High Performance Parallel Computing Assignment 1

António Maschio ; Dimitrios Anastasiou ; Georgios Sevastakis

February 2024

Objective

The goal is to simulate the dynamics of the SIR model in the $C++$ language for a given set of parameters and initial values.

Description of the SIR model

In this assignment we study the SIR model, which constitutes a simple mathematical model that describes the spread of a disease amongst individuals under some assumptions. It owes its name to its dynamical variables $S(t)$, $I(t)$ and $R(t)$ which stand for Susceptible, Infected and Recovered respectively.

According to this model, there is a number of individuals, the Susceptible, who are prone to the disease and can get infected by coming in contact with an individual from the Infected population. The spread is quantified by the parameter $\beta\,(days^{-1})$, which describes how infectious the disease is. After transitioning from the Susceptible to the Infected population, the recovery follows, whose rate is quantified by another parameter, $\gamma\,(days^{-1})$.

The SIR model assumes that the total number of individuals remains constant, meaning births and deaths are not included. Apart from that, Recovered individuals can not be infected again and all of the individuals are assumed to be prone to the disease (i.e. no vaccination is assumed). Last but not least, the parameters β and γ are assumed to be constant throughout time.

The following set of differential equations constitutes the SIR model:

$$\frac{dS}{dt} = -\frac{\beta}{N} I \cdot S \tag{1}$$

$$\frac{dI}{dt} = \frac{\beta}{N} I \cdot S - \gamma I \tag{2}$$

$$\frac{dR}{dt} = \gamma I \tag{3}$$

where N is the total number of individuals amongst all compartments.

Treatment

We solved the system of differential equations above numerically by implementing the forward Euler method in $C++$. By discretizing time and choosing an appropriate time step and initial values, we computed the values of S, I and R at the various time values as follows

$$S_{n+1} = S_n + \frac{dS}{dt}|_n \, dt \tag{4}$$

$$I_{n+1} = I_n + \frac{dI}{dt}|_n \, dt \tag{5}$$

$$R_{n+1} = R_n + \frac{dR}{dt}|_n \, dt \tag{6}$$

These values (t, S, I and R) were stored in vectors in our code and saved into a file (see Appendix A). The values were subsequently read by a python script tasked to plot S, I and R with respect to time t (see Appendix B).

Results

In this section, we present the output of our code

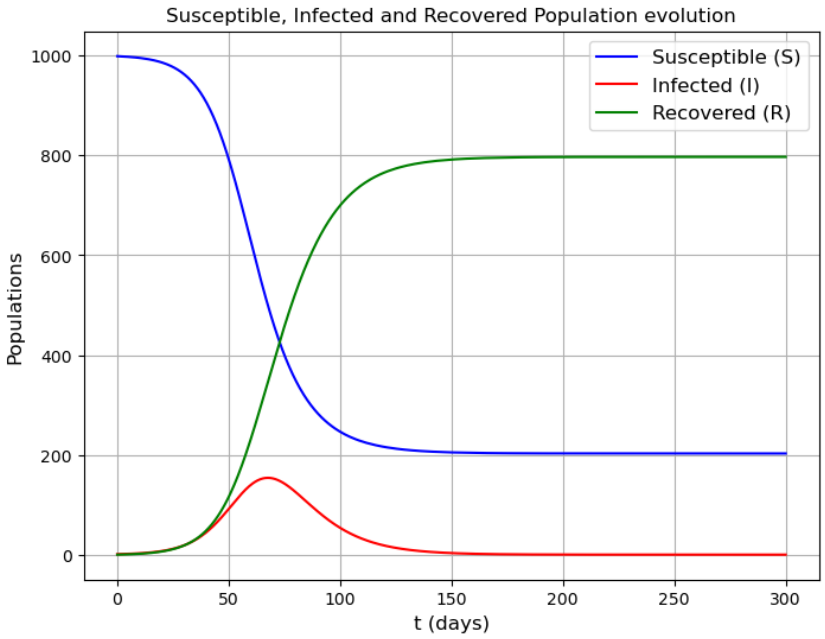

Figure 1: Susceptible (S), Infected (I) and Recovered (R) populations over a time span of 300 days with $dt = 0.1$, $\beta = 1/5$ and $\gamma = 1/10$ and $S(0) = N = 1000$, $I(0) = 1$ and $R(0) = 0$.

