# Model 2: Basic SIR Model

Katie Montovan & Aaron King

2024-12-22

This document contains an example of a modified model disease spread. We structure this file more in the way you might structure a finished modeling file. We have excluded the step by step instructions and instead focus more on the global perspective of the process. We assume that you have already read the chapter and worked through the "Basic SI" r code and understand it reasonably well. We are still using the same simulated data so that we can compare the performance of this model with the fit of the basic SI model. The model we will implement in a basic SIR model and in presented in more detail in the book chapter.

#General Setup These are things that you usually won't need to change as you make different version of your model for the same dataset.

## Load in needed libraries

```
options(
  tidyverse.quiet=TRUE,
  pomp_archive_dir="results",
  stringsAsFactors=FALSE
)
library(tidyverse)
library(pomp)
stopifnot(packageVersion("pomp")>="6.0.4")
library(doParallel)
library(doRNG)
library(ggplot2)
library(dplyr)
```

## Setup for saving files

```
library(rstudioapi)
# Get the path to this file
current_path = rstudioapi::getActiveDocumentContext()$path
#Set the working directory to this location
setwd(dirname(current_path ))
```

## Setup a plotting function

We create a function that helps with plotting the pairs fit parameters. This function helps automate some pieces of the plotting that can get cumbersome otherwise.

```
plot_pairs <- function (dataset, fit_params) {
  paste0(
    "pairs(~loglik + ",
    paste0(
      names(fit_params),
      collapse=" + "
    ),
    ",data=dataset,pch=16)"
  ) |>
    str2expression() |>
    eval()
}
```

### Import data

```
library(readr) #this library is only used to load the csv file.
simuldata <-read_csv("https://raw.githubusercontent.com/kmontovan/pomp/refs/heads/main/SimulatedPompitis
```

## Model setup

These are the things you may need to adjust for each new model, or, when making a major change to your model. It can be helpful to have this information all in one place as it makes changing the model a bit easier.

```
fixed_params = c(N=1000, #population size
                 phi = 0.004 #number of infections at time 0
)
fit_params = c(Beta=0.0024, # rate of spread of illness - per year
               gamma=0.1, #recovery rate
               r = 0.01, #rate of individuals loosing immunity
               rho=0.8, #fraction of new infections that are recorded
               k=5 ) #changes the variability of reporting accuracy

partrans <- parameter_trans(log=c("k"), #stay positive
                            logit=c("r","gamma","Beta","rho")) #between 0 and 1

savefile <- "SIR_Model_results.csv" #should be different than for other models
```

```
sir_step <- Csnippet("
  double infections = rbinom(S,1-exp(-Beta*I*dt));
  double recoveries = rbinom(I,1-exp(-gamma*dt));
  double loseimmunity = rbinom(R,1-exp(-r*dt));
  S += -infections + loseimmunity;
  I += infections - recoveries;
  R += recoveries - loseimmunity;
  NewI += infections;
")

sir_rinit <- Csnippet("
  S = nearbyint((1-phi)*N);
  I = nearbyint(phi*N);
  R = 0;
```

```
  NewI = 0;
")

si_dmeas <- Csnippet("
  lik = dnbinom_mu(NewCases,k,rho*NewI,give_log);
")

si_rmeas <- Csnippet("
  NewCases = rnbinom_mu(k,rho*NewI);
")
```

```
SIR_Model <-
  pomp(
    data=simuldata, #the timeseries data
    times="day", #timesteps in the simuldata dataset
    t0=0, #the time to start the simulations.
    rprocess=euler(
      sir_step,
      delta.t=1/4 # steps of <= 6hr
    ),
    rinit=sir_rinit,
    rmeasure=si_rmeas,
    dmeasure=si_dmeas,
    params=c(fixed_params,fit_params), #pulls parameters defined above
    accumvars="NewI",  #This state variable resets to 0 at each observation in simuldata
    statenames=c("S","I","R","NewI"), #The names of all state variables being modeled

    #param names are pulled automatically to make it easier to change your model and not create errors
    paramnames=c(names(fixed_params), names(fit_params))
  )
```

