Colfiorito Simulation Example

Francesco Serafini

2022-08-27

Import libraries and System Requirements

We start importing all the libraries and scripts needed for the example.

```r
source('R/modelling_utils.R')
source('R/code_for_etas2.R')
library(maps)
library(inlabru)
library(INLA)
library(sp)
library(sf)
library(raster)
library(rgeos)
library(RColorBrewer)
library(ggplot2)
library(rgdal)
library(dplyr)
library(viridis)
library(R.filesets)
```

Regarding INLA and Inlabru, I am using R vesion 4.2.1, Inlabru version 2.5.2.9000 and INLA version 22.7.23.

Summary

Here a brief summary of the components of this documents.

1. Import data
2. Estimate background field spatial variation
3. Estimate ETAS model's parameters
4. Simulate the parameters of the ETAS model and a catalogue
5. Re-estimate backfround field spatial variation and ETAS model's parameters

The following code will produce a number of warnings messages. This is due to the projection which contains some extra information. Nothing to worry about.

Import data

The first step is importing the data that we are going to use. This section can be divided in the following steps:

1. Import Italy map
2. Create CRS object used for projection later on
3. Load the `data.frame` representing the observed catalogue

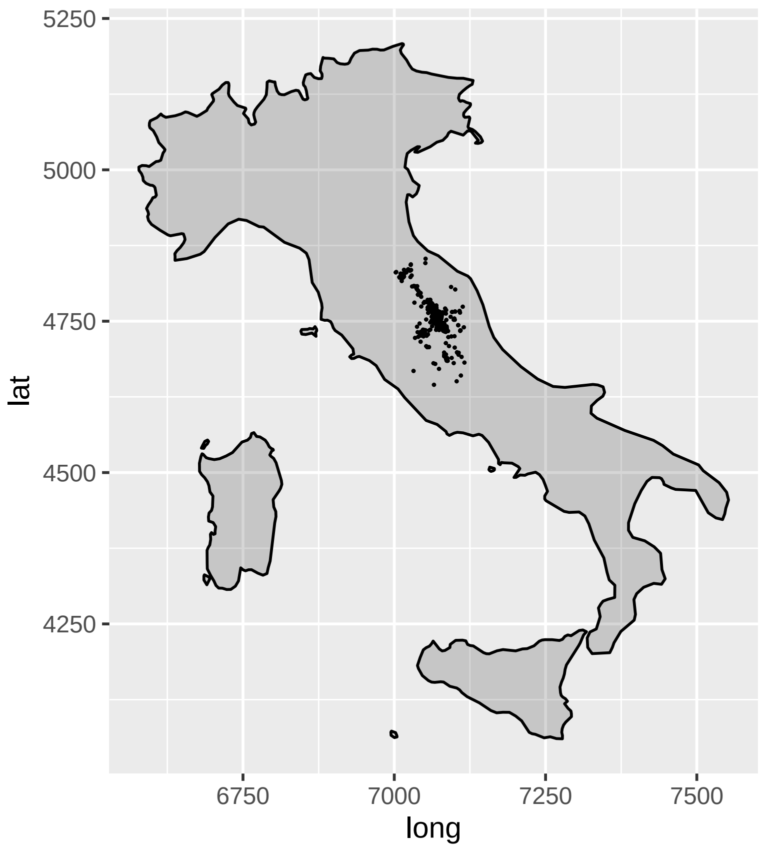
4. Rearrange informations and create a SpatialPointsDataFrame representing the observed catalogue
5. Create a polygon representing the spatial domain

```
# Import Italy map
IM <- map("world", "Italy", plot = FALSE, fill = TRUE)

## Sometimes we want the sf object (tmap)
italy_map <- st_as_sf(IM)
##  And sometimes we want a polygon (ggplot)
italy_map_poly <- as_Spatial(italy_map)

## create CRS objects (ignore warnings)
crs_Ita <- CRS(SRS_string='EPSG:6875')
crs_Ita_km <- fm_crs_set_lengthunit(crs_Ita, "km")
italy_km <- spTransform(italy_map_poly, crs_Ita_km)
```

The data in this case is a subset of the ISIDE catalogue and goes from 1997-01-23 20:50:01 GMT to 1998-12-27 06:30:04 GMT with longitude between 12.0302 and 13.4095 and latitude between 42 and 43.8782. Here notice that the name used for the columns of the created object `df.bru` are mandatory and should not be changed. These are used internally to retrieve the correct column during the calculations for the ETAS model.

