ETAS example - Amatrice (new)

Francesco Serafini

05/04/2022

Set-up

In this section we set-up the environment with which we are going to work in the reminder of the document. This section is the one requiring the most changes by the user and some of the quantities and libraries used afterward depends on the machine used to run the present code. Specifically, in this section, we are going to import all the libraries and data files necessary for the example, as well as we will define the domain of the problem (e.g. area of interest, magnitude of completeness, etc.). Also, we going to set-up machine specific quantities such as number of cores to be used and the possibility to run the code in parallel; the default is to use only one core. This section is divided in a Library set-up section reporting all the required libraries and importing them; a Data set-up section where we import all the data files and set up the data in the correct format for our implementation.

Libraries set-up

Below the list of libraries (name and version) that we will use in this example. The only optional library is the `parallel` library which is imported only if the number of cores is greater than one.

```
load('session.info.Rds')
cbind(Name = sapply(session.info$otherPkgs, \(x) x$Package),
      Version = sapply(session.info$otherPkgs, \(x) x$Version))
```

```
##                Name            Version
## viridis       "viridis"       "0.6.2"
## viridisLite   "viridisLite"   "0.4.0"
## ggplot2       "ggplot2"       "3.3.5"
## rnaturalearth "rnaturalearth" "0.1.0"
## mnormt        "mnormt"        "2.0.2"
## rgeos         "rgeos"         "0.5-9"
## raster        "raster"        "3.5-15"
## MASS          "MASS"          "7.3-55"
## mvtnorm       "mvtnorm"       "1.1-3"
## matrixStats   "matrixStats"   "0.61.0"
## metR          "metR"          "0.12.0"
## data.table    "data.table"    "1.14.2"
## dplyr         "dplyr"         "1.0.8"
## INLA          "INLA"          "22.03.16"
## foreach       "foreach"       "1.5.2"
## Matrix        "Matrix"        "1.3-4"
## inlabru       "inlabru"       "2.5.2.9000"
## sp            "sp"            "1.4-6"
## sf            "sf"            "1.0-6"
```

Then, we can import the code and libraries needed for plotting and handling spatial objects.

```
#source('../../../Code/Time-dependent-forecast/code_for_etas.R')
source('code_for_etas2.R')
library(rnaturalearth) # library for Italy map
library(ggplot2) # library for plotting
library(viridis) # library for color scales
library(sp) # library to handle Spatial objects
```

Now we are going to set the number of cores, the default is 1 which works for any machine. If we set the number of cores to a number greater than 1, then, we will need the library `parallel` to perform parallel computations. All the functions here has 1 as default, so if the user wants to be sure to use one core can just delete the line `ncore = my_cores` to all the functions.

```
# set number of cores
my_cores <- 5

# import parallel library if necessary
if(my_cores > 1){
  library(parallel)
}
```

Data set-up

This section is dedicated to import the dataset that we will be using to fit the model and to be forecasted. In this section we also define the domain in which the data lives which is constitute by: a time interval (T_1, T_2), a 2D region $W \subset \mathbb{R}^2$ and a magnitude of completeness M_0 such that any observed magnitude $m > M_0$.

We start importing the file containing the space-time-magnitude locations of the observed events. The file is contained in the `data` folder in the `Amatrice_Example` folder. We start importing data with magnitude greater than 2.5. The present code assumes that the data is in csv format, if this is not the case we recommend changing the following line to import the data in the format at hand. Also, we set the magnitude of completeness M_0 and we select only the observations with magnitude greater than M_0. In this example, we set $M_0 = 2.99$ meaning that the minimum magnitude in the catalog will be 3. We also transform the time column in to be in `Date`. We then create a column representing the number of days between the time immediately before the first event in the sequence and the observed events. This is the times used to fit the model. We recommend also to always check and remove duplicate rows based on the space-time location of the events. One of the assumptions on which many model are based is that two point can not occur at the same location.

```
M0 <- 2.99
# load the data
my_data <- read.csv("data/M2.5.csv" , header=TRUE) %>%
  mutate(time_date = as.POSIXct(time_date, format = "%d/%m/%Y %H:%M")) %>% # tranform time column
  filter(Mw > M0)  %>% # filer for magnitude
  distinct(time_date, Lat, Lon, .keep_all = TRUE) # remove duplicated lines
# there were 2 duplicated lines

# set starting date (a second before the first event in the sequence)
start.date <- min(my_data$time_date) - 1

my_data <- my_data %>%
  mutate(time.diff = as.numeric(difftime(time_date, start.date, units = 'days')))
```

We proceed setting up the time domain that will go from $T_1 = 0$ to the time immediately after the last event in the sequence. We set up alos a Polygon representing the study region. In this example, we take a rectangular region defined by the minimum and maximum observed values of longitude and latitude with a

buffer of 0.01. The values of the vertices defining the spatial domain may be changed providing values for `x.lim` and `y.lim` below.

```
# set up time interval
T1 = 0
T2 = as.numeric(difftime(max(my_data$time_date) + 1, start.date, units = 'days'))

# set the value of the buffer
buff <- 0.01
# Set up region verteces
x.lims <- c(min(my_data$Lon), max(my_data$Lon)) + c(-buff, buff)
y.lims <- c(min(my_data$Lat), max(my_data$Lat)) + c(-buff, buff)

# create sp object representing the area
x_coords <- c(x.lims[1],x.lims[2],x.lims[2],x.lims[1],x.lims[1])
y_coords <- c(y.lims[1],y.lims[1],y.lims[2],y.lims[2],y.lims[1])
poly1 <- sp::Polygon(cbind(x_coords,y_coords))
bdy <- sp::Polygons(list(poly1), ID = "A")
bdy <- sp::SpatialPolygons(list(bdy))
```

Inlabru dataset

Here we set up the `data.frame` that would be ultimately given to Inlabru to fit the model. It has to respect a specific format: 1. longitude and latitude columns are called `x` and `y` 2. the occurence times (reported as time differences from the starting date T_1) are called `ts` 3. the magnitudes are called `mags`

```
df.bru <- data.frame(x = my_data$Lon, # x for longitude
                     y = my_data$Lat, # y for latitude
                     ts = my_data$time.diff, # ts for time (in secs, hours, days)
                     mags = my_data$Mw) # mags for magnitudes
```

Weeks data set up

In the subsequent sections we will provide an example on how to produce weekly forecasts. To do this, we need first of all to split the data in weeks. We do that creating a `list` of data.frames representing the different weeks. We calculate also the vector of number of events per week.

```
# divide the first 100 weeks of data
list.weeks <- foreach(i = 1:ceiling(T2/7)) %do% {
  df.bru[df.bru$ts >= (i-1)*7 & df.bru$ts <= (i)*7,]
}
# number of events per week
N.weeks <- unlist(lapply(list.weeks, nrow))
```

Some of the elements may be empty reflecting the absence of events in that week. For example, we have 5 weeks having 0 events (reported below) and the corresponding list elements is empty. We reccommend to keep this in mind for the next code parts in which we will be using elements from `list.weeks` and we will need to check if they are empty or not.

```
which(N.weeks == 0)
```

```
## [1]  9 32 43 47 49
```

```
list.weeks[[9]]
```

```
## [1] x    y    ts   mags
## <0 rows> (or 0-length row.names)
```

Data plotting

In this section, we have a look at the selected data that we are going to use to fit the model and retrospectively test it. We are going to use `sf` objects for plotting. The procedure comprises the following steps: i) transform your data in an `sp` object; ii) transform the `sp` object in an `sf` object; iii) project it using the Italy map projection; iv) plot it. This is done every time we need to plot a spatial object with a specific projection.

```
# i) tranform data in sp object
my_data.sp <- data.frame(my_data)
sp::coordinates(my_data.sp) <- c("Lon", "Lat")

# load italy map
it.map <- ne_countries(country = 'Italy', returnclass = "sf", scale = 'medium')
# extract crs
italy.crs <- crs(it.map)

# ii) tranform sp in  sf object
my_data.sf <- st_as_sf(my_data.sp)
# iii) project the sf object
st_crs(my_data.sf) <- italy.crs#"+proj=longlat +datum=WGS84 +no_defs"#"+proj=utm +zone=33"
my_data.sf <- st_transform(my_data.sf, italy.crs)

# same for study region:
# transform sp in sf object
bdy.sf <- st_as_sf(bdy)
# project it
st_crs(bdy.sf) <- italy.crs
bdy.sf <- st_transform(bdy.sf, italy.crs)

# plots
# map with Italy
pl.it <- ggplot() +
  geom_sf(data = it.map, fill=alpha("lightgrey", 0), color = 'green') +
  geom_sf(data = bdy.sf) +
  geom_sf(data = my_data.sf, size = 0.2)
# zoom on the sequence
pl.seq <- ggplot() +
  geom_sf(data = bdy.sf) +
  geom_sf(data = my_data.sf[order(my_data.sf$Mw),], shape = 21, mapping = aes(fill = Mw)) +
  geom_sf(data = my_data.sf[my_data.sf$Mw >= 5,], shape = 24, fill = 'red', size = 2) +
  scale_fill_viridis()
# time vs magnitude plot
pl.mag <- ggplot(my_data.sf, aes(x = time_date, y = Mw)) +
  geom_point() +
  xlab('date')

# histogram of times
pl.hist <- ggplot(my_data.sf, aes(x = time_date,)) +
  geom_histogram(bins = 50) +
  xlab('date')
# actual plot
multiplot(pl.it, pl.seq, pl.mag, pl.hist,
          layout = matrix(1:4, ncol = 2, byrow = TRUE))
```

We focus on the first 20 weeks given that are the most interesting ones (where the majority of the events

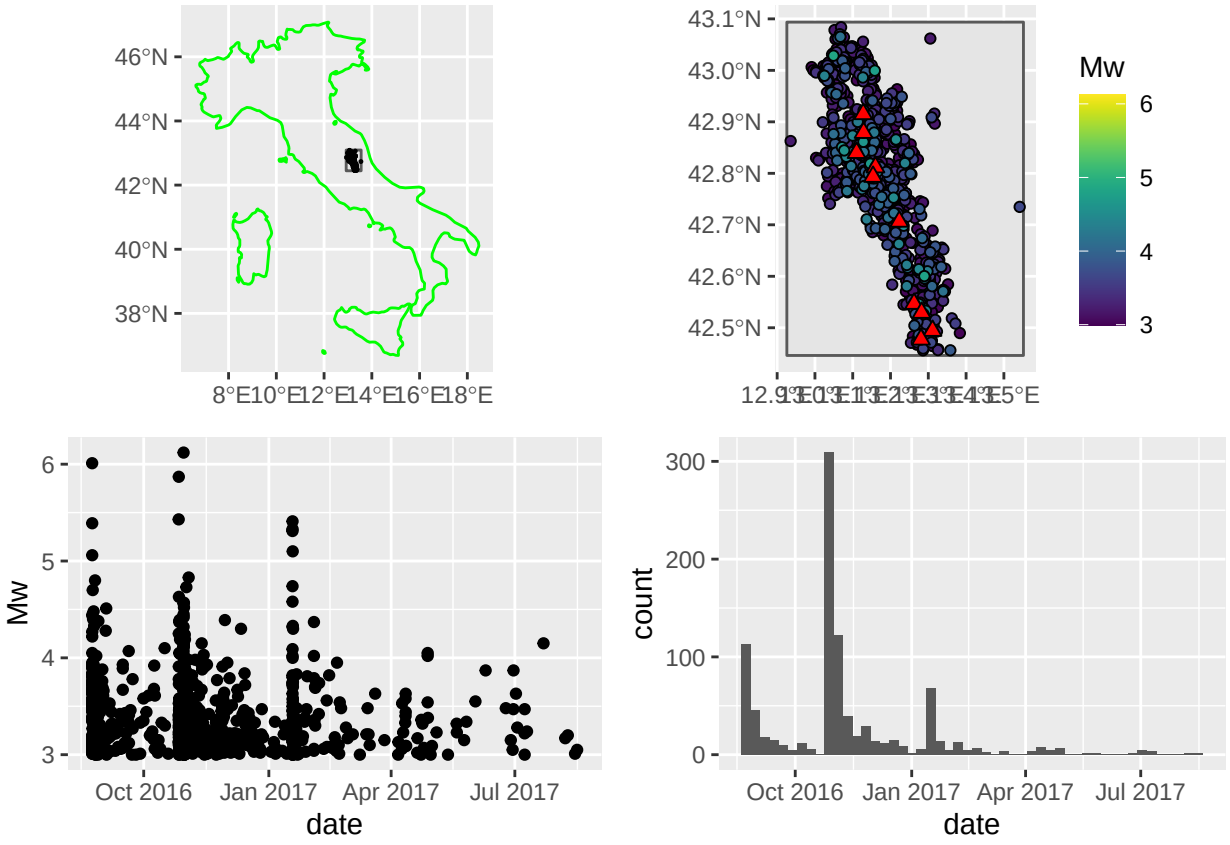
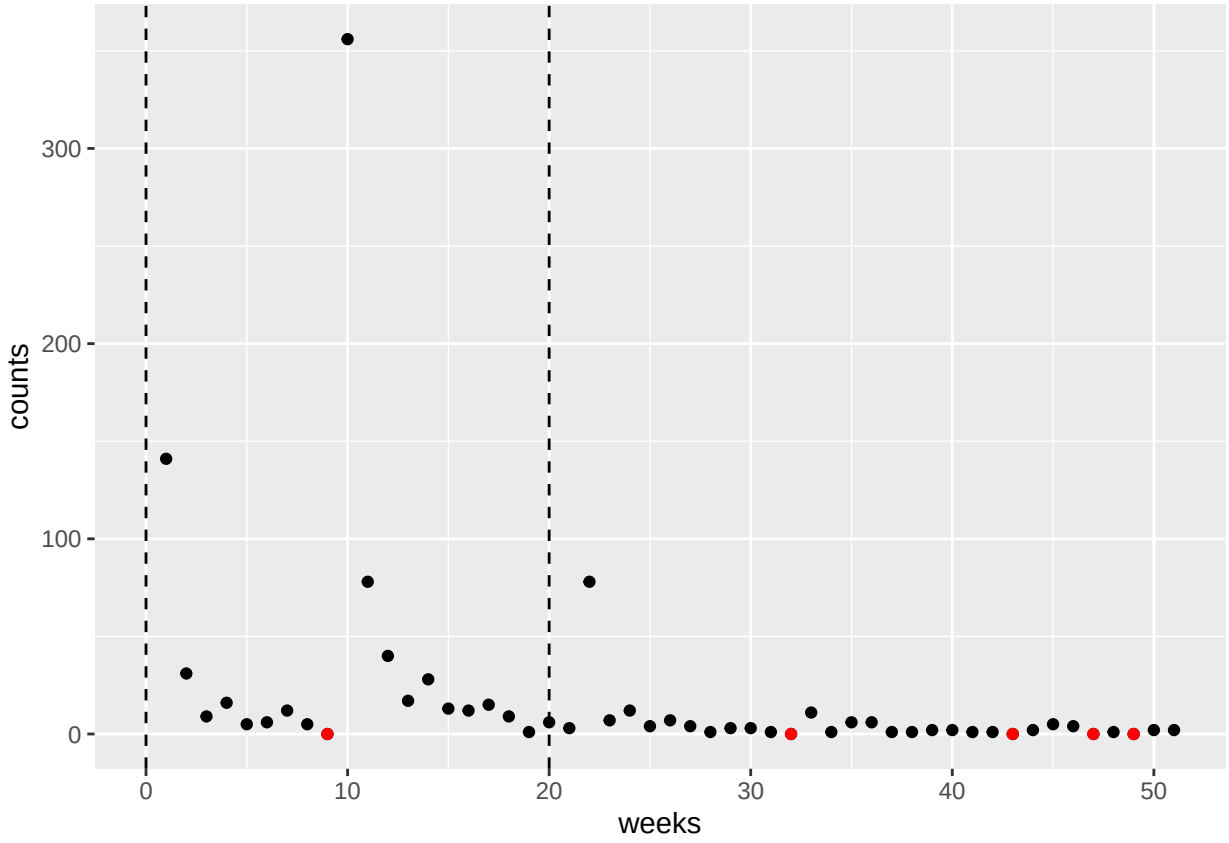