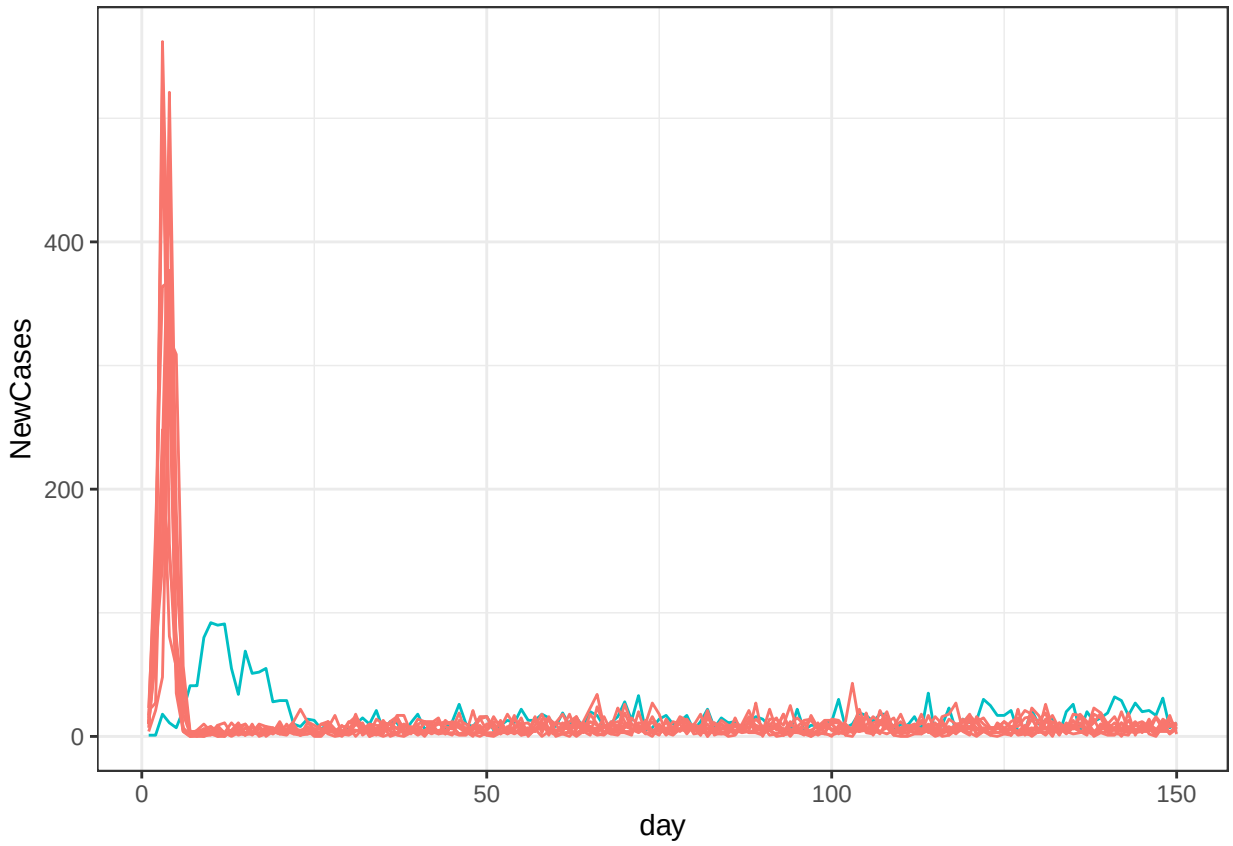
## Optional: Check that simulations and loglikelihoods run

These pieces can be omitted from your final code but are often useful to getting a new model working.

Run simulations for a parameter set. Recall: The data is shown in teal and the simulation results are shows in red.

```
SIR_Model |>
  simulate(
    params=c(N=1000, #population size
             phi = 0.004, #number of infections at time 0
             Beta=0.0024, # rate of spread of illness - per year
             gamma=0.1, #recovery rate
             r = 0.01, #rate of individuals loosing immunity
             rho=0.8, #fraction of new infections that are recorded
             k=5 ), #changes the variability of reporting accuracy
    nsim=10,
    format="data.frame",
    include.data=TRUE
  ) -> sims

sims |>
  ggplot(aes(x=day,y=NewCases,group=.id,color=.id=="data"))+
```

```
  geom_line()+
  guides(color="none")+
  theme_bw()
```




Estimate the log likelihood to make sure that the rmeasure and dmeasure functions are working right.

```
pf <- SIR_Model |>
  pfilter(Np=500)
logLik(pf)
```

