GreatMod: SIR model

Beccuti Marco, Castagno Paolo, Pernice Simone

Contents

Introduction

In this document we describe how to use the R library *epimod*. In details, *epimod* implements a new general modeling framework to study epidemiological systems, whose novelties and strengths are:

1. the use of a graphical formalism to simplify the model creation phase;
2. the automatic generation of the deterministic and stochastic process underlying the system under study;
3. the implementation of an R package providing a friendly interface to access the analysis techniques implemented in the framework;
4. a high level of portability and reproducibility granted by the containerization (Veiga Leprevost et al. 2017) of all analysis techniques implemented in the framework;
5. a well-defined schema and related infrastructure to allow users to easily integrate their own analysis workflow in the framework.

The effectiveness of this framework is showed through the wellknown and simple SIR model.

How to start

Before starting the analysis we have to install (1) GreatSPN GUI, (2) docker, and (3) the R package **devtools** for installing *EPIMOD*. GreatSPN GUI, the graphical editor for drawing Petri Nets formalisms, is available online (link to install GreatSPN), and it can be installed following the steps showed therein. Then, the user must have docker installed on its computer for exploiting the *epimod*'s docker images (for more information on the docker installation see: link to install docker), and to have authorization to execute docker commands reported in the command page of function install docker. To do this the following commands must be executed.

1. Create the docker group.

```
    $ sudo groupadd docker
```

2. Add your user to the docker group.

```
    $ sudo usermod -aG docker $USER
```

The R package *devtools* has to be installed to run *epimod*:

```
install.packages("devtools")
library(devtools)
install_github("qBioTurin/epimod", dependencies=TRUE)
```

```
library(epimod)
```

Then, the following function must be used to download all the docker images used by *epimod*:

```
downloadContainers()
```

Something to know

All the *epimod* functions print the following information:

- *Docker ID*, that is the CONTAINER ID which is executed by the function;
- *Docker exit status*, if 0 then the execution completed with success, otherwise an error log file is saved in the working directory.

Cases of study

In this section we show the steps necessary to model, study and analyze a simple case study. To this aim, we choose to study the diffusion of a disease following the SIR dynamics. We refer to (Keeling and Rohani 2011) for all the details.

SIR model

The S-I-R models the diffusion of an infection in a population assuming three possible states (or compartments) in which any invidual in the population may move. Specifically, (1) *Susceptible*, individuals unexposed to the disease, (2) *Infected*, individuals currently infected by the disease, and (3) *Recovered*, individuals which were successfully recovered by the infection. To consider the simplest case, we ignore the population demography (i.e., births and deaths of individuals are omitted), thus we consider only two possible events: the infection (passage from *Susceptible* to *Infected*), and the recovery (passage from *Infected* to *Recovered*). We are also assuming to neglect complex pattern of contacts, by considering an homogeneous mixing. From a mathematical point of view, the system behaviors can be investigated by exploiting the deterministic approach (Kurtz 1970) which approximates its dynamics through a system of ordinary differential equations (ODEs):

$$\begin{aligned}\frac{dS}{dt} &= -\frac{\beta}{N}SI,\\ \frac{dI}{dt} &= \frac{\beta}{N}SI - \gamma I,\\ \frac{dR}{dt} &= \gamma I,\end{aligned}$$

where:

- S, I, R are the number of susceptible, infected, and recovered individuals, respectively;
- β is the infection rate;
- N is the constant population size;
- γ is the recovery rate, which determines the mean infectious period.

Model generation

The first step is the model construction. Starting with the GreatSPN editor tool it is possible to draw the model using the PN formalism and its generalizations. We recall that the Petri Nets are bipartite graphs in which we have two type of nodes, places and transitions. Graphically, places are represented as circles and those are the variables of our systems. On the other hand, transitions are depicted as rectangles and are the possible events happening in the system. Variables and events (i.e., places and transitions) are connected through arcs, showing what variable(s) is (are) affected by a specific event. For more details we refer to (Marsan et al. 1995).

Therefore, as represented in figure 1, we add one place for each variable of the system (i.e., S, I, and R represent the susceptible, infected, and recovered individuals respectively), and one transition for each possible event (i.e., *Infection* and *Recovery*). Finally, we save the PN model as a file with extension *.PNPRO* .

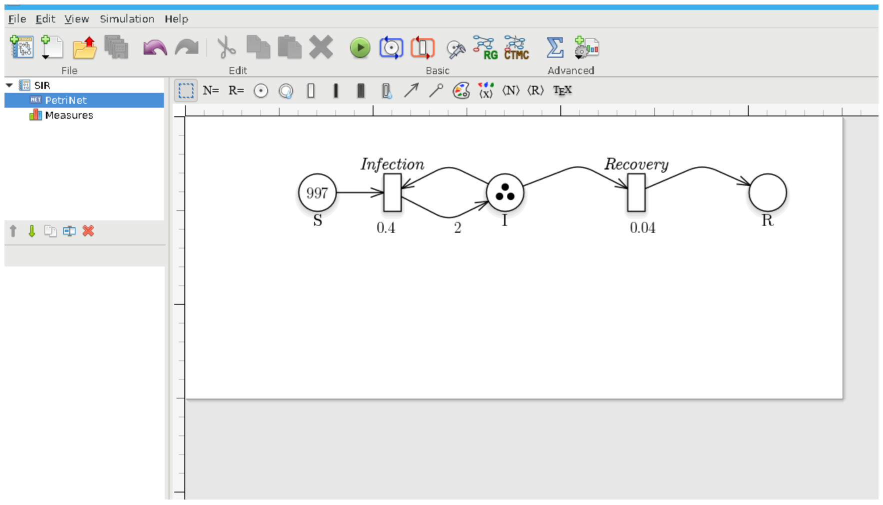


Figure 1: Petri Net representation of the SIR model.

