Tutorial: how to use Rstan to fit a SIR deterministic model

Andrea Bizzotto

8/3/2022

The purpose of this tutorial is to provide advice, in order to fit epidemic data, using the Hamiltonian Monte Carlo method, implemented with package Rstan.

Genetion of the synthetic data

The synthetic data are useful because are obtained from a framework presettled by ourself. So the results that we want to obtain are known. Therefore we need to create the data to be fitted using our statistical model. Let's start with a simple SIR model: Simptomatics, Infected, Removed, and our goal is to obtain a reasonable (posterior) distribution of the parameters involved in the model. We recall that a SIR model is described by the differential equations

$$S' = -\beta \frac{I}{N} SI' = \beta \frac{I}{N} S - \gamma IR' = \gamma I$$

where β is the infectivity and $\frac{1}{\gamma}$ is called recovery rate. In this mode one can prove that the basic reproduction number is the multiplication between the infectivity and the average time spent in the Infected compartment:

$$R_0 = \frac{\beta}{\gamma}$$

Now we do some hypothesis to create our data. We take into account a time period of 100 days, $T = 100$ and a population of 1000 individuals, where the infected at the beginning of the epidemic are only 5. We suppose that at the beginning of the epidemic, one infected generates two infected for day, this means $R_0 = 2$. Finally the time spent in the infected compartment is one week, so that $\gamma = \frac{1}{7}$. The last two hyphothesis imply $\beta = \frac{2}{7}$. Let's start with the coding. The first thing we need is a function, that describes the dynamical system. This function take as input the time, the variables and the parameters.

```
library(deSolve)
library(ggplot2)
library(rstan)
```

```
## Caricamento del pacchetto richiesto: StanHeaders
```

```
## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
library(truncnorm)
```

```
SIR_equations <- function(time, variables, parameters){

  t = time;

  S =  variables[1];
  I =  variables[2];
  R =  variables[3];

  beta = parameters[1];
  gamma = parameters[2];
  N = parameters[3];

  dS_dt = - beta * I/N * S;
  dI_dt =   beta * I/N * S - gamma * I;
  dR_dt =                    gamma * I;

  return(list(c(dS_dt, dI_dt, dR_dt)));
}
```

Now we define the values of the parameters and the initial conditions, baseed on the hyphotesis made before.

```
T <- 100
N <- 1000;
I0 <- 5;
beta_value <- 2/7;
gamma_value <- 1/7;

time_values <- seq(1:T)

initial_values <- c(S = N - I0, I = I0, R = 0);

parameters_values <- c(beta = beta_value, gamma = gamma_value, N = N);
```

It remains only to solve the ODEs system, with the function `ode`.

```
SIR <- ode(
    y = initial_values,
    times = time_values,
    func = SIR_equations,
    parms = parameters_values
          )
```

The variable `ODE` is a matrix with dimension 100×4, where columns represents the time and the three variables (S,I,R), and each row corresponds to the value of the state at the various time. For instance

```
SIR[30,]
```

```
##     time        S        I        R
##  30.0000 665.9546 133.2848 200.7606
```

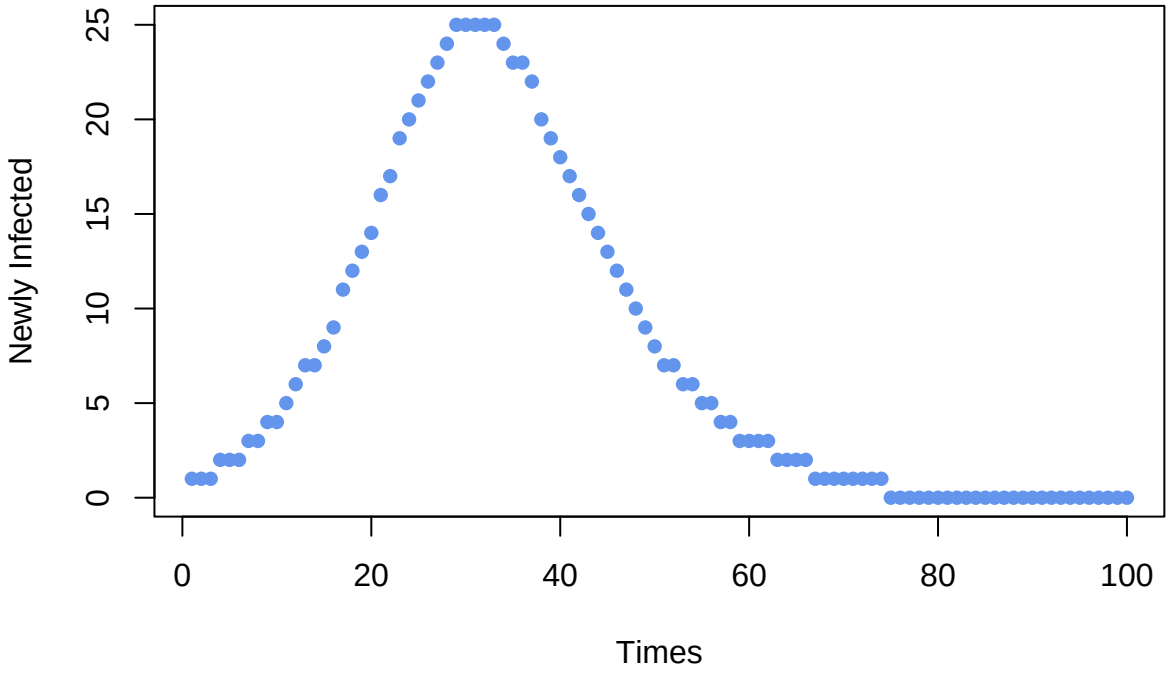
means that at time $t = 10$, there are 665.9546 Susceptibles, 133.2848 Infected and 200.7606 Removed. Note that for each time t we have $N = S(t) + I(t) + R(t)$. Our purpose is to fit the daily incidence of the Infected (Y_{data}), i.e.