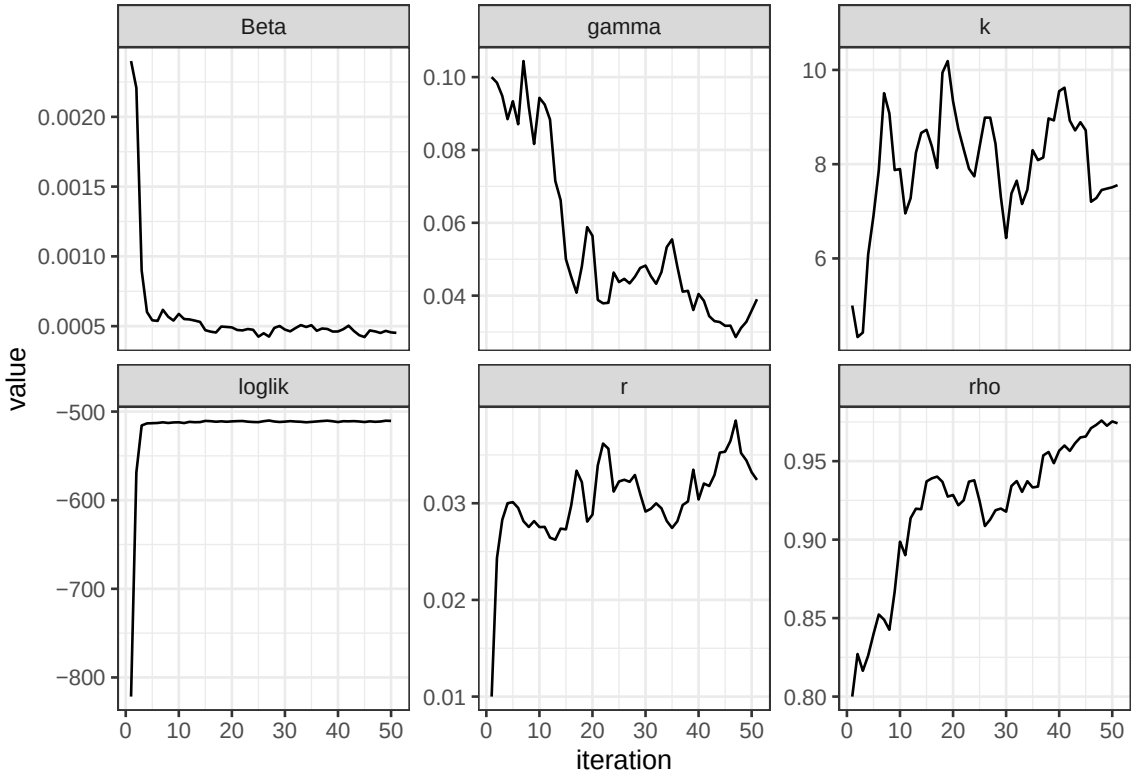
```
## [1] -931.8858
```

## Create mifs_local

Run the local search and save the options. This sets things up for the next step and makes sure things are working.

```
mifs_local <-
  SIR_Model |>
  mif2(
    params=c(fixed_params,fit_params),
    Np=1000,
    Nmif=50, #number of iterations to perform in the local search
    cooling.fraction.50=0.5,
    partrans=parameter_trans(
      log=c("Beta","gamma","k","r"), #parameters to fit that are always positive
      logit=c("rho") #parameters to fit that are always between 0 and 1
```

```
    ),
    paramnames=c("Beta","gamma","r","k","rho"), #names of all parameters to fit
    rw.sd=rw_sd(
      Beta=0.02,
      gamma=0.02,
      r=0.02,
      rho=0.02,
      k=0.02
    )
  )

#plot
mifs_local |>
  traces(c("Beta","gamma","r","rho","k","loglik")) |>
  melt() |>
  ggplot(aes(x=iteration,y=value))+
  geom_line()+
  facet_wrap(~name,scales="free_y")+
  theme_bw()
```




local-1.pdf

## Global Search

Start at a lot of different places in parameter space and search for nearby parameter sets that maximize the log likelihood. The `runif_design` creates a sequence of each parameter spanning the range from the lower to the upper bound and giving `nseq` parameter set "guesses''.

```
guesses <-
  runif_design(
```