Figure 1: Amatrice sequence on different domains

happened).

```
# plot
ggplot() +
  geom_point(aes(x = 1:length(N.weeks), y = N.weeks)) +
  geom_point(aes(x = which(N.weeks == 0), y = 0), color = 'red') +
  xlab('weeks') +
  ylab('counts') +
  geom_vline(xintercept = c(0,20), linetype = 2)
```



Below we provide a plot of the spatial distribution of the events in each week. This is done creating a `data.frame` containing all the data with a column indicating the week. Weeks with no events are recorded as NA on the other columns. Then we plot only the weeks with events in them and add as many empty plots as many empty weeks.

```
# select weeks to show
weeks.to.fore <- list.weeks[1:20]
# create a data.frame with all the observations and a column that indicate the week
df.plot <- foreach(i = 1:length(weeks.to.fore), .combine = rbind) %do% {
  if(nrow(weeks.to.fore[[i]]) == 0){
    data.frame(x = NA, y = NA, ts = NA, mags = NA, week = 1 + (i - 1))
  }
  else{
    data.frame(x = weeks.to.fore[[i]]$x,
               y = weeks.to.fore[[i]]$y,
               ts = weeks.to.fore[[i]]$ts,
               mags = weeks.to.fore[[i]]$mags,
               week = 1 + (i - 1))
    }
```

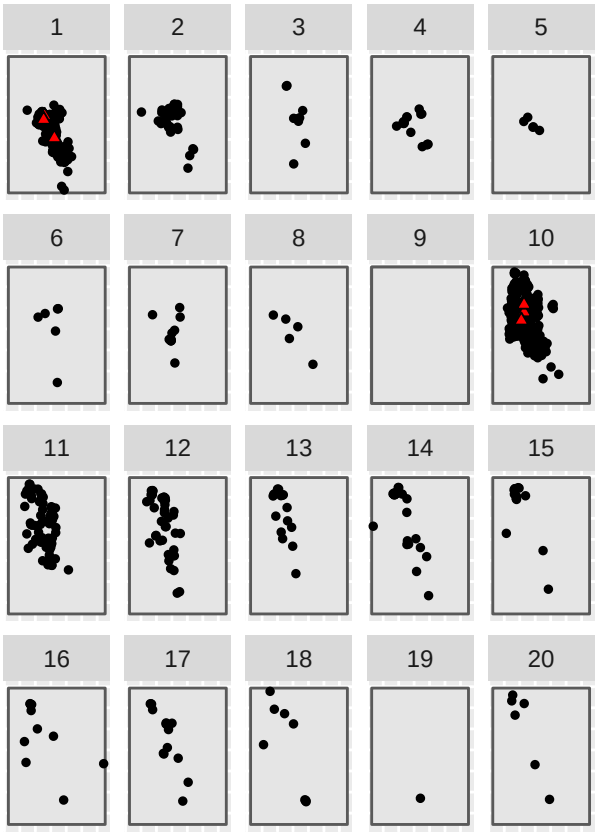
```
}

# turn it in an sf object (only weeks with events)
df.plot.sp <- df.plot[!is.na(df.plot$x),]
coordinates(df.plot.sp) <- c('x', 'y')
df.plot.sf <- st_as_sf(df.plot.sp)
st_crs(df.plot.sf) <- italy.crs
df.plot.sf <- st_transform(df.plot.sf, italy.crs)

# plot it
pl.week <- ggplot() +
  geom_sf(data = bdy.sf) +
  geom_sf(data = df.plot.sf, size = 1) +
  geom_sf(data = df.plot.sf[df.plot.sf$mags >= 5, ], shape = 24, fill = 'red', size = 1.5) +
  facet_wrap(facets = vars(week)) +
  theme(axis.text = element_blank(),
        axis.ticks = element_blank())

# add empty plot for empty weeks
for(i in 1:sum(is.na(df.plot$x))){
  pl.week <- pl.week + geom_sf(data = bdy.sf %>%
            mutate(week = df.plot$week[is.na(df.plot$x)][i]))
}

# plot it
pl.week
```



Model for Background field

We are considering the background rate as given by:

$$\mu(\mathbf{s}) = \mu u(\mathbf{s})$$

Where $\mu \geq 0$ can be seen as the usual background rate, while we refer to $u(\mathbf{s}) \geq 0$ as the background field, it represents the variation of the background field depending on location and it is normalized such that $\int_W u(\mathbf{s}) d\mathbf{s} = 1$. Our goal now, is to estimate the logairthm of the background field, namely $\log u(\mathbf{s})$

The first thing to do is to fit a model for the log background field, which in this case is a simple model with just an intercept and a spatially varying SPDE random effect. The background field may include covariates (as in Kirsty's model). In general, however the model is, a mesh and and the value of the log-intensity at the mesh nodes.

Here there are some parameters that needs to be adjusted to the problem at hand: the mesh parameters and the PC priors for the background field parameters. Regarding the mesh, we choose to use a maximum triangle edge equal to 0.05 for the interior, and 0.2 for the extended boundary. In total there are 510 vertices. This is just an example and we chose those number to maintain the computational time small, real applications would benefit from a finer mesh. We recommend setting those values to reflect the level of diversity of the data, for example setting the maximum triangle edge to some small multiple (< 10) of the minimum distance between points.

The parameters of the PC priors regulate the obtained background field. The parameters are range r and standard deviation σ. The range regulates the degree of smoothing (the higher the range the smoother the field), the standard deviation regulates the amplitude of the oscillations. The PC priors are defined by two values p_0, α which determines a PC prior such that $\Pr[r < p_0] = \alpha$ for the range and $\Pr[\sigma > p_0] = \alpha$ for the standard deviation. Here we choose to set PC priors such that $Pr[r < 0.01] = 0.01$, this is done to avoid too small values of the range which would induce overfitting; and $\Pr[\sigma > 0.1] = 0.01$ which is done to prevent the standard deviation to be too high. This parameters would need to be adjusted to the problem at hand.

```
# contruct the mesh
mesh_col <- inla.mesh.2d(boundary = bdy, max.edge = c(0.05, 0.2))

# transform data in SpatialPointsDataFrame
df.bru.pp <- df.bru
coordinates(df.bru.pp) <- c('x', 'y')

# set the spde object and the components
spde.o <- inla.spde2.pcmatern(mesh_col,
                              prior.sigma = c(0.1, 0.01), #Pr[sigma > 0.1] = 0.01
                              prior.range = c(0.01, 0.01)) #Pr[range < 0.01] = 0.01
cmp <- coordinates ~ field(coordinates, model = spde.o) + Intercept(1)
```

Then, we fit the model. We are aware that the present code may produce different errors on different machines, or produce unrealistic results which do not fit the data at all. We recommend to try running the same code with `inla.mode = 'classic'` or to set explicitly the number of threads to be used equal 1 adding the line `num.threads=1:1` in the options list or both. Also changing strategy may help, visit the website for a list of possible strategies (Sec 2.5.1 https://becarioprecario.bitbucket.io/inla-gitbook/ch-INLA.html)

```
# fit the model - takes around 50 secs
bru.bkg <- lgcp(components = cmp,
                data = df.bru.pp,
                samplers = bdy,
                domain = list(coordinates = mesh_col),
                options = list(inla.mode = 'experimental',
                               control.inla = list(int.strategy="eb")))
```

```
# value of the bkg field at pixel coordinates (for plotting only)
pred.bkg <- predict(bru.bkg, pixels(mesh_col), ~ field + Intercept, samplers = bdy)

# plot log-intensity
ggplot() + gg(pred.bkg['median']) + gg(bdy) + gg(df.bru.pp) + scale_fill_viridis()
```

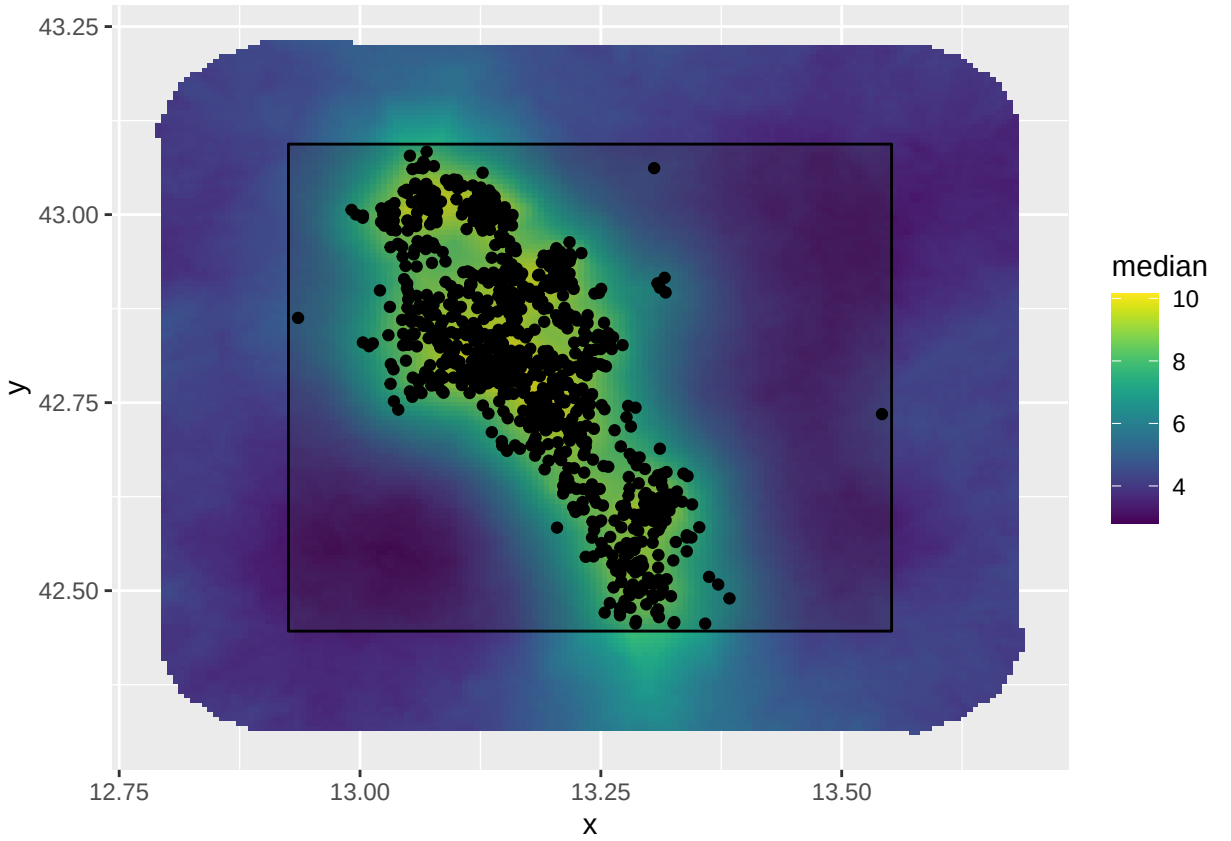


Figure 2: Estimated background field

In the next steps, we are going to use the median of the posterior log-intensity field as background field estimate. We now have to normalize the estimates to make them integrate to one.

```
# background rate at the mesh points
pred.bkg.mesh <- predict(bru.bkg, mesh_col, ~ exp(field + Intercept))

# create projection to find value of bkg field at the obs points
proj <- inla.mesh.project(mesh_col, cbind(df.bru$x, df.bru$y))

# add a bkg column to the data
df.bru$bkg <- as.numeric(proj$A %*% pred.bkg.mesh$median)

# approximate integral
integ.field <- sum(ipoints(samplers = bdy, mesh_col)$weight*crop(pred.bkg.mesh, bdy)$median)

df.bru$bkg <- df.bru$bkg/integ.field
```

We will consider the background field as given to estimate the ETAS parameters, however, in a forecasting

framework, for each simulated catalogue we can extract a different background field from the posterior. Below three examples.

```
sample.bkg <- generate(bru.bkg, pixels(mesh_col), ~ field + Intercept, 3)
pix <- pixels(mesh_col)
pix$samp1 <- sample.bkg[,1]
pix$samp2 <- sample.bkg[,2]
pix$samp3 <- sample.bkg[,3]

lims <- range(sample.bkg)

pl1 <- ggplot() + gg(pix['samp1']) + scale_fill_viridis(limits = lims) + theme(legend.position = 'none')
pl2 <- ggplot() + gg(pix['samp2']) + scale_fill_viridis(limits = lims) + theme(legend.position = 'none')
pl3 <- ggplot() + gg(pix['samp3']) + scale_fill_viridis(limits = lims) + theme(legend.position = 'none')
multiplot(pl1, pl2, pl3, cols = 3)
```