Having constructed the model, the generation of both the stochastic (the Continuous Time Markov Chain) and deterministic (ODEs) processes underlying the model is implemented by the *model_generation()* function. This function takes as input the file generated by the graphical editor, in this case called *SIR.PNPRO*, and automatically derives the processes.

```
model_generation(net_fname = "./Net/SIR.PNPRO")
```

The binary file *SIR.solver* is generated in which the derived processes and the library used for their simulation are packaged.

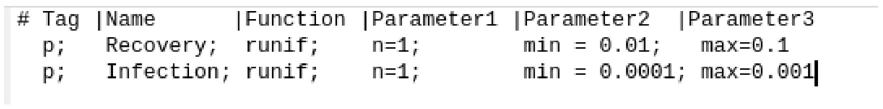
Sensitivity analysis

iniziare da quello più semplice e passo per passo si complica! tolgo functions nel primo e rifai le immagini togliendo il cancelletto, caso senza marcatura iniziale ma fai una composizione tra due dostribuzioni per un paramtero. Frase per dire che le tracee posso essere generate dallo script!

The second step is represented by the sensitivity analysis, in which the deterministic process is solved several times varying the values of the unknown parameters to identify which are the sensitive ones (i.e., those that have a greater effect on the model behavior), by exploiting the Pearson Ranking Correlation Coefficients (PRCCs). This may simplify the calibration step reducing (1) the number of variables to be estimated and (2) the search space associated with each estimated parameter. With this purpose, the function *sensitivity_analysis()* calculates the PRCCs, and, given a reference dataset and a distance measure, it ranks the simulations according to the distance of each solution with respect to the reference one. In details, the function *sensitivity_analysis()* takes in input

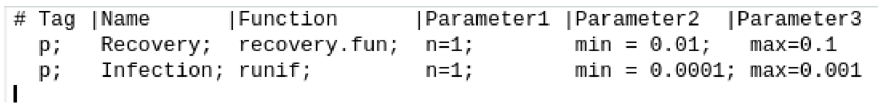
1. **solver_fname**: the *.solver* file generated by the *model_generation* function, that is *SIR.solver*;
2. **n_config**: the total number of samples to be performed, for instance 200;
3. **f_time**: the final solution time, for instance 10 weeks (70 days);
4. **s_time**: the time step defining the frequency at which explicit estimates for the system values are desired, in this case it could be set to 1 day;
5. **parameters_fname**: a textual file in which the parameters to be studied are listed associated with their range of variability. This file is defined by three mandatory columns: (1) a tag representing the parameter type: *i* for the complete initial marking (or condition), *p* for a single parameter (either a single rate or initial marking), and *g* for a rate associated with general transitions (Pernice et al.

2019) (the user must define a file name coherently with the one used in the general transitions file); (2) the name of the transition which is varying (this must correspond to name used in the PN draw in GreatSPN editor), if the complete initial marking is considered (i.e., with tag *i*) then by default the name *init* is used; (3) the function used for sampling the value of the variable considered, it could be either a R function or an user-defined function (in this case it has to be implemented into the R script passed through the *functions_fname* input parameter). Let us note that the output of this function must have size equal to the length of the varying parameter, that is 1 when tags *p* or *g* are used, and the size of the marking (number of places) when *i* is used. The remaining columns represent the input parameters needed by the functions defined in the third column. An example is given by the file *Functions_list.csv*, where we decided to vary the rates of the *Recovery* and *Infection* transitions by using the R function which generates values following the uniform probability distribution on the interval from *min* to *max*. We set *n=1* because we must generate one value for each sample.



```
# Tag |Name      |Function |Parameter1 |Parameter2   |Parameter3
  p;   Recovery;  runif;    n=1;        min = 0.01;   max=0.1
  p;   Infection; runif;    n=1;        min = 0.0001; max=0.001
```

6. **functions_fname**: an R file storing the user defined functions to generate instances of the parameters summarized in the *parameters_fname* file. An example is given by *Functions.R*, where the function *recovery.fun* introduced in *Functions_list2.csv* file (see the next image) is defined in order to uniformly sample the recovery transition rate between *min* and *max*.



```
# Tag |Name      |Function      |Parameter1 |Parameter2   |Parameter3
  p;   Recovery;  recovery.fun;  n=1;        min = 0.01;   max=0.1
  p;   Infection; runif;         n=1;        min = 0.0001; max=0.001
```

```
recovery.fun<-function(min , max)
{
   # min/max are the bounds for the uniform ditribution

    p_1=runif(n=1,min=min,max=max)

    return(p_1)
}
```

7. **target_value_fname**: an R file providing the function to obtain the place or a combination of places from which the PRCCs over the time have to be calculated. In details, the function takes in input a *data.frame*, namely *output*, defined by a number of columns equal to the number of places plus one corresponding to the time, and number of rows equals to number of time steps defined previously. Finally, it must return the column (or a combination of columns) corresponding to the place (or combination of places) for which the PRCCs have to be calculated for each time step. An example is given in *Target.R*, where the PRCCs are calculated with respect to place *I* (infected individuals).

```
Target<-function(output)
{
    I <- output[,"I"]
    return(I)
}
```

8. **reference_data**: a csv file storing the data to be compared with the simulations' result. In *reference_data.csv* we report the SIR evolution starting with 997 susceptible, three infected and zero recovered, with a recovery and infection rates equals to 0.04 and 0.0004 respectively. Notice that the **reference_data**'s rows must be the variable time serie, and so the colums the corresponding values at a specific time.

```
#>           Time        S         I         R
#> TimeStep1    0 997.0000  3.000000 0.0000000
#> TimeStep2    1 995.5620  4.293673 0.1443379
#> TimeStep3    2 993.5082  6.140960 0.3508461
#> TimeStep4    3 990.5796  8.774355 0.6460555
#> TimeStep5    4 986.4130 12.519415 1.0675618
#> TimeStep6    5 980.5043 17.827326 1.6683746
#> TimeStep7    6 972.1632 25.314077 2.5227070
```

9. **distance__measure__fname**: the R file storing the function to compute the distance (or error) between the model output and the reference dataset itself. The function defining the distance takes in input only the reference data and the simulation's output (i.e. a trajectory); an example is given by *msqd.R* where a distance measure (based on the squared error distance) as function of the infected individuals is defined:

```
msqd<-function(reference, output)
{
    reference[1,] -> times_ref
    reference[3,] -> infect_ref

    # We will consider the same time points
    Infect <- output[,"I"]

    diff.Infect <- 1/length(times_ref)*sum(( Infect - infect_ref )^2 )

    return(diff.Infect)
}
```

Let us observe that: (i) the distance and target functions must have the same name of the corresponding R file,(ii) *sensitivity_analysis* exploits also the parallel processing capabilities, and (iii) if the user is not interested on the ranking calculation then the **distance__measure__fname** and **reference__data** are not necessary and can be omitted.