$$Y_{data}(t) = \beta \frac{I(t)}{N} S(t)$$

```
Y_data <- beta_value * SIR[,3]/N * SIR[,2]
Y_data <- as.integer(Y_data)
```

We can now visualize the bahaviour of the newly infected per day

```
plot(time_values,Y_data, xlab = "Times", ylab = "Newly Infected", col = "cornflowerblue",pch = 16)
```



Rstan model

In this section we describe the model implementation, with the Rstan language. We need to create a `file.stan` (File -> New File -> Stan file). We call this file `Model.stan`.

```
# functions{
#   real[] SIR_equations(real t,
#               real[] y,
#               real[] theta,
#               real[] x_r,
#               int[] x_i)
# {
# real S;
# real I;
# real R;
#
# int N;
#
# real dS_dt;
# real dI_dt;
```

```
# real dR_dt;
#
# real beta;
# real gamma;
#
# int t_seed;
#
# S = y[1];
# I = y[2];
# R = y[3];
#
# N = x_i[1];
#
# beta = theta[1];
# gamma = theta[2];
#
#
#   dS_dt = - beta * I/N * S             ;
#   dI_dt =   beta * I/N * S - gamma * I ;
#   dR_dt =                    gamma * I ;
#
# return{dS_dt,   dI_dt,   dR_dt};
# }
# }
#
# data {
#        int  time_simulation;
#        int  number_data;
#        real initial_time;
#        real time_sequence[time_simulation];
#        int  N;
#
#        real rel_tol;
#        real abs_tol;
#        real max_num_steps;
#
#        int Y_data[number_data];
# }
#
# transformed data {
#
#     real x_r[0] ;
#     int x_i[1] = {N};
#
# }
#
# parameters {
#
# real<lower=0, upper=1> beta;
# real<lower=0, upper=1> gamma;
#
# real<lower=1, upper=10> I0;
#
```

```
# real<lower=0> d_inv;
#   }
#
# transformed parameters{
#
#   real SIR_ode[time_simulation, 3];
#   real Initial_Conditions[3];
#   real theta[2];
#
#   real  I_incidence[time_simulation];
#
#   real  d;
#
#   Initial_Conditions = {N-I0, I0, 0};
#
#   theta[1] = beta;
#   theta[2] = gamma;
#
#   SIR_ode = integrate_ode_rk45(SIR_equations, Initial_Conditions, initial_time, time_sequence, theta,
#
#  for (i in 1 : time_simulation){
#
#     I_incidence[i]   = beta * SIR_ode[i,2]/N * SIR_ode[i,1];
#                                }
#     d = 1/d_inv;
# }
#
# model {
#    beta ~ normal(0.3, 0.1);
#    gamma ~ normal(0.15,0.1);
#
#    I0 ~ normal(2, 10);
#    d_inv ~ exponential(5);
#
#      for (i in 1 : number_data){
#      target += neg_binomial_2_lpmf(Y_data[i]   | I_incidence[i] + 1e-5, d);}
#
# }
#
# generated quantities {
# real prediction_SIR[time_simulation];
# real R0;
#
# for (i in 1 : time_simulation){
#
# prediction_SIR[i]  =  neg_binomial_2_rng(I_incidence[i]+1e-5,d);}
# R0 = beta/gamma;
# }
```

the execution of the fit

```
rstan_options (auto_write = TRUE)
options (mc.cores = parallel::detectCores ())
```

```
n_data <- length(Y_data)

time_sequence <- 1: (T+20)

data_model <- list(time_simulation = T + 20,
                   number_data = n_data,
                   initial_time = 0,
                   time_sequence = time_sequence,
                   N = N,
                   Y_data =  Y_data,
                   rel_tol = 1e-3,
                   abs_tol = 1e-3,
                   max_num_steps= 2000
                   )

init = function(){
 list( beta =  rtruncnorm(1, a=0, b=1,   mean=0.3,  sd=0.1),
       gamma = rtruncnorm(1, a=0, b=1,   mean=0.15, sd=0.1),
       I0 =    rtruncnorm(1, a=1, b=10,  mean=2,    sd=0.1),
       d_inv = rtruncnorm(1, a=0, b=1,   mean=0.5,  sd=1)
       )}

model <- stan_model("Model.stan")


# control <- list(adapt_engaged = TRUE,
#                 adapt_gamma = 0.05,
#                 adapt_delta = 0.6,
#                 adapt_kappa = 0.75,
#                 adapt_t0 = 10,
#                 adapt_init_buffer = 75,
#                 adapt_term_buffer = 50,
#                 adapt_window = 25
#                 )

fit <- sampling(model,
                init = init,
                data = data_model,
                iter = 1500,
                chains = 4,
                save_warmup = TRUE,
                refresh = 200)
```

Now we want to check how the fitting has been executed, and how it is good. Let's see if we arrived to convergence, basing on the two statistics R_{hat} and n_{eff}. R_{hat} represents the variance between the chain of each parameter, n_{eff} instead describe the effort in the sample. For a good convergence we should have $R_{hat} \approx 1$, at most $R_{hat} = 1.10$ and $n_{eff} \geq 1000$.