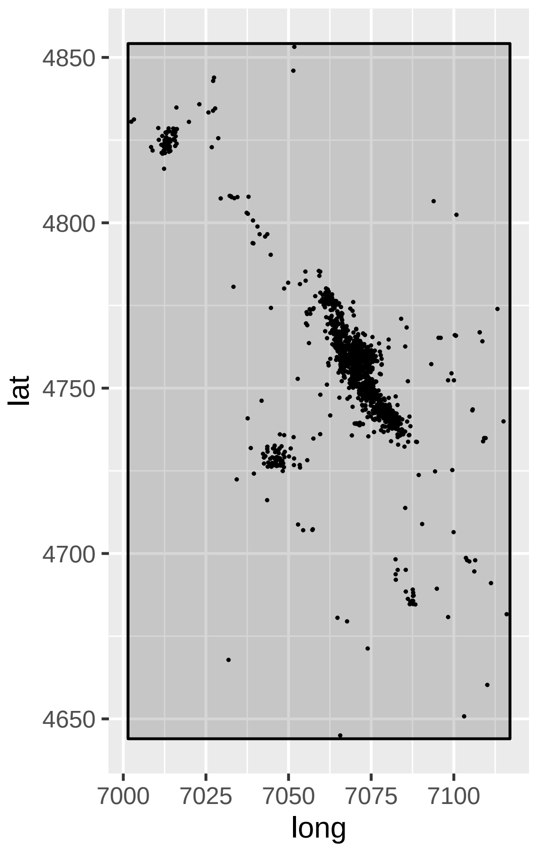
```
## load colfiorito data
load('input/colfiorito.97_99.Rds')

# remove the first event in the catalogue because very isolated.
colfiorito.ds.97_99 <- colfiorito.ds.97_99[,-1]

# find starting date of the catalogue
start.date <- min(colfiorito.ds.97_99$time_date)

# create data.frame (variable names are mandatory)
df.bru <- data.frame(x = colfiorito.ds.97_99$Lon, # x for longitude
                     y = colfiorito.ds.97_99$Lat, # y for latitude
                     ts = as.numeric(difftime(colfiorito.ds.97_99$time_date,
                                              start.date,
                                              units = 'days')), # ts for time (in secs, hours, days)
                     mags = colfiorito.ds.97_99$Mw) # mags for magnitudes

# setting the coordinates of a data.frame it turns it in a SpatialPointsDataFrame
coordinates(df.bru) <- c('x' , 'y')
# set projection
proj4string(df.bru) <- proj4string(italy_map_poly)
# change projection
df.bru.km <- spTransform(df.bru, crs_Ita_km)

# plot it
ggplot() + gg(italy_km) + gg(df.bru.km, size = 0.1) + coord_equal()
```

```
## Regions defined for each Polygons
```



The only thing missing is to create a SpatialPolygon representing the study region, we do that using the `Square.poly` function that can be found in the `modelling_utils` script. As extremes of the rectangular polygon representing the spatial polygon we use the minimum and maximum longitude and latitude of the catalogue and we apply a buffer of 1.

```
# create the spatial domain
colfiorito.col <- Square.poly(x.lim = df.bru.km@bbox[1,],
                              y.lim = df.bru.km@bbox[2,],
                              crs_obj = crs_Ita_km, buff = 1)
# plot it
ggplot() + gg(colfiorito.col) + gg(df.bru.km, size = 0.1) + coord_equal()
```



Estimate background field spatial variation

We are considering an ETAS model with spatially varying background rate, the value at location $\mathbf{s} \in W$

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

and we assume that $u(\mathbf{s})$ is such that its integral over the domain W is 1. In this way, the expected number of background points in the area W is exactly μ. We are going to estimate $u(\mathbf{s})$ in this section, while μ will be determined in the ETAS model section.

This part comprises the following steps:

1. Creating a mesh
2. Creating a SPDE object
3. Fit an LGCP model - it gives us unnormalized $u(\mathbf{s})$
4. Calculate the Integral of $u(\mathbf{s})$
5. Rescale $u(\mathbf{s})$ to integrate to 1

The first step is creating the mesh. The mesh resolution will determine the resolution of the background rate spatial variation, this can be regulated by the `max.edge` argument. The first number regulates the maximum edge length inside the domain while the second number referred to the extension. It is needed to extend the domain because these models present some boundary effect and thus, to be safe in the domain of interest we have to consider a larger domain.

```
# creating a mesh
mesh_col <- inla.mesh.2d(boundary = colfiorito.col, max.edge = c(2, 5),
                         crs = crs_Ita_km)
```

```
# number of mesh vertices - the more points the slower the computation but the higher the resolution
mesh_col$n
```

```
## [1] 18115
```

Then, we need to create a SPDE object. This is because we are modelling the background rate spatial variation as a LGCP model with linear log-internsity given by

$$\log \lambda_{bkg}(\mathbf{s}) = \beta_0 + f(\mathbf{s})$$

Where the function $f()$ is a Gaussian Markov Random Field with Matern Covariance matrix. This simply mean that points close to each other will tend to have a similar value. This is regulated by two parameters (for which we have to set priors), the range and the standard deviation. The range determines the distance at which two points have a week dependency (correlation around 0.1), thus high range means not much variation, while low range means more variations. The standard deviation represents how much far from zero this values can go.

We use PC priors for the range and standard deviation parameters. PC priors are defined by a number and a probability, and for the range is such that the prior probability of the range being below that number is equal to the probability, while it is the opposite for the standard deviation (probability that the standard deviation is above…). We choose to just use priors such that we have a small probability to have small values of the range (so the field will vary smoothly over space) and a low probability of having a large standard deviation, that will lead to a wildly varying field.

In this part, it is possible to also introduce covariates in the model or to use declustered data. We are going to use the same data that we will be used later on for ETAS.

```
# creating an spde object
spde.o <- inla.spde2.pcmatern(mesh_col,
                              prior.sigma = c(0.1, 0.01), # prob(standard deviation > 0.05) = 0.99
                              prior.range = c(0.5, 0.01)) # prob(range < 0.5) = 0.01
# for the range 0.5 ,20 and 50 produce very similar results.
```

Then, we need just to create the component of the models and to pass everything to Inlabru. These steps are usually very safe and do not produce problems. Possible problems may derive from using different projections. The user needs to be sure that the data and the mesh (and any covariate) have the same projection. Different `inla.mode` arguments produce slightly different results, from our experience this is the choice using less memory, in fact, an error that may happen is,

Error: C stack usage 34724900 is too close to the limit

which means that the user doesn't have enough memory on their machine. We suggest to use a mesh with less points and to rerun the model.