For a span of 300 days, we observe the spread of the disease among the total population ($N = 1000$) with just one infected individual at $t = 0\,days$. We observe that the disease spreads at an ever increasing rate as the days pass by until $t \approx 70\,days$. After that, the infection rate drops and the Susceptible line starts to plateau. The epidemic comes finally to an end after approximately 160 days, with 200 out of 999 susceptible individuals (not counting patient zero) never contracting the disease and with the rest recovering fully.

Questions

How many people should be vaccinated?

In order for an epidemic not to burst out, the rate of the Infected individuals at $t = 0$ needs to be negative. By substituting $\frac{dI}{dt}|_{t=0} < 0$ in Equation 2 and solving for the number of Susceptible individuals we get

$$\frac{\beta}{N} I \cdot S - \gamma I < 0 \tag{7}$$

$$\Leftrightarrow \left(\frac{\beta}{N} \cdot S - \gamma \right) I < 0 \tag{8}$$

Since $I(0) = 1$ for $t = 0$ and $N = 1000$, $\beta = \frac{1}{5}$, and $\gamma = \frac{1}{10}$, the number of vaccinated individuals at $t = 0$ should be more than 500, meaning that $S(0) < 500$.

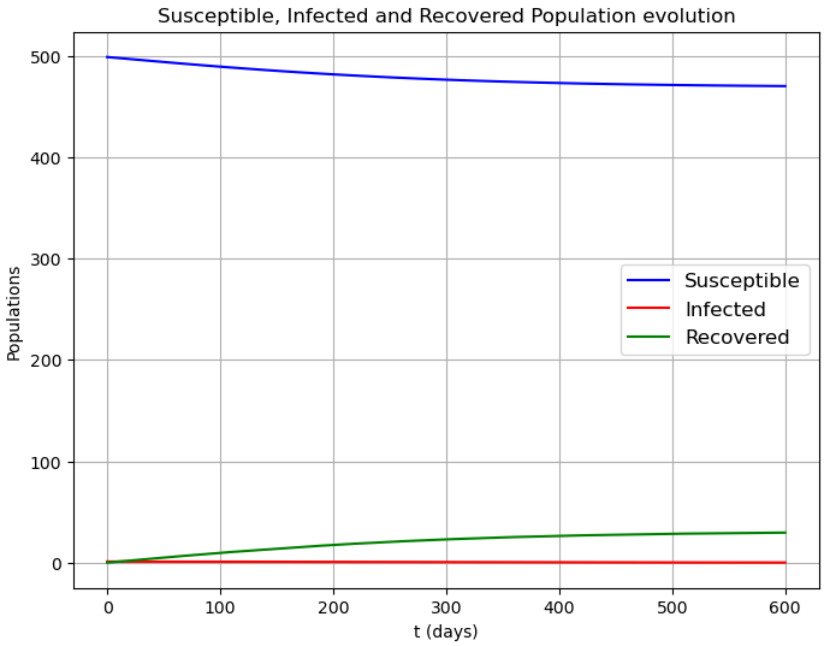

Figure 2: Susceptible (S), Infected (I) and Recovered (R) populations over a time span of 600 days with $dt = 0.1$, $\beta = 1/5$ and $\gamma = 1/10$ and $S(0) = 499$, $I(0) = 1$ and $R(0) = 0$.

How big should the time step be?

The choice of the time step should be based on a balance between the accuracy of the results and the computational cost. With smaller time steps, convergence and greater accuracy is achieved with a heavier computational cost. Larger time steps bring about faster execution times (low computational cost), but they could possibly lead to divergence

In our case, the choice of the time step is based on the values of the parameters β and γ, whose reciprocals constitute the time scales. Since $1/\beta = 5$ and $1/\gamma = 10$, the time step $dt = 0.1$ is a good choice and yields precise results.

β and γ are uncertain estimates of things that are probably not quite constant. How do you deal with this?

