GreatMod: SIR model

Baccega Daniele, 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 well-known 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 the authorization to execute docker commands reported in the command page of 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("https://github.com/qBioTurin/epimod", ref = "master", dependencies = TRUE)
```

Now we need to include the *epimod* library:

```
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):

$$\frac{dS}{dt} = - \beta \ SI,$$
$$\frac{dI}{dt} = \ \beta \ SI - \gamma I,$$
$$\frac{dR}{dt} = \gamma I,$$

where:

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

Analysis

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*.
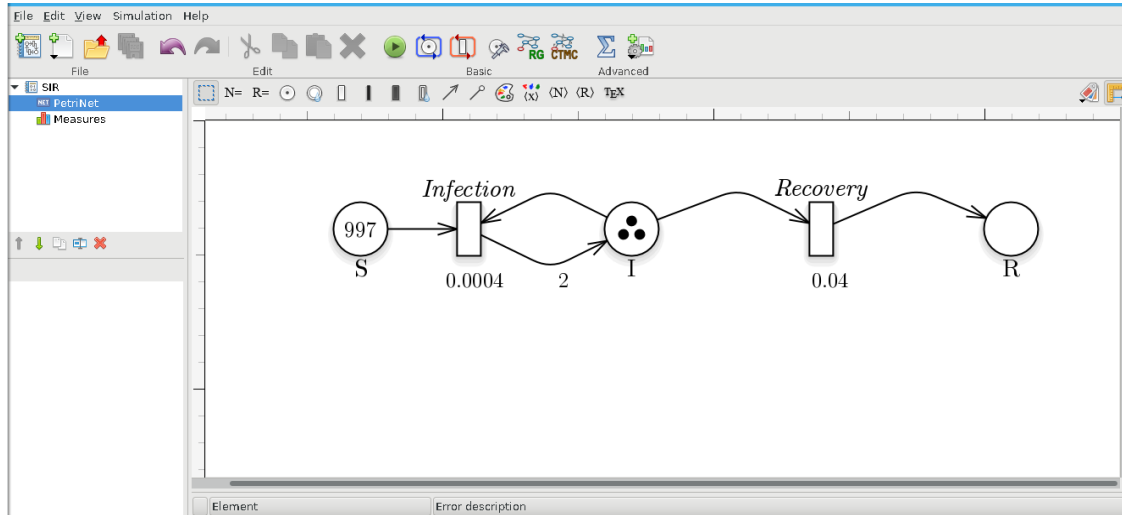

Figure 1: Petri Net representation of the SIR model.

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("SIR.PNPRO")
```

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

Model Analysis

A possible 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.

```
model.analysis(solver_fname = "SIR.solver",
               i_time = 1,
               f_time = 100, # days
               s_time = 1)
```

1. **solver_fname**: the *.solver* file generated by the *model.generation* function, that is *SIR.solver*;
2. **i_time**: the initial solution time, for instance day 1;
3. **f_time**: the final solution time, for instance 100 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.

```
## How to generate the plots
source("Rfunction/ModelAnalysisPlot.R")
AnalysisPlot = ModelAnalysisPlot(solverName_path = "./SIR.analysis/SIR-analysys-1.trace",
                                 Stoch = F,
                                 print = F)

AnalysisPlot$plAll
```

It is also possible to simulate the stochastic behavior of the system exploiting the Gillespie algorithm, namely **SSA**, which is an exact stochastic method widely used to simulate chemical systems whose behavior can be described by the Master equations.

```
model.analysis(solver_fname = "SIR.solver",
               solver_type = "SSA",
               i_time = 1,
               f_time = 100, # days
               s_time = 1,
               n_run = 500,
               parallel_processors = 2)
```

1. **solver_type**: type of solver to use;
2. **n_run**: number of stochastic simulations to run;
3. **parallel_processors**: number of processors to use for parallelize the simulations.

