Programming project: Infectuous diseases simulation

Victor Eijkhout

2020
Abstract

Infectious diseases have been an important issues in recent years. From the ‘anti-vax’ movement to the recent Covid-19 epidemic, infectious diseases are having a noticeable impact on society. Of the many aspects that can be studied, here we take a look at the statistics of the way a disease spreads through the population.

This project walks a student through the development of a ‘network model’ of a population. After the full model has been implemented, there are questions about the sensitivity of the disease propagation to various parameters that students can explore.

Any programming language can be used here, but the project description targets an object-oriented formulation.

This project can be done by a single undergraduate in about a week, at the end of a first semester programming course.
| **Summary** | Model the spread of infectious diseases, applied to diseases for which a vaccine exists. |
| **Topics** | Arrays, classes |
| **Audience** | Undergraduate or AP high school |
| **Difficulty** | Moderate |
| **Strengths** | Relatively easy to program, lots of opportunity for initiative |
| **Weaknesses** | Open to sloppy programming and minimal exploration |
| **Dependencies** | No graphics output, so the interpretation of output requires some imagination |
| **Variants** | Application to diseases without vaccine (Ebola, Covid-19); this requires extra programming and exploration. |
Chapter 1

Infectuous diseases simulation

This section contains a sequence of exercises that builds up to a somewhat realistic simulation of the spread of infectious diseases.

1.1 Model design

It is possible to model disease propagation statistically, but here we will build an explicit simulation: we will maintain an explicit description of all the people in the population, and track for each of them their status.

We will use a simple model where a person can be:

- sick: when they are sick, they can infect other people;
- susceptible: they are healthy, but can be infected;
- recovered: they have been sick, but no longer carry the disease, and can not be infected for a second time;
- vaccinated: they are healthy, do not carry the disease, and can not be infected.

In more complicated models a person could be infectious during only part of their illness, or there could be secondary infections with other diseases, et cetera. We keep it simple here: any sick person is can infect others while they are sick.

In the exercises below we will gradually develop a somewhat realistic model of how the disease spreads from an infectious person. We always start with just one person infected. The program will then track the population from day to day, running indefinitely until none of the population is sick. Since there is no re-infection, the run will always end.

1.1.1 Other ways of modeling

Instead of capturing every single person in code, a ‘contact network’ model, it is possible to use an Ordinary Differential Equation (ODE) approach to disease modeling. You would then model the percentage of infected persons by a single scalar, and derive relations for that and other scalars [1].


This is known as a ‘compartmental model’, where each of the three SIR states is a compartment: a section of the population. Both the contact network and the compartmental model capture part of the truth. In
1. Infectuous diseases simulation

Infectious diseases simulation fact, they can be combined. We can consider a country as a set of cities, where people travel between any pair of cities. We then use a compartmental model inside a city, and a contact network between cities. In this project we will only use the network model.

1.2 Coding up the basics

We start by writing code that models a single person. The main methods serve to infect a person, and to track their state. We need to have some methods for inspecting that state.

The intended output looks something like:

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 sick (5 to go)
On day 15, Joe is sick (4 to go)
On day 16, Joe is sick (3 to go)
On day 17, Joe is sick (2 to go)
On day 18, Joe is sick (1 to go)
On day 19, Joe is recovered

Exercise 1.1. Write a Person class with methods:

- `status_string()`: returns a description of the person’s state as a string;
- `update()`: update the person’s status to the next day;
- `infect(n)`: infect a person, with the disease to run for `n` days;
- `is_stable()`: return a bool indicating whether the person has been sick and is recovered.

Your main program could for instance look like:

```cpp
Person joe;

int step = 1;
for (; ; step++) {
    joe.update();
    float bad_luck = (float) rand() / (float) RAND_MAX;
    if (bad_luck > .95)
        joe.infect(5);

    cout << "On day " << step << " , Joe is "
        << joe.status_string() << endl;
    if (joe.is_stable())
        break;
}
```

Here is a suggestion how you can model disease status. Use a single integer with the following interpretation:
1.3. Population

- healthy but not vaccinated, value 0,
- recovered, value $-1$,
- vaccinated, value $-2$,
- and sick, with $n$ days to go before recovery, modeled by value $n$.

The `Person::update` method then updates this integer.

**Remark 1** Consider a point of programming style. Now that you’ve modeled the state of a person with an integer, you can use that as

```cpp
void infect(n) {
    if (state==0)
        state = n;
}
```

But you can also write

```cpp
bool is_susceptible() {
    return state==0;
}
void infect(n) {
    if (is_susceptible())
        state = n;
}
```

Which do you prefer and why?

### 1.3 Population

Next we need a `Population` class. Implement a population as a `vector` consisting of `Person` objects. Initially we only infect one person, and there is no transmission of the disease.

The trace output should look something like:

```
Size of population?
Disease ran its course by step 6
```

**Exercise 1.2.** Program a population without infection.

- Write the `Population` class. The constructor takes the number of people:
  ```cpp
  Population population(npeople);
  ```
- Write a method that infects a random person:
  ```cpp
  population.random_infection();
  ```
1. Infectuous diseases simulation

- Write a method `count_infected` that counts how many people are infected.
- Write an `update` method that updates all persons in the population.
- Loop the `update` method until no people are infected: the `Population::update` method should apply `Person::update` to all persons in the population.

Write a routine that displays the state of the population, using for instance: ? for susceptible, + for infected, – for recovered.

1.4 Contagion

This past exercise was too simplistic: the original patient zero was the only one who ever got sick. Now let’s incorporate contagion, and investigate the spread of the disease from a single infected person.

We start with a very simple model of infection.