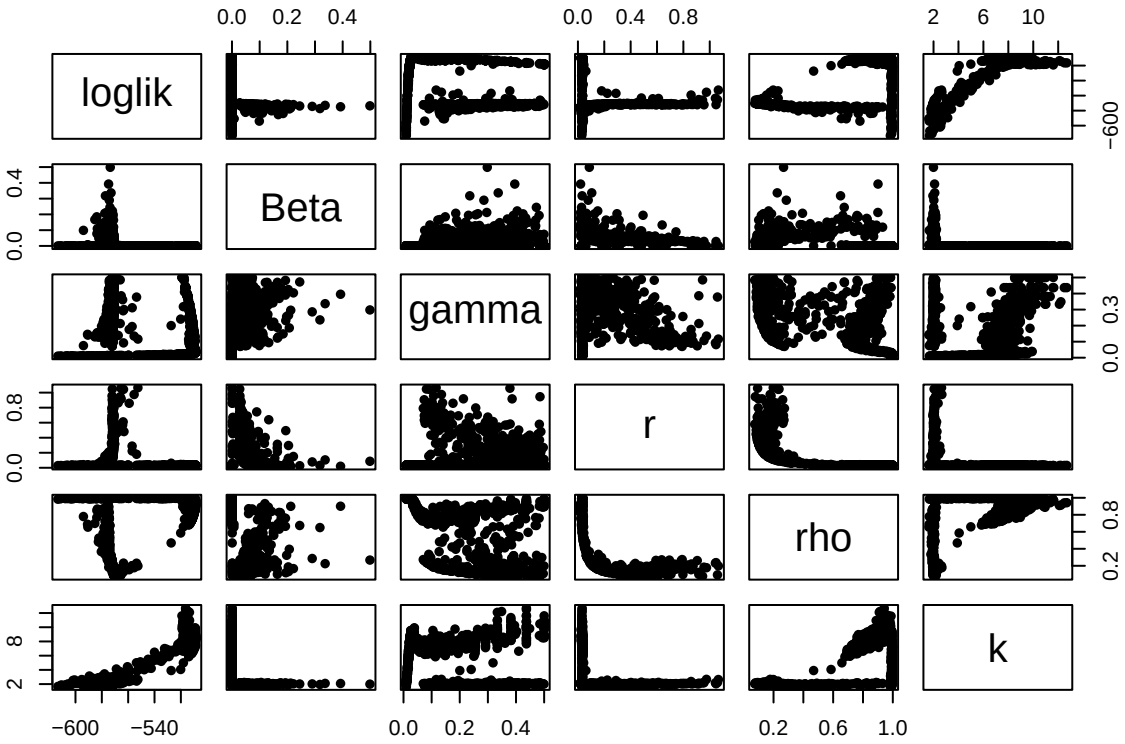
```
    lower=c(Beta=0.0001,gamma=0.07,r=0.001, rho=0.1,k=1,N=1000,phi=0.004),
    upper=c(Beta=0.05,gamma=0.5,rho=0.9,r=0.05,k=10,N=1000,phi=0.004),
    nseq=400
  )
```

```
foreach(
    guess=iter(guesses,"row"),
    .combine=bind_rows
  ) %dopar% {
      #for parallel processing it is necessary to include library(pomp) here
      library(pomp)
      mifs_local |>
        ## first iterated filtering mif2 run: slow cooling
        ## to find general area of better parameter combinations
        mif2(
          params=guess,
          cooling.fraction.50=0.5,
          Nmif=50,
          rw.sd=rw_sd(
            Beta=0.02,
            rho=0.02,
            r=0.02,
            k=0.02
          )
        ) |>
        ## second mif2 run: faster cooling
        ## to refine the parameter values and come up with the "better fitting" parameter set
        mif2(
          params=guess,
          cooling.fraction.50=0.1,
          Nmif=50,
          rw.sd=rw_sd(
            Beta=0.02,
            rho=0.02,
            r=0.02,
            k=0.02
          )
        ) -> mf

      #For the resulting "better fitting" parameter set, use the particle filter
      #multiple times to estimate the likelihood and variance
      mf |>
        pfilter(Np=1000) |>
        logLik() |>
        replicate(n=10) |>
        logmeanexp(se=TRUE,ess=TRUE) -> ll
      #Pull out the coefficients and return the logliklihood value
      #This is added to results along with the "better fitting" parameter set.
      mf |>
        coef() |>
        c(loglik=ll)
    } -> results
```

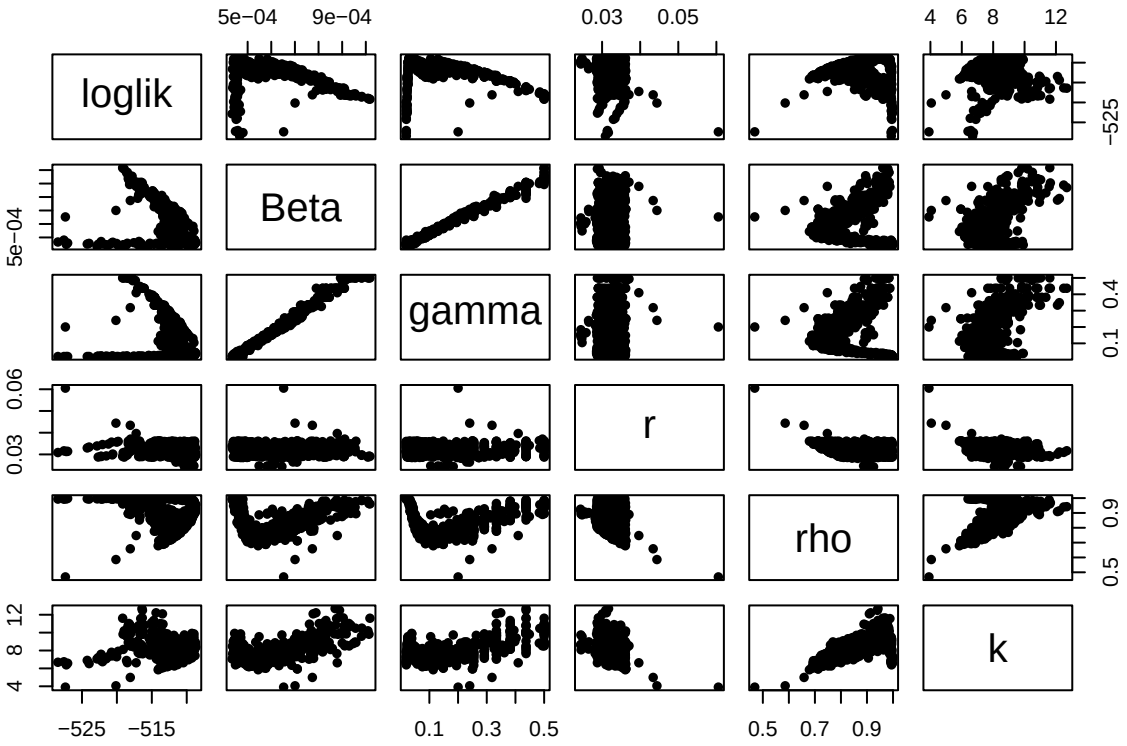
```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
#Clean the data, rename the logliklihood, and remove any rows with non-finite loglik
results <- results |>
  rename(loglik=loglik.est) |>
  filter(is.finite(loglik))

#add results to saved file using `append_data()' in pomp
#You can overwrite saved results using the option ``overwrite=TRUE"
all <- append_data(data=results,file=savefile)
```

```
#plot all results saved so far
plot_pairs(all, fit_params)
```




```
#plot just the results within 20 of the max log likelihood value
all|>
  filter(loglik>max(loglik)-20) |>
  plot_pairs(fit_params)
```