```
## How to generate the plots
source("Rfunction/ModelAnalysisPlot.R")
AnalysisPlot = ModelAnalysisPlot(solverName_path = "./SIR_analysis/SIR-analysys-1.trace",
                                 Stoch = T,
                                 print = F)

AnalysisPlot$plAll
AnalysisPlot$plAllMean
```
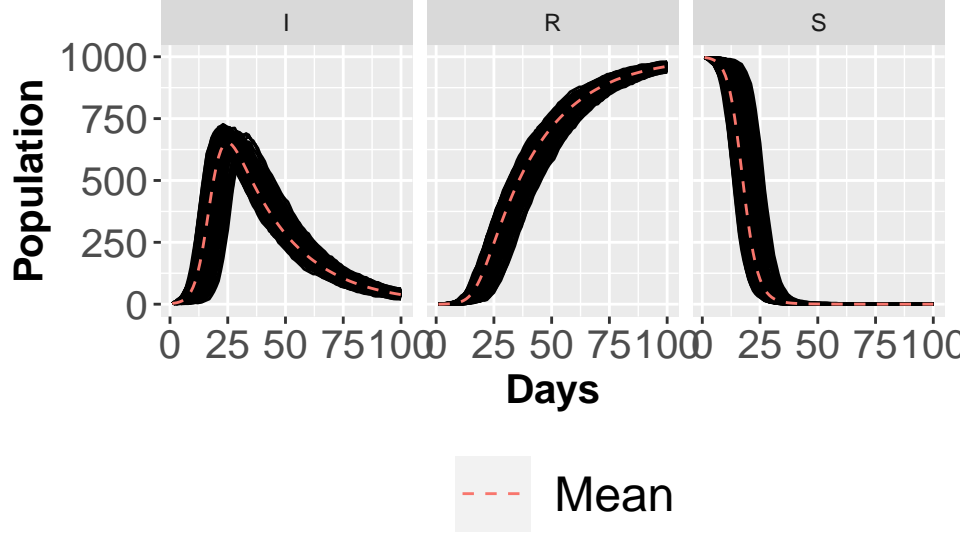
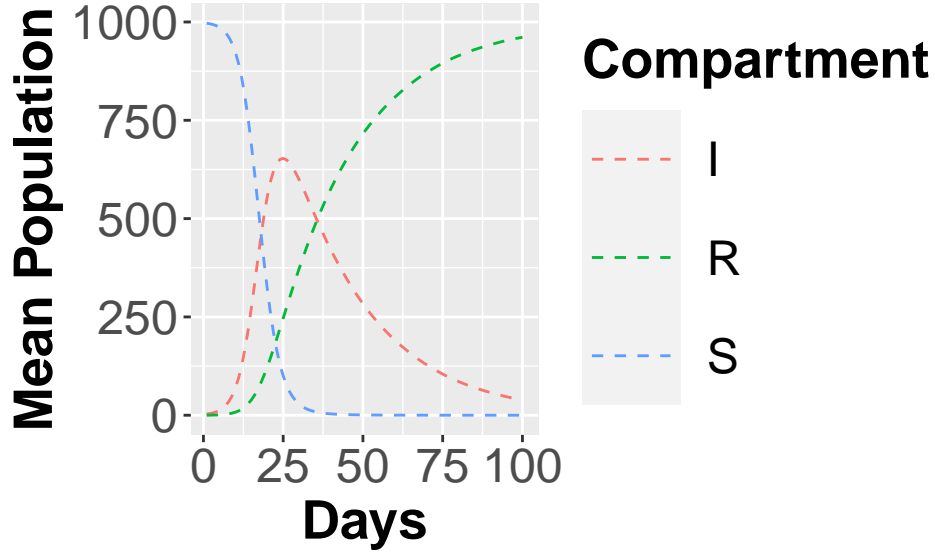Figure 2: Stochastic trajectories.

Figure 3: Average of the stochastic trajectories.

A further feature is given by the possibility to change the transition rates without modifying the Petri Net model, but directly from R.

```
model.analysis(solver_fname = "SIR.solver",
               parameters_fname = "Functions_list_ModelAnalysis.csv",
               i_time = 1,
               f_time = 100, # days
               s_time = 1)
```

1. **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 (*which must separeted using ;*): (1) a tag representing the parameter type: i for the complete initial marking (or condition), m for the initial marking of a specific place, c for a single constant rate, 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 the 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 *m*, *c* 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.

```
1   c;Recovery;0.04;
2   c;Infection;0.0002;
3
```

This function solves the system given a specific parameters configuration which is passed through the function parameter, *parameters_fname*. In details, we split in half the *Infection* rate (from 0.0004 to 0.0002).