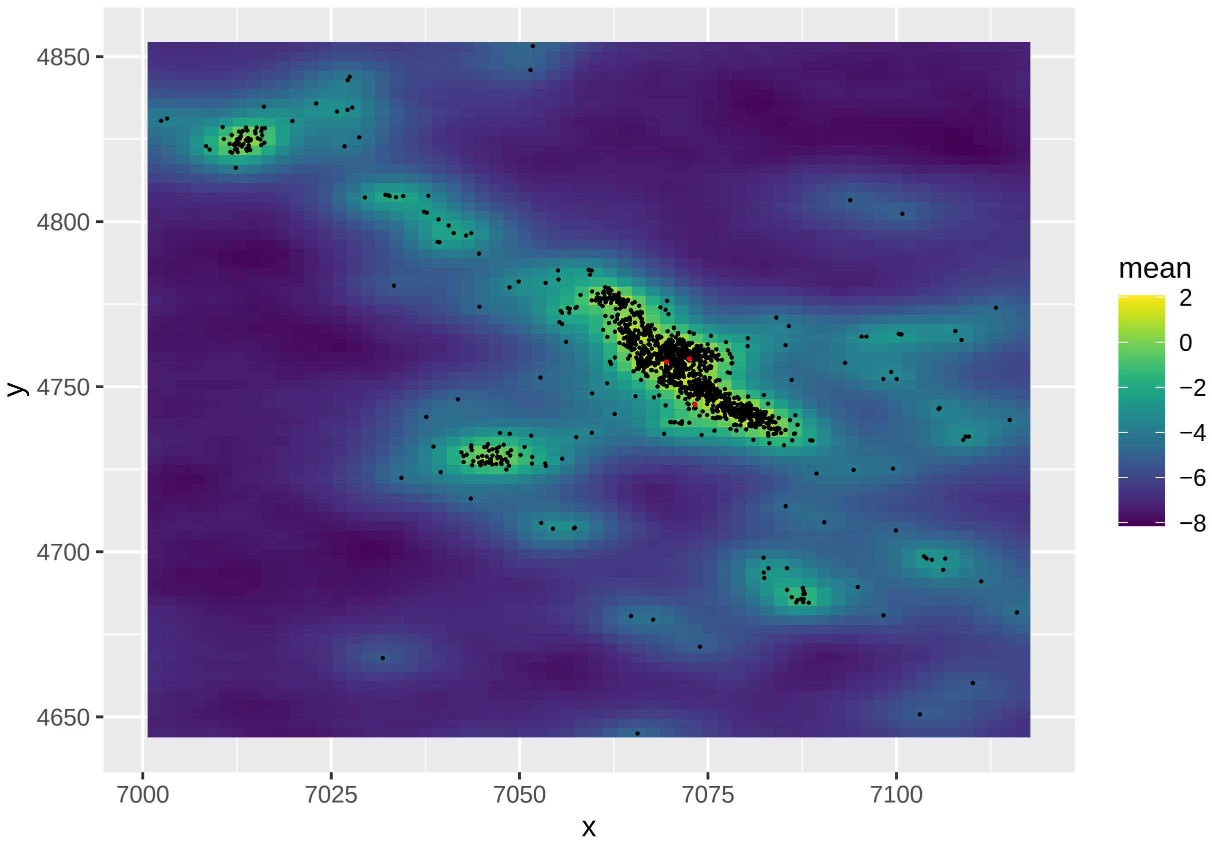
```
# no need to run this piece of code, the models results are stored and loaded in the next code chunk.

# components (if we ave covariates we need to put them here as linear effect like + cov(covariate, mode
cmp <- coordinates ~ field(coordinates, model = spde.o) + Intercept(1)

# fit background model
bru.bkg <- lgcp(components = cmp, # components
                data = df.bru.km, # SpatialPointsDataFrame representing the data
                samplers = colfiorito.col, # spatial domain in which we are interested
                domain = list(coordinates = mesh_col), # extended spatial domain as mesh
                options = list(inla.mode = 'experimental', # inlabru options
                               num.threads = 5, # number of cores
                               control.inla = list(int.strategy = "eb")))
```

```
# save the object - the user needs to create a models_fit folder before this command is runned
save(bru.bkg, file = 'models_fit/bkg.col.Rds')
```

```
load('models_fit/bkg.col.Rds')
```

First of all, let's take a view at the logarithm of the background field. We use the mean of the posterior of the background field as point estimate of the field.

```
# find SpatialPixels object corresponding to the mesh the mask argument specify the spatial domain in w
pix.pred <- pixels(mesh_col, nx = 100, ny = 100, mask = colfiorito.col)
# calculate posterior summaries of the field for each pixel
loglambda.pix <- predict(bru.bkg, pix.pred, ~ field + Intercept)
# plot the mean, if we are interested in other functionals (like the mode, the standard deviation or qu
pl.bkg <- ggplot() + gg(loglambda.pix['mean']) + scale_fill_viridis() + gg(df.bru.km, size = 0.1) +
  gg(df.bru.km[df.bru.km$mags > 5.5,], color = 'red', size = 0.2)
pl.bkg
```



Next, we have to retrieve the value of the background field at the observed points and normalize it to make the integral of the background field over the domain equal to one. This is done in two steps. The first step is to calculate the value of the unnormalized background field at the observed points. Second is to calculate the integral of the background field over the domain and divide the values retrieved at step one for this value. This ensures that the background field will integrate to 1 over the domain.