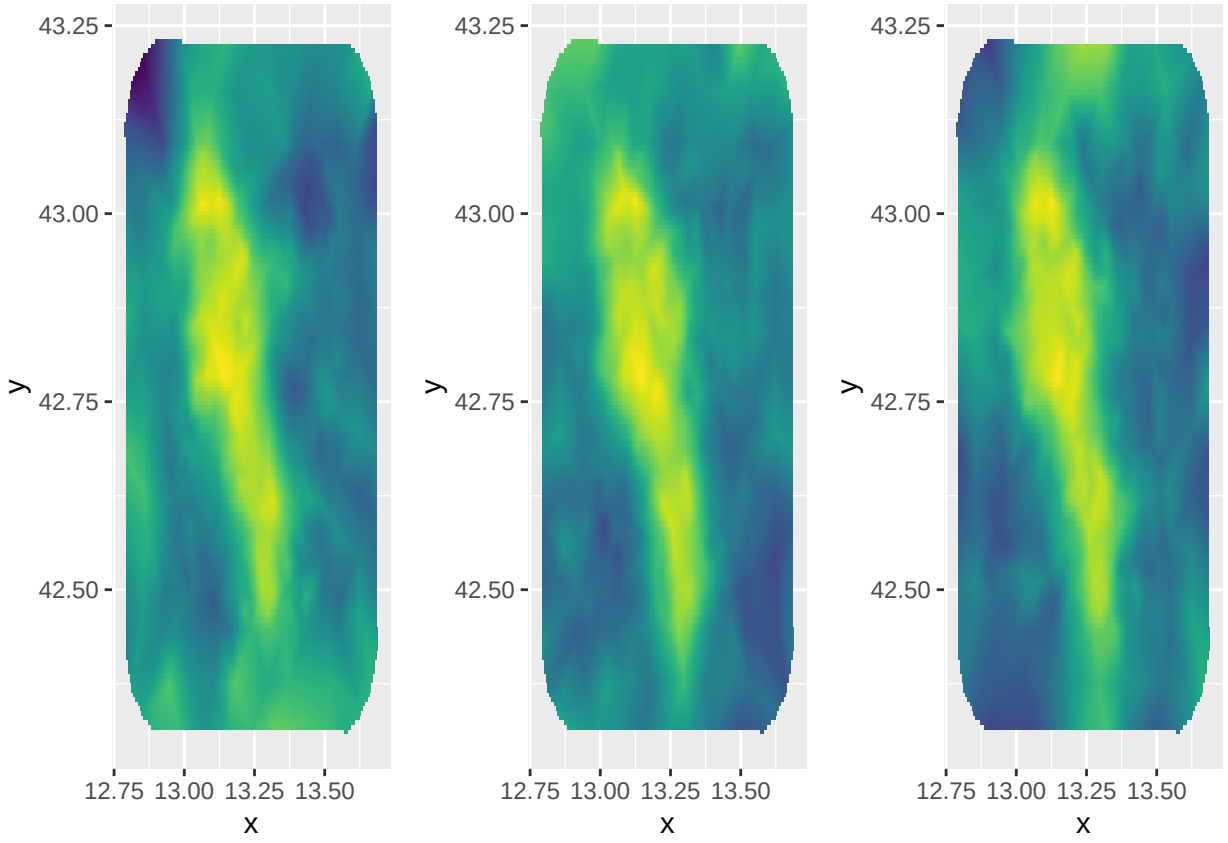


Figure 3: Sample from the posterior of the background field

Fit the model knowing Σ

In this first example, we are going to consider the covariance matrix Σ defining the spatial triggering function as fixed. To have a realistic Σ matrix we are going to consider

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2^2 \end{pmatrix}$$

where $\sigma_1 = \sigma_2 = \gamma \text{Med}(d_{min})$. The quantity $\text{Med}(d_{min})$ is the median of the minimum distance between points in the observed sequence, we are going to try different values of $\gamma = 1, 2, 3, 4, 5$. For the present example we are going to consider an isotropic kernel for which $\rho = 0$. We create a list to store the matrices that we are going to use for model fitting.

```
# vector of minimum distances
min.dists <- sapply(1:nrow(df.bru), \(idx)
                    min(sqrt(colSums((c(df.bru$x[idx], df.bru$y[idx]) -
                          rbind(df.bru$x[-idx], df.bru$y[-idx]))^2)))
                    )

# set sigma.1 sigma.2 and rho
sigma.1 <- median(min.dists)
sigma.2 <- sigma.1
ro_ <- cor(df.bru$x, df.bru$y)

# set Sigma
Sigma.list <- list(s1.ro0 = matrix(c(sigma.1^2, 0,
                                     0, sigma.2^2),
                                   byrow = TRUE, ncol = 2),
                   s2.ro0 = matrix(c((sigma.1*2)^2, 0,
                                     0, (sigma.2*2)^2),
                                   byrow = TRUE, ncol = 2),
                   s3.ro0 = matrix(c((sigma.1*3)^2, 0,
                                     0, (sigma.2*3)^2),
                                   byrow = TRUE, ncol = 2),
                   s4.ro0 = matrix(c((sigma.1*4)^2, 0,
                                     0, (sigma.2*4)^2),
                                   byrow = TRUE, ncol = 2),
                   s5.ro0 = matrix(c((sigma.1*5)^2, 0,
                                     0, (sigma.2*5)^2),
                                   byrow = TRUE, ncol = 2))
```

Among the input of the function to fit the model, there is a list of link functions to be applied to the parameters. Each parameter of the ETAS model is internally represented by a parameter $\theta \sim N(0, 1)$ which is then transformed to the correct scale. More formally, we have 5 parameters μ, K, α, c, p, such that $\mu, K, \alpha, c > 0$ and $p > 1$. Internally they are represented by 5 parameters $\theta_\mu, \theta_K, \theta_\alpha, \theta_c, \theta_p$ and 5 functions $\eta_\mu, \eta_K, \eta_\alpha, \eta_c, \eta_p$ such that $\mu = \eta_\mu(\theta_\mu)$, same for the others. This is done for two reasons:

1. The internal parameters $\theta_\mu, \theta_K, \theta_\alpha, \theta_c, \theta_p$ are constraint-free
2. We can impose different priors through the link functions.

On the second point, to obtain a random variable Y with distribution function (CDF) $F_Y(y) = \Pr[Y < y]$ from a random variable $X \sim N(0, 1)$ is sufficient to apply the function η such that

$$\eta(x) = F_Y^{-1}(F_X(x))$$

The method relies on the Inverse Transform method for sampling random variables, the first transformation $F_X(x)$ transform the variable in a $\text{Unif}(0, 1)$ and the second transformation takes the uniformily distributed values and turn into the desired distribution.

In this way, we can consider differnt priors changing the link function. For this example, we are going to consider

$$\eta_\mu(x) = \eta_K(x) = \eta_\alpha(x) = \eta_c(x) = e^x,\ \eta_p(x) = 1 + e^x$$

which induce a log-Gaussian distribution on the ETAS parameters. The code below contains a function `gamma.t()` which takes parameters with a standard normal distribution and turn them into parameters with a Gamma distribution. The link functions has to be stored into a list with elements names as the ETAS parameters. The ability of the algorithm to converge also depends on the choice of prior, so we recommend trying different ones and check how the result changes.

```
# use this as link function to have gamma distributed parameters
gamma.t <- function(x, a, b){
  bru_forward_transformation(qgamma, x, a, b)
}


link.f <- list(mu = \(x) exp(x),
               K = \(x) exp(x),
               alpha = \(x) exp(x),
               cc = \(x) exp(x),
               pp = \(x) exp(x) + 1)
```

Now, we can fit the model in a for loop using the different covariance matrices created before. We warn the reader that the convergence may depend on the problem at hand and the algorithm depends upon a number of free-parameters which are problem-dependent. These free-parameters can be grouped in two classes, parameters for the binning and parameters for the Inlabru algorithm. The first class is composed by `N.breaks.min, t.jump, max.length, N.tail, min.tail.length`. The binning works as follows, for each t_h we take equally spaced bins between t_h and $t_h + \texttt{t.jump}$. The number of bins is such that the bins are minimum `N.breaks.min` and have maximum length `max.length`. The other part of the interval, namely $(t_h + \texttt{t.jump}, T_2)$ is divided in larger bins such that they are maximum `N.tail` and has minimum length `min.tail.length`. The binning determines the ability to the algorithm to converge. For example, if the algorithm *jumps* regularly between states, considering an higher number of bins may be helpful. In the next section (Check convergence) we show how to plot the values assumed by the parameters in the algorithm iterations from which would be possible to identify those *jumps*. The parameters regulating the Inlabru algorithm instead are `max_step, inla.mode, bru_max_iter, num.threads`. The parameter `max_step` regulates the possible states that can be visited, starting from a given state. Lowering this parameters may also be helpful in case of regular jumps. The parameters `bru_max_iter` regulates the maximum number of iterations that the algorithm may do. This is particularly relevant when `max_step` is active because, in this case, the algorithm does not stop when convergence is reached but only when it has performed exactly `bru_max_iter` iterations. Other possible solutions in case the algorithm did not converge are:

1. set `inla.mode = 'classic'`
2. set `num.threads = 1:1`
3. set `control.inla = list(strategy = 'simplified.laplace')` or `control.inla = list(strategy = 'gaussian')`

We have noted that also trying combinations of the above settings may help convergence on different problems and on the same problem on different machines.

```
fit_list_1 <- vector(mode = 'list', length = length(Sigma.list))
names(fit_list_1) <- names(Sigma.list)
max_iter_vec <- c(15, 15, 40, 40, 40)
#1:length(Sigma.list)
for(S.idx in 3){
  cat('name : ', names(Sigma.list)[S.idx], '\n')
  Sigma.p <- Sigma.list[[S.idx]]
  cat('Sigma : ', Sigma.p, '\n')
  fit_ <- ETAS.fit.B_bkg(sample.s = df.bru, # data.frame representing data points
                         N.breaks.min = 10, # minimum number of bins (Integral decomposition)
                         t.jump = 10,
```

```
                              max.length = 10/30, # maximum length of a time bin
                              N.tail = 10,
                              min.tail.length = 0.05,
                              Sigma = Sigma.p, # Sigma matrix for spatial trig function
                              link.fun = link.f, # prior precision for thetas of temporal ETAS
                              M0 = M0, # magnitude of completeness
                              T1 = T1, T2 = T2 , # extremes of time interval
                              bdy = bdy, # SpatialPolygon representing study region (has to be squared)
                              bru.opt = # bru options
                                  list(bru_method = list(max_step = 0.5),
                                       inla.mode = 'experimental', # inla mode
                                       bru_max_iter = max_iter_vec[S.idx] , # max number of iterations
                                       bru_verbose = 3), # verbose changes what inlabru prints
                                bin.strat = 'alt', # strategy for the bins (recommend to use alt when T2>
                                ncore = my_cores) # set number of cores
  save(fit_, file = paste0('utils/fit_sigma.',names(Sigma.list)[S.idx],'.Rds'))
  fit_list_1[[S.idx]] <- fit_
}
save(fit_list_1, file = 'utils/fit_list_1.Rds')
```

```
load('utils/fit_list_1.Rds')
```

Check convergence

It is crucial to have a reliable posterior of the parameters that the algorithm converged. The Inlabru output already provides informations about the convergence. The first thing to check would be the log which is automatically printed out when we run Inlabru imposing `bru_verbose = 3`. It is stored in the final object and we can check the convergence printing out. For example, for the first and last model on the list is

```
cat('First model \n',
    fit_list_1$s1.ro0$bru_iinla$log[length(fit_list_1$s1.ro0$bru_iinla$log) - 2], '\n',
    fit_list_1$s1.ro0$bru_iinla$log[length(fit_list_1$s1.ro0$bru_iinla$log) - 1], '\n')
```

```
## First model
##  2022-04-07 18:06:29: iinla: Max deviation from previous: 0.239% of SD [stop if: <1%]
##  2022-04-07 18:06:29: iinla: Maximum iterations reached, running final INLA integration.
```

```
cat('Last model \n',
    fit_list_1$s5.ro0$bru_iinla$log[length(fit_list_1$s5.ro0$bru_iinla$log) - 2], '\n',
    fit_list_1$s5.ro0$bru_iinla$log[length(fit_list_1$s5.ro0$bru_iinla$log) - 1], '\n')
```

```
## Last model
##  2022-04-08 20:06:19: iinla: Max deviation from previous: 55.4% of SD [stop if: <1%]
##  2022-04-08 20:06:19: iinla: Maximum iterations reached, running final INLA integration.
```

From which we can see that both of them reached the maximum number of iterations (it was expected given that we set `max_step = 0.5`) and that the first one reached convergence (Max deviation $< 1\%$) while the last one not (Max deviation $> 1\%$).

To have a better picture of the algorithm convergence, we can plot the value assumed by the linearization point (supposed to be the posterior mode) at each iteration. From this plot is clear if the parameters converged or not. Below, the code to plot it and to compare different models. The parameters would be plotted in their internal scale.

```
fit_1 <- fit_list_1$s1.ro0
# single plot
ggplot(fit_1$bru_iinla$track, aes(x = iteration, y = mode, color = effect)) +
```

```
  geom_line() +
  geom_ribbon(aes(ymin = mode - 2*sd, ymax = mode + 2*sd, fill = effect), alpha = 0.2) +
  geom_hline(data = data.frame(v = c(0,0,0,0,0),
                               effect = c('th.mu', 'th.K', 'th.alpha',
                                          'th.c', 'th.p')),
             mapping = aes(yintercept = v)) +
  facet_wrap(facets = vars(effect))
```

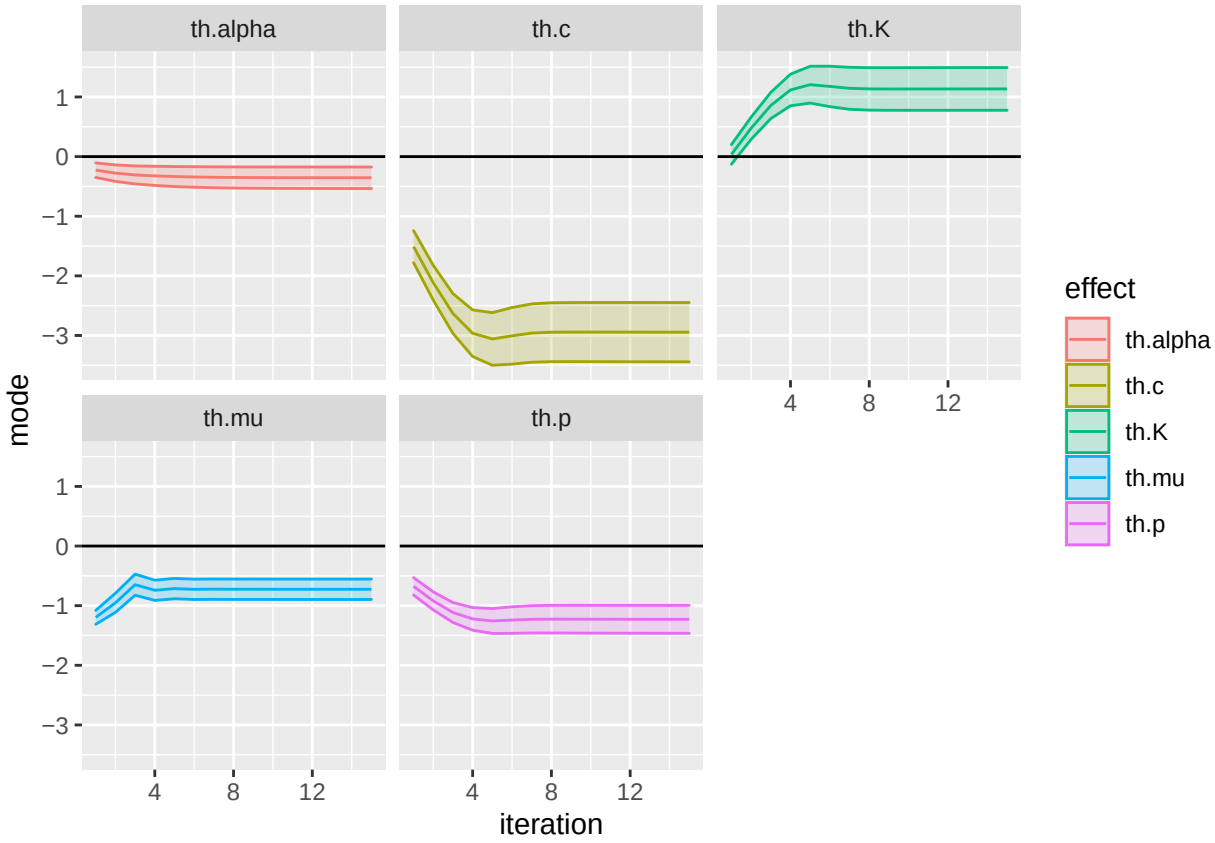


Figure 4: Single model convergence check, black horizontal lines represents the prior mean for the parameter

```
fit_2 <- fit_list_1$s5.ro0

df.plot <- rbind(data.frame(fit_1$bru_iinla$track, model = 'First'),
                 data.frame(fit_2$bru_iinla$track, model = 'Last'))


# single plot
ggplot(df.plot, aes(x = iteration, y = mode, color = model)) +
  geom_line() +
  geom_ribbon(aes(ymin = mode - 2*sd, ymax = mode + 2*sd, fill = model), alpha = 0.2) +
  geom_hline(data = data.frame(v = c(0,0,0,0,0),
                               effect = c('th.mu', 'th.K', 'th.alpha',
                                          'th.c', 'th.p')),
             mapping = aes(yintercept = v)) +
  facet_wrap(facets = vars(effect))
```