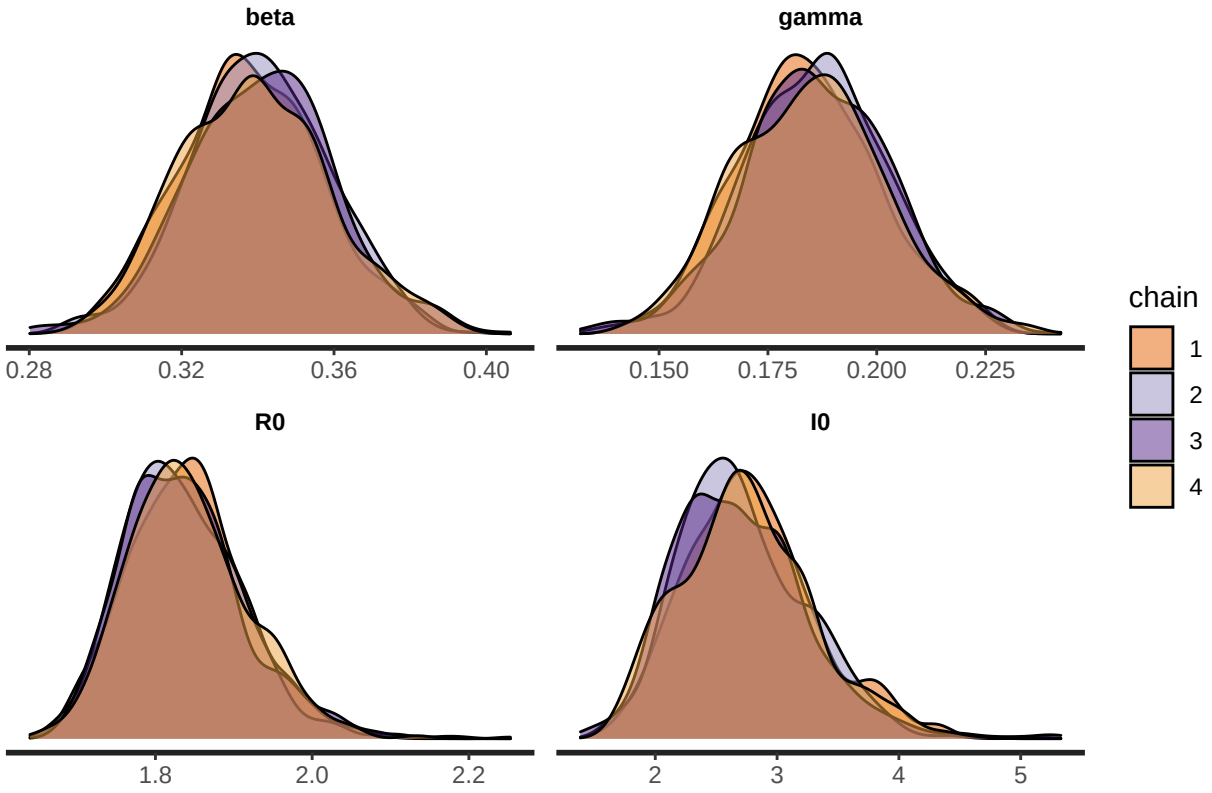
```
parameters <- c("beta", "gamma", "R0", "I0")

print(fit, pars = parameters)
```

```
## Inference for Stan model: Model.
## 4 chains, each with iter=1500; warmup=750; thin=1;
```

```
## post-warmup draws per chain=750, total post-warmup draws=3000.
##
##       mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## beta  0.34    0.00 0.02 0.30 0.33 0.34 0.35  0.38   746    1
## gamma 0.19    0.00 0.02 0.15 0.17 0.19 0.20  0.22   815    1
## R0    1.84    0.00 0.08 1.71 1.78 1.83 1.88  2.01  1114    1
## I0    