Hence, considering the SIR model we can run the *sensitivity_analysis* varying the *Infection* and *Recovery* transitions rates in order to characterized their effect on the number of infected individuals.

```
sensitivity<-sensitivity_analysis(n_config = 200,
                                  parameters_fname = "Input/Functions_list.csv",
                                  solver_fname = "Net/SIR.solver",
                                  target_value_fname = "Rfunction/Target.R" ,
                                  parallel_processors = 2,
                                  f_time = 100, # days
                                  s_time = 1 # day
                                  )
```

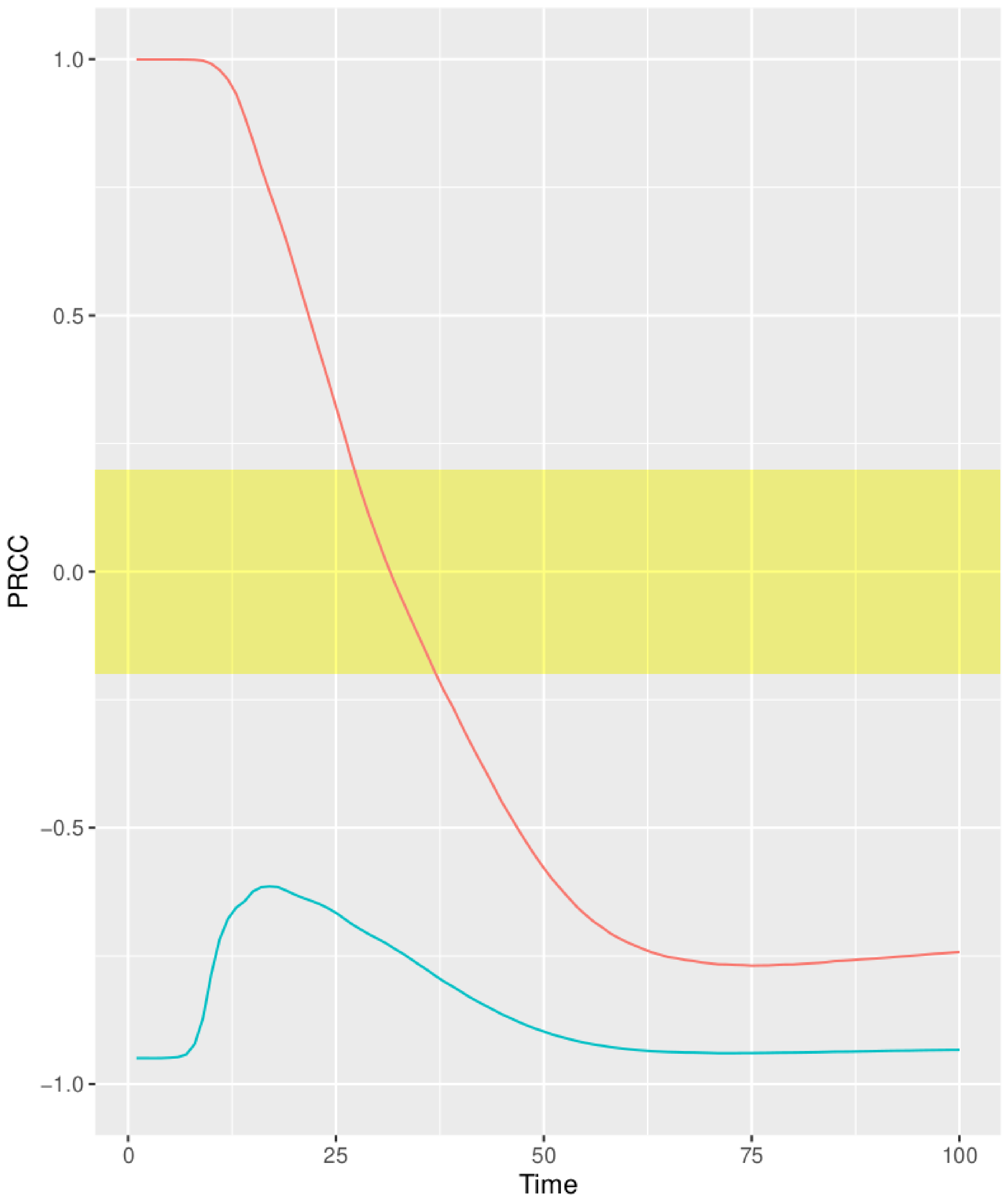
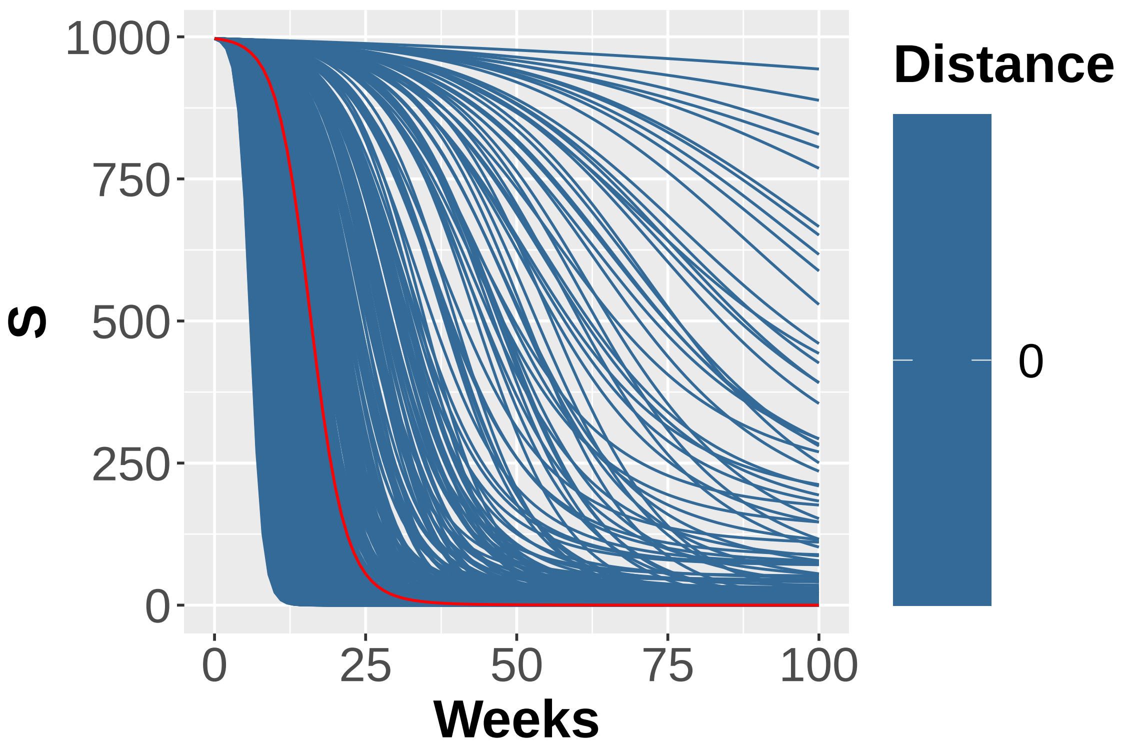
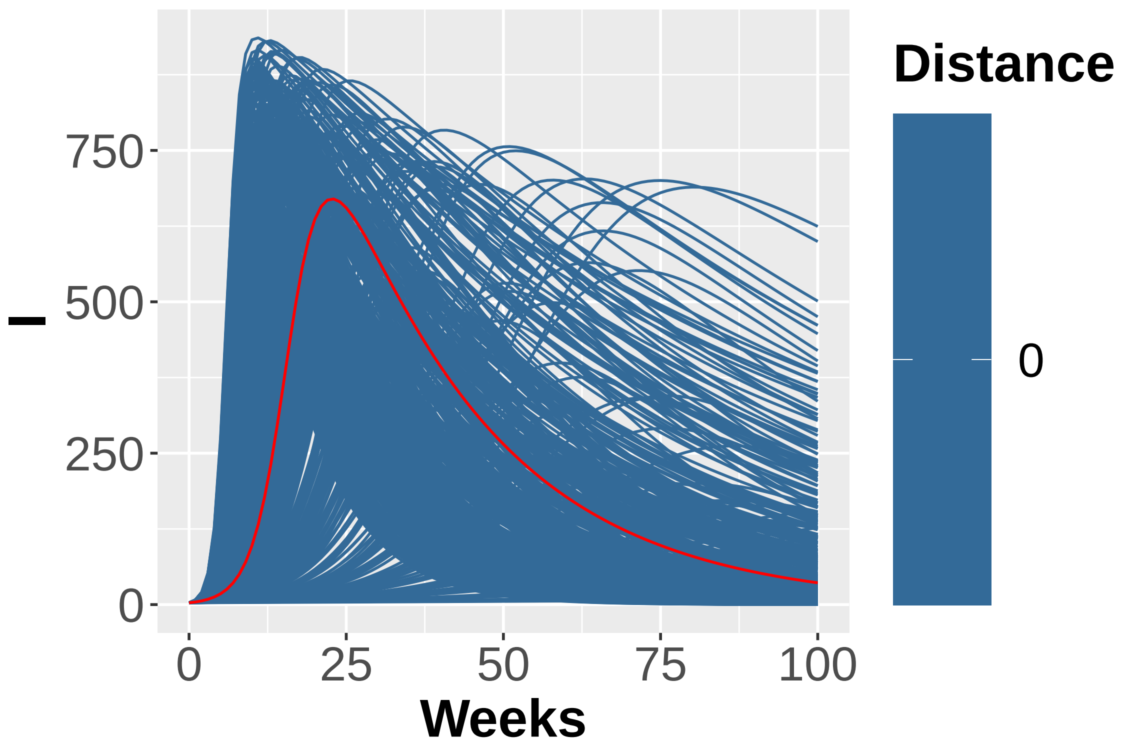