From which we can see that the last model did not reach converge and wouldn't reached even increasing the

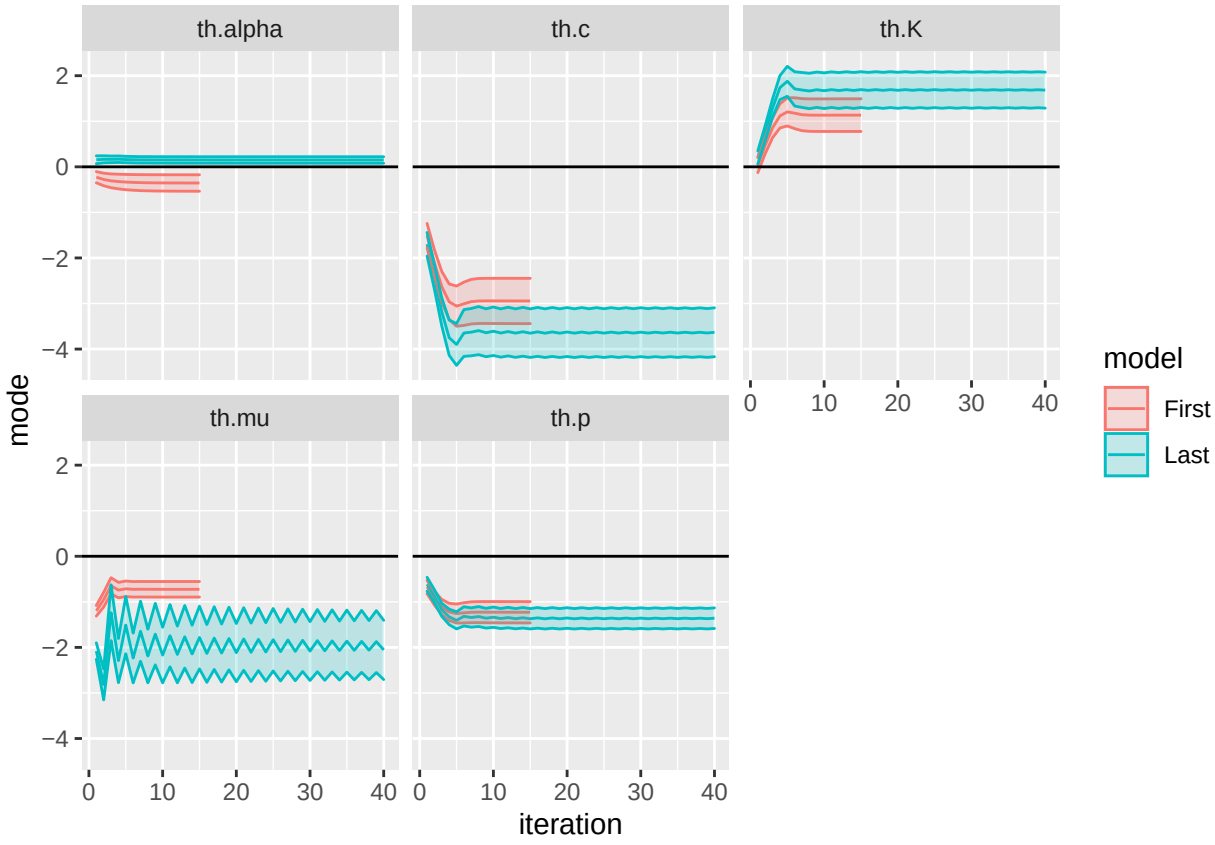


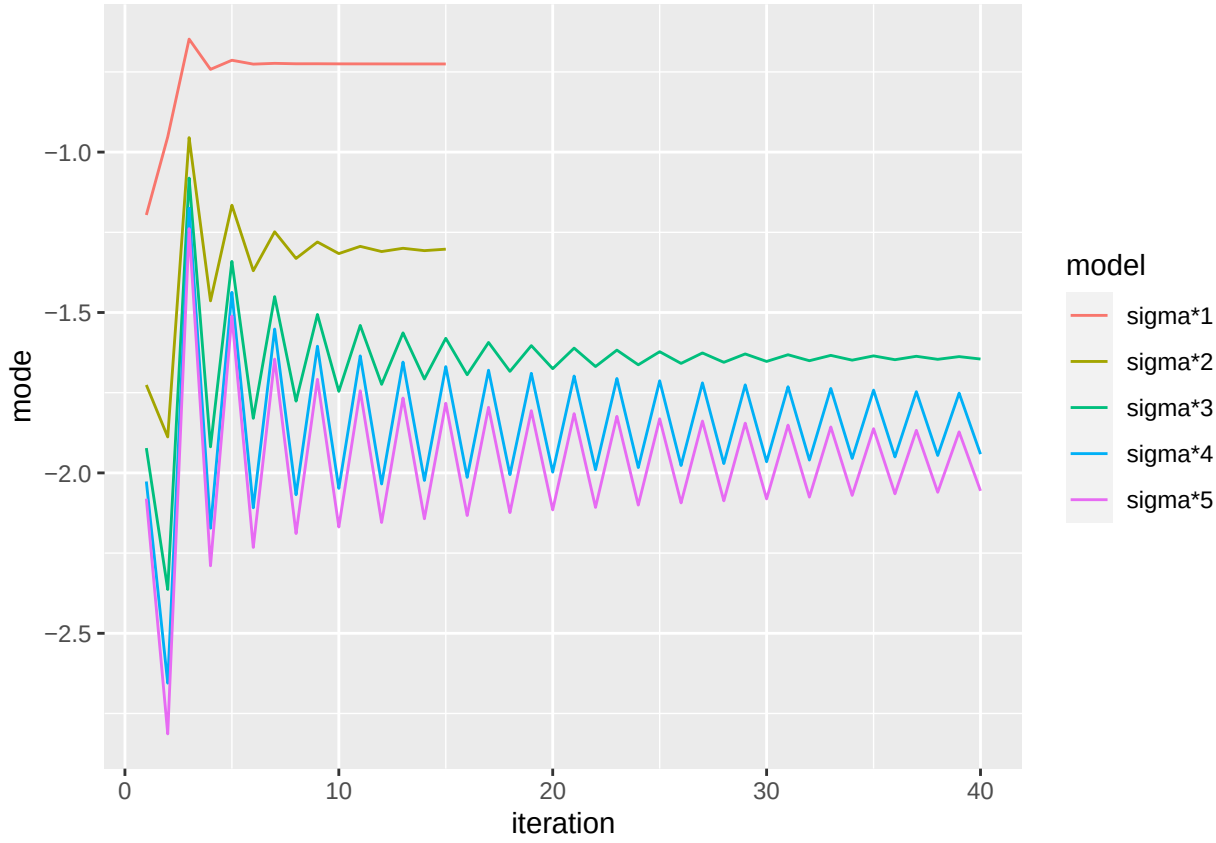
Figure 5: Convergence comparison, black horizontal lines represents the prior mean for the parameter

number of iterations given that the parameters θ_μ is jumping regularly between states. This problem may be mitigated considering stronger priors or smaller `max_step`.

We analyse the states track record for the different models considered, plotting only the linearization point at the different iterations of $\theta_m u$ which is the most problematic parameter. We can see that using $\gamma = 4, 5$ the model seems not to converge, while seems to be converging for $\gamma = 2, 3$ and to have converged for $\gamma = 1$.

```
df.plot.mu <- rbind(
  data.frame(fit_list_1$s1.ro0$bru_iinla$track[fit_list_1$s1.ro0$bru_iinla$track$effect == 'th.mu',],
             model = 'sigma*1'),
  data.frame(fit_list_1$s2.ro0$bru_iinla$track[fit_list_1$s2.ro0$bru_iinla$track$effect == 'th.mu',],
             model = 'sigma*2'),
  data.frame(fit_list_1$s3.ro0$bru_iinla$track[fit_list_1$s3.ro0$bru_iinla$track$effect == 'th.mu',],
             model = 'sigma*3'),
  data.frame(fit_list_1$s4.ro0$bru_iinla$track[fit_list_1$s4.ro0$bru_iinla$track$effect == 'th.mu',],
             model = 'sigma*4'),
  data.frame(fit_list_1$s5.ro0$bru_iinla$track[fit_list_1$s5.ro0$bru_iinla$track$effect == 'th.mu',],
             model = 'sigma*5')
  )


ggplot(df.plot.mu, aes(iteration, mode, color = model)) +
  geom_line()
```



Increase number of iterations

We can increase the number of iterations without the need of re-running the whole model. This is useful when we set a `max_step` parameter for which the only stopping mechanism is the number of iterations. To do that we have simply to set the initial values of the parameter to the value of the last state. We are going to do that for $\gamma = 2, 3$ which seems to be converging but not converged yet. We start checking the log of the model. We can see that at iteration 14 we are almost there, this is indicated by Max deviation close to be < 1 and the quantities `|lin1 - lin0|, <eta-lin1,delta>/|delta| , |eta-lin0 - delta <delta,eta-lin0>/<delta,delta>|` all close to zero (ideal point of convergence). From this we can imagine we do not need much more steps to get to convergence.

```
fit_ <- fit_list_1$s2.ro0
for(i in 8:2){
  cat(fit_$bru_iinla$log[length(fit_$bru_iinla$log) - i], '\n')
}
```

```
## 2022-04-07 18:29:11: iinla: Iteration 14 [max:15]
## 2022-04-07 18:29:30: iinla: Step rescaling: 162%, Expand
## 2022-04-07 18:29:38: iinla: Step rescaling: 100%, Overstep
## 2022-04-07 18:29:46: iinla: Step rescaling: 100.1%, Optimisation
## 2022-04-07 18:29:46: iinla: |lin1-lin0| = 4.952
##         <eta-lin1,delta>/|delta| = 8.064e-05
##         |eta-lin0 - delta <delta,eta-lin0>/<delta,delta>| = 0.0009398
## 2022-04-07 18:29:54: iinla: Step rescaling: 50%, Maximum step length
## 2022-04-07 18:30:37: iinla: Max deviation from previous: 4.33% of SD [stop if: <1%]
```

We are going to run the model for another 10 iterations. First thing is to retrieve the last state, which is stored as a list of 5 elements, each one containing a scalar indicating the value of the parameters at the state. Then, we can just re-run the model setting the number of iteration to 5 and the initial value of the parameters.

```
# select last state
last.state <- fit_$bru_iinla$states[length(fit_$bru_iinla$states)][[1]]
new_iter <- 5
# run the model
fit_ext <- ETAS.fit.B_bkg(sample.s = df.bru,
                          N.breaks.min = 10,
                          t.jump = 10,
                          max.length = 10/30,
                          N.tail = 10,
                          min.tail.length = 0.05,
                          Sigma = Sigma.list$s2.ro0, # set Sigma
                          link.fun = link.f,
                          M0 = M0,
                          T1 = T1, T2 = T2 ,
                          bdy = bdy,
                          bru.opt =
                              list(bru_initial = last.state, # set initial values
                                   bru_method = list(max_step = 0.5),
                                   inla.mode = 'experimental',
                                   bru_max_iter = new_iter, # change max number of iterations
                                   bru_verbose = 3),
                            bin.strat = 'alt',
                            ncore = my_cores)
#save(fit_ext, file = paste0('utils/fit_sigma.',names(Sigma.list)[S.idx],'.Rds'))
save(fit_ext, file = 'utils/fit_sigma.s2.ro0_ext.Rds')
```

```
load('utils/fit_sigma.s2.ro0_ext.Rds')
fit_ext$bru_iinla$track <- rbind(fit_$bru_iinla$track[nrow(fit_$bru_iinla$track),],
                                 fit_ext$bru_iinla$track)
fit_ext$bru_iinla$track$iteration <- fit_ext$bru_iinla$track$iteration +
  max(fit_$bru_iinla$track$iteration) - 1



df.plot.s2 <- rbind(fit_$bru_iinla$track %>%
                      mutate(fit = 'first 15'),
                    fit_ext$bru_iinla$track %>%
                      mutate(fit = 'Extended'))


ggplot(df.plot.s2 %>%
         filter(effect == 'th.mu'),
       aes(iteration, mode, color = fit)) +
  geom_ribbon(aes(ymin = mode - 2*sd, ymax = mode + 2*sd), alpha = 0.2) +
  geom_line()
```

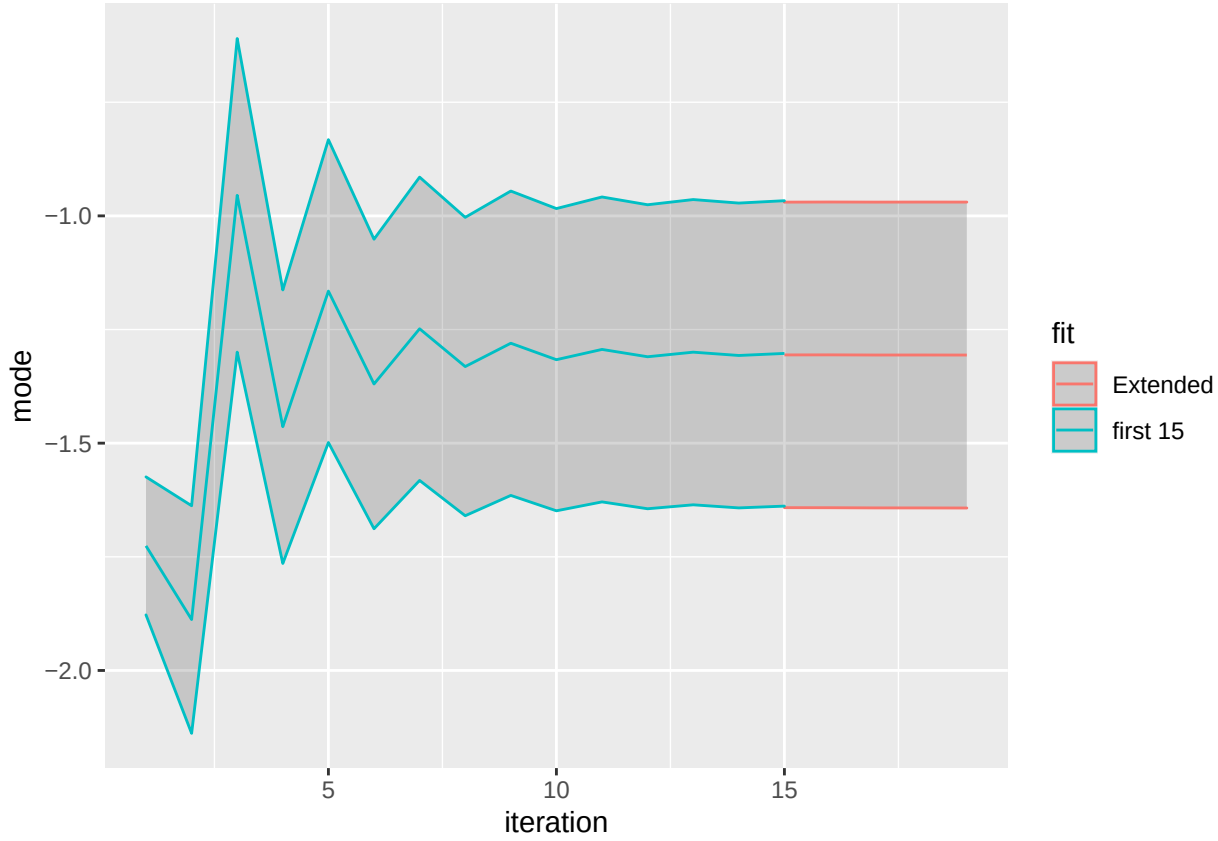


Figure 6: Track plot for background rate

```
#fit_list$s2.ro0 <- fit_ext
#save(fit_list, file = 'utils/fit_list.Rds')
```

