Hawkes process with Inlabru - Tutorial

Francesco Serafini

2022-09-26

Scope

In this tutorial we show how to implement Hawkes process models with the R-package `inlabru`. We consider a specific Hawkes process model, namely, the temporal *Epidemic-Type Aftershock Sequence* (ETAS) model, which is the most widely used model for short term forecast of seismicity. We use this model to illustrate our technique but it can be generalized to different type of Hawkes process model.

Introduction - ETAS model

The temporal ETAS model is a marked Hawkes process model, where the marking variable is the magnitude of the events. The ETAS model is composed by two parts: a Hawkes process model for the location of he events (in this case the location is the time but can be generalized to include also space) and an independent distribution for the magnitude. In fact, in seismology, the magnitude distribution is usually assumed to be independent of space and time. Therefore, we focus on the Hawkes process part of the model and the parameters of the magnitude distribution are determined independently. Our technique can be generalized to include the case of a space-time varying marking variable distribution, the only thing that changes is the functional form of the functions to be provided which has to include also the marking variable distribution.

The ETAS Hawkes process part conditional intensity evaluated a generic time point $t \in (T_1, T_2), T_1, T_2 \geq 0, T_1 < T_2$ having observed the events $\mathcal{H}_t = \{(t_h, m_h) : t_h < t, m_h > M_0, \forall h = 1, ..., N(t) - 1\}$, where $M_0 \geq 0$ is the minimum magnitude in the catalogue, and $N(t)$ is the counting process associated with the Hawkes process representing the number of events recorded up to time t (included), is given by:

$$\lambda_{Hawkes}(t|\mathcal{H}_t) = \mu + \sum_{(t_h,m_h)\in\mathcal{H}_t} K e^{\alpha(m_h - M_0)} \left(\frac{t - t_h}{c} + 1\right)^{-p}$$

The parameters of the model are $\mu, K, \alpha, c \geq 0$ and $p > 1$. Different parametrizations of the ETAS model exists. We focus on this one because it has proven to be the most suitable parametrization for our method.

Calling $\pi(m)$ the magnitude distribution, the conditional intensity of the Hawkes process model is given by:

$$\lambda_{ETAS}(t, m|\mathcal{H}_t) = \left(\mu + \sum_{(t_h,m_h)\in\mathcal{H}_t} K e^{\alpha(m_h - M_0)} \left(\frac{t - t_h}{c} + 1\right)^{-p}\right) \pi(m)$$

In the following sections, we only focus on $\lambda_{Hawkes}(\cdot)$ and, for ease of notation, we drop the subscript, so that the expression $\lambda(\cdot)$ refers to the ETAS Hawkes process part. Instead, we keep using $\lambda_{ETAS}(\cdot)$ when referring to the full ETAS model conditional intensity.

Introduction - Log-likelihood approximation

Our technique is based on decomposing the Hawkes process log-likelihood in multiple parts, the sum of which returns the exact log-likelihood. Our idea is to linearly approximate the single components with respect to the posterior mode and to apply the *Integrated Nested Laplace Approximation* (INLA) method to perform inference on the parameters of the model. Both the linearisation and the optimization to find the posterior mode are performed internally by `inlabru` and the user only has to provide the functions representing the log-likelihood components to be approximated. More details on the methodology to find the posterior mode can be found at the link.

The Hawkes process log-likelihood having observed a catalogue of events $\mathcal{H} = \{(t_i, m_i) : t_i \in [T_1, T_2], m_i \in (M_0, \infty)\}$ is given by:

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{H}) = -\Lambda(T_1, T_2) + \sum_{(t_i,m_i)\in\mathcal{H}} \log \lambda(t_i|\mathcal{H}_{t_i})$$

Where $\mathcal{H}_{t_i} = \{(t_h, m_h) \in \mathcal{H} : t_h < t_i\}$ is the history up to time t_i, and

$$\begin{aligned}
\Lambda(T_1, T_2) &= \int_{T_1}^{T_2} \lambda(t|\mathcal{H}_t)dt \\
&= (T_2 - T_1)\mu + \sum_{(t_i,m_i)\in\mathcal{H}} \int_{T_1}^{T_2} K e^{\alpha(m_i - M_0)} \left(\frac{t - t_i}{c} + 1\right)^{-p} \mathbb{I}(t > t_i)dt \\
&= (T_2 - T_1)\mu + \sum_{(t_i,m_i)\in\mathcal{H}} K e^{\alpha(m_i - M_0)} \int_{\max(T_1,t_i)}^{T_2} \left(\frac{t - t_i}{c} + 1\right)^{-p} dt \\
&= (T_2 - T_1)\mu + \sum_{(t_i,m_i)\in\mathcal{H}} K e^{\alpha(m_i - M_0)} \frac{c}{p-1} \left(\left(\frac{\max(t_i, T_1) - t_i}{c} + 1\right)^{1-p} - \left(\frac{T_2 - t_i}{c} + 1\right)^{1-p} \right) \\
&= \Lambda_0(T_1, T_2) + \sum_{(t_i,m_i)\in\mathcal{H}} \Lambda_i(T_1, T_2)
\end{aligned}$$

