Assignment 0

High Performance Parallel Computing 2022

Kimi Cardoso Kreilgaard & Linea Stausbøll Hedemark

January 2022

The differential equations for the SIR-model are as follows:

$$\frac{dS}{dt} = 0 - \beta I \cdot \frac{S}{N} \qquad \frac{dI}{dt} = \beta I \cdot \frac{S}{N} - \gamma I \qquad \frac{dR}{dt} = \gamma I$$

Discretizing these equations, we obtain the expressions for the next step values of S, I and R in the simulation:

$$\begin{aligned} S_{k+1} &= S_k - \left(\beta I_k \cdot \frac{S_k}{N}\right) \Delta t \\ I_{k+1} &= I_k + \left(\beta I_k \cdot \frac{S_k}{N} - \gamma I_k\right) \Delta t \\ R_{k+1} &= R_k + (\gamma I_k)\, \Delta t \end{aligned}$$

Using a population size of $N = 1000$, a rate of infection of $\beta = 0.2$, and a recovery time of $\gamma = 10\text{days}^{-1}$, along with the initial value of a single infected person $I(0) = 1$, we implement the code in `c++` as seen in the bottom of the document, and simulate $N = 2e4$ iterations.
Loading the values created by the `c++` code into `Python`, we plot S, I and R as a function of time, as seen in Fig. 1.

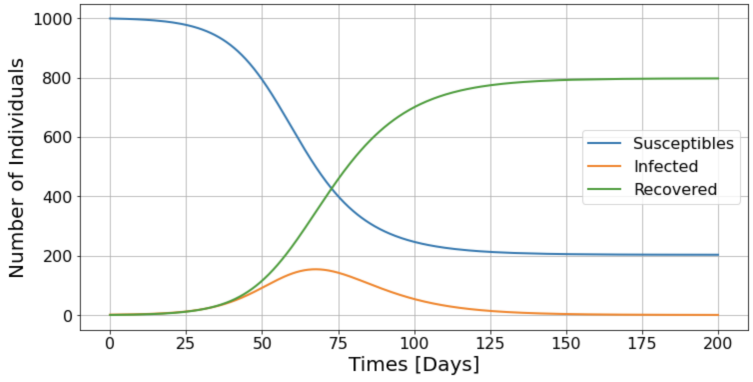

Figure 1: SIR-model simulation with $\Delta t = 0.01$, $N = 1000$, $\beta = 0.2$, $\gamma = 10\text{days}^{-1}$, $I(0) = 1$, $R(0) = 0$.

To avoid an epidemic, one method is to vaccinate the population, which in the SIR-model corresponds to moving a number of individuals from the Susceptible group to the recovered group, assuming that the vaccination prevents infection. The question is worded very vaguely, but if we assume that an epidemic

implies an increase in infections, we can find the value of $R(0)$, i.e. the required size of the vaccinated population, by demanding that $\frac{dI}{dt}$ should be zero. This would give the model a decrease in infection, and thus we are absolutely sure to have avoided an epidemic.

$$\beta I \frac{S}{N} - \gamma I = 0 \tag{1}$$

$$\beta I \frac{S}{N} = \gamma I \tag{2}$$

$$S = \frac{\gamma}{\beta} N = \frac{1/10}{0.2} 1000 = 500 \tag{3}$$

So we find that there can be no more than 500 people susceptible at the start, and 499 people at least need to be vaccinated at the start of the spread, to avoid disease. The result of this simulation is displayed in Fig. 2. Notice that the step size Δt had to be smaller for this simulation, since we are working with a specific edge case, where small changes are important. There are still some numerical errors present, since we don't expect the number of susceptibles or recovered individuals to change more than 1.

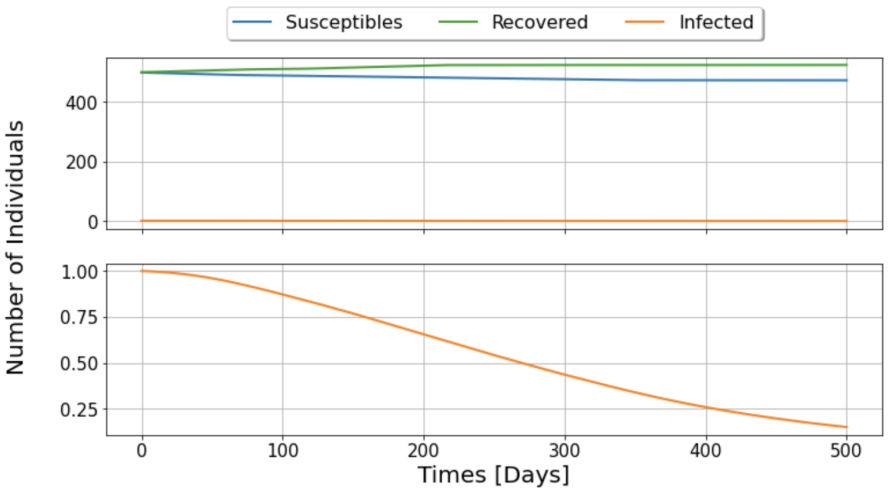

Figure 2: SIR-model simulation with $\Delta t = 0.0005$, $N = 1000$, $\beta = 0.2$, $\gamma = 10\text{days}^{-1}$, $I(0) = 1$ and $R(0) = 499$.