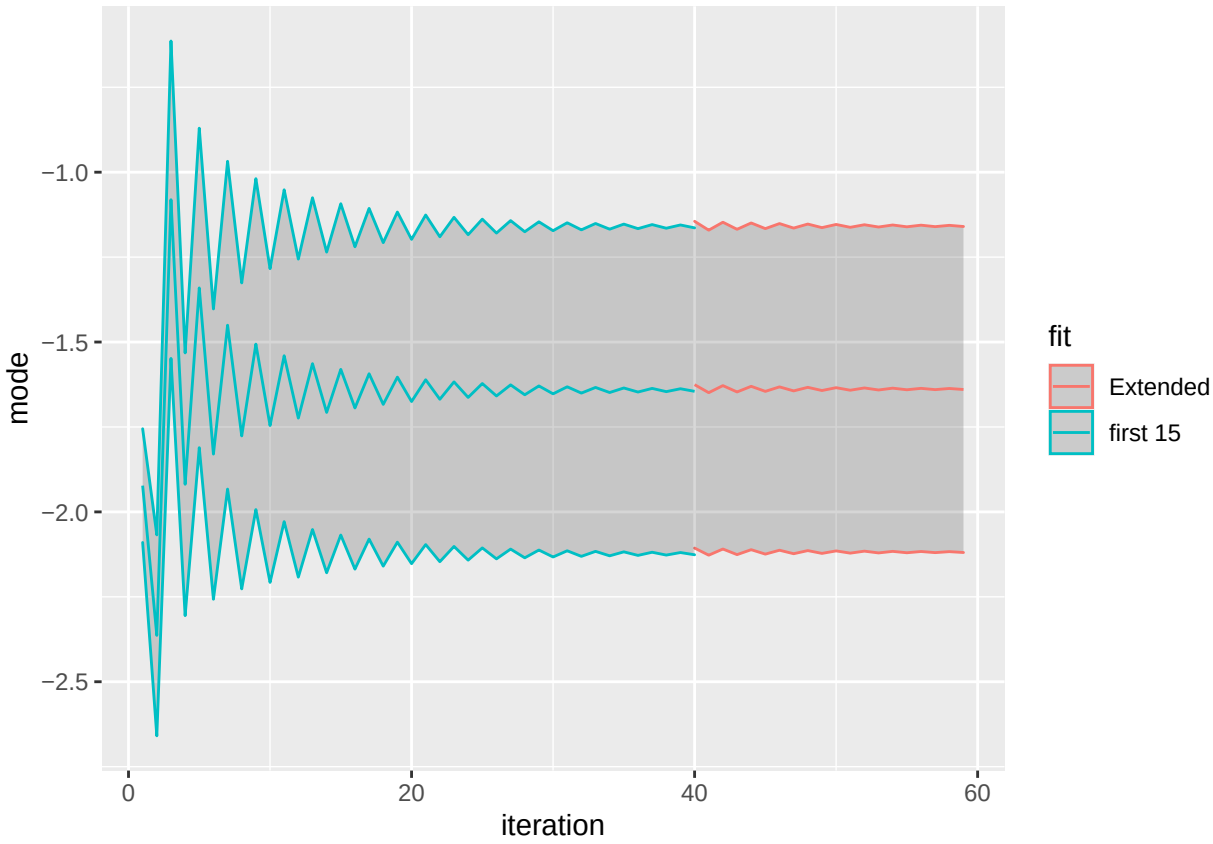
We do the same for $\gamma = 3$ considering 20 additional iterations.

```
fit_ <- fit_list$s3.ro0
last.state_ <- fit_$bru_iinla$states[length(fit_$bru_iinla$states)][[1]]
new_iter <- 20
# run the model
fit_ext_ <- ETAS.fit.B_bkg(sample.s = df.bru,
                           N.breaks.min = 10,
                           t.jump = 10,
                           max.length = 10/30,
                           N.tail = 10,
                           min.tail.length = 0.05,
                           Sigma = Sigma.list$s3.ro0, # set Sigma
                           link.fun = link.f,
                           M0 = M0,
                           T1 = T1, T2 = T2 ,
                           bdy = bdy,
                           bru.opt =
                               list(bru_initial = last.state_, # set initial values
                                    bru_method = list(max_step = 0.5),
                                    inla.mode = 'experimental',
                                    bru_max_iter = new_iter, # change max number of iterations
                                    bru_verbose = 3),
                             bin.strat = 'alt',
                             ncore = my_cores)
#save(fit_ext, file = paste0('utils/fit_sigma.',names(Sigma.list)[S.idx],'.Rds'))
save(fit_ext_, file = 'utils/fit_sigma.s3.ro0_ext.Rds')
```

```
load('utils/fit_sigma.s3.ro0_ext.Rds')
fit_ <- fit_list_1$s3.ro0
fit_ext_$bru_iinla$track <- rbind(fit_$bru_iinla$track[nrow(fit_$bru_iinla$track),],
                                  fit_ext_$bru_iinla$track)
fit_ext_$bru_iinla$track$iteration <- fit_ext_$bru_iinla$track$iteration +
  max(fit_$bru_iinla$track$iteration) - 1


df.plot.s3 <- rbind(fit_$bru_iinla$track %>%
                      mutate(fit = 'first 15'),
                    fit_ext_$bru_iinla$track %>%
                      mutate(fit = 'Extended'))

ggplot(df.plot.s3 %>%
         filter(effect == 'th.mu'),
       aes(iteration, mode, color = fit)) +
  geom_ribbon(aes(ymin = mode - 2*sd, ymax = mode + 2*sd), alpha = 0.2) +
  geom_line()
```




```
fit_list$s3.ro0 <- fit_ext_
save(fit_list, file = 'utils/fit_list.Rds')
```

Increase number of iterations - Change algorithm parameters

The possibility of running the algorithm from a initial state may also be used to run the algorithm with different parameters. For example, it is possible to run the first x iteration using a maximum step size or a binning and the subsequent using different values. We are going to use this possibility to reduce the maximum step size for the models with $\gamma = 4, 5$ to see if it makes the model converge.

```
last.state_ <- fit_list$s4.ro0$bru_iinla$states[length(fit_list$s4.ro0$bru_iinla$states)][[1]]
new_iter <- 6
# run the model
fit_ext_s4 <- ETAS.fit.B_bkg(sample.s = df.bru,
                             N.breaks.min = 10,
                             t.jump = 10,
                             max.length = 10/30,
                             N.tail = 10,
                             min.tail.length = 0.05,
                             Sigma = Sigma.list$s4.ro0, # set Sigma
                             link.fun = link.f,
                             M0 = M0,
                             T1 = T1, T2 = T2 ,
                             bdy = bdy,
                             bru.opt =
                                 list(bru_initial = last.state_, # set initial values
                                      bru_method = list(max_step = 0.3),
```

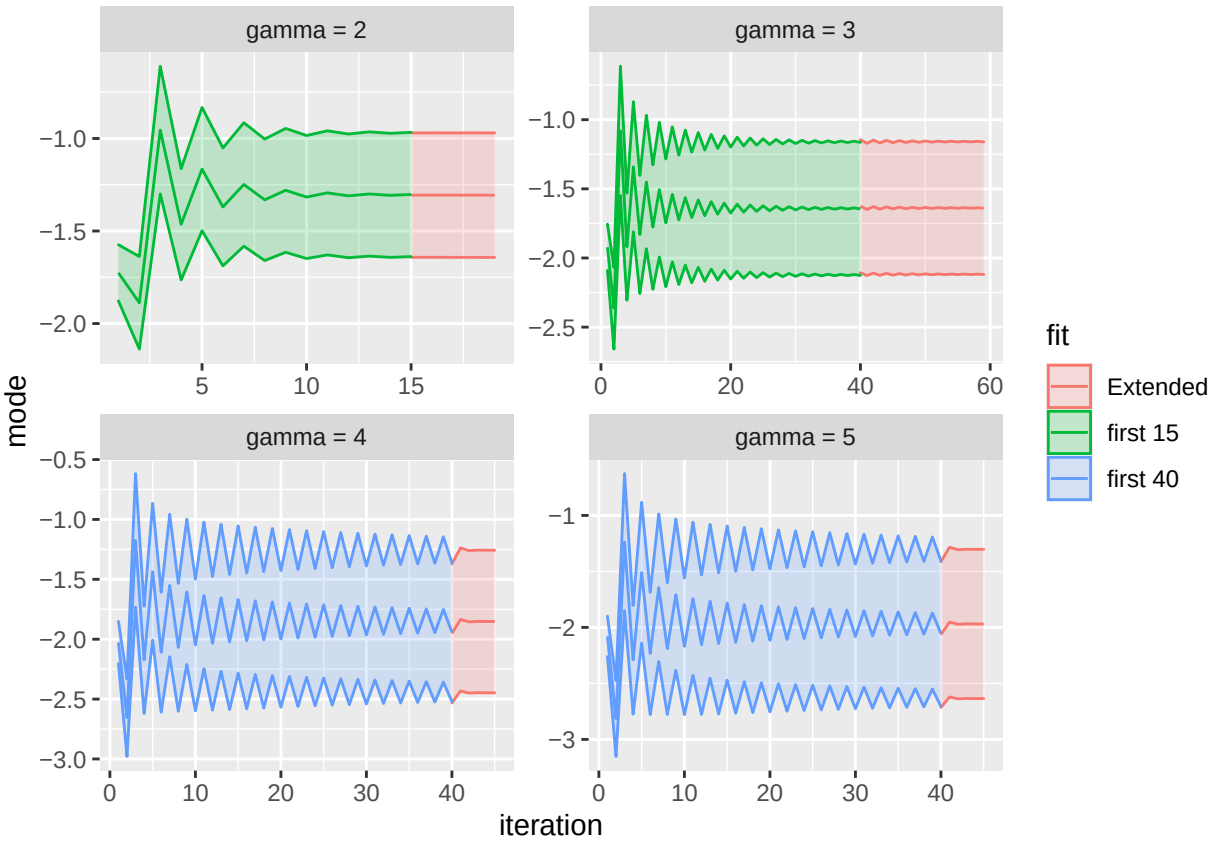
```
                                    inla.mode = 'experimental',
                                    bru_max_iter = new_iter, # change max number of iterations
                                    bru_verbose = 3),
                             bin.strat = 'alt',
                             ncore = my_cores)
#save(fit_ext, file = paste0('utils/fit_sigma.',names(Sigma.list)[S.idx],'.Rds'))
save(fit_ext_s4, file = 'utils/fit_sigma.s4.ro0_ext.Rds')
```

```
load('utils/fit_sigma.s4.ro0_ext.Rds')
load('utils/fit_sigma.s5.ro0_ext.Rds')
fit_ext_s4$bru_iinla$track <- rbind(fit_list_1$s4.ro0$bru_iinla$track[nrow(fit_list_1$s4.ro0$bru_iinla$
                                    fit_ext_s4$bru_iinla$track)
fit_ext_s4$bru_iinla$track$iteration <- fit_ext_s4$bru_iinla$track$iteration +
  max(fit_list_1$s4.ro0$bru_iinla$track$iteration) - 1

fit_ext_s5$bru_iinla$track <- rbind(fit_list_1$s5.ro0$bru_iinla$track[nrow(fit_list_1$s5.ro0$bru_iinla$
                                    fit_ext_s5$bru_iinla$track)
fit_ext_s5$bru_iinla$track$iteration <- fit_ext_s5$bru_iinla$track$iteration +
  max(fit_list_1$s5.ro0$bru_iinla$track$iteration) - 1


df.plot.s <- rbind(df.plot.s2 %>%
                     mutate(model = 'gamma = 2'),
                   df.plot.s3 %>%
                     mutate(model = 'gamma = 3'),
                   fit_list_1$s4.ro0$bru_iinla$track %>%
                       mutate(fit = 'first 40',
                              model = 'gamma = 4'),
                   fit_ext_s4$bru_iinla$track %>%
                       mutate(fit = 'Extended',
                              model = 'gamma = 4'),
                   fit_list_1$s5.ro0$bru_iinla$track %>%
                       mutate(fit = 'first 40',
                              model = 'gamma = 5'),
                   fit_ext_s5$bru_iinla$track %>%
                       mutate(fit = 'Extended',
                              model = 'gamma = 5'))

ggplot(df.plot.s %>%
         filter(effect == 'th.mu'), aes(iteration, mode, color = fit)) +
  geom_line() +
  geom_ribbon(aes(ymin = mode - 2*sd, ymax = mode + 2*sd, fill = fit), alpha = 0.2) +
  facet_wrap(facets = vars(model), scales = 'free')
```




```
fit_list$s4.ro0 <- fit_ext_s4
fit_list$s5.ro0 <- fit_ext_s5
save(fit_list, file = 'utils/fit_list.Rds')
```

Goodness-of-fit

In this section we investigate the goodness-of-fit of each model to decide which one fits the data best. We are going to do that visually. Specifically, we are going to use the Random Time Change Theorem which says that, if $t_1, ..., t_n$ have been observed, then the tranformed quantities $\Lambda(t_1), ..., \Lambda(t_n)$, where,

$$\Lambda(t) = \int_0^t \lambda(x|\mathcal{H}_x)dx$$

form a sample from a Poisson process with unit rate.