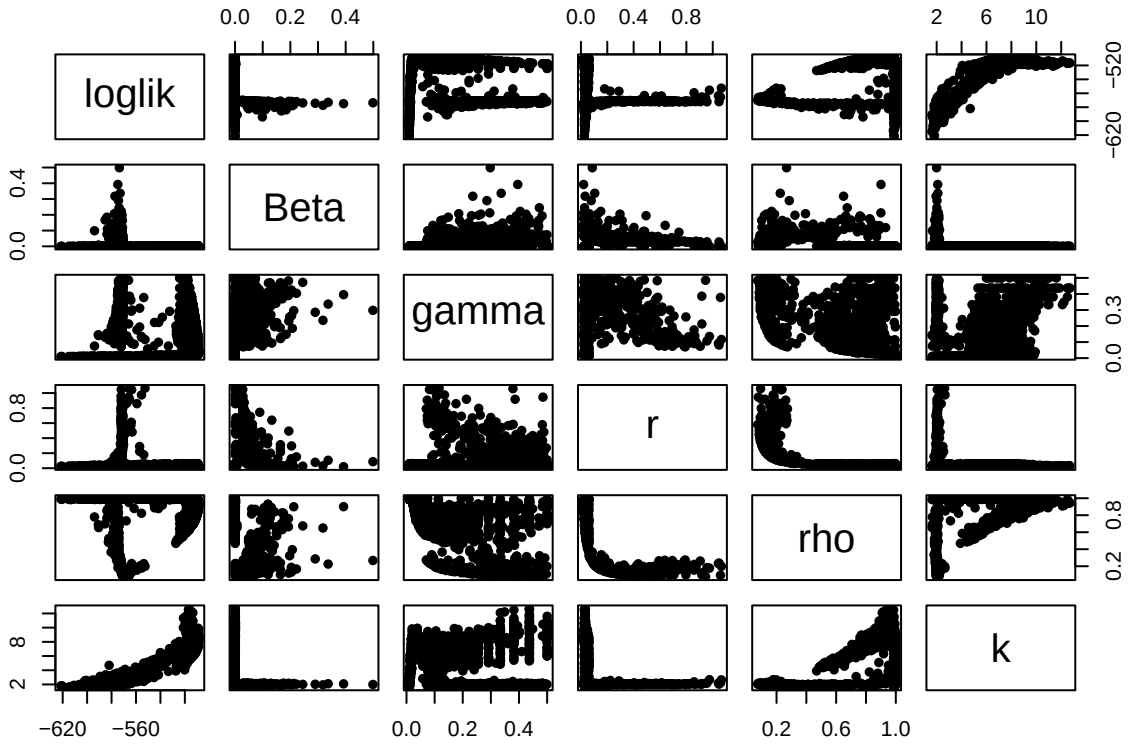



**Profile likelihood**

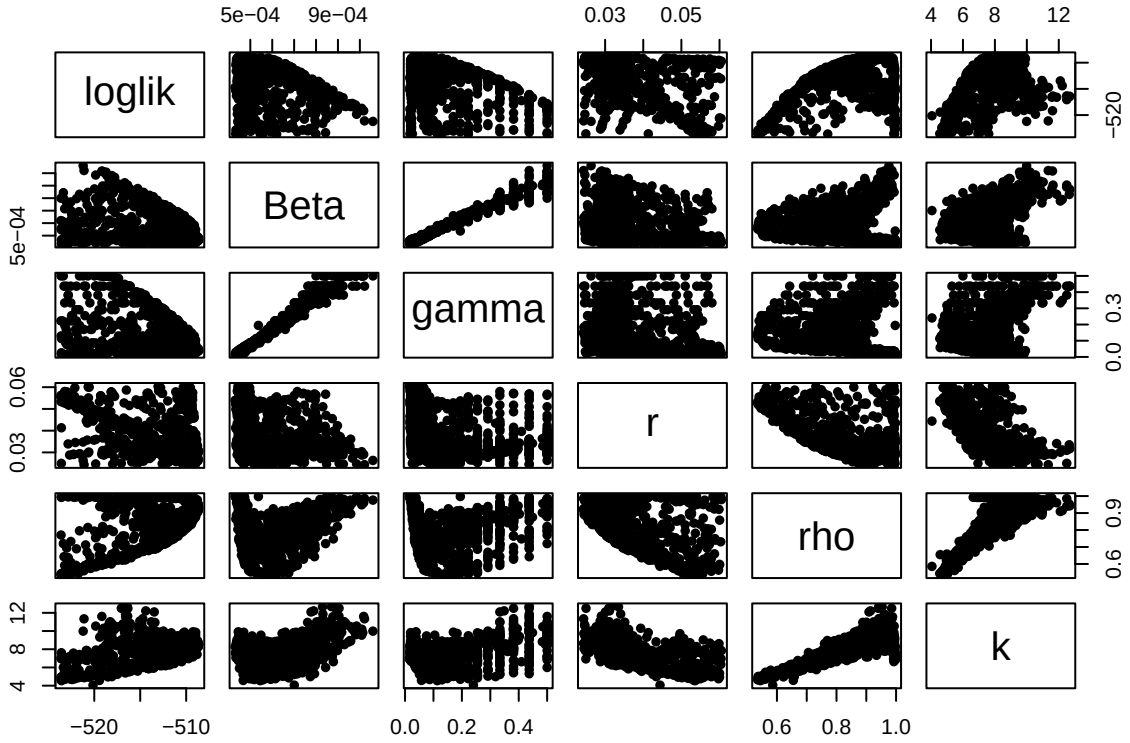
This will make a profile likelihood for gamma. We setup a range of plausible parameters for gamma and a randomly chosen set of guesses for the other parameters. When we do local search around the parameter space we hold gamma constant and search for the set of the other parameters that maximizes the likelihood. Our goal is to get a pretty good estimate of the maximum likelihood of the model for each value pf gamma set at a particular value. We repeat this for the range of gamma values to create a likelihood profile.

```
box <-
      all |>
      filter(loglik>max(loglik)-20) |>
      sapply(range)

## Create parameter ranges for each fit parameter and the parameter of interest
guesses <-
    profile_design(
        gamma=exp(seq(log(0.01),log(0.5),length=30)),
        #corresponds to illness lasting 3 to 14 days with a buffer
        #the log scale samples smaller values more heavily.
        lower=box[1,c("Beta","rho","r","k","N","phi")],
        upper=box[2,c("Beta","rho","r","k","N","phi")],