Figure 2: PRCC for the I place over time.

The PRCCs values for these two parameters, depicted in figure 2, with respect the number of infections over the entire simulated period are both meaningful, especially in the first part of the simulation, corresponding to the transient part where the parameters affect mostly the output. Differently, this effect decreases after the fifth week where all the deterministic trajectories obtained with different parameters configurations converge to the same states, see figure 3.





With ranking

```r
sensitivity<-sensitivity_analysis(n_config = 100,
                                  parameters_fname = "Input/Functions_list.csv",
                                  functions_fname = "Rfunction/Functions.R",
                                  solver_fname = "Net/SIR.solver",
                                  reference_data = "Input/reference_data.csv",
                                  distance_measure_fname = "Rfunction/msqd.R" ,
                                  target_value_fname = "Rfunction/Target.R" ,
                                  parallel_processors = 2,
                                  f_time = 100, # days
                                  s_time = 1 # day
                                  )
```

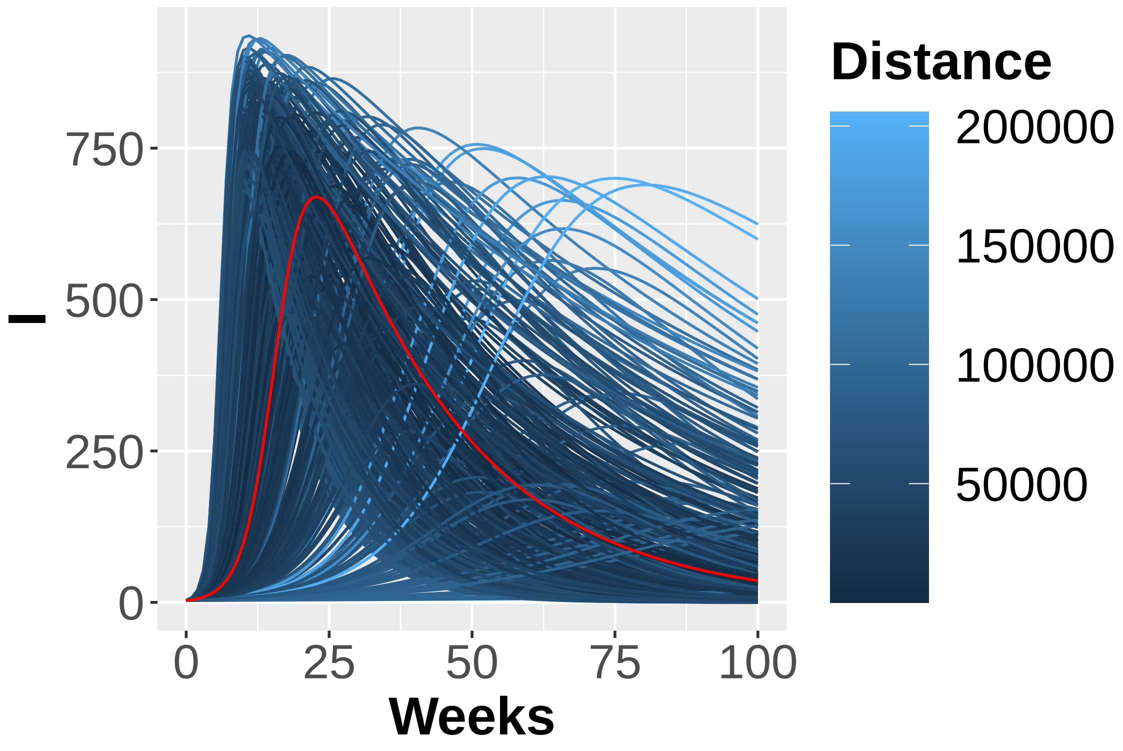


Figure 3: The 200 trajectories considering the I place obtained from different parameters configurations.

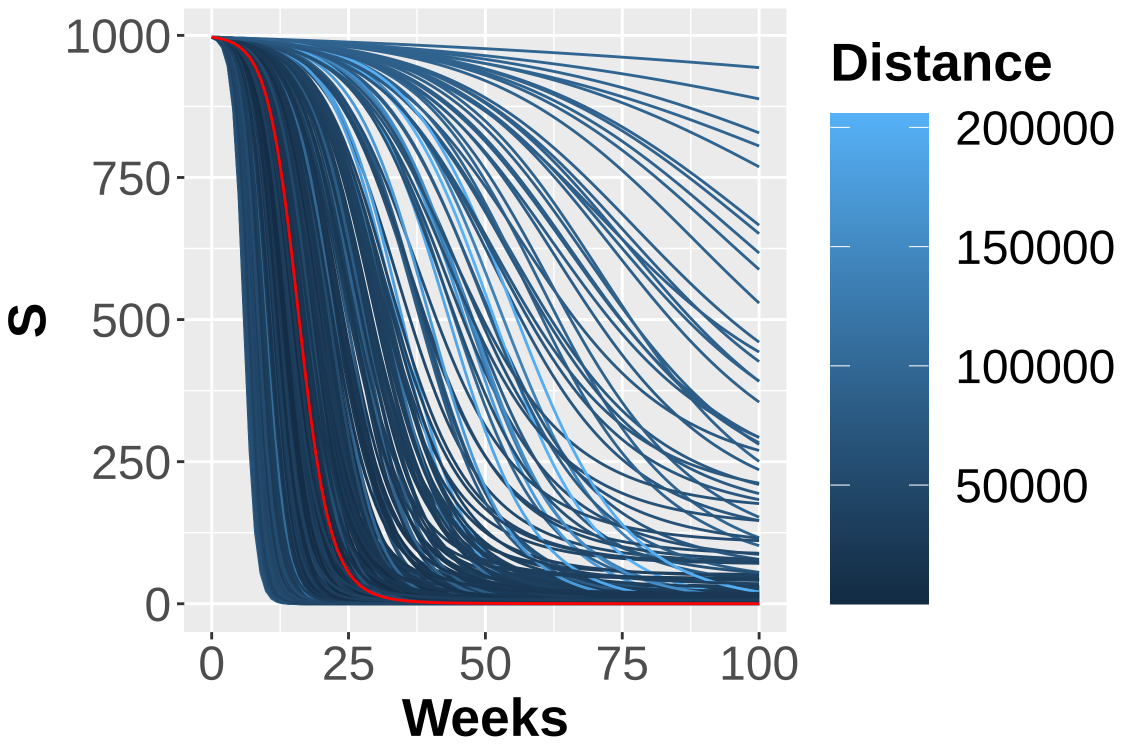


Figure 4: The 200 trajectories considering the S place obtained from different parameters configurations.