The first thing to remark is that the quantity $\Lambda(t)$ can be seen as the expected number of points in $(0, t)$ and thus the sequence $\Lambda(t_1), ..., \Lambda(t_n)$ has to match the cumulative number of observed points at locations $t_1, ..., t_n$. Therefore, the first plot is going to show $\Lambda(t_1), ..., \Lambda(t_n)$ and the cumulative number of points as function of time. For the parameters we are going to use the median of the posterior distribution, confidence bans for $\Lambda(t)$ can be obtained repeating the below procedure using samples from the posterior instead of the median. Below we retrieve the posterior median of the parameters for each of the models to be compared.

```
load('utils/fit_list.Rds')
# retrieve median estimates
bru.est <- lapply(fit_list, \(x) predict(x, data.frame(1),
                    ~ data.frame(mu = link.f$mu(th.mu),
                                 K = link.f$K(th.K),
                                 alpha = link.f$alpha(th.alpha),
```

```
                                c = link.f$cc(th.c),
                                p = link.f$pp(th.p) ) ) )

medians.est <- lapply(bru.est, function(x) sapply(x, \(y) y$median))
bind_rows(medians.est) %>%
  mutate(gamma = 1:5)
```

```
## # A tibble: 5 x 6
##      mu     K alpha      c     p gamma
##   <dbl> <dbl> <dbl>  <dbl> <dbl> <int>
## 1 0.482  3.20 0.705 0.0497  1.28     1
## 2 0.264  4.42 0.858 0.0382  1.27     2
## 3 0.190  5.12 1.01  0.0295  1.25     3
## 4 0.144  5.41 1.08  0.0272  1.25     4
## 5 0.127  5.29 1.16  0.0269  1.26     5
```

Here, for each model, we calculate $\Lambda()$ for the observed points. The plot below shows that the models with $\gamma > 1$ fit the data better than the model assuming $\gamma = 1$. This is probably due to the fact that $\gamma = 1$ induces a small spatial kernel and the model has to consider an higher background rate in trying to match the observed sequence (This can be seen also from the medians which present a decreasing background back for increasing values of γ).

```
tranf.points <- lapply(1:length(medians.est), \(idx) sapply(
  df.bru$ts[-1], \(x) x*medians.est[[idx]][1] +
                       sum(exp(logLambda.h.vec(theta = as.numeric(medians.est[[idx]]),
                                               th = df.bru$ts[df.bru$ts < x],
                                               xh = df.bru$x[df.bru$ts < x],
                                               yh = df.bru$y[df.bru$ts < x],
                                               mh = df.bru$mags[df.bru$ts < x],
                                               M0 = M0,
                                               T1 = 0,
                                               T2 = x,
                                               bdy = bdy,
                                               Sigma = Sigma.list[[idx]])))))
  )
save(tranf.points, file = 'utils/tran.Rds')
```

```
load('utils/tran.Rds')
cums <- sapply(df.bru$ts, \(x) sum(df.bru$ts < x))

ggplot() +
  geom_line(aes(x = df.bru$ts, y = cums)) +
  geom_line(aes(x = df.bru$ts[-1], y = tranf.points[[1]], color = 'gamma = 1')) +
  geom_line(aes(x = df.bru$ts[-1], y = tranf.points[[2]], color = 'gamma = 2')) +
  geom_line(aes(x = df.bru$ts[-1], y = tranf.points[[3]], color = 'gamma = 3')) +
  geom_line(aes(x = df.bru$ts[-1], y = tranf.points[[4]], color = 'gamma = 4')) +
  geom_line(aes(x = df.bru$ts[-1], y = tranf.points[[5]], color = 'gamma = 5')) +
  ylab('Cumulative count') +
  xlab('days')
```

Another way to check the goodness-of-fit is to plot the cumulative counts of $\Lambda(t_1), ..., \Lambda(t_n)$. In theory they should resemble a sample from a unit rate Poisson process and therefore, their cumulative counts has to resemble a straight line. The straighter the line the better the fit. The plot below is in accordance with the previous one, showing increasing *goodness-of-fit* for increasing values of γ. Also, it is clear the difference between $\gamma = 1$ and $\gamma > 1$

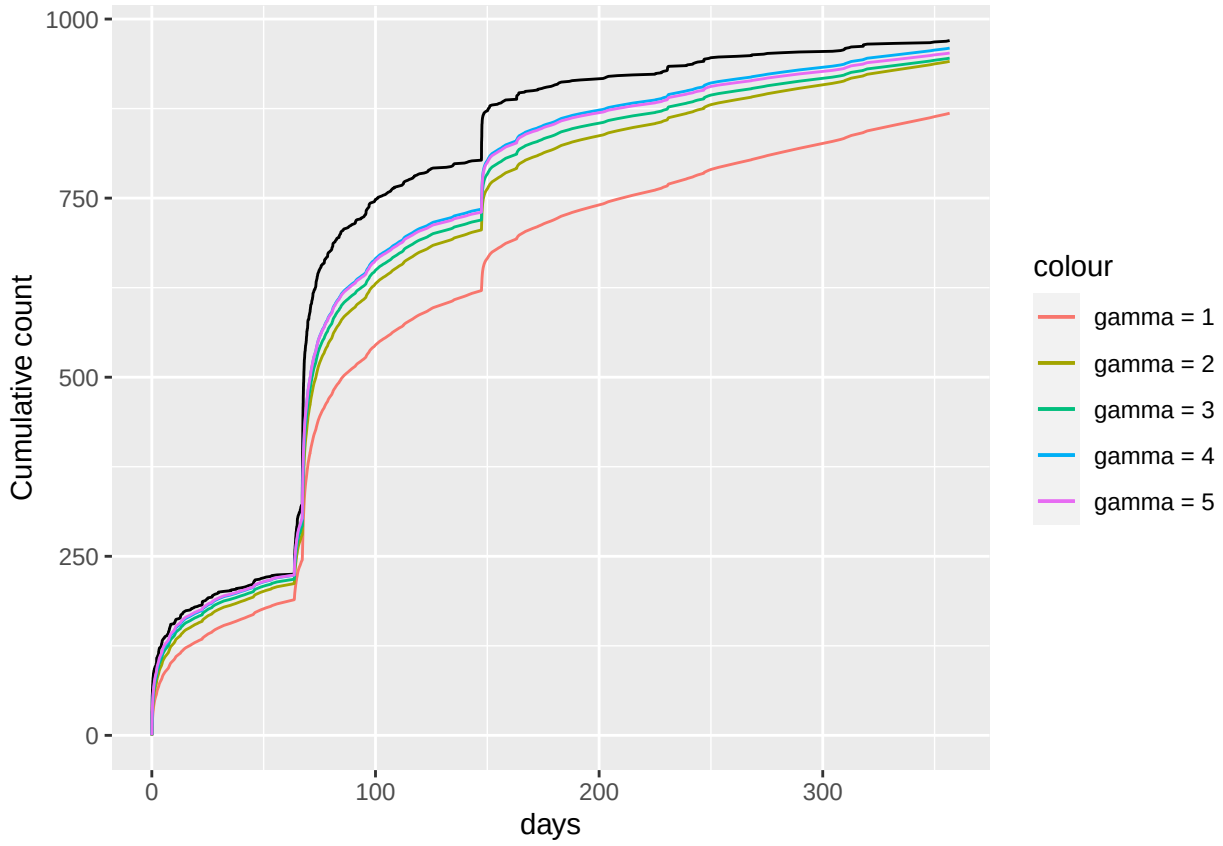


Figure 7: Observed vs expected cumulative number of points

```
cum.points <- lapply(tranf.points, \(x) sapply(x, \(y) sum(x < y)))

ggplot() +
  geom_line(aes(x = tranf.points[[1]], y = cum.points[[1]], color = names(Sigma.list)[1])) +
  geom_line(aes(x = tranf.points[[2]], y = cum.points[[2]], color = names(Sigma.list)[2])) +
  geom_line(aes(x = tranf.points[[3]], y = cum.points[[3]], color = names(Sigma.list)[3])) +
  geom_line(aes(x = tranf.points[[4]], y = cum.points[[4]], color = names(Sigma.list)[4])) +
  geom_line(aes(x = tranf.points[[5]], y = cum.points[[5]], color = names(Sigma.list)[5]))+
  geom_abline(slope = 1, linetype = 2) +
  xlab('Lambda') +
  ylab('Cumulative count')
```

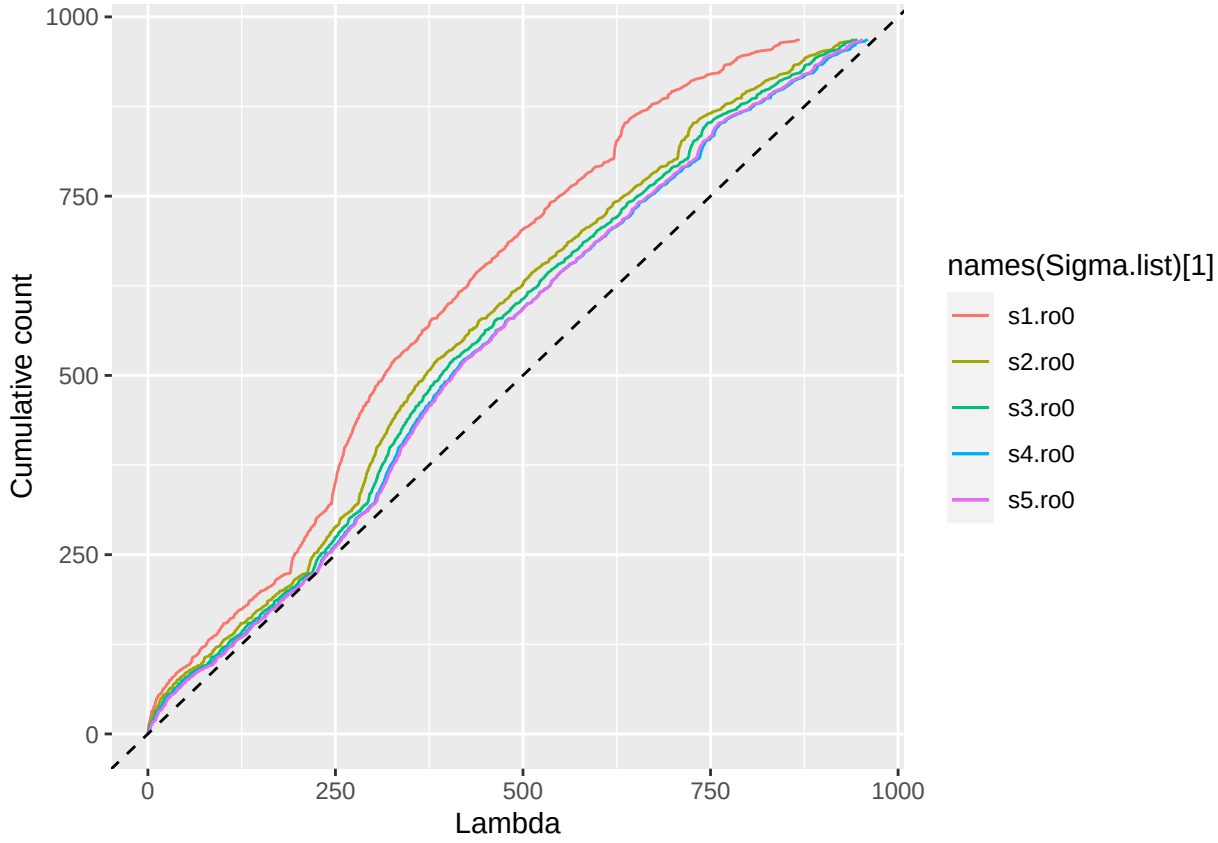


Figure 8: Theretical vs Observed cumulative number of $\Lambda(t)$.

Also, this type of plots give us an idea of what we can expect from the model in the different periods of times. For example, we can see that the first 50 days (circa the first 300 events) are well represented by the models with $\gamma > 1$. On the other hand, all the models have issues in fitting the middle part of the sequence where there is the biggest jump in the cumulative number of points. Missing that part induces underestimation also in the third part of the sequence which would have been well represented if the previous part was.

Stability

An important aspects of Hawkes process model is the stability of the parametrization. Specifically, there are parametrization for which we can have a huge (possibly infinite) number of events in a finite interval of time. This is due, usually, to a strong productivity and large numbers of triggered events. Therefore, if we wish to test the models using catalogue-based forecast we have to be careful with the simulations. A pivotal quantity

in this context is the expected number of events triggered by another. The quantity in question is

$$\psi(\mathbf{x}_h) = \int_W \int_0^\infty g(\mathbf{x}, \mathbf{x}_h) d\mathbf{x} = K \exp\{\alpha(m_h - M_0)\} \frac{c}{p-1} \Phi(\mathbf{s}_h, \Sigma, W)$$

Where $g(\mathbf{x}, \mathbf{x}_h)$ is the triggering function, which in this case is

$$g(\mathbf{x}, \mathbf{x}_h) = K \exp\{\alpha(m_h - M_0)\} \left(1 + \frac{t - t_h}{c}\right)^{-p} \phi(\mathbf{s}, \mathbf{s}_h, \Sigma)$$

Where $\phi(\mathbf{s}, \mathbf{s}_h, \Sigma)$ is a 2D Gaussian density calculated at point $\mathbf{s}$ with mean $\mathbf{s}_h$ and covariance matric Σ. Tha quantity $\Phi(\mathbf{s}_h, \Sigma, W)$ is the integral of $\phi(\mathbf{s}, \mathbf{s}_h, \Sigma)$ over W. The table below reports the expected number of earthquakes triggered by an event with magnitude indicated by the row under the model indicated by the column. We can appreciate the difference in productivity induced by the different models. Again, the values below are calculated using the median of the parameters, we can obtain confidence bans for those quantities using replicate from the posterior instead of the median. Also, we assume that the 2D location of the point is far enough from the boundary to ensure that $\Phi(\mathbf{s}_h, \Sigma, W) = 1$

```
rapid.psi <- function(mh_) {
  exp(c(logLambda.h.vec(medians.est$s1.ro0, th = 0.5, xh = mean(bdy@bbox[1,]),
                                yh =  mean(bdy@bbox[2,]), mh = mh_, M0 = M0, T1 = 0.5, T2 = Inf, bdy = bdy
                Sigma = Sigma.list$s1.ro0),
              logLambda.h.vec(medians.est$s2.ro0, th = 0.5, xh = mean(bdy@bbox[1,]),
                                yh =  mean(bdy@bbox[2,]), mh = mh_, M0 = M0, T1 = 0.5, T2 = Inf, bdy = bdy
                Sigma = Sigma.list$s2.ro0),
              logLambda.h.vec(medians.est$s3.ro0, th = 0.5, xh = mean(bdy@bbox[1,]),
                                yh =  mean(bdy@bbox[2,]), mh = mh_, M0 = M0, T1 = 0.5, T2 = Inf, bdy = bdy
                Sigma = Sigma.list$s3.ro0),
              logLambda.h.vec(medians.est$s4.ro0, th = 0.5, xh = mean(bdy@bbox[1,]),
                                yh =  mean(bdy@bbox[2,]), mh = mh_, M0 = M0, T1 = 0.5, T2 = Inf, bdy = bdy
                Sigma = Sigma.list$s4.ro0),
              logLambda.h.vec(medians.est$s5.ro0, th = 0.5, xh = mean(bdy@bbox[1,]),
                                yh =  mean(bdy@bbox[2,]), mh = mh_, M0 = M0, T1 = 0.5, T2 = Inf, bdy = bdy
                Sigma = Sigma.list$s5.ro0)))
  }


dd <- rbind(rapid.psi(3),
      rapid.psi(4),
      rapid.psi(5),
      rapid.psi(6),
      rapid.psi(7))

rownames(dd) <- paste0('Mw = ', 3:7)
colnames(dd) <- paste0('gamma = ', 1:5)
dd
```

```
##         gamma = 1  gamma = 2  gamma = 3  gamma = 4  gamma = 5
## Mw = 3 0.5634687  0.6310228  0.5991054  0.5855443  0.5514696
## Mw = 4 1.1402402  1.4881884  1.6426674  1.7324311  1.7621916
## Mw = 5 2.3074000  3.5097067  4.5039761  5.1256879  5.6309894
## Mw = 6 4.6692749  8.2772051 12.3493046 15.1652072 17.9935269
## Mw = 7 9.4487854 19.5207549 33.8601545 44.8688086 57.4973568
```