2.74    0.02 0.54 1.85 2.35 2.69 3.06  3.93   956    1
##
## Samples were drawn using NUTS(diag_e) at Fri May 20 11:18:20 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

We can note, that we have a very good convergence of the method, as we can see the well mixed graph of the posteriors.

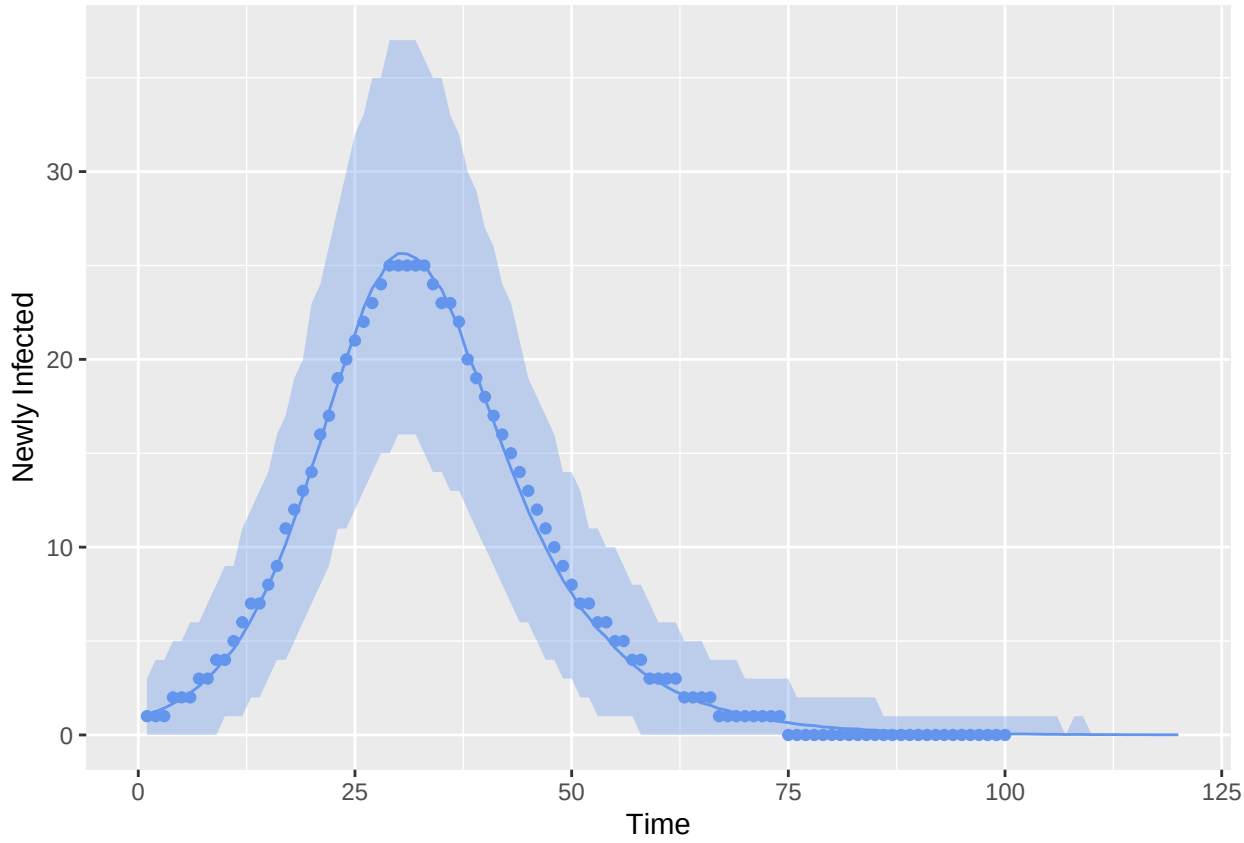
```
stan_dens(fit, pars = parameters, separate_chains = TRUE)
```



However the parameters are estimated differently, from the known values used to create the synthetic data. Let's try to analyze the simulated data

