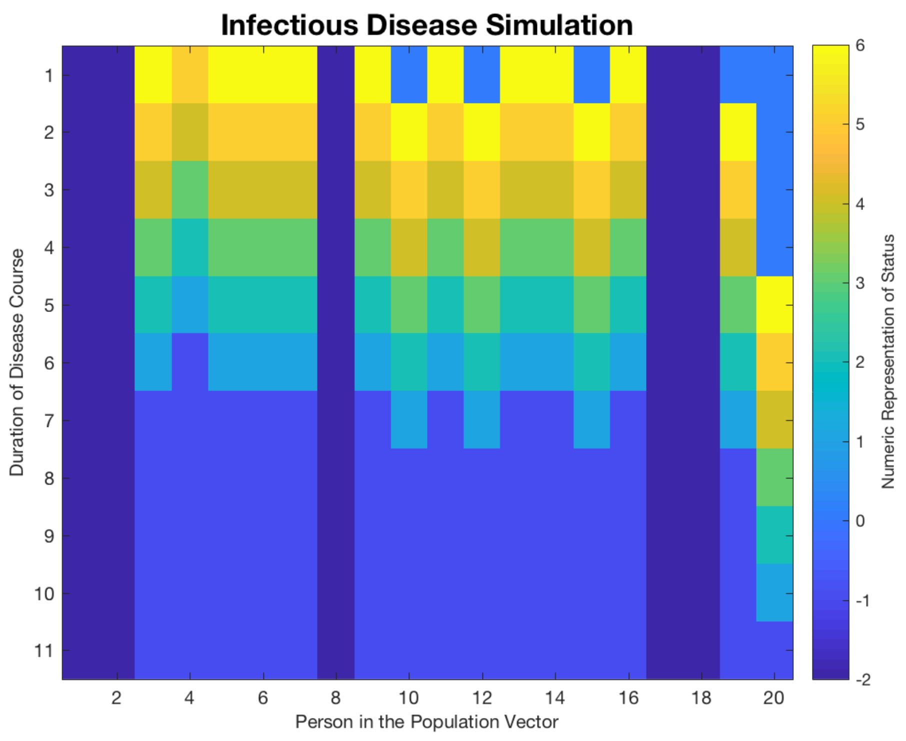
```
close all;
load('run1.txt');

imagesc(run1(:,:));
a = caxis;
cbar = colorbar;
ylabel(cbar,'Numeric Representation of Status');

xlabel('Person in the Population Vector', 'fontsize', 10);
ylabel('Duration of Disease Course', 'fontsize', 10);

saveas(gcf, 'graph', ‘png');
```

Resulting Figure


Infectious Disease Simulation
Duration of Disease Course
Person in the Population Vector
Numeric Representation of Status