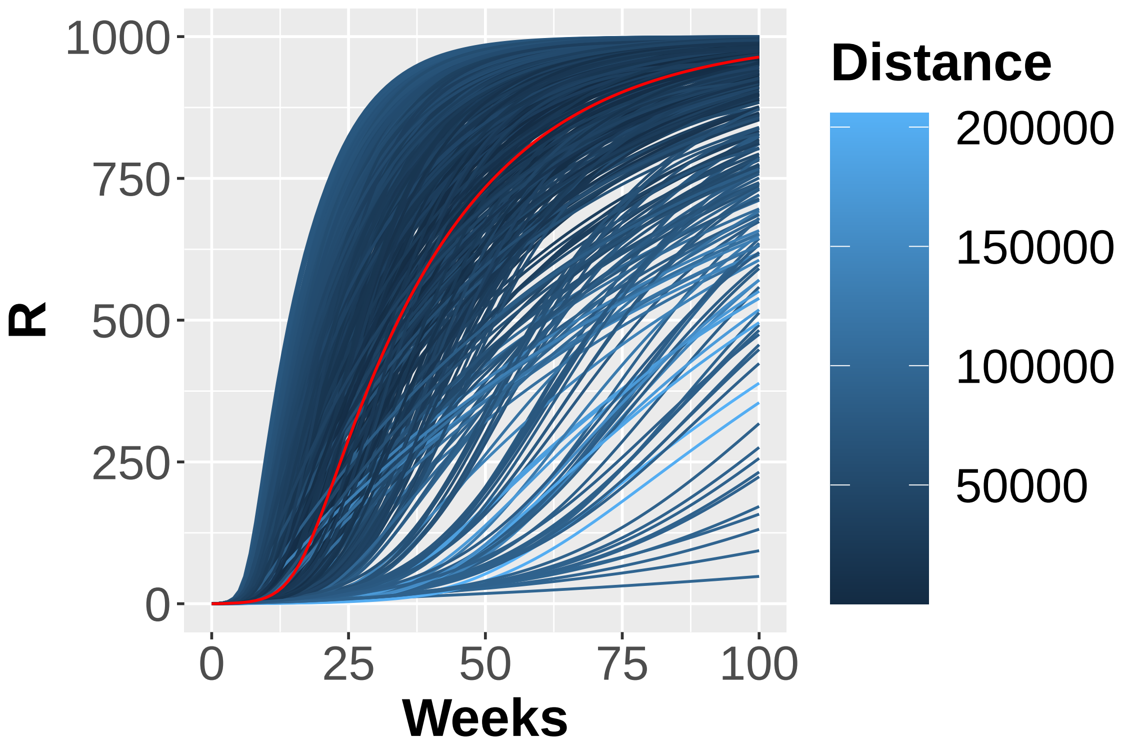


Figure 5: The 200 trajectories considering the R place obtained from different parameters configuration.

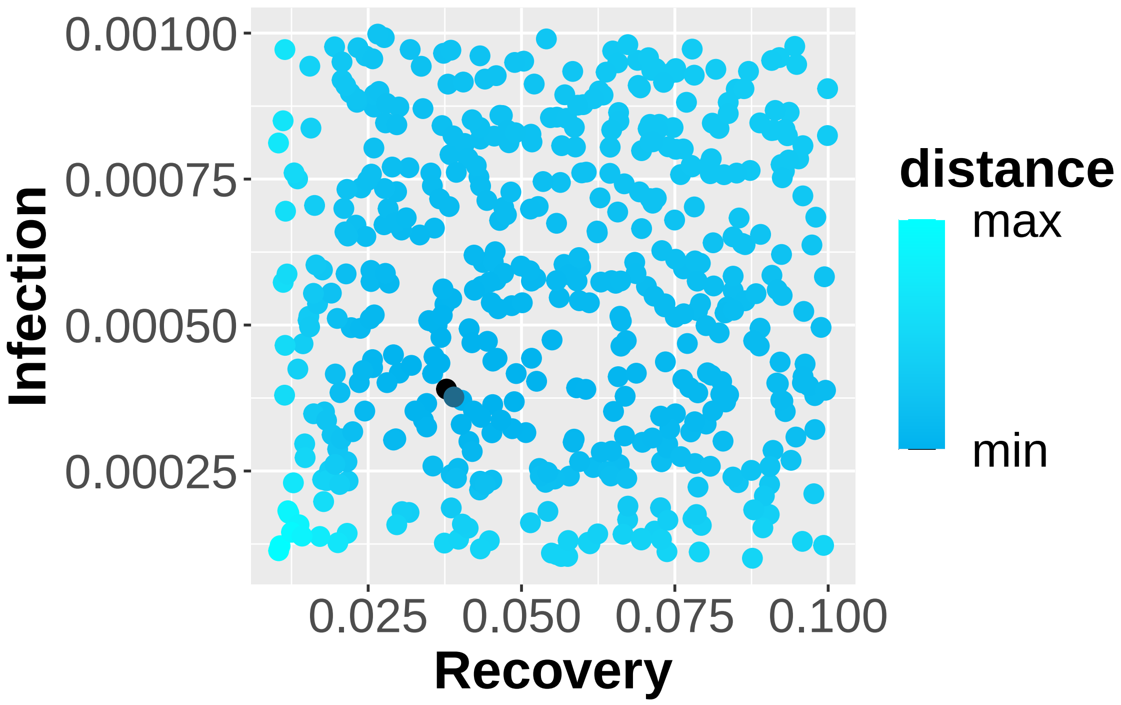


Figure 6: Scatter plot showing the squared error between the reference data and simulated number of infected. The dark blue points represent the parameters configuration with minimum error.

From the figures 3, 5, and 4, it is possible to observe the different trajectories obtained by solving the system of ODEs, represented by eq. **??**, with different parameters configurations, sampled by exploiting the function passed through **parameters_fname**. In figure 6 the distance values, obtained using the measure definition described before, are plotted varying the *Recovery* parameter (on the x-axis) and *Infection* parameter (on the y-axis). Each point is colored according to a nonlinear gradient function starting from color dark blue (i.e., lower value) and moving to color light blue (i.e., higher values). From this plot we can observe that lower squared errors are obtained when *Recovery* is between [0.025,0.05] and *Infection* [0.00025,0.0005] thus we can reduce the search space associated with the two parameters around these two values.