Integrating out the magnitude with respect a distribution $\pi()$, we can retrieve the expected number of points generated by a random event. If this quantity is greater than one the model can be considered unstable and

care has to be taken in performing simulations using this model especially for long periods of time. The GR law induces an exponential distribution with parameter β on the quantity $m - M_0$, and

$$\begin{aligned}
\Psi &= \int_{M_0}^{\infty} K \exp\{\alpha(m - M_0)\} \frac{c}{p-1} \Phi(\mathbf{s}_h, \Sigma, W) \pi(m) dm \\
&= K \frac{c}{p-1} \Phi(\mathbf{s}_h, \Sigma, W) \int_{M_0}^{\infty} \exp\{\alpha(m - M_0)\} \beta \exp\{-\beta(m - M_0)\} dm \\
&= K \frac{c}{p-1} \Phi(\mathbf{s}_h, \Sigma, W) \frac{\beta}{\beta - \alpha} \exp\{(\beta - \alpha) M_0\} \int_{M_0}^{\infty} (\beta - \alpha) \exp\{-(\beta - \alpha) m\} dm \\
&= K \frac{c}{p-1} \Phi(\mathbf{s}_h, \Sigma, W) \frac{\beta}{\beta - \alpha} \exp\{(\beta - \alpha) M_0\} \Pr[Z > M_0]
\end{aligned}$$

Where Z has an exponential distribution with parameter $\beta - \alpha$. A model can be said unstable if $\Psi > 1$, which means that, on average each earthquake triggers more than one event. We can clearly see that, the first and second model are the only one to be clearly stable, while the others present values of Ψ very close to one. For this reason, we are going to use the model with $\gamma = 2$ in the next section.

```
beta.p <- 1/mean(df.bru$mags - M0)

PSI <- foreach(i = medians.est, .combine = c) %do% {
  med.par <- i
  exp.par <- beta.p - med.par[3]
  med.par[2]*med.par[4]/(med.par[5] - 1)*( beta.p/exp.par )*exp(exp.par*M0)*(1 - pexp(M0, exp.par))
}

names(PSI) <- paste0('gamma = ', 1:5)
PSI
```

```
## gamma = 1 gamma = 2 gamma = 3 gamma = 4 gamma = 5
## 0.7731698 0.9427264 0.9810097 1.0078073 1.0009962
```

Generate forecasts and Imposing events.

Here, we explore the possibilities of generating earthquakes sequences with some event imposed. First of all, in this section we use an higher number of samples from the posterior (10000) compared to the weekly forecast example and we are going to generate sequences for the first week only. To do that we need a modified function, which directly write a txt file containing all the simulated catalogues binded together by row and with a column identifying the sample to which the observation belongs to.

```
# clean the environment
rm(fit_list_1)
rm(fit_1)
rm(fit_2)
rm(fit_)
rm(fit_ext_)
rm(fit_ext)
rm(fit_ext_s4)
rm(fit_ext_s5)

# extract gamma = 2 model
fit_s2 <- fit_list$s2.ro0
```

```
# sample 10000 times from the posterior, sampling 10 samples of size 1000 (increase stability wrt 1 of
n.rep <- 10
```

```
s.size <- 1000
s.param.10000.s2 <- foreach(i = 1:n.rep, .combine = cbind) %do% {
  generate(fit_s2, data.frame(x = 0, y = 0, ts = 0, mags = 0),
                   ~ c(th.mu, th.K, th.alpha, th.c, th.p), n.samples = s.size)
}

save(s.param.10000.s2, file = 'utils/sample.param10000.s2.Rds')
```

Below the function to generate a forecast using for each simulated catalogue a sample from the posterior of the parameters. The function takes as input an Inlabru model (`fit_`), the known history of the process before forecastig time interval (`total.data`), imposed observations (`imposed.data`), the extremes of the time forecasting interval (`T1.f, T2.f`), the magnitude of completeness (`M0`), the posterior sample of the parameters in the Inlabru internal scale (`sample.param`), if this is not provided a sample of numerosity `n_sample` is extracted, the spatial area of interest as SpatialPolygon (`bdy_`), the covariance matrix for the spatial kernel (`Sigma.p`), the logarithm of the spatially varying part of the background rate $u(\mathbf{s})$ calculated at the mesh locations (`log.bkg.field_`), the mesh (`mesh_`), the value of the β of the GR law (`beta.p`), the CRS object (`crs_`), the set of link functions (`link.fun`), the name of file that will be written (`file.name`), the file will be saved as (`fore_full/file.name.txt`). The last argument (`time.out.sec`) is the seconds allowed to try to sample. This is done to prevent huge sequences (possibly infinite) to freeze the machine, due to memory or computing overload. The default value is 60 which means that if the sampling takes more than 60 seconds it will stop. This has been chosen because we are going to use this function to produce weekly forecast and usually a minute is more than enough to generate a weekly sample with the present parameters. In general, we strongly suggest to set this parameter according to the problem and the number of events we expect to be generated.

```
library(R.utils)
log.bkg.field <- predict(bru.bkg, mesh_col, ~ Intercept + field)
fore.batch <- function(fit_,
                       total.data,
                       imposed.data = data.frame(),
                       T1.f, T2.f,
                       M0,
                       sample.param = NULL,
                       n_sample = 1000,
                       bdy_ = bdy, Sigma_ = Sigma.p,
                       log.bkg.field_ = log.bkg.field,
                       mesh_ = mesh_col,
                       beta.p = beta.p_,
                       crs_ = italy.crs, ncore = 1,
                       link.fun,
                       file.name,
                       time.out.sec = 60,
                       display_ = TRUE
                       ){
  # if posterior sample of the parameter is NULL extract it
  if(is.null(sample.param)){

    if(display_){
      cat('No posterior samples provided - sampling posterior ', n_sample , ' times', '\n')
    }

    sample.param = generate(fit_, data.frame(x = 0, y = 0, ts = 0, mags = 0),
                    ~ c(th.mu, th.K, th.alpha, th.c, th.pm1), n.samples = n_sample)
```

```
  if(display_){
    cat('Finished posterior sampling', '\n')
  }
}
# select all the data recorded before the forecasting period and merged it with imposed data
known.data = rbind(total.data[total.data$ts < T1.f, ], imposed.data)

if(display_){
    cat('Start simulating ', ncol(sample.param), ' catalogues - timeout ', time.out.sec ,
        ' seconds', '\n', 'Imposed events : ', nrow(imposed.data), ' - Known events : ',
        nrow(known.data), '\n')
}

# for each posterior sample of the parameters
for(i in 1:ncol(sample.param)){

  # transform parameters in the ETAS scale
  th_ = c(link.fun$mu(sample.param[1,i]),
          link.fun$K(sample.param[2,i]),
          link.fun$alpha(sample.param[3,i]),
          link.fun$cc(sample.param[4,i]),
          link.fun$pp(sample.param[5,i])
          )
  # if no known points just sample with Ht = NULL
  if(nrow(known.data) == 0){
    ss = withTimeout(expr = sample.ETAS(theta = th_,
                                        beta.p = beta.p, M0 = M0,
                                        T1 = T1.f, T2 = T2.f,
                                        loglambda.bkg = log.bkg.field_,
                                        mesh.bkg = mesh_, crs_obj = crs_,
                                        bdy = bdy_, Sigma = Sigma_, Ht = NULL, ncore = ncore),
                     timeout = time.out.sec,
                     onTimeout = 'silent')
  } else{ # else set Ht = known.data
    ss = withTimeout(expr = sample.ETAS(theta = th_,
                                        beta.p = beta.p,
                                        M0 = M0,
                                        T1 = T1.f, T2 = T2.f,
                                        loglambda.bkg = log.bkg.field_,
                                        mesh.bkg = mesh_, crs_obj = crs_,
                                        bdy = bdy_, Sigma = Sigma_, Ht = known.data, ncore = ncore),
                     timeout = time.out.sec,
                     onTimeout = 'silent')
  }
  if(is.null(ss)){
    cat('Time out reached at catalogue ', i, '\n')
    next
  } else {
    # combine generations, arrange by time, add catalogue identifier
    ss = bind_rows(ss) %>%
      arrange(ts) %>%
      mutate(cat.idx = i)
    # merge with imposed data if any
```

```
      if(!is.null(imposed.data)){
        imposed.data = imposed.data %>%
        mutate(cat.idx = i,
               gen = 0)
        ss <- rbind(ss, imposed.data)
      }
    }
    # if it is the first sample initialize the file
    if(i == 1){
      write.table(ss, file = paste0('fore_full/', file.name, '.txt'),
                  col.names = TRUE, row.names = FALSE)
    } else { # else append the new rows to the file
      write.table(ss, file = paste0('fore_full/', file.name, '.txt'), append = TRUE,
                  col.names = FALSE, row.names = FALSE)
    }
    # print completeness
    if( display_ & (i/ncol(sample.param)) %% 0.25 == 0){
      cat('completed : ', i/ncol(sample.param), '\n')
    }
  }
  # return the list of samples
}
```

```
load('utils/sample.param10000.s2.Rds')
n.samp <- 1000
```

We are going to forecast only the first week and to show the difference between imposing and not imposing events. We are going to use only the first 1000 samples from the posterior and base the analysis on only 1000 catalogues. This is done to save time on a general Laptop, on a more powerful machine this can be increased. We are going to consider three cases, one with 0 events imposed, one where we impose the first event in the sequence (Magnitude 6.01), one where we impose the first three events with magnitude greater than 5, all recorded in the first week. If this code will be run through this markdown file, we suggest to separate the three forecast in different code chunks to be runned one at the time. Also, we suggest to empty the envirnoment from all but the strictly necessary elements to run the following code.

```
n.samp <- 1000
# 0 event imposed
rr <- fore.batch(fit_ = fit_s2,
                 total.data = df.bru, T1.f = 0, T2.f = 7,
                 sample.param = s.param.10000.s2[,1:n.samp],
                 M0 = M0,
                 link.fun = link.f,
                 log.bkg.field_ =  log.bkg.field$median,
                 beta.p = 1/mean(df.bru$mags - M0),
                 Sigma_ = Sigma.list$s2.ro0,
                 ncore = my_cores,file.name = 'fore.s2_1k')
```

```
# one event imposed
imp.data <- df.bru[1,]
imp.data$bkg <- NULL
rr2 <- fore.batch(fit_ = fit_s2,
                  total.data = df.bru,
                  imposed.data = imp.data, # imposed events
                  T1.f = 0, T2.f = 7,
                  sample.param = s.param.10000.s2[,1:n.samp],
```

```
                  M0 = M0,
                  link.fun = link.f,
                  log.bkg.field_ =  log.bkg.field$median,
                  beta.p = 1/mean(df.bru$mags - M0),
                  Sigma_ = Sigma.list$s2.ro0,
                  ncore = my_cores,
                  file.name = 'fore.s2_1k_imp1')
```

```
# three events imposed
imp.data <- df.bru %>% filter(ts < 7, mags > 5)
imp.data$bkg <- NULL
rr3 <- fore.batch(fit_ = fit_s2,
                  total.data = df.bru,
                  imposed.data = imp.data, # imposed events
                  T1.f = 0, T2.f = 7,
                  M0 = M0,
                  link.fun = link.f,
                  sample.param = s.param.10000.s2[,1:n.samp],
                  log.bkg.field_ =  log.bkg.field$median,
                  beta.p = 1/mean(df.bru$mags - M0),
                  Sigma_ = Sigma.list$s2.ro0,
                  ncore = my_cores,
                  file.name = 'fore.s2_1k_imp3')
```

```
fore.noimp <- read.table(file = 'fore_full/fore.s2_1k.txt', header = TRUE)
fore.imp1 <- read.table(file = 'fore_full/fore.s2_1k_imp1.txt', header = TRUE)
fore.imp3 <- read.table(file = 'fore_full/fore.s2_1k_imp3.txt', header = TRUE)
```

```
# Number of earthquakes
N.noimp <- sapply(1:n.samp, \(x) sum(fore.noimp$cat.idx == x))
N.imp1 <- sapply(1:n.samp, \(x) sum(fore.imp1$cat.idx == x))
N.imp3 <- sapply(1:n.samp, \(x) sum(fore.imp3$cat.idx == x))

ggplot() +
  geom_density(aes(x = N.noimp, y = ..scaled.., color = 'Imp 0')) +
  geom_vline(aes(xintercept = median(N.noimp),color = 'Imp 0'), linetype = 2) +
  geom_density(aes(x = N.imp1, y = ..scaled.., color = 'Imp 1')) +
  geom_vline(aes(xintercept = median(N.imp1),color = 'Imp 1'), linetype = 2) +
  geom_density(aes(x = N.imp3, y = ..scaled.., color = 'Imp 3')) +
  geom_vline(aes(xintercept = median(N.imp3),color = 'Imp 3'), linetype = 2) +
  geom_vline(xintercept = sum(df.bru$ts < 7), linetype = 2) +
  xlim(0, 300)
```

```
# Code to get 2D plot of the sample representing a given quantile of the number of events
quant_ <- 0.9
# Explore distribution
idxx.cat <- which(N.imp3 == quantile(N.imp3, quant_))
if(length(idxx.cat) == 1){
  idxx <- idxx.cat
} else{
  idxx <- sample(x = idxx.cat, size = 1)
}

f.imp <- fore.imp3[fore.imp3$cat.idx == idxx,]
```