When choosing the size of the time step we have to find the right balance between a time step small enough that the solution is approximately continual, but not so small that the code takes too long to run or that the model is not able to see changes, making the values constant over time. To investigate this more quantitatively, we repeat the original simulation changing only the size of the time step Δt. We explore 10 different values of Δt between 1 and 0.0001. The number of iterations are chosen, so all time step values simulate 200 days. The resulting number of susceptibles (along with recovered and infected) should be converging for smaller time steps, therefor the number of susceptibles after 200 days is plotted against the value of Δt used for the given simulation. This can be seen in Fig. 3. $S(t)$ starts to converge around $\Delta t = 0.01$, which is the value we have chosen for this assignment.

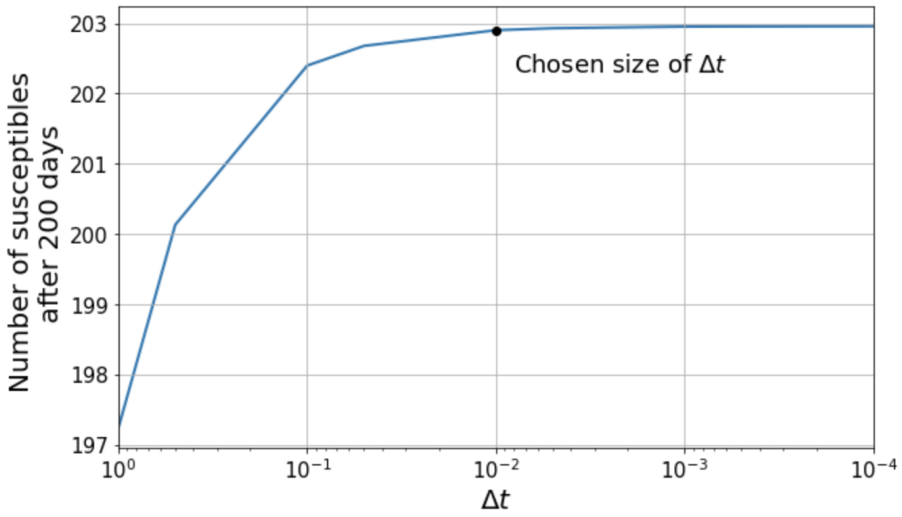

Figure 3: Final number of susceptibles in the end of a 200 day simulation, as a function of different values of Δt used in the simualtions. Notice that the x-axis is decreasing.

The rate of infection and recovery time β and γ are used as constants though they are factors that likely vary greatly in true situations. So in doing the simulation we have to make the assumption that the infection rate is somewhat stable (i.e. no new variants of disease) and that the population of people in question have some sort of average mode of behaviour, meaning we don't account for a sudden change in the level of interactions people have, for example a holiday season increasing the amounts of close contacts. In practise, the way we deal with it, would be to compare models with different parameter values to real data (fitting) and thus being able to analyse whether the parameter values change over time.

There are several ways one could cross-check the solution. First, we could examine an edge case, setting the starting parameters such that everyone in the population was infected, and then check that the amount of recovered people rose to the max of $N_S = 1000$, after 10 days. With the same logic, we can check that when nobody is infected from the beginning there is no disease in the population. Secondly, if we choose a specific number of susceptible from the start, say 50, and let β and γ be sufficiently large, we would expect exactly 50 individuals to be infected during the simulation (for a suitable period of time). Thirdly, we could solve the problem analytically, and compare that solution to our numerical model. Fourthly, one could adjust the parameters β and γ up or down, and check that the results correspond to faster/slower infection or slower/faster recovery, respectively. Finally, we could compare the numerical solution, with an appropriate choice of parameters, to the infection and recovery rates of an actual pandemic.

The c++ code can be seen below.

```
/* This is our assignment for the SIR model */


/* Including various libraries */
#include <iostream>
#include <vector>
#include <fstream>

/* Defining constants */
const float N=1000;
const float beta = 0.2;
const float gamma = 1.0/10.0;

/* Defining or constant step size */
const float t_step = 0.01;

/* Defining the number of iterations */
const int N_ite = 2e4;

/* Initial parameters */
std::vector<float> I = {1.0};
std::vector<float> R = {0.0};
std::vector<float> S = {N-I.back()};

/* Define a function calculating next step of S_(n+1) */
void update_vectors( std::vector<float> &S, std::vector<float> &I,
                     std::vector<float> &R  ) {

    /* Take the previous S value, aka the last element in the vector.
    Same for I and R */
    float S_now = S.back();
    float I_now = I.back();
    float R_now = R.back();

    /* Discretized equation for dS ---------------------------- */
    float S_next = S_now - ( beta * I_now * (S_now/N) )*t_step;
```

```
    /* Append S_next to the S vector */
    S.push_back(S_next);


    /* Discretized equation for dI ------------------------------ */
    float I_next = I_now + ( beta * I_now * (S_now/N) - gamma * I_now )*t_step;

    /* Append I_next to the I vector */
    I.push_back(I_next);


    /* Discretized equation for dR ------------------------------ */
    float R_next = R_now + ( gamma * I_now )*t_step;

    /* Append I_next to the I vector */
    R.push_back(R_next);
}

/* Update the vector N_ite times */
int main() {

    /* Define name of file with results */
    std::ofstream myfile("./sir_output.txt");

    /* Define the header */
    myfile << "SIR \n";

    /* Insert the initial parameters on the first row */
    myfile << S.back() << ' '
           << I.back() << ' '
           << R.back() << '\n';

    /* Make a for loop to update the vectors and export values to file */
    for (int i = 0; i < N_ite; ++i) {
        update_vectors(S, I, R);

        /* Insert to file, a row for each time step containing S, I, R */
        myfile << S.back() << ' '
               << I.back() << ' '
               << R.back() << '\n';
```

```
    }

    return 0;
}
```