Other possible examples of how to use this function are reported hereafter:

```
sensitivity<-sensitivity_analysis(n_config = 100,
                                  parameters_fname = "Input/Functions_list2.csv",
                                  functions_fname = "Rfunction/Functions.R",
                                  solver_fname = "Net/SIR.solver",
                                  reference_data = "Input/reference_data.csv",
                                  distance_measure_fname = "Rfunction/msqd.R" ,
                                  target_value_fname = "Rfunction/Target.R" ,
                                  parallel_processors = 2,
                                  f_time = 100, # days
                                  s_time = 1 # days
                                  )
```

Calibration analysis

The aim of this phase is to optimize the fit of the simulated behavior to the reference data by adjusting the parameters associated with both Recovery and Infection transitions. This step is performed by the function *model_calibration()*, characterized by the solution of an optimization problem in which the distance between the simulated data and the reference data is minimized, according to the definition of distance provided by the user (**distance_fname**).

```r
model_calibration(parameters_fname = "Input/Functions_list_Calibration.csv",
                  functions_fname = "Rfunction/FunctionCalibration.R",
                  solver_fname = "Net/SIR.solver",
                  reference_data = "Input/reference_data.csv",
                  distance_measure_fname = "Rfunction/msqd.R" ,
                  f_time = 100, # days
                  s_time = 1, # day
                  # Vectors to control the optimization
                  ini_v = c(0.035,0.00035),
                  ub_v = c(0.05, 0.0005),
                  lb_v = c(0.025, 0.00025),
                  max.time = 1
                  )
```

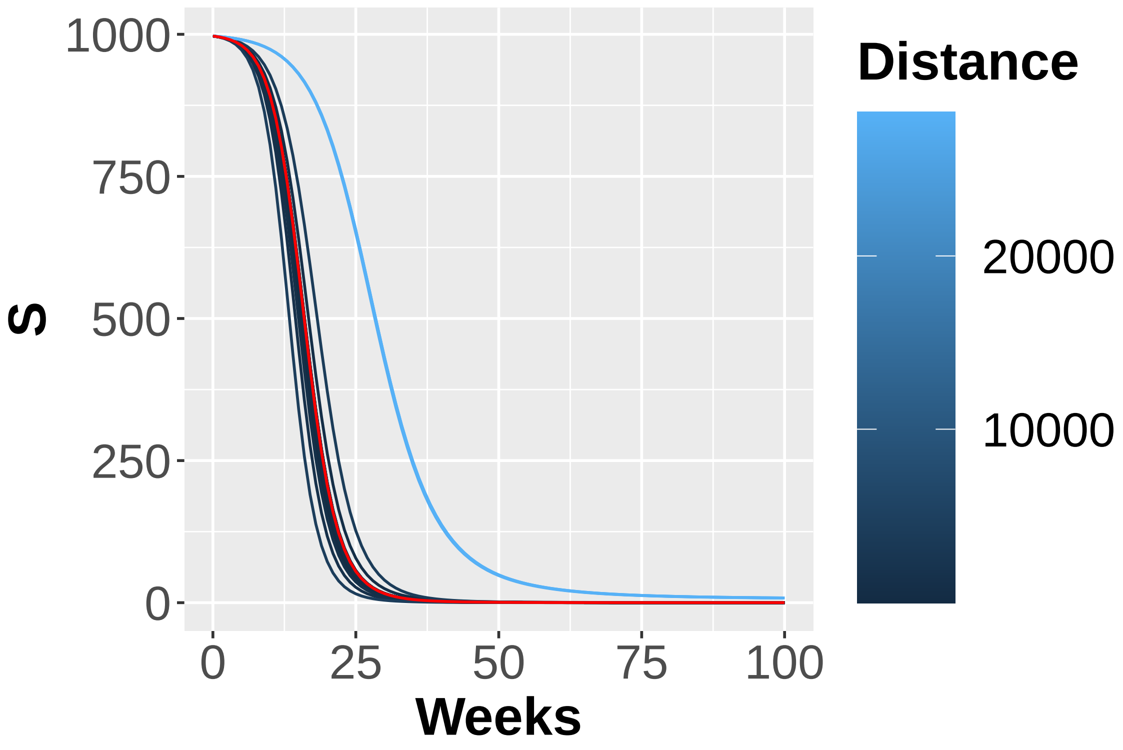


Figure 7: Trajectories considering the S place.