```
## How to generate the plots
source("Rfunction/ModelAnalysisPlot.R")
AnalysisPlot = ModelAnalysisPlot(solverName_path = "./SIR_analysis/SIR-analysys-1.trace",
                                 Stoch = F,
                                 print = F)
AnalysisPlot$plAll
```
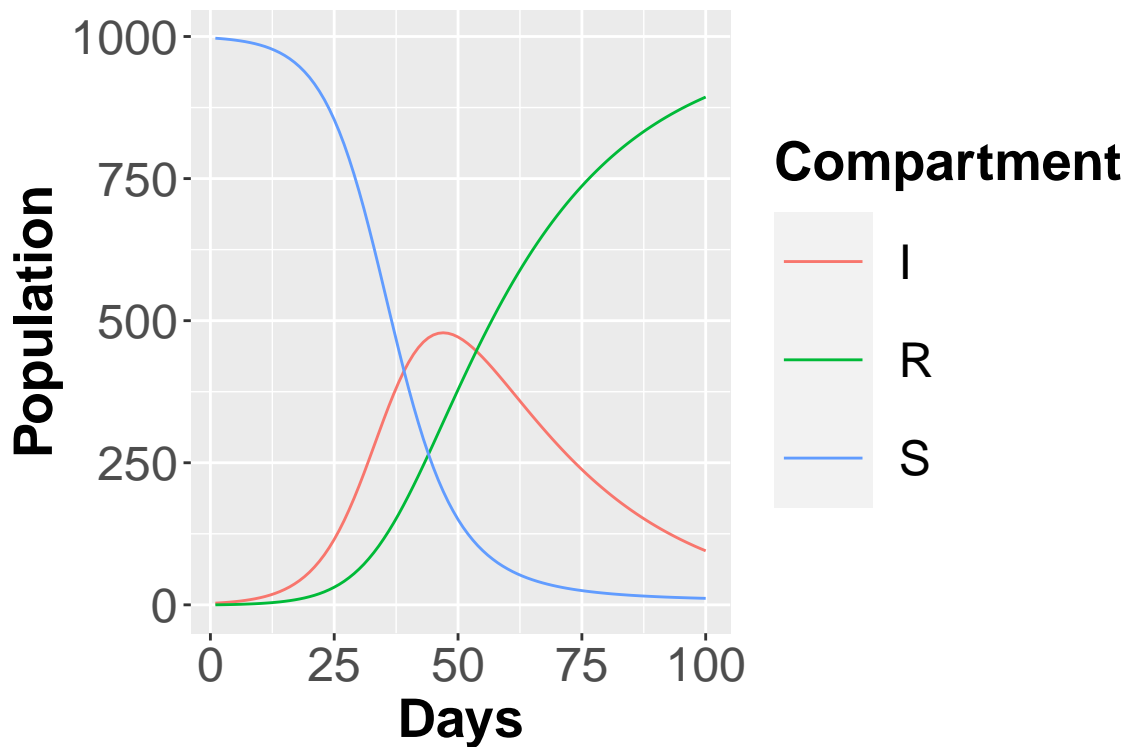

Figure 4: Deterministic Trajectory considering all places

Considering the stochastic simulations, we can obtain the following results.

```
model.analysis(solver_fname = "SIR.solver",
               parameters_fname = "Input/Functions_list_ModelAnalysis.csv",
```

```
            solver_type = "SSA",
            i_time = 1,
            f_time = 100, # days
            s_time = 1,
            n_run = 500,
            parallel_processors = 2)
```

```
## How to generate the plots
source("Rfunction/ModelAnalysisPlot.R")
AnalysisPlot = ModelAnalysisPlot(solverName_path = "./SIR_analysis/SIR-analysis-1.trace",
                                 Stoch = T,
                                 print = F)