```
pred_Y <- cbind(as.data.frame(summary(fit, pars="prediction_SIR",
            probs=c(0.025, 0.50, 0.975))$summary), 1:data_model$time_simulation)
colnames(pred_Y) <- make.names(colnames(pred_Y))
Y_data <- data.frame(Y_data)

p = ggplot() +
   geom_line(data = pred_Y, aes(x = 1:data_model$time_simulation, y = mean), color = "cornflowerblue") +
```

```
  geom_ribbon(data = pred_Y,aes(ymin = X2.5., ymax = X97.5.,
              y = mean, xmin = 0, xmax = data_model$time_simulation,
        x = 1:data_model$time_simulation ), fill = "cornflowerblue", alpha = 0.35)+
geom_point(data = Y_data, aes(x = 1:n_data, y = Y_data), color = "cornflowerblue")+
  xlab('Time') +
  ylab('Newly Infected')
plot(p)
```



Age-structured SIR model

Now we want to extend the previous model, considering two different age-groups with a fictitious contact matrix, which represents the rate of contact per unit time between the various groups. For instance we can divide the population in the individuals with maximum age 50 (Group 1) and the complementary (Group 2). It is reasonable to assume that in the younger group there are more individuals:

$$N_1 = 750; \qquad N_2 = 250$$

and more contacts:

$$C = \begin{pmatrix} 4 & 2 \\ 6 & 5 \end{pmatrix}$$

We note that the matrix satisfies the equation

$$N_i \cdot C_{ij} \approx N_j \cdot C_{ji}$$

In this case the force of the infection is defined as

$$\lambda_i = \beta \cdot \sum_{j=i} C_{ij} \cdot \frac{I_j}{N_j}$$

So the system composed by the six differential equation with $i = 1, 2$ is

$$S_i' = -\lambda_i \frac{I}{N} S I_i' = \lambda_i \frac{I}{N} S - \gamma_i I R_i' = \gamma_i I$$

Where we supposed two different recovery rates depending on the age group, for instance $\frac{1}{\gamma_1} = 7$ and $\frac{1}{\gamma_2} = 10$. In this model the basic reproduction number is given by the radius of the Next Generation Matrix (K).

$$K = \beta \cdot \begin{pmatrix} \frac{1}{\gamma_1} \cdot C_{11} & \frac{1}{\gamma_2} \cdot C_{12} \cdot \frac{N_1}{N_2} \\ \frac{1}{\gamma_1} \cdot C_{21} \cdot \frac{N_2}{N_1} & \frac{1}{\gamma_2} \cdot C_{22} \end{pmatrix}$$

To be more consistent now we briefly explain which is the classical procedure to build the next generation matrix. 0) Define an infectious disease equilibrium, i.e. a state in which there are no infectious individuals. $E = (S_1 = N_1, I_1 = 0, R_1 = 0, S_2 = N_2, I_2 = 0, R_2 = 0)$. 1) Choose what means "infected individuals" and the corresponding compartments. In our model there no possibility to be wrong in the choice of "infected individuals" and the corresponding compartments: I_1 and I_2. 2) Define the function $\mathbf{f}$, in which each element represents the rate at which new infected enter in the compartments chosen before. $\mathbf{f} = (\lambda_1 \cdot S_1,\ \lambda_2 \cdot S_2)$. 3) Compute the gradient of this function respect to the variables that represent the infected compartments, and evaluate it in the disease free equilibrium.

$$F = \beta \cdot \begin{pmatrix} C_{11} & C_{12} \cdot \frac{N_1}{N_2} \\ C_{21} \cdot \frac{N_2}{N_1} & C_{22} \end{pmatrix}$$

4) Define the function $\mathbf{v}$, in which each element represents the difference between the rates of transfer of individuals out and into each compartment, avoiding the new infected. $\mathbf{v} = (\gamma_1 \cdot I_1,\ \gamma_2 \cdot I_2)$. 5) Compute the gradient of this function respect to the variables that represent the infected compartments, and evaluate it in the disease free equilibrium.

$$V = \begin{pmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \end{pmatrix}$$

6) compute the inverse of the last matrix.

$$V^{-1} = \begin{pmatrix} \frac{1}{\gamma_1} & 0 \\ 0 & \frac{1}{\gamma_2} \end{pmatrix}$$

7) Compute the next generation matrix as $K = F \cdot V^{-1}$

$$K = \beta \cdot \begin{pmatrix} \frac{1}{\gamma_1} \cdot C_{11} & \frac{1}{\gamma_2} \cdot C_{12} \cdot \frac{N_1}{N_2} \\ \frac{1}{\gamma_1} \cdot C_{21} \cdot \frac{N_2}{N_1} & \frac{1}{\gamma_2} \cdot C_{22} \end{pmatrix}$$

In our specific case the characteristic polynomial of this matrix is given by

$$p_K(x) = x^2 - x \cdot trace(K) + det(K)$$

then the two eigenvalues are

$$x_1 = \frac{trace(K) + \sqrt{[trace(K)]^2 - 4 \cdot det(K)}}{2} \qquad x_2 = \frac{trace(K) - \sqrt{[trace(K)]^2 - 4 \cdot det(K)}}{2}$$

Insering the previour values

$$trace(K) = \beta \cdot (\frac{1}{\gamma_1} \cdot C_{11} + \frac{1}{\gamma_2} \cdot C_{22}) = \beta \cdot 78$$

$$det(K) = \beta^2 \cdot \frac{1}{\gamma_1} \cdot \frac{1}{\gamma_2} \cdot (C_{11} \cdot C_{22} - C_{21} \cdot C_{12}) = \beta^2 \cdot 560$$

Then

$$x_1 = \frac{\beta \cdot 78 + \sqrt{[\beta \cdot 78]^2 - 4 \cdot \beta^2 \cdot 560}}{2} = 70 \cdot \beta \qquad x_2 = \frac{\beta \cdot 78 - \sqrt{[\beta \cdot 78]^2 - 4 \cdot \beta^2 \cdot 560}}{2} = 8 \cdot \beta$$

Since $\beta > 0$ we have the following formula for the basic reproduction number

$$R_0 = 70 \cdot \beta$$

We suppose another time $R_0 = 2$, then we set $\beta = \frac{1}{35}$. Finally as initial values we suppose $I_1(0) = 3$ and $I_2(0) = 2$. Now we are ready to create our synthetic data. First of all we define the equations.

```
SIR_age_equations <- function(time, variables, parameters){

  t = time;

  S1 =  variables[1];
  I1 =  variables[2];
  R1 =  variables[3];
  S2 =  variables[4];
  I2 =  variables[5];
  R2 =  variables[6];

  beta = parameters[1];
  gamma1 = parameters[2];
  gamma2 = parameters[3];
  N1 = parameters[4];
  N2 = parameters[5];
  C11 = parameters[6];
  C12 = parameters[7];
  C21 = parameters[8];
  C22 = parameters[9];

  lambda1 = beta * (C11 * I1 / N1 + C12 * I2 / N2);
  lambda2 = beta * (C21 * I1 / N1 + C22 * I2 / N2);

  dS1_dt = - lambda1 * S1;
  dI1_dt =   lambda1 * S1 - gamma1 * I1;
  dR1_dt =                          gamma1 * I1;
  dS2_dt = - lambda2 * S2;
  dI2_dt =   lambda2 * S2 - gamma2 * I2;
  dR2_dt =                          gamma2 * I2;

  return(list(c(dS1_dt, dI1_dt, dR1_dt,dS2_dt, dI2_dt, dR2_dt)));
}
```

and the initial values