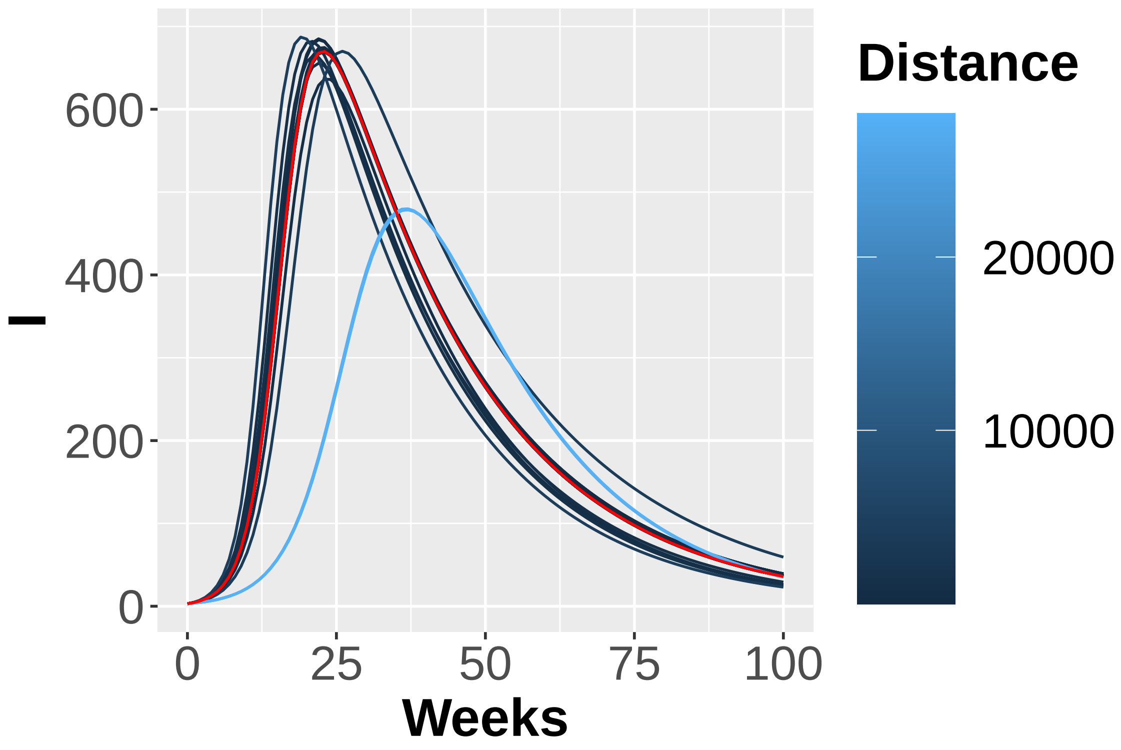


Figure 8: Trajectories considering the I place.

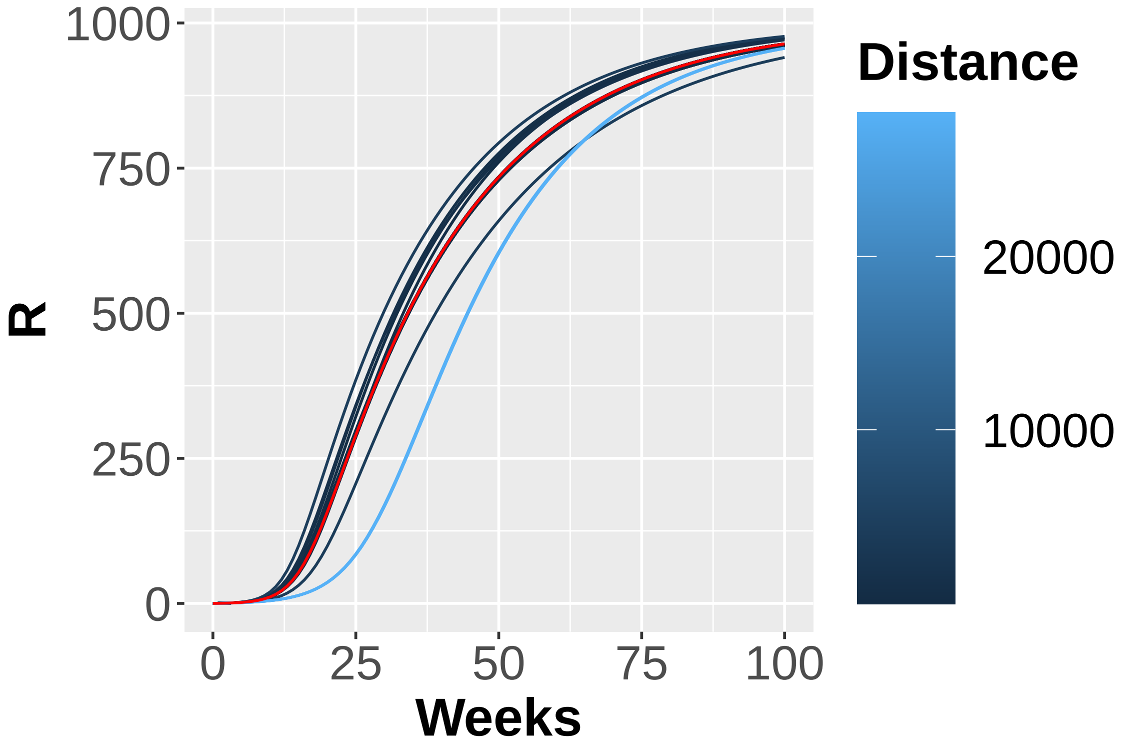


Figure 9: Trajectories considering the R place.

In figures 8,7 and 9 the trajectories with color depending on the squared error w.r.t. reference trend are plotted. In this case, fixing a maximum number of objective function calls, we obtain the following optimal value for the two parameters:

```
#> [1] 0.0399999567 0.0004000018
```

Model Analysis

The last step is the model analysis, where the corresponding function *model_analysis()* executes and tests the behavior of the developed model. Furthermore, by changing the input parameters, it is possible to perform a *what-if* analysis or forecasting the evolution of the diffusion process. This function solves the system given a specific parameters configuration which is passed through the function parameter, *parameters_fname*. In this case, instead of writing a function to sample or define the parameter variability, we can pass the specific value obtained from the calibration for generating the corresponding trajectory.

```
#>   Tag      Name Specific value NA
#> 1   p  Recovery          4e-02 NA
#> 2   p Infection          4e-04 NA
```

```r
model_analysis(out_fname = "model_analysis",
               solver_fname = "Net/SIR.solver",
               parameters_fname = "Results/Functions_list_ModelAnalysis.csv",
               f_time = 7*10, # weeks
               s_time = 1
               )
```

References

Keeling, Matt J, and Pejman Rohani. 2011. *Modeling Infectious Diseases in Humans and Animals*. Princeton University Press.

Kurtz, T. G. 1970. "Solutions of Ordinary Differential Equations as Limits of Pure Jump Markov Processes." *J. Appl. Probab.* 1 (7): 49–58.

Marsan, M. Ajmone, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. 1995. *Modelling with Generalized Stochastic Petri Nets*. New York, NY, USA: J. Wiley.

Pernice, S., M. Pennisi, G. Romano, A. Maglione, S. Cutrupi, F. Pappalardo, G. Balbo, M. Beccuti, F. Cordero, and R. A. Calogero. 2019. "A Computational Approach Based on the Colored Petri Net Formalism for Studying Multiple Sclerosis." *BMC Bioinformatics*.

Veiga Leprevost, Felipe da, Björn A Grüning, Saulo Alves Aflitos, Hannes L Röst, Julian Uszkoreit, Harald Barsnes, Marc Vaudel, et al. 2017. "BioContainers: an open-source and community-driven framework for software standardization." *Bioinformatics* 33 (16): 2580–2.