More specifically, to retrieve the value of the field at the observed points we calculate the value of the field at the mesh points and multiply it for a projection matrix which will give us the value of the background field at the observed points. For the integral, we approximate it numerically. We find a set of integration points and weights. We calculate the value of the field at the integration points and then, the integral is just the sum of the values of the background field at the integration points each one multiplied by the corresponding integration weight.

```
# value of the field 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
# the first argument is the mesh and the second is a matrix of coordinates of points in which we want t
proj <- inla.mesh.project(mesh_col, df.bru.km@coords)

# add unnormalized bkg field column to the data, the %*% is the classic matrix inner-product
df.bru.km$bkg <- as.numeric(proj$A %*% pred.bkg.mesh$mean)

# find integration points and weights associated with mesh points, the samplers argument specify the do
ip <- ipoints(mesh_col, samplers = colfiorito.col)
# find projection matrix from mesh points to integration points coordinates
proj.ip <- inla.mesh.project(mesh_col, ip@coords)
# value of the field at the integration points
bkg.ip <- as.numeric(proj.ip$A %*% pred.bkg.mesh$mean)
# approximate integral
integ.field <- sum(ip$weight*bkg.ip)
# normalize the value of the field at the observed points
df.bru.km$bkg <- df.bru.km$bkg/integ.field
# we save the value data so we do not need to run all this part again if we want to repeat the analysis
save(df.bru.km, file = 'df.bru.colfiorito.Rds')
# to load the SpatialPointsDataFrame just run the following command.
# load('df.bru.colfiorito.Rds')
```

Estimate ETAS model's parameters

This section is composed by the following steps:

1. Define the transformation function for the parameters prior
2. Set the priors
3. Set the initial value of the parameters
4. Set the time domain and the magnitude of completeness
5. Select observations and tranform them in `data.frame`
6. Fit the ETAS model

Now, we have are ready to set up and fit our ETAS model. The following example assumes as triggering function a Gaussian density with covariance matrix given by

$$\Sigma = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}$$

The other parameters are $\mu, K, \alpha, c \geq 0$ and $p \geq 1$. The first step is to set up the priors of the parameters. Specifically, we are going to consider, internally, the parameters as having a standard Gaussian distribution and we are going to transform them in order to have the desired distribution. We use this method so we can ensure the constraints to be respected, while working in a free-constraints environment internally. For example, considering an exponential distribution for the parameter μ ensures that it will be greater than 0, given that negative values, in this case, have zero probability.

The priors need to be setted reflecting information on the parameters. This helps our algorithm because, given the large plateau at the mode that the ETAS model parameters usually exhibit, having a prior information is essential to approach this plateau from the desired direction. This may be useful to get *stable* ETAS parametrizations. Below, some examples of these functions that ensure that the parameters will have the desired prior distribution.

```
# Exponential distribution copula transformation
exp.t <- function(x, rate){
  bru_forward_transformation(qexp, x, rate)
}
# Gamma distribution copula transformation
gamma.t <- function(x, a, b){
  bru_forward_transformation(qgamma, x, a, b)
}
# Uniform distribution copula transformation
unif.t <- function(x, a, b){
  bru_forward_transformation(qunif, x, min = a, max = b)
}
# Log-Gaussian distribution copula transformation
loggaus.t <- function(x, m, s){
  bru_forward_transformation(qlnorm, x, meanlog = m, sdlog = s)
}
```

For the present example we consider the following set of priors which ensures a reasonable prior interquantile range. Again, this depends on the problem at hand and they might be need to be changed changing the area or the sequence of interest.

$$
\begin{aligned}
\mu &\sim \text{Exp}(3)\\
K &\sim \text{LogN}(0, 0.8)\\
\alpha &\sim \text{LogN}(0,\ 0.8)\\
c &\sim \text{Exp}(3)\\
p - 1 &\sim \text{LogN}(-0.5, 0.8)\\
\sigma^2 &\sim \text{Exp}(1)
\end{aligned}
$$

This set of priors is implemented creating the following list of functions. Each element of the list is the transformation required by each parameters. Again, the names of the elements of the list are mandatory and they must not be changed.

```
# list of functions setting the priors for each parameters, the notation \(x) stands for function of x
link.f.exp <-   list(mu = \(x) exp.t(x, 3),
                     K = \(x) loggaus.t(x, 0, 0.8),
                     alpha = \(x) loggaus.t(x, 0, 0.8),
                     cc = \(x) exp.t(x, 3),
                     pp = \(x) 1 + loggaus.t(x, -0.5, 0.8),
                     sigma = \(x) exp.t(x, 1))
```

The last bit is to set the starting points. This has to be done in the internal scale of the parameters. For this is useful to have inverse transformation functions for which we can specify the value that we want for the parameter in the ETAS scale and get the corresponding value in the internal scale. This is necessary because the default starting point is 0 for all the parameters and this might be an unrealistic set of parameters which in turns may cause convergence issues and slow down the algorithm which would need more steps to get to convergence. The set of inverse functions corresponding to the above transformations are

```
# inverse trasnformations useful for starting points
# inverse exponential copula transformation
inv.exp.t <- function(x, rate){
  qnorm(pexp(x, rate))
}
# inverse gamma copula transformation
inv.gamma.t <- function(x, a, b){
```

```r
  qnorm(pgamma(x, a, b))
}
# inverse uniform copula transformation
inv.unif.t <- function(x, a, b){
  qnorm(punif(x, a, b))
}
# inverse log-gaussian copula transformation
inv.loggaus.t <- function(x, m, s){
  qnorm(plnorm(x, meanlog = m, sdlog = s))
}
```

Let's check what the default starting value of the parameters would be. Afterwards, we are going to change only the ones that we need. In general, it is good practice to start from values that we know (from the literature) would be reasonable. The default starting values are

```r
c(mu = link.f.exp$mu(0),
  K = link.f.exp$K(0),
  alpha = link.f.exp$alpha(0),
  c = link.f.exp$cc(0),
  p = link.f.exp$pp(0),
  sigma2 = link.f.exp$sigma(0))
```

```
##        mu         K     alpha         c         p    sigma2
## 0.2310491 1.0000000 1.0000000 0.2310491 1.6065307 0.6931472
```

Those are mostly fine except for the parameter K which would benefit from a higher starting value (again, this depends on the problem at hand). Let's suppose we want to start from a value of K equals 5. We just need to run

```r
inv.loggaus.t(5, 0, 0.8)
```

```
## [1] 2.011797
```