        #what does nprof do here?
        nprof=20,
        type="runif"
    )

registerDoParallel()
registerDoRNG(294113554)
foreach(
    guess=iter(guesses,"row"),
```

```
    .combine=bind_rows
  ) %dopar% {
      library(pomp)
      mifs_local |>
        ## first mif2 run: slow cooling
        mif2(
          params=guess,
          cooling.fraction.50=0.5,
          Nmif=50,
          rw.sd=rw_sd(
            Beta=0.02,
            rho=0.02,
            k=0.02
          )
        ) |>
        ## second mif2 run: faster cooling
        mif2(
          params=guess,
          cooling.fraction.50=0.1,
          Nmif=50,
          rw.sd=rw_sd(
            Beta=0.02,
            rho=0.02,
            k=0.02
          )
        ) -> mf
      mf |>
        pfilter(Np=1000) |>
        logLik() |>
        replicate(n=10) |>
        logmeanexp(se=TRUE,ess=TRUE) -> ll
      mf |>
        coef() |>
        c(loglik=ll)
    } -> results

attr(results,"ncpu") <- getDoParWorkers()

results |>
  rename(loglik=loglik.est) |>
  filter(is.finite(loglik)) -> results

all <- append_data(data=results,file=savefile)
plot_pairs(all,fit_params)
```