```
N1 <- 750;
N2 <- 250;
I10 <- 3;
I20 <- 2;

beta_value <- 1/35;
```

```
gamma1_value <- 1/7;
gamma2_value <- 1/10;
C11 <- 4
C12 <- 2
C21 <- 6
C22 <- 5

initial_values <- c(S1 = N1 - I10, I1 = I10, R1 = 0,
                    S2 = N2 - I20, I2 = I20, R2 = 0);

parameters_values <- c(beta = beta_value, gamma1 = gamma1_value,
                       gamma2 = gamma2_value, N1 = N1, N2 = N2,
                       C11 = C11, C12 = C12, C21 = C21, C22 = C22);
```

We solve the equations

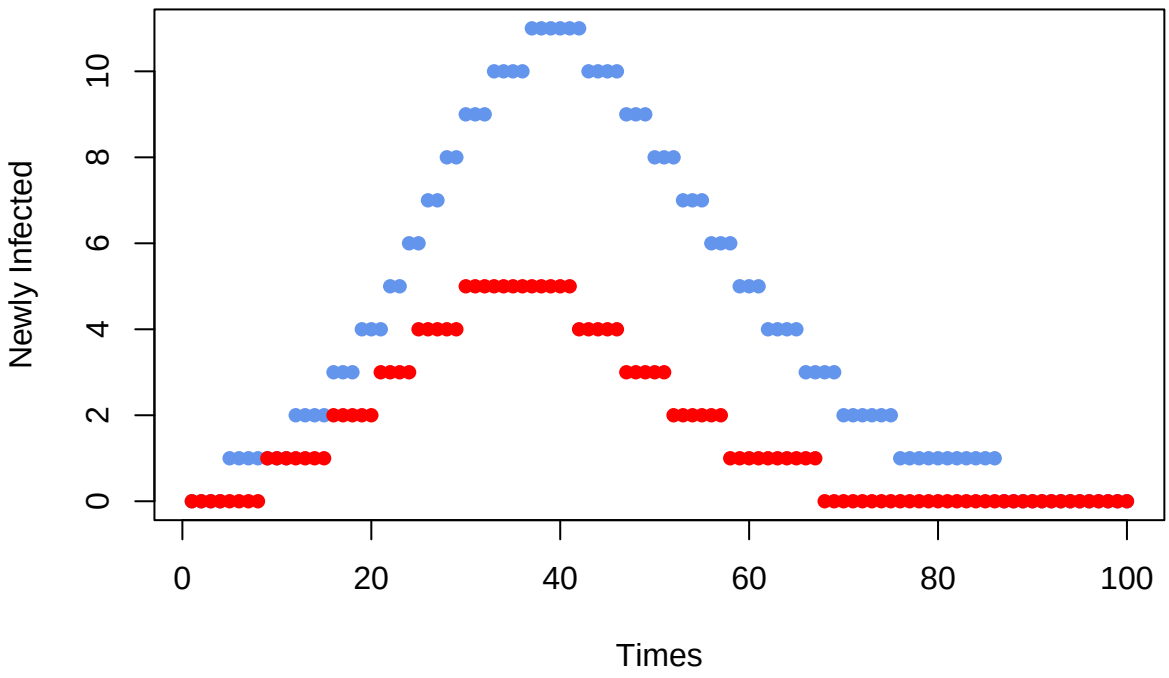
```
SIR_age <- ode(
    y = initial_values,
    times = time_values,
    func = SIR_age_equations,
    parms = parameters_values
          )
```

Now we compute the incidence per day.

```
lambda1 = beta_value * (C11 * SIR_age[,3]/N1 + C12 * SIR_age[,6]/N2);
lambda2 = beta_value * (C21 * SIR_age[,3]/N1 + C22 * SIR_age[,6]/N2);

Y1_data <- as.integer(lambda1 * SIR_age[,2])
Y2_data <- as.integer(lambda2 * SIR_age[,5])
```

We can observe the two trajectories

```
plot(time_values,Y1_data, xlab = "Times", ylab = "Newly Infected",col = "cornflowerblue",pch=16)+
points(time_values,Y2_data, col = "red", pch = 16)
```