This is the value of the starting point that we need to set. This is done creating a list of values that will be passed to the ETAS model fitting function afterwards. Also here, the names of the list elements are mandatory. We do not need to set also the ones that are equals to the default value, we just report them for completeness and in case one wishes to change them.

```r
init.values <- list(th.mu = 0,
                    th.K = inv.loggaus.t(5, 0, 0.8),
                    th.alpha = 0,
                    th.c = 0,
                    th.p = 0,
                    th.sigma = 0)
```

Next, we need to transform the `SpatialPointsDataFrame` representing the observations in a simple `data.frame` in order to fit the model correctly. We also need to set the boundaries of the time domain T_1 and T_2 (all the observed times needs to be between this two values) and a magnitude of completeness value M_0, we consider only the events with magnitude greater or equal than M_0. I have chosen $M_0 = 2.75$ because otherwise we have too many events and my computer runs out of memory.

```r
df.bru.etas <- as(df.bru.km, 'data.frame')
# set domain parameters
T1 = 0
T2 = ceiling(max(df.bru.etas$ts))
M0 = 2.75
# select data with magnitude above M0
```

```
df.bru.etas <- df.bru.etas[df.bru.etas$mags >= M0, ]
```

Now, we are ready to fit an ETAS model. This is done through the function below. This functions takes a number of arguments in input which are

- `sample.s` = `data.frame` containing the observations, the names of the columns must be the same used in this example.
- `coef_t` and `delta.t` = parameters of the time-binning strategy, lower values of `delta.t` increase the number of bins, while higher values of `coef_t` decreases the number of temporal bins.
- `delta.sp`, `n.layer.sp` and `min.edge.sp` = parameters of the space-binning strategy, lower values of `delta.sp` increase the number of bins, while higher values of `n.layer.sp` increases the number of bins. The parameters `min.edge.sp` corresponds to the minimum possible edge length.
- `link.fun` = list of link functions representing the priors.
- `M0` = magnitude of completeness.
- `T1`, `T2` = boundaries of the time domain.
- `bdy` = `SpatialPolygon` representing the spatial domain, must be a rectangle.
- `bru_opt` = list of inlabru options. More details below
- `bin.strat` = binning strategy for now it is fixed.
- `ncore` = number of cores used in parallel computing

```
# fit the ETAS model
fit_etas_iso <- ETAS.fit.isotropic(sample.s = df.bru.etas, # data.frame representing data points
                                   coef_t = 1, # time-binning parameter
                                   delta.t = 0.1, # time-binning parameter
                                   delta.sp = 0.1, # space-binning parameter
                                   n.layer.sp = 8, # space-binning parameter
                                   min.edge.sp = 0.1, # space-binning parameter
                                   link.fun = link.f.exp, # list of prior transformations functions
                                   M0 = M0, # magnitude of completeness
                                   T1 = T1, T2 = T2 , # extremes of time interval
                                   bdy = colfiorito.col, # SpatialPolygon representing study region (re
                                   bru.opt = # bru options
                                     list(
                                       bru_method = list(max_step = 0.2), # max step
                                       inla.mode = 'experimental', # inla mode
                                       bru_max_iter = 40, # max number of iterations -
                                       bru_verbose = 3, # verbose changes what inlabru prints
                                       bru_initial = init.values), # initial value of the ETAS paramete
                                   bin.strat = 'exp', # strategy for the bins
                                   ncore = 5) # set number of cores
# save the model
save(fit_etas_iso, file = 'models_fit/fit.ama_iso.Rds')
```

```
load('models_fit/fit.ama_iso.Rds')
```

We give here more details on the inlabru options which are essential to the convergence of the algorithm. The most important are the `max_step` and the `bru_max_iter` arguments. The first one determines *how much* the parameters value can change from step to step. The algorithm without setting this parameter it usually make too large jumps and ends up in a loop. We can recognize to be in this case when, in the Inlabru output, from one iteration to another, the values of the *Max difference from previous step* start going in a loop and repeat themselves. In these cases, lowering the `max_step` argument may be helpful. The other parameter, `bru_max_iter` determines the maximum number of iterations. This is needed because changing the `max_step` argument invalidate the internal Inlabru stopping rule and the algorithm will do exactly the number of steps defined with `bru_max_iter`. The best choice for this is to set it to the value at which the variation from one iteration to the other becomes less than 1% of the SD. For the other, `inla.mode` needs to be equal to exponential.

Possible two most common issues with this function are: the loop problem described above and memory errors. Usually, memory errors can manifest in three ways. The function returns an error saying that the stack is too close to the limit. Otherwise, the R session just freezes or becomes white or R abort the session. When this happens reducing changing the parameters to have less spatio-temporal bins, or using less data points may be helpful. On the other hand, there is a trade-off. Having too few spatio-temporal bins may induce the loop behavior we described earlier. Multiple trials are usually needed to find the right balance.

Simulate the parameters of the ETAS model and a catalogue

Here, we want to simulate a value from the posterior distribution of the parameters and use those values to simulate a catalogue. This is useful if we have to provide catalogue-based forecast in which case is sufficient to repeat this steps for the number of simulated catalogues composing the forecast. The steps in this section are

1. Sample from the parameters' posterior distribution.
2. Find Maximum Likelihood estimate of the GR-law's b-value
3. Retrieve the value of the logarithm of the background field at the mesh points.
4. Simulate and store a catalogue.