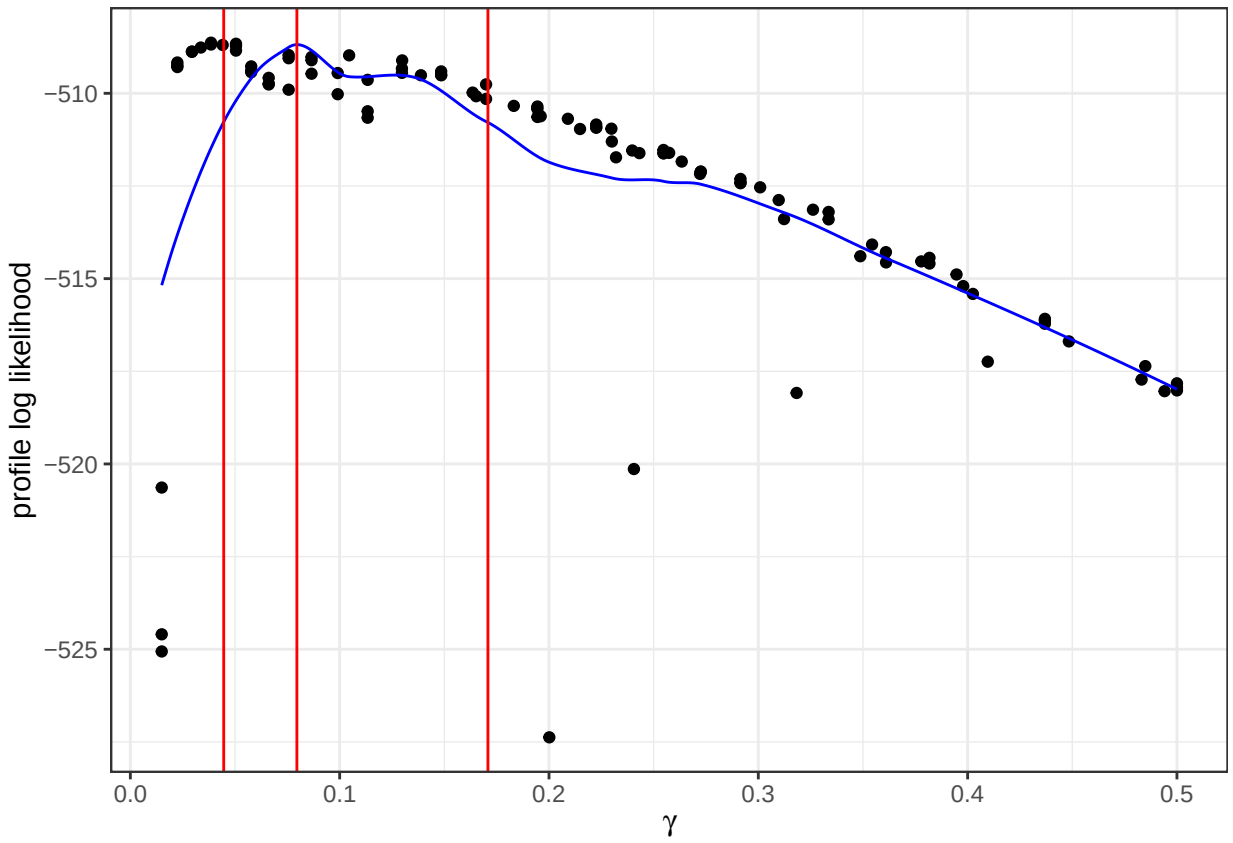



```
all |>
  filter(loglik>max(loglik)-15)|>
  plot_pairs(fit_params)
```




The function `mcap` adjusts the profile likelihood for Monte Carlo variability. We can see here that we haven't searched sufficiently broadly to estimate the CI.

```
#Filter to only logliklihood values within 20 of the max
#And to the top 4 logliklihood values for each band of gamma
#The level of rounding can change based on the range of the data
prof <- all |>
  filter(loglik>(max(loglik)-20) )|>
  group_by(round(gamma,2)) |>
  #group_by(round(log(gamma),3)) |>
  filter(rank(-loglik)<=3) |>
  ungroup()

#Use `mcap`: Monte Carlo-adjusted profile likelihood
prof |>
  with(
    mcap(loglik,gamma,span=0.5)
  ) -> mc

prof |>
  ggplot(aes(x=gamma,y=loglik))+
  geom_point()+
  geom_line(data=mc$fit,aes(x=parameter,y=smoothed),color="blue")+
  geom_vline(xintercept=c(mc$mle,mc$ci),color="red")+
  labs(x=expression(gamma),y="profile log likelihood")+
  theme_bw()
```