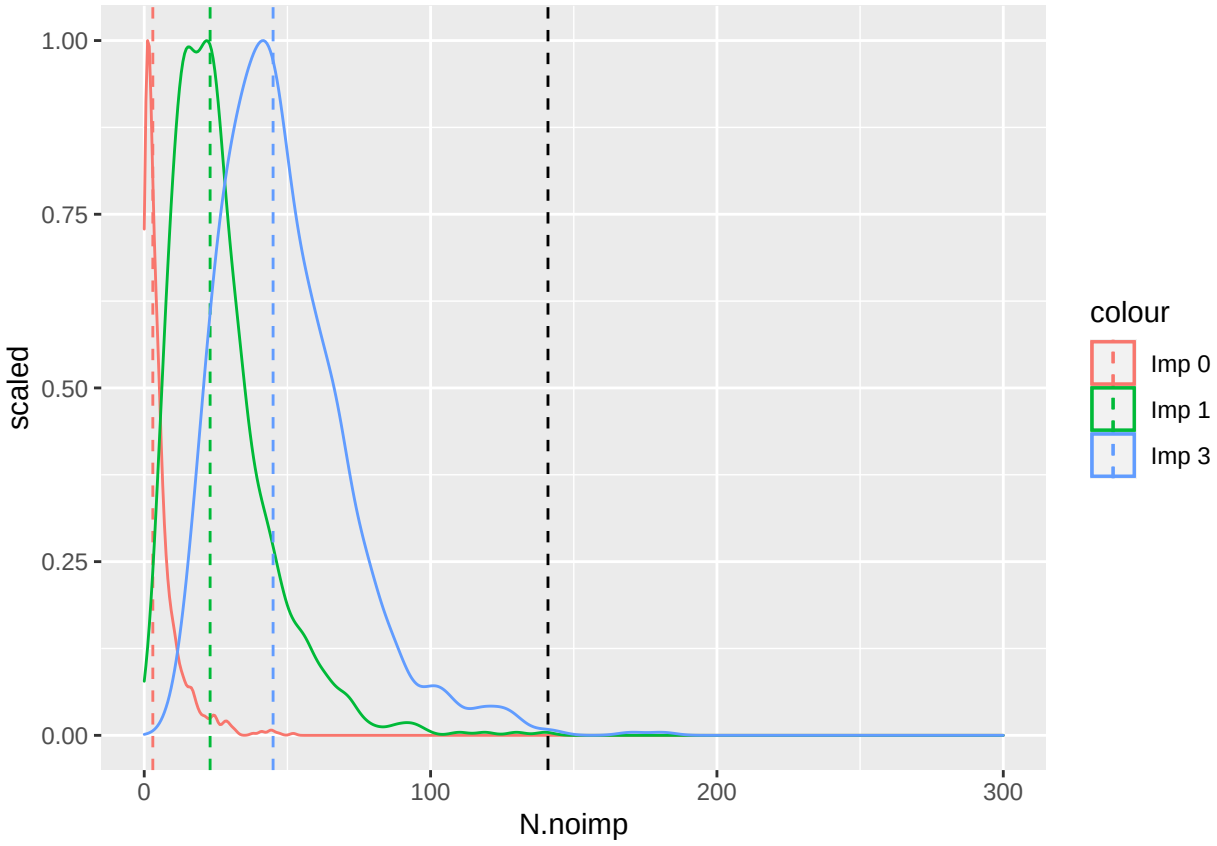


Figure 9: Stable model weekly forecast

```
pl.sim <- ggplot(f.imp, aes(x = x, y = y)) +
  geom_point() +
  geom_point(data = f.imp[f.imp$mags > 5, ], fill = 'red', shape = 23) +
  gg(bdy) +
  labs(title = paste0('Simulated - N = ', nrow(f.imp)))

pl.tr <- ggplot(df.bru[df.bru$ts < 7, ], aes(x = x, y = y)) +
  geom_point() +
  geom_point(data = df.bru[df.bru$ts < 7, ][df.bru$mags > 5, ], fill = 'red', shape = 23) +
  gg(bdy) +
  labs(title = paste0('Observed - N = ', nrow(df.bru[df.bru$ts < 7,])))

multiplot(pl.sim, pl.tr, cols = 2)
```

Comparison with unstable model

```
fit_s5 <- fit_list$s5.ro0

# sample 10000 times from the posterior, sampling 10 samples of size 1000 (increase stability wrt 1 of
n.rep <- 10
s.size <- 1000
s.param.10000.s5 <- foreach(i = 1:n.rep, .combine = cbind) %do% {
  generate(fit_s5, data.frame(x = 0, y = 0, ts = 0, mags = 0),
                   ~ c(th.mu, th.K, th.alpha, th.c, th.p), n.samples = s.size)
}

save(s.param.10000.s5, file = 'utils/sample.param10000.s5.ro0.Rds')
```

```
load('utils/sample.param10000.s5.Rds')
n.samp <- 1000
# 0 event imposed
rr <- fore.batch(fit_ = fit_s5,
                 total.data = df.bru, T1.f = 0, T2.f = 7,
                 sample.param = s.param.10000.s5[,1:n.samp],
                 M0 = M0,
                 link.fun = link.f,
                 log.bkg.field_ =  log.bkg.field$median,
                 beta.p = 1/mean(df.bru$mags - M0),
                 Sigma_ = Sigma.list$s5.ro0,
                 ncore = my_cores,file.name = 'fore.s5_1k')
```

```
# one event imposed
imp.data <- df.bru[1,]
imp.data$bkg <- NULL
rr2 <- fore.batch(fit_ = fit_s5,
                  total.data = df.bru,
                  imposed.data = imp.data, # imposed events
                  T1.f = 0, T2.f = 7,
                  sample.param = s.param.10000.s5[,1:n.samp],
                  M0 = M0,
                  link.fun = link.f,
                  log.bkg.field_ =  log.bkg.field$median,
                  beta.p = 1/mean(df.bru$mags - M0),
```

```
                  Sigma_ = Sigma.list$s5.ro0,
                  ncore = my_cores,
                  file.name = 'fore.s5_1k_imp1')
```

```
# three events imposed
imp.data <- df.bru %>% filter(ts < 7, mags > 5)
imp.data$bkg <- NULL
rr3 <- fore.batch(fit_ = fit_s5,
                  total.data = df.bru,
                  imposed.data = imp.data, # imposed events
                  T1.f = 0, T2.f = 7,
                  M0 = M0,
                  link.fun = link.f,
                  sample.param = s.param.10000.s5[,1:n.samp],
                  log.bkg.field_ =  log.bkg.field$median,
                  beta.p = 1/mean(df.bru$mags - M0),
                  Sigma_ = Sigma.list$s5.ro0,
                  ncore = my_cores,
                  file.name = 'fore.s5_1k_imp3')
```

```
fore.noimp.s5 <- read.table(file = 'fore_full/fore.s5_1k.txt', header = TRUE)
fore.imp1.s5 <- read.table(file = 'fore_full/fore.s5_1k_imp1.txt', header = TRUE)
fore.imp3.s5 <- read.table(file = 'fore_full/fore.s5_1k_imp3.txt', header = TRUE)
```

```
# Number of earthquakes
N.noimp.s5 <- sapply(1:n.samp, \(x) sum(fore.noimp.s5$cat.idx == x))
N.imp1.s5 <- sapply(1:n.samp, \(x) sum(fore.imp1.s5$cat.idx == x))
N.imp3.s5 <- sapply(1:n.samp, \(x) sum(fore.imp3.s5$cat.idx == x))

ggplot() +
  geom_density(aes(x = N.noimp.s5, y = ..scaled.., color = 'Imp 0')) +
  geom_vline(aes(xintercept = median(N.noimp),color = 'Imp 0'), linetype = 2) +
  geom_density(aes(x = N.imp1.s5, y = ..scaled.., color = 'Imp 1')) +
  geom_vline(aes(xintercept = median(N.imp1),color = 'Imp 1'), linetype = 2) +
  geom_density(aes(x = N.imp3.s5, y = ..scaled.., color = 'Imp 3')) +
  geom_vline(aes(xintercept = median(N.imp3),color = 'Imp 3'), linetype = 2) +
  geom_vline(xintercept = sum(df.bru$ts < 7), linetype = 2) +
  xlim(0, 300)
```

```
## Warning: Removed 8 rows containing non-finite values (stat_density).
```

```
# comparison

df.plot <- rbind(data.frame(N = c(N.noimp, N.imp1, N.imp3),
                            imposed = rep(c('No imposed', 'Imposed 1', 'Imposed 3'), each = n.samp),
                            model = 'gamma = 2'),
                 data.frame(N = c(N.noimp.s5, N.imp1.s5, N.imp3.s5),
                            imposed = rep(c('No imposed', 'Imposed 1', 'Imposed 3'), each = n.samp),
                            model = 'gamma = 5'))

ggplot(df.plot, aes(x = N, y = ..scaled..,color = model)) +
  geom_density() +
  geom_vline(xintercept = sum(df.bru$ts < 7), linetype = 2) +
  xlim(0,250) +
  facet_wrap(facets = vars(imposed))
```

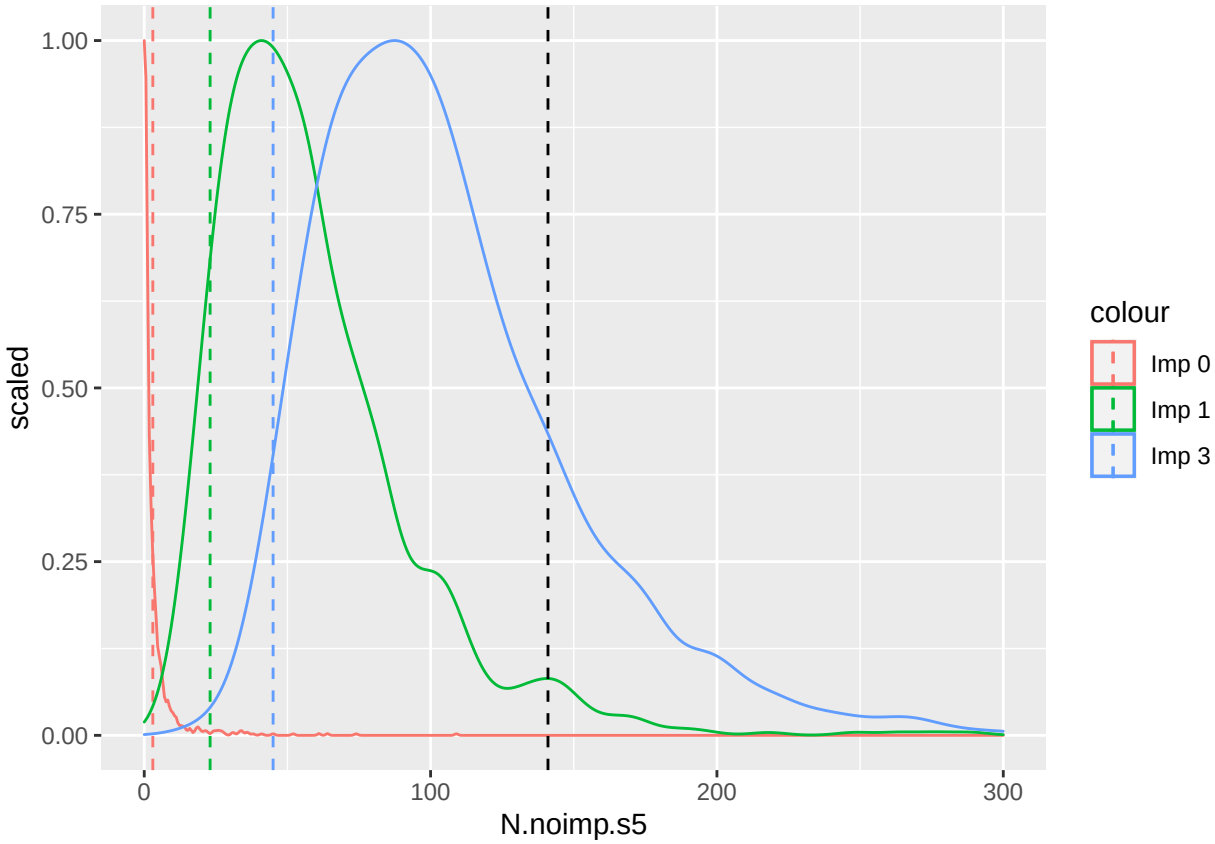


Figure 10: Unstable model first week forecast

```
## Warning: Removed 20 rows containing non-finite values (stat_density).
```

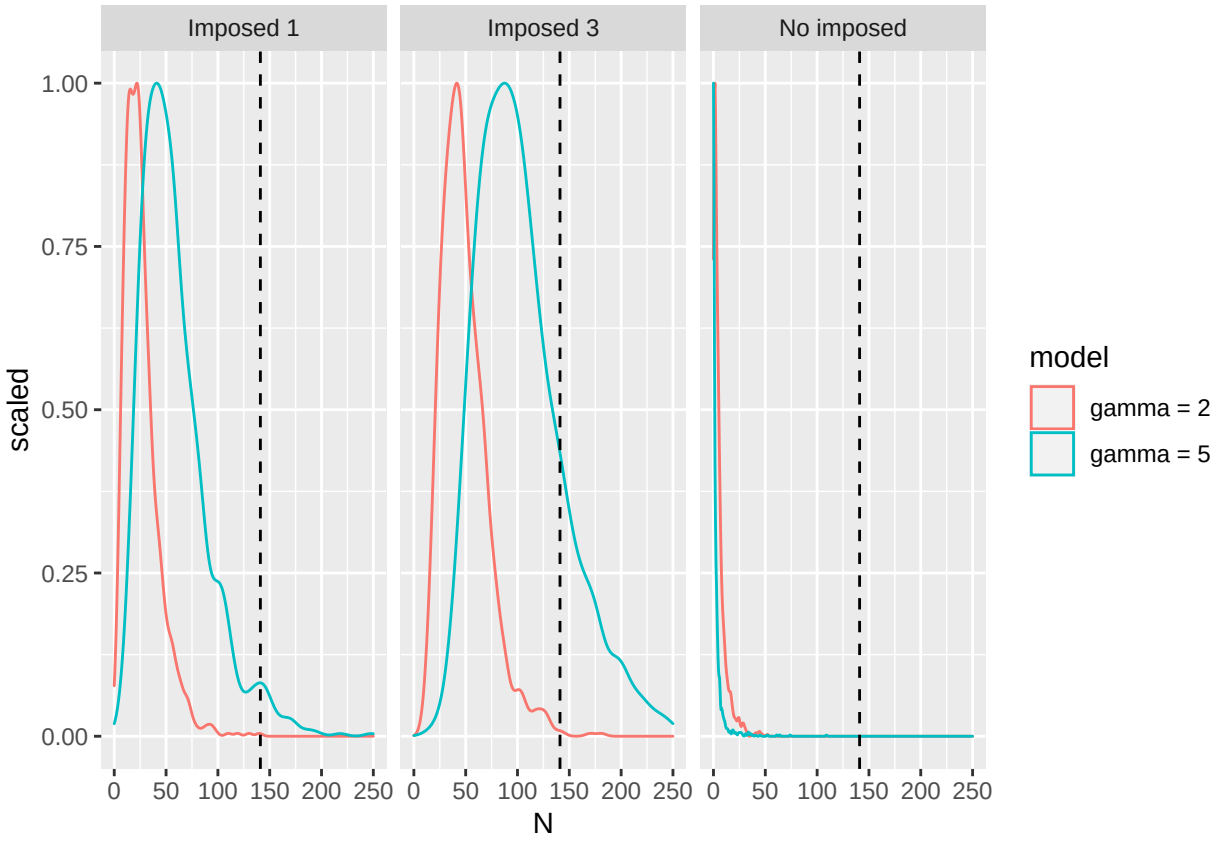


Figure 11: Forecast comparison stable vs unstable