For the first step we can use the inlabru `generate` function. This extract a sample of the parameters from the posterior distribution in the internal scale which would need to be trasformed in the ETAS scale to perform the sampling.

```r
# simulate parameters value
params <- generate(fit_etas_iso, df.bru.etas, ~ c(th.mu, th.K, th.alpha, th.c, th.p, th.sigma),
                   n.samples =  1, # number of samples from the posterior
                   seed = 1) # seed to obtain always the same results

# tranform them in ETAS scale
etas.params <- c(link.f.exp$mu(params[1]),
                 link.f.exp$K(params[2]),
                 link.f.exp$alpha(params[3]),
                 link.f.exp$cc(params[4]),
                 link.f.exp$pp(params[5]),
                 link.f.exp$sigma(params[6]))
# create Sigma as matrix
Sigma.sim <- matrix(c(etas.params[6], 0,
                      0, etas.params[6]), byrow = TRUE, ncol = 2)

# maximum likelihood estimate beta for GR law
beta.ml <- 1/mean(df.bru.etas$mags - M0) # classic beta (GR law) estimator
```

Then, we retrieve the value of the spatial variation of the background field $u(\mathbf{s})$ introduced before. In this case, we don't need to be normalized, indeed we are going to use it just to determine the location of the background events (the number is determined by the parameter μ) and thus, the value of the integral of $u(\mathbf{s})$ is irrelavant here.

```r
# value of the log bkg field at the mesh points
bkg.field.mesh <- predict(bru.bkg, mesh_col, ~ field + Intercept)
```

Next, we simulate a catalogue using the `sample.ETAS` function. This function takes as input:

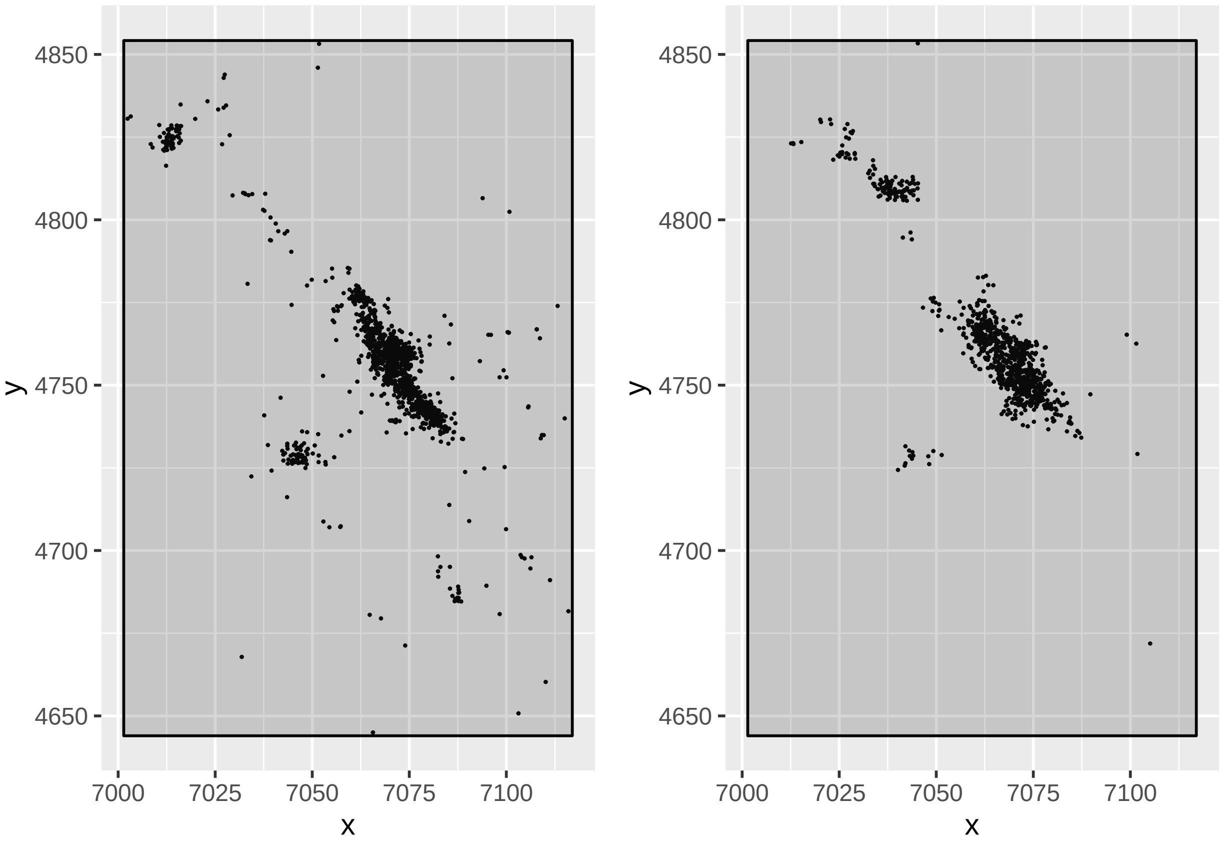
- `theta` = value of the parameters μ, K, α, c, p in the ETAS scale (the order matters!).
- `Sigma` = covariance matrix for the spatial triggering function.
- `beta.p` = parameter of the GR law ($b = \beta/\log(10)$).
- `M0` = magnitude of completeness.

- `T1` and `T2` = boundaries of the time domain.
- `bdy` = spatial domain as `SpatialPolygon`.
- `loglambda.bkg` = value of the spatial variation of the background field at the mesh nodes.
- `mesh.bkg` = mesh at which the spatial variation of the background field is calculated.
- `crs_obj` = CRS object, it is the projection in which the simulated catalogue will be.
- `ncore` = number of cores.