AnalysisPlot$plAll
AnalysisPlot$plAllMean
```
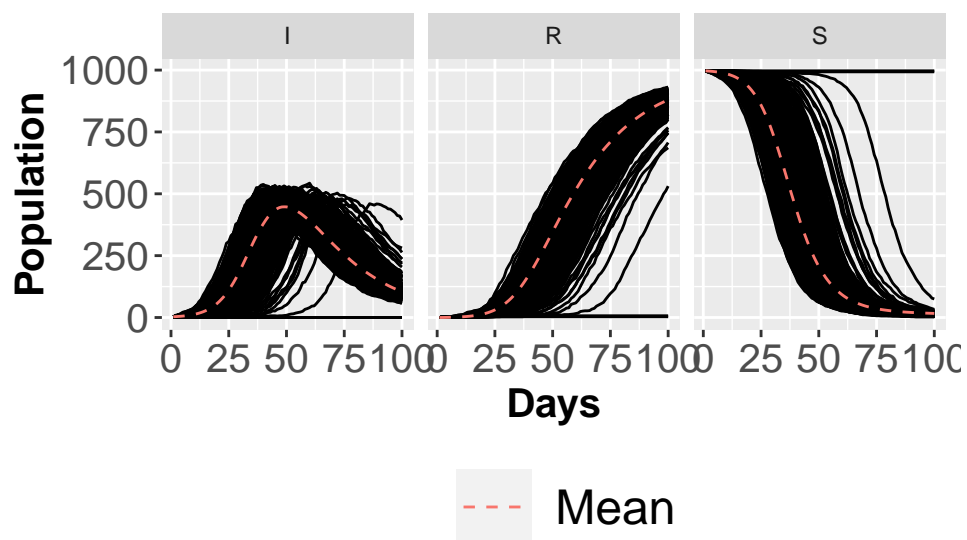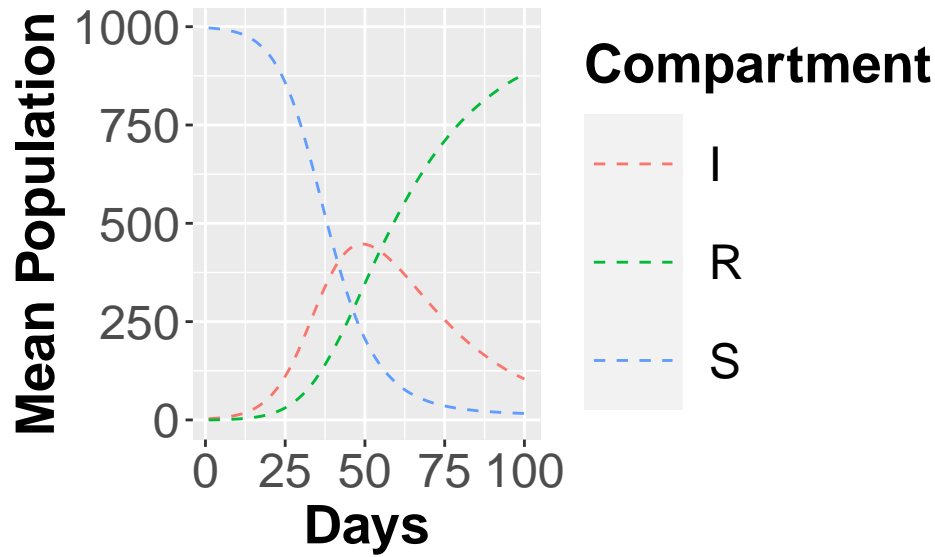

Figure 5: Stochastic trajectories.

Figure 6: Average of the stochastic trajectories.

What-if analysis It is possible to change the place marking directly from the *parameters_fname*:

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**).
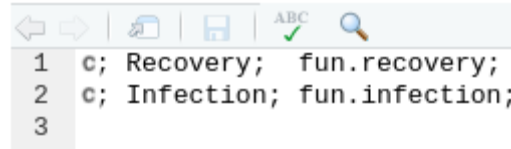
```
model.calibration(solver_fname = "./Net/SIR.solver",
                  parameters_fname = "./Input/Functions_list_Calibration.csv",
                  functions_fname = "./Rfunction/FunctionCalibration.R",
                  reference_data = "./Input/reference_data.csv",
                  distance_measure = "mse",
                  i_time = 1,
                  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)
```

1. **solver_fname**: the *.solver* file generated by the *model.generation* function, that is *SIR.solver*;

2. **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 (*which must separeted using ;*): (1) a tag representing the parameter type: *i* for the complete initial marking (or condition), *m* for the initial marking of a specific place, *c* for a single constant rate, 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 the 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 *m*, *c* 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_Calibration.csv* where the rates of the *Recovery* and *Infection* transitions can be calibrated by using the R functions stored in the R script *functions_fname*:

```
1  c; Recovery;  fun.recovery;
2  c; Infection; fun.infection;
3
```

3. **functions_fname**: an R file storing: 1) the user defined functions to generate instances of the parameters summarized in the *parameters_fname* file, and 2) the function to compute the distance (or error) between the model output and the reference dataset itself. An example is given by *FunctionCalibration.R*, where three functions are implemented: *fun.recovery*, *fun.infection*, and *mse*. The first two are introduced in *Functions_list_Calibration.csv* file, and they are defined in order to return the value (or a linear transformation) of the vector of the unknown parameters generated from the optimization algorithm, namely **optim_v**, whose size is equal to number of parameters in *parameters_fname*. Let us note that the output of these functions must return a value for each input parameter. For instance, to calibrate the transition rates associated with *Recovery* and *Infection*, the functions recovery and infection have to be defined, returning just the corresponding value from the vector **optim_v**, where **optim_v[1]** = *"Recovery rate"*, **optim_v[2]** = *"Infection rate"*, since we do not want to change the vector generated from the optimization algorithm. The order of values in **optim_v** is given by the order of the parameters in *parameters_fname*. Finally, the function *mse* defines the distance measure (based on the squared error distance) between the reference data and the simulations; it takes in input only the reference data (defined in *reference_data.csv*), and the *simulation output* with the following structure:

```
  Time        S         I          R
1    1 997.0000  3.000000 0.0000000
2    2 995.5117  4.347906 0.1404350
3    3 993.3593  6.296759 0.3438917
4    4 990.2522  9.109392 0.6383857
5    5 985.7778 13.158063 1.0640958
6    6 979.3576 18.964074 1.6783310
7    7 970.1924 27.245395 2.5621912
```

Thus, these three functions are defined as follows:

```
fun.recovery <- function(optim_v)
{
    return(optim_v[1])
}

fun.infection <- function(optim_v)
```

```
{
    return(optim_v[2])
}

mse <- function(reference, output)
{
    reference[1,] -> times_ref
    reference[2,] -> infect_ref

    # We will consider the same time points
    Infect <- output[which(output$Time %in% times_ref), "I"]
    infect_ref <- infect_ref[which(times_ref %in% output$Time)]

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

    return(diff.Infect)
}
```

4. **reference_data**: a csv file storing the data to be compared with the simulations' results. 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 columns the corresponding values at a specific time.

5. **distance_measure_fname**: the distance function name to exploit for ranking the simulations, which is implemented in *functions_fname*;
6. **i_time**: the initial solution time, for instance day 1;
7. **f_time**: the final solution time, for instance 10 weeks (70 days);
8. **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;
9. **ini_v**: initial values for the parameters to be optimized.
10. **lb_v, ub_v**: vectors with length equal to the number of parameters which are varying. Lower/Upper bounds for each parameter.
11. **max_time**: maximum running time.