```
## integer(0)
```

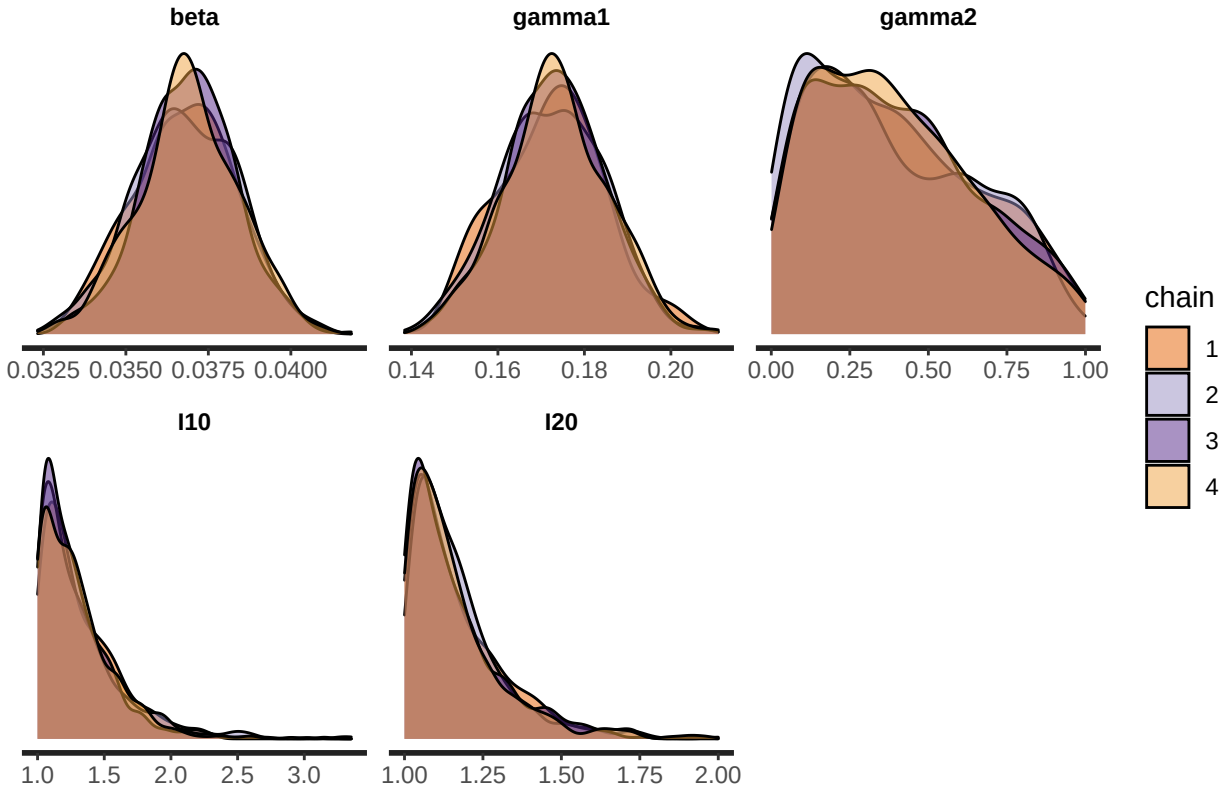
```
n_data <- length(Y1_data)

data_model <- list(time_simulation = 120,
                   number_data = n_data,
                   initial_time = 0,
                   time_sequence = 1:120,
                   N1 = N1,
                   N2 = N2,
                   contact_matrix = matrix(c(C11,C21,C12,C22),nrow=2,ncol=2),
                   Y1_data =  Y1_data,
                   Y2_data = Y2_data,
                   rel_tol = 1e-4,
                   abs_tol = 1e-4,
                   max_num_steps= 2000
                   )

init = function(){
 list( beta =   rtruncnorm(1, a=0, b=1,   mean=0.01,  sd=0.01),
       gamma1 = rtruncnorm(1, a=0, b=1,   mean=0.15, sd=0.1),
       gamma2 = rtruncnorm(1, a=0, b=1,   mean=0.15, sd=0.1),
       I10 =    rtruncnorm(1, a=1, b=10,  mean=2,    sd=0.1),
       I20 =    rtruncnorm(1, a=1, b=10,  mean=2,    sd=0.1),
       d_inv =  rtruncnorm(1, a=0, b=1,   mean=0.5,  sd=1)
       )}
```

```
model <- stan_model("Model_age.stan")