The function returns a `list`. Each element of the list is a generation of points, meaning that the first element are the background events, the second element are the events generated by the background ones, the third element are the events generated by the second element, and so on. We merge them together in the last step of the code chunk.

```
# load the catalogue afterwards if you want to have the same as in my example.
# generate catalogue
# it returns a list of Generations - EXPLAIN output
sim.cat <- sample.ETAS(theta = etas.params[1:5],
                       Sigma = Sigma.sim,
                       beta.p = beta.ml,
                       M0 = M0,
                       T1 = 0, T2 = T2,
                       bdy = colfiorito.col,
                       loglambda.bkg = bkg.field.mesh$mean,
                       mesh.bkg = mesh_col,
                       crs_obj = crs_Ita_km, ncore = 5)
# merge different generations and arrange events by time
sim.cat <- bind_rows(sim.cat) %>%
  arrange(ts)
save(sim.cat, file = 'sim.cat.Rds')
```

```
# load the simulated catalogue
load('sim.cat.Rds')
# to plot is best to trasform it in SpatialPointsDataFrame
sim.cat.sp <- sim.cat
coordinates(sim.cat.sp) <- c('x', 'y') # this line turns it into a SpatialPointsDataFrame
# set projetcion
proj4string(sim.cat.sp) <- crs_Ita_km

# plot
pl.true <- ggplot() + gg(df.bru.km, size = 0.1) + gg(colfiorito.col)
pl.sim <- ggplot() + gg(sim.cat.sp, size = 0.1) + gg(colfiorito.col)
# combine the plots
multiplot(pl.true, pl.sim, cols = 2)
```

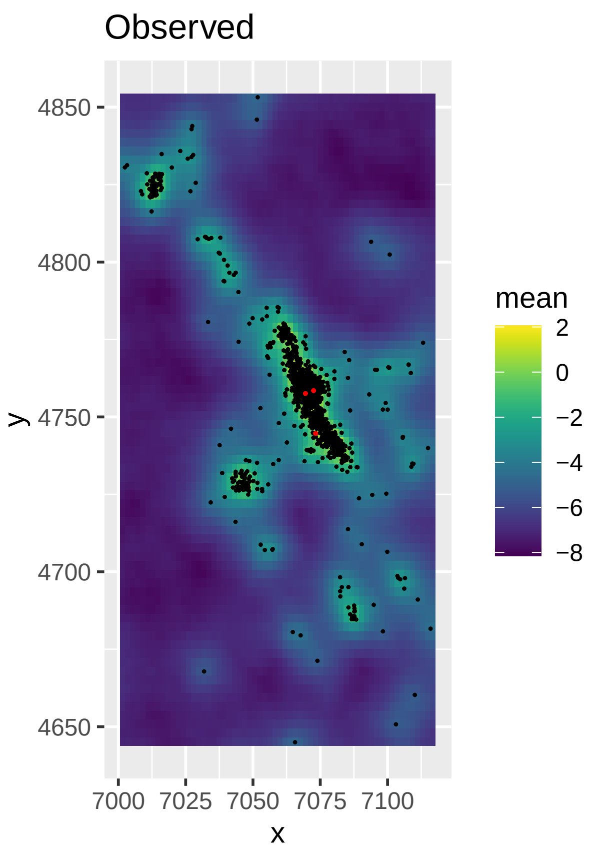


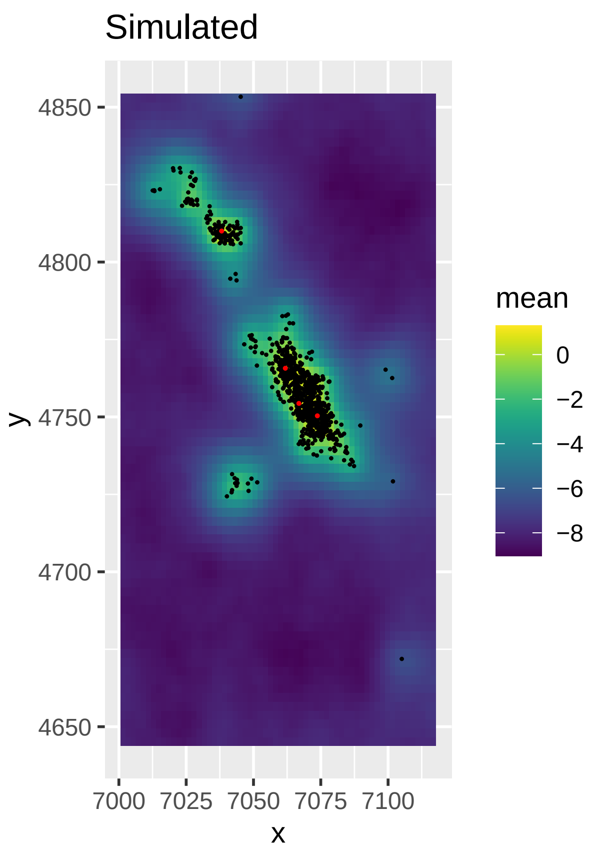
Re-estimate backfround field spatial variation and ETAS model's parameters

Now, we just have to re-do all the steps we have done before on this new catalogue and check if we retrieve correctly the value of the parameters.

```
# not need to run just load the file in the end
# estimate bkg field again
bru.bkg.sample <- lgcp(components = cmp,
                       data = sim.cat.sp,
                       samplers = colfiorito.col,
                       domain = list(coordinates = mesh_col),
                       options = list(inla.mode = 'experimental',
                               num.threads = 1,
                               control.inla = list(int.strategy = "eb")))
save(bru.bkg.sample, file = 'models_fit/bkg.sim.Rds')
```

```
load('models_fit/bkg.sim.Rds')
# find SpatialPixelsDataFrame for plotting
sample.bkg <- predict(bru.bkg.sample, pix.pred, ~ field + Intercept)

pl.bkg.sample <- ggplot() + gg(sample.bkg) + scale_fill_viridis() + gg(sim.cat.sp, size = 0.1) +
  gg(sim.cat.sp[sim.cat.sp$mags > 5.5,], color = 'red', size = 0.2) + labs(title = 'Simulated')

multiplot(pl.bkg + labs(title = 'Observed'), pl.bkg.sample, cols = 2)
```