By definition, β constitutes the average number of individuals to be infected over a specific time frame. Equivalently, γ corresponds to the average number of infected individuals to recover over a time frame. It stands to reason to assume that these parameters vary with time or even have a direct dependency of S, I and/or R. For example, when people notice that there is a flu outbreak, many of them will probably be more cautious (i.e. wear masks) or even self-quarantine, thus dropping the value of β. As far as γ is concerned, a mutation or the development of a new drug against the disease could lead to a higher γ value.

How do you know your results are correct? You can never be sure, but provide 5 different methods to strengthen the trust in your results.

We could strengthen out trust in the results by:

- Using 4^{th} order Runge-Kutta method (or another method) as a solver, which is more computationally costly but yields more trustworthy results, and compare with the forward Euler method.
- Setting the initial value $I(0) = 0$. In this case, we expect none to be infected (horizontal lines in the plot).
- Repeating the simulation for smaller time steps and expecting the same outcome.
- Setting $\beta = 0$ or $\gamma = 0$. In the former case, we expect none of the susceptible individuals to get infected (other than patient zero) and in the latter, we expect none of the infected individuals to recover.
- Checking that the total number of individuals remains constant throughout the simulation, since $S + I + R = N$.

Appendices

A C++ code

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>

using namespace std;

int main(){

    // Declaration and Initialization of the problem's parameters.
    const int N = 1000;
    const double b = 1/5., g = 1/10.;

    // Declaration and Initialization of the variables S, I and R and Declaration of the rates and individual update variables.
    double S = N - 1, S_dot, S_new;
    double I = 1, I_dot, I_new;
    double R = 0, R_dot, R_new;

    // Declaration and Initialization of time parameters.
    const double T = 300, dt = 0.1;
    int Nt = 0;
    double t_i = 0;

    // Declaration of the vectors that will store the values.
    vector<double> t;
    vector<double> S_store;
    vector<double> I_store;
    vector<double> R_store;

    // Initial conditions in the vectors
    t.push_back(0);
    S_store.push_back(S);
    I_store.push_back(1);
    R_store.push_back(0);

    // Calculating total number of steps.
    Nt = floor(T / dt);

    // For-loop which implements the forward Euler method to compute the S, I and R variables over time.
    for(int i = 0; i < Nt; i++){
        // Counting the time.
        t_i += dt;
        t.push_back(t_i);

        // Updating and storing S.
        S_dot = -b/N * I * S;
        S_new = S + S_dot * dt;
        S_store.push_back(S_new);

```

```
        // Updating aand storing I.
        I_dot = b/N * I * S - g * I;
        I_new = I + I_dot * dt;
        I_store.push_back(I_new);

        // Updating and storing R.
        R_dot = g * I;
        R_new = R + R_dot * dt;
        R_store.push_back(R_new);

        // Updating the variables for the next iteration.
        S = S_new;
        I = I_new;
        R = R_new;
}
}
// We import the variables S, I and R, as well as the time t in a file,
   namely "sir_output.txt", with headers.
ofstream myfile;
myfile.open("sir_output.txt");
myfile << "Time (days)" << " " << "Susceptible" << " " << "Infected" << " "
   << "Recovered" << endl;
for (int j = 0; j <= Nt; j++) {
    myfile << t[j] << " " << S_store[j] << " " << I_store[j] << " " <<
   R_store[j] << endl;
    }
myfile.close();
return 0;
}
```

Forward Euler method in C++ to graph the dynamical evolution of the S I and R populations over time.

B Python code

```
import numpy as np
import matplotlib.pyplot as plt

t, S, I, R = np.loadtxt('c:\\Users\\George\\Desktop\\Cpp_codes\\sir_output.
   txt', skiprows=1, unpack=True)
print(S)
fig, ax = plt.subplots(figsize = (8,6))
plt.xlabel('t (days)')
plt.ylabel('Populations')
plt.title('Susceptible, Infected and Recovered Population evolution')
plt.plot(t, S, 'b', label = 'Susceptible')
plt.plot(t, I, 'r', label = 'Infected')
plt.plot(t, R, 'g', label = 'Recovered')
ax.legend(fontsize = 12)

plt.grid()
plt.show()
```

Python code that imports the data (t S I and R) and plots the S I and R over time