fit <- sampling(model,
                init = init,
                data = data_model,
                iter = 1000,
                chains = 4,
                save_warmup = TRUE,
                refresh = 50)
```

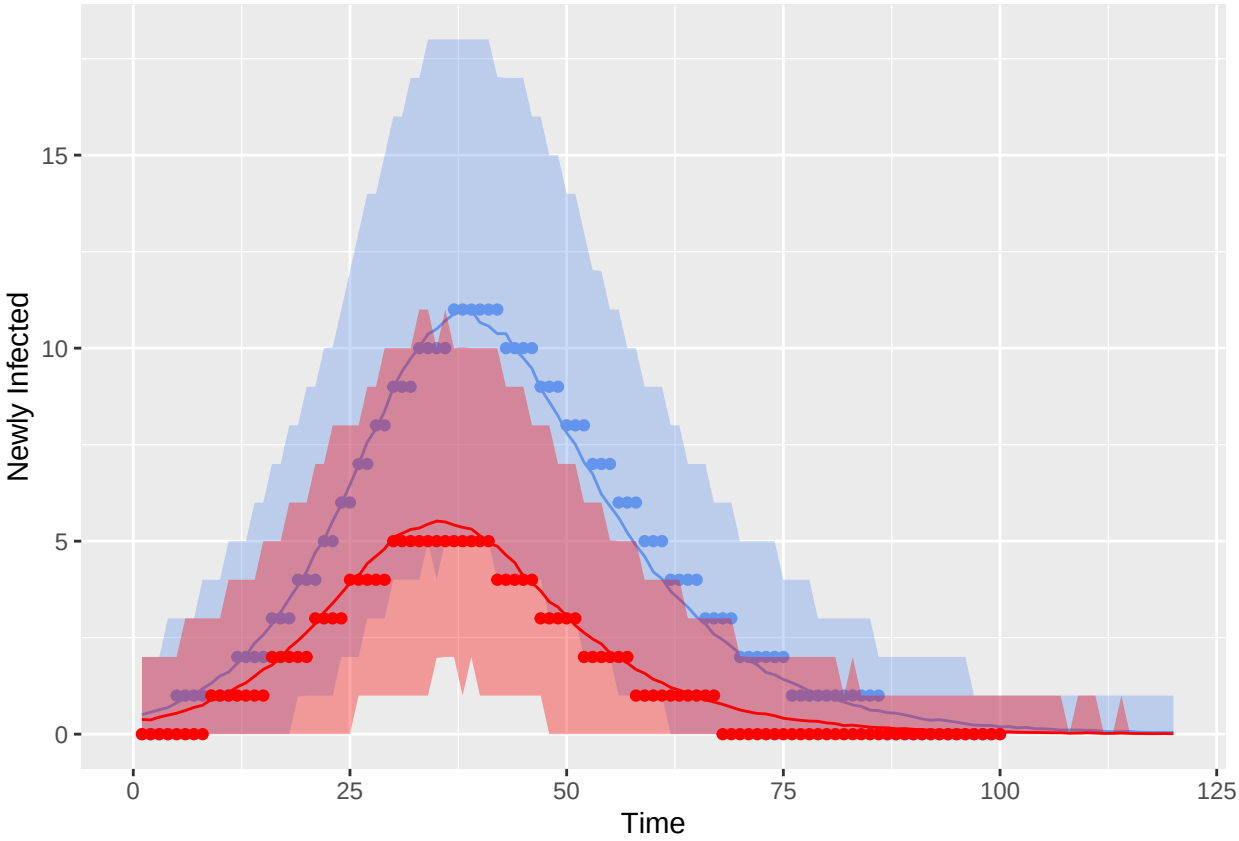
```
parameters <- c("beta", "gamma1", "gamma2", "I10", "I20")

print(fit, pars = parameters)
```

```
## Inference for Stan model: Model_age.
## 4 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=2000.
##
##        mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## beta   0.04    0.00 0.00 0.03 0.04 0.04 0.04  0.04   986 1.00
## gamma1 0.17    0.00 0.01 0.15 0.16 0.17 0.18  0.20  1092 1.00
## gamma2 0.39    0.01 0.26 0.02 0.17 0.35 0.58  0.91  1871 1.00
## I10    1.30    0.01 0.29 1.01 1.08 1.22 1.43  2.08   947 1.01
## I20    1.16    0.00 0.15 1.01 1.05 1.12 1.22  1.58  1523 1.00
##
## Samples were drawn using NUTS(diag_e) at Fri May 20 11:21:40 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
stan_dens(fit, pars = parameters, separate_chains = TRUE)
```




```
pred_Y1 <- cbind(as.data.frame(summary(fit, pars="prediction1_SIR",
            probs=c(0.025, 0.50, 0.975))$summary), 1:data_model$time_simulation)
colnames(pred_Y1) <- make.names(colnames(pred_Y1))
pred_Y2 <- cbind(as.data.frame(summary(fit, pars="prediction2_SIR",
            probs=c(0.025, 0.50, 0.975))$summary), 1:data_model$time_simulation)
colnames(pred_Y2) <- make.names(colnames(pred_Y2))

Y1_data <- data.frame(Y1_data)
Y2_data <- data.frame(Y2_data)

p = ggplot() +
  geom_line(data = pred_Y1, aes(x = 1:data_model$time_simulation, y = mean), color = "cornflowerblue") +
  geom_ribbon(data = pred_Y1,aes(ymin = X2.5., ymax = X97.5.,
              y = mean, xmin = 0, xmax = data_model$time_simulation,
        x = 1:data_model$time_simulation ), fill = "cornflowerblue", alpha = 0.35)+
  geom_point(data = Y1_data, aes(x = 1:n_data, y = Y1_data), color = "cornflowerblue")+
  geom_line(data = pred_Y2, aes(x = 1:data_model$time_simulation, y = mean), color = "red") +
  geom_ribbon(data = pred_Y2,aes(ymin = X2.5., ymax = X97.5.,
              y = mean, xmin = 0, xmax = data_model$time_simulation,
        x = 1:data_model$time_simulation ), fill = "red", alpha = 0.35)+
  geom_point(data = Y2_data, aes(x = 1:n_data, y = Y2_data), color = "red")+
  xlab('Time') +
  ylab('Newly Infected')
plot(p)
```



General advice

How to be precise and efficent in the ode solver

We can set the absolute and relative tolerances. The default tolerance are: . Increasing the tolerance your benefit is that you reduce the time of execution of the algorithm. However you have to be careful, because high tolerances brings to results affected by numerical errors, so when the method has to compute the gradient of the posterior distribution, used in the hamiltonian equations, it commits errors that don't permit to obtain good estimates and fitting.

Why and how to choose initial values (of the parameters)

By default the algorithm choose initial values of the parameters between -2 and +2. However some values brings to incidence very close to zero, but negative. This means that you could obtain errors due to the fact the a negative (or null) mean for the negative binomial is not accepted. The advice is to define a function able to extract random values, from a distribution in a specific interval. Moreover since the function extracts random values, the algorithm starts from random initial points, and it helps to detect the modals in a multimodal distribution