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

# create projection to find value of bkg field at the obs points
proj.sample <- inla.mesh.project(mesh_col, sim.cat.sp@coords)

# add a bkg column to the data
sim.cat$bkg <- as.numeric(proj.sample$A %*% pred.bkg.mesh.sample$mean)

# find integration weights for the mesh points
ip <- ipoints(mesh_col, samplers = colfiorito.col)
# find projection matrix from mesh points to integration points coordinates
proj.ip <- inla.mesh.project(mesh_col, ip@coords)
# value of the field at the integration points
bkg.ip <- as.numeric(proj.ip$A %*% pred.bkg.mesh.sample$mean)

# approximate integral
integ.field.sample <- sum(ip$weight*bkg.ip)

# normalize bkg values
sim.cat$bkg <- sim.cat$bkg/integ.field.sample
#save(sim.cat, file = 'sim.cat.with.bkg.Rds')

# fit the etas model
fit_etas_sample <- ETAS.fit.isotropic(sample.s = sim.cat,
                                      coef_t = 1,
                                      delta.t = 0.1,
```

```r
                                   delta.sp = 0.1,
                                   n.layer.sp = 8,
                                   min.edge.sp = 0.1,
                                   link.fun = link.f.exp,
                                   M0 = M0,
                                   T1 = T1, T2 = T2 ,
                                   bdy = colfiorito.col,
                                   bru.opt =
                                     list(
                                       bru_method = list(max_step = 0.2),
                                       inla.mode = 'experimental',
                                       bru_max_iter = 40,
                                       bru_verbose = 3,
                                       bru_initial = init.values),
                                   bin.strat = 'exp',
                                   ncore = 5)
save(fit_etas_sample, file = 'models_fit/fit.etas.sample.Rds')
```

After having fitted a model, it useful to check if it arrived at convergence. In fact, the number of steps needed to reach converge with one catalogue may be not enough with a different one. The way to check if we arrived at converge is to print the last lines of the log element of the model.

```r
# load model
load('models_fit/fit.etas.sample.Rds')
# check convergence
tail(fit_etas_sample$bru_iinla$log)
```

We can see that the maximum deviation from previous step in this case is 16.5% which is above 1. We need to run some more iterations. To avoid to restart from iteration one we can extract the value of the last state of the parameters and use it as starting point. In this way, we just need to run the additional steps.

```r
# retrieve last state - 40 because we did 40 iterations before
last.state <- fit_etas_sample$bru_iinla$states[[40]]

# rerun the model for 27 more iterations
fit_etas_sample <- ETAS.fit.isotropic(sample.s = sim.cat,
                                   coef_t = 1,
                                   delta.t = 0.1,
                                   delta.sp = 0.1,
                                   n.layer.sp = 8,
                                   min.edge.sp = 0.1,
                                   link.fun = link.f.exp,
                                   M0 = M0,
                                   T1 = T1, T2 = T2 ,
                                   bdy = colfiorito.col,
                                   bru.opt =
                                     list(
                                       bru_method = list(max_step = 0.2),
                                       inla.mode = 'experimental',
                                       bru_max_iter = 27,  # changed
                                       bru_verbose = 3,
                                       bru_initial = last.state), # changed
                                   bin.strat = 'exp',
                                   ncore = 5)
save(fit_etas_sample, file = 'models_fit/fit.etas.sample.Rds')
```

To check if the posterior estimates of the parameters we take the $0.025, 0.5, 0.975$ quantiles of the posterior distribution of the parameters in the internal scale and we compare it with the parameters values used to simulate the data. We do the same with the model fitted on the observed data. There is significant overlap between the posterior intervals for the parameters in the two cases which prove that our algorithm is coherent and is able to identify the parameters of the model.

```
load('models_fit/fit.etas.sample.Rds')
# compare with model on the simulated data
cbind(`0.025quant` =fit_etas_sample$summary.fixed$`0.025quant`,
      `0.5quant` =fit_etas_sample$summary.fixed$`0.5quant`,
      `0.975quant` =fit_etas_sample$summary.fixed$`0.975quant`,
      True = as.numeric(params))
```

```
##      0.025quant   0.5quant  0.975quant         True
## [1,] -0.7399829 -0.3857660 -0.03154917 -0.379598542
## [2,]  1.2010922  1.4733298  1.74556745  1.389765448
## [3,] -0.2785783 -0.1574850 -0.03639183 -0.007356891
## [4,] -1.0117273 -0.8278199 -0.64391248 -0.956602990
## [5,] -0.4551145 -0.2813640 -0.10761338 -0.517083870
## [6,]  1.0463614  1.2747737  1.50318606  1.281050774
```

```
# compare with model on the observed data
cbind(`0.025quant` = fit_etas_iso$summary.fixed$`0.025quant`,
      `0.5quant` = fit_etas_iso$summary.fixed$`0.5quant`,
      `0.975quant` = fit_etas_iso$summary.fixed$`0.975quant`,
      True = as.numeric(params))
```

```
##      0.025quant    0.5quant  0.975quant         True
## [1,] -0.8232678 -0.43958011 -0.05589243 -0.379598542
## [2,]  1.1617003  1.45280730  1.74391428  1.389765448
## [3,] -0.1167821 -0.00385799  0.10906607 -0.007356891
## [4,] -1.1598735 -0.97614062 -0.79240770 -0.956602990
## [5,] -0.7369043 -0.55553147 -0.37415867 -0.517083870
## [6,]  0.9946444  1.20993069  1.42521695  1.281050774
```