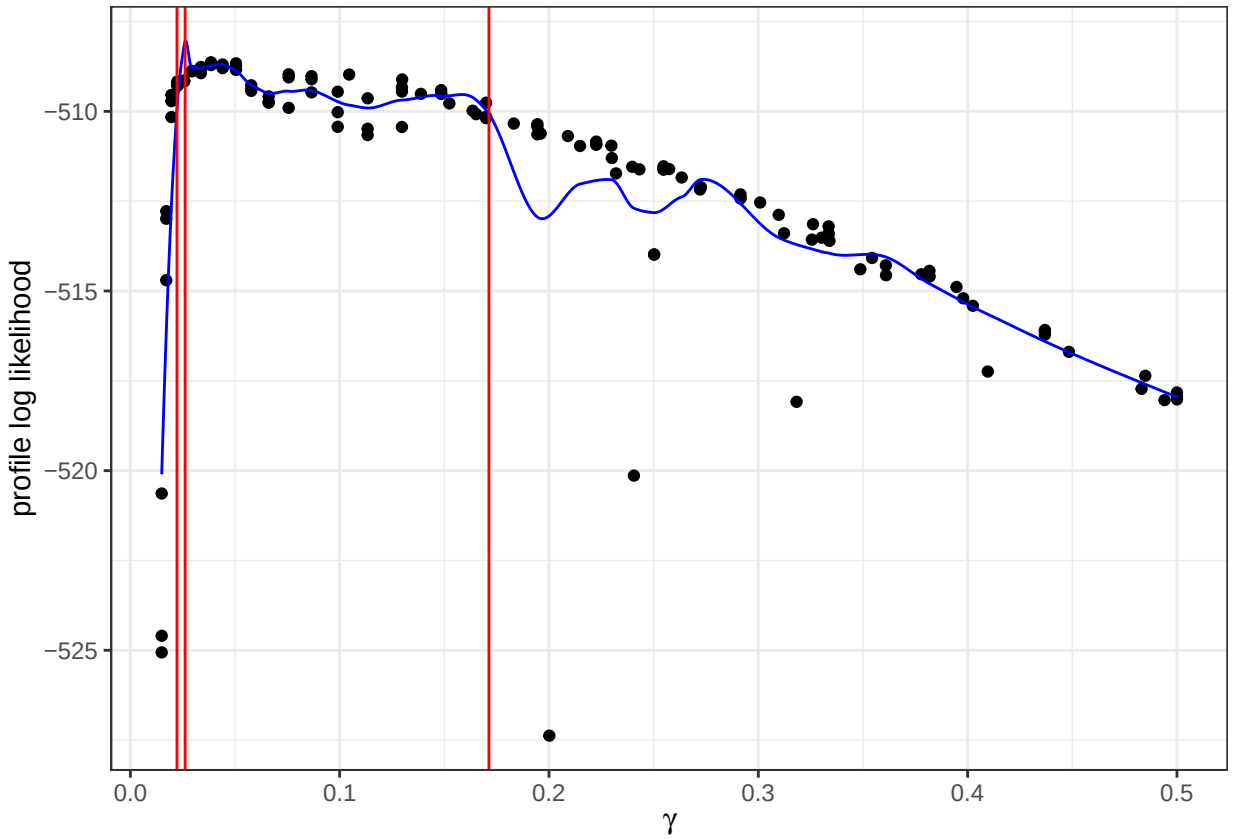



```
#Compute and return the confidence interval
(gammaCI= c(mc$mle,mc$ci))
```

```
## [1] 0.07955952 0.04460373 0.17083297
```

Grouping by the log of gamma gives denser sampling near zero where things are a bit less clear. Shorting the `span` in mcap also eliminates some of the pull caused by larger values. This makes it more clear where the left side of the confidence interval is.

```
#Filter to only logliklihood values within 20 of the max
#And to the top 4 logliklihood values for each band of gamma
#The level of rounding can change based on the range of the data
prof <- all |>
  filter(loglik>(max(loglik)-20) )|>
  group_by(round(log(gamma),3)) |>
  #group_by(round(log(gamma),3)) |>
  filter(rank(-loglik)<=3) |>
  ungroup()

#Use `mcap`: Monte Carlo-adjusted profile likelihood
prof |>
  with(
    mcap(loglik,gamma,span=0.2)
  ) -> mc

prof |>
  ggplot(aes(x=gamma,y=loglik))+
  geom_point()+
  geom_line(data=mc$fit,aes(x=parameter,y=smoothed),color="blue")+
  geom_vline(xintercept=c(mc$mle,mc$ci),color="red")+
  labs(x=expression(gamma),y="profile log likelihood")+
  theme_bw()
```




```
#Compute and return the confidence interval
(gammaCI= c(mc$mle,mc$ci))
```

```
## [1] 0.02615484 0.02227086 0.17131847
```

We could repeat this for each parameter if we want to get the confidence intervals for each.

If we are more interested in comparing the model and not so worried about the confidennce intervals, we can move on from here to calculate the AIC.

## Find the best log likelihood value and compute the AIC

```
all |>
  filter(loglik==max(loglik))|>
  summarize(
    loglik=loglik,
    K=length(fit_params),
    AIC=-2*loglik+2*K
  )
```

```
##      loglik K      AIC
## 1 -508.6336 5 1027.267
```

We will use these final pieces when comparing this model to another model.

Under this model, the infectious period is estimated to be 5.8–45days.