```
# How to generate the plots
source("Rfunction/CalibrationPlot.R")
plots <- calibration.plot(solverName_path = "./SIR_calibration/SIR-calibration-1.trace",
                          reference_path = "reference_data.csv")
plots$plS
plots$plI
plots$plR
```
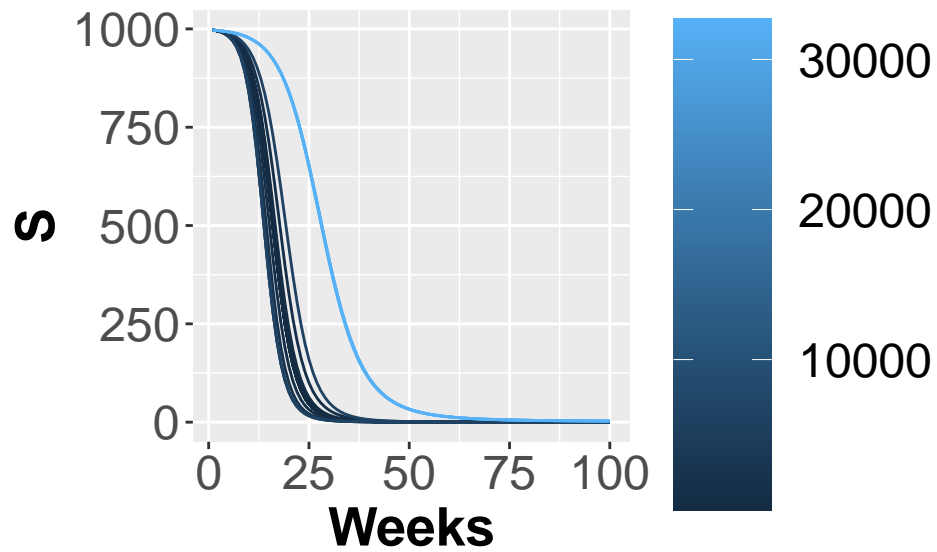

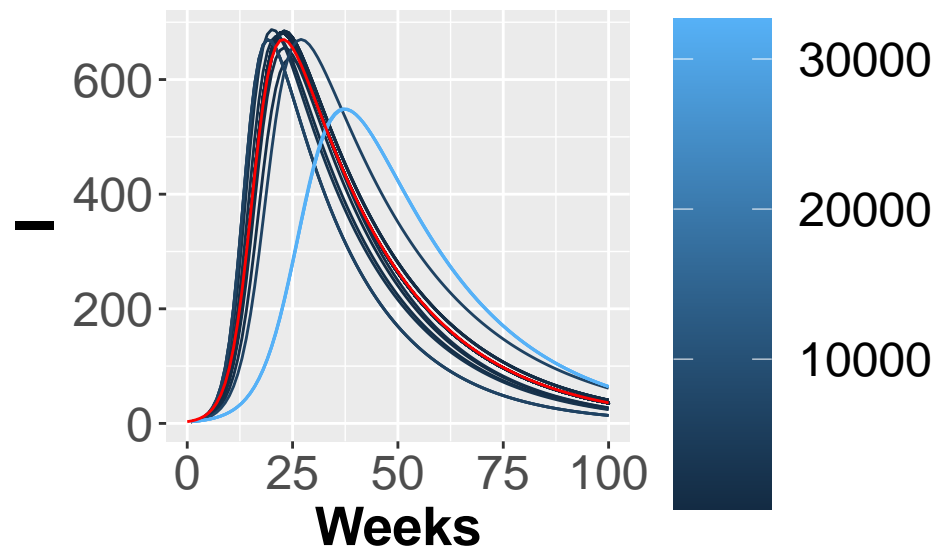Figure 7: Trajectories considering the S place.

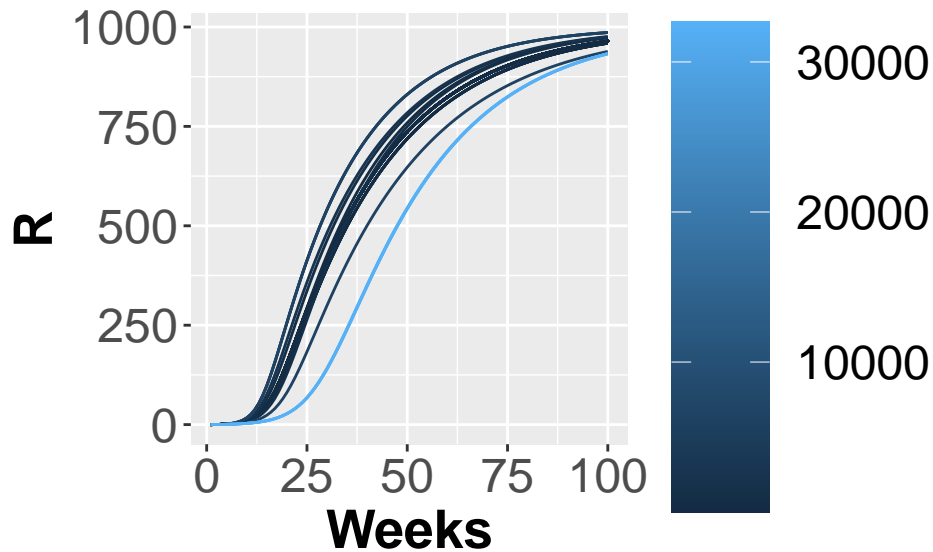Figure 8: Trajectories considering the I place.

Figure 9: Trajectories considering the R place.

In Figures 7, 8 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] 0.0401615154 0.0004227339
```

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): 2580–82.