Exercise 1.3. Read in a number $0 \leq p \leq 1$ representing the probability of disease transmission upon contact. Incorporate this into the program: in each step the direct neighbours of an infected person can now get sick themselves.

```
| population.set_probability_of_transfer(probability);
```

Run a number of simulations with population sizes and contagion probabilities. Are there cases where people escape getting sick?

Exercise 1.4. Incorporate vaccination: read another number representing the percentage of people that has been vaccinated. Choose those members of the population randomly.

Describe the effect of vaccinated people on the spread of the disease. Why is this model unrealistic?

1.5 Spreading

To make the simulation more realistic, we let every sick person come into contact with a fixed number of random people every day. This gives us more or less the SIR model; [https://en.wikipedia.org/wiki/Epidemic_model](https://en.wikipedia.org/wiki/Epidemic_model).

Set the number of people that a person comes into contact with, per day, to 6 or so. (You can also let this be an upper bound for a random value, but that does not essentially change the simulation.) You have already programmed the probability that a person who comes in contact with an infected person gets sick themselves. Again start the simulation with a single infected person.

Exercise 1.5. Code the random interactions. Now run a number of simulations varying

- The percentage of people vaccinated, and
- the chance the disease is transmitted on contact.
1.6 Diseases without vaccine: Ebola and Covid-19

The project so far applies to diseases for which a vaccine is available, such as MMR for measles, mumps and rubella. The analysis becomes different if no vaccine exists, such as is the case for ebola and covid-19, as of this writing.

Instead, you need to incorporate ‘social distancing’ into your code: people do not get in touch with random others anymore, but only those in a very limited social circle. Design a model distance function, and explore various settings.

The difference between ebola and covid-19 is how long an infection can go unnoticed: the incubation period. With ebola, infection is almost immediately apparent, so such people are removed from the general population and treated in a hospital. For covid-19, a person can be infected, and infect others, for a number of days before they are sequestered from the population.

Add this parameter to your simulation and explore the behavior of the disease as a function of it.

1.7 Project writeup and submission

1.7.1 Program files

In the course of this project you have written more than one main program, but some code is shared between the multiple programs. Organize your code with one file for each main program, and a single ‘library’ file with the class methods. This requires you to use separate compilation for building the program, and you need a header file.

Submit all source files with instructions on how to build all the main programs. You can put these instructions in a file with a descriptive name such as README or INSTALL, or you can use a makefile.

1.7.2 Writeup

In the writeup, describe the ‘experiments’ you have performed and the conclusions you draw from them. The exercises above give you a number of questions to address.

For each main program, include some sample output, but note that this is no substitute for writing out your conclusions in full sentences.

The last exercise asks you to explore the program behaviour as a function of one or more parameters. Include a table to report on the behaviour you found. You can use Matlab or Matplotlib in Python (or even Excell) to plot your data, but that is not required.
1. Infectuous diseases simulation

1.8 Bonus: mathematical analysis

The SIR model can also be modeled through coupled difference or differential equations.

1. The number $S_i$ of susceptible people at time $i$ decreases by a fraction

$$S_{i+1} = S_i(1 - \lambda_i \, dt)$$

where $\lambda_i$ is the product of the number of infected people and a constant that reflects the number of meetings and the infectiousness of the disease. We write:

$$S_{i+1} = S_i(1 - \lambda I_i \, dt)$$

2. The number of infected people similarly increases by $\lambda S_i I_i$, but it also decreases by people recovering (or dying):

$$I_{i+1} = I_i(1 + \lambda S_i \, dt - \gamma \, dt).$$

3. Finally, the number of ‘removed’ people equals that last term:

$$R_{i+1} = R_i(1 + \gamma I_i).$$

**Exercise 1.6.** Code this scheme. What is the effect of varying $dt$?

**Exercise 1.7.** For the disease to become an epidemic, the number of newly infected has to be larger than the number of recovered. That is,

$$\lambda S_i I_i - \gamma I_i > 0 \iff S_i > \gamma/\lambda.$$  

Can you observe this in your simulations?

The parameter $\gamma$ has a simple interpretation. Suppose that a person stays ill for $\delta$ days before recovering. If $I_t$ is relatively stable, that means every day the same number of people get infected as recover, and therefore a $1/\delta$ fraction of people recover each day. Thus, $\gamma$ is the reciprocal of the duration of the infection in a given person.
Chapter 2

Style guide for project submissions

The purpose of computing is insight, not numbers. (Richard Hamming)

Your project writeup is at least as important as the code. Here are some common-sense guidelines for a good writeup. However, not all parts may apply to your project. Use your good judgement.

**Style**  First of all, observe correct spelling and grammar. Use full sentences.

**Completeness**  Your writeup needs to have the same elements as a good paper:

- Title and author, including EID.
- A one-paragraph abstract.
- A bibliography at the end.

**Introduction**  The reader of your document need not be familiar with the project description, or even the problem it addresses. Indicate what the problem is, give theoretical background if appropriate, possibly sketch a historic background, and describe in global terms how you set out to solve the problem, as well as your findings.

**Code**  Your report should describe in a global manner the algorithms you developed, and you should include relevant code snippets. If you want to include full listings, relegate that to an appendix: code snippets should only be used to illustrate especially salient points.

Do not use screen shots of your code: at the very least use a `verbatim` environment, but using the `listings` package (used in this book) is very much recommended.

**Results and discussion**  Present tables and/or graphs when appropriate, but also include verbiage to explain what conclusions can be drawn from them.

You can also discuss possible extensions of your work to cases not covered.

**Summary**  Summarize your work and findings.
2. Style guide for project submissions
Chapter 3

Bibliography