The above integral can be considered as the sum of two parts, the number of background events $\Lambda_0(T_1, T_2)$ and the remaining summation which is referred as the sum of the number of triggered events by each event t_i, namely $\Lambda_i(T_1, T_2)$. Our technique approximate the integral linearising the functions $\Lambda_0(T_1, T_2)$ and $\Lambda_i(T_1, T_2)$. This means that the resulting approximate integral is the sum of $|\mathcal{H}| + 1$ linear functions of the parameters. However, we discovered that this approximation is not accurate enough. To increase the accuracy of the approximation, for each integral $\Lambda_i(T_1, T_2)$, we consider a partition of the integration interval $[\max(T_1, t_i), T_2]$ in B_i bins, $t_0^{(b_i)}, ..., t_{B_i}^{(b_i)}$ such that $t_0^{(b_i)} = \max(T_1, t_i)$, $t_{B_i}^{(b_i)} = T_2$ and $t_j^{(b_i)} < t_k^{(b_i)}$ if $j < k$. Doing this, the integral becomes

$$\Lambda(T_1, T_2) = \Lambda_0(T_1, T_2) + \sum_{(t_i,m_i)\in\mathcal{H}} \sum_{j=0}^{B_i - 1} \Lambda_i(t_j^{(b_i)}, t_{j+1}^{(b_i)})$$

In this way, the integral is decomposed in $\sum_i B_i + 1 > |\mathcal{H}| + 1$ terms providing a more accurate approximation.

The Hawkes process log-likelihood can be written as

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{H}) = -\Lambda_0(T_1, T_2) - \sum_{(t_i,m_i)\in\mathcal{H}} \sum_{j=0}^{B_i - 1} \Lambda_i(t_j^{(b_i)}, t_{j+1}^{(b_i)}) + \sum_{(t_i,m_i)\in\mathcal{H}} \log \lambda(t_i|\mathcal{H}_{t_i})$$

Our approximation method consists in linearising the logarithm of the single elements of the summations with respect to the posterior mode $\boldsymbol{\theta}^*$, the approximate log-likelihood is then,

$$\overline{\mathcal{L}}(\boldsymbol{\theta}|\mathcal{H}) = -\exp\{\overline{\log\Lambda_0}(T_1, T_2)\} - \sum_{(t_i,m_i)\in\mathcal{H}}\sum_{j=0}^{B_i-1}\exp\{\overline{\log\Lambda_i}(t_j^{(b_i)}, t_{j+1}^{(b_i)})\} + \sum_{(t_i,m_i)\in\mathcal{H}}\overline{\log\lambda}(t_i|\mathcal{H}_{t_i})$$

where for a generic function $f(\boldsymbol{\theta})$ with argument $\boldsymbol{\theta}\in\Theta\subset\mathbb{R}^m$, the linearised version with respect a point θ^* is given by:

$$\overline{f}(\boldsymbol{\theta}) = f(\boldsymbol{\theta}^*) + \sum_{k=1}^{m}(\theta_k - \theta_k^*)\frac{\partial}{\partial\theta_k}f(\boldsymbol{\theta})\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}$$

Functions to be provided by the user

The functions that need to be provided by the user can be grouped in two families: functions to be linearised and functions for the binning. The first group consists of the functions illustrated in the previous section, while the latter consists in all the function needed to retrieve the values $t_0^{(b_i)}, ..., t_{B_i}^{(b_i)}$ for each point t_i.

Functions to be linearized

The linearisation and the finding of the mode $\boldsymbol{\theta}^*$ are performed automatically by the `inlabru` package. The user only has to provide the functions to be linearised. Specifically, we need to provide the logarithm of the functions needed to approximate the integral and the logarithm of the conditional intensity. More formally, for this example, the user needs to provide

$$\log\Lambda_0(T_1, T_2) = \log(T_2 - T_1) + \log(\mu)$$

$$\log\Lambda_i(t_j^{(b_i)}, t_{j+1}^{(b_i)}) = \log(K) + \alpha(m_i - M_0) + \log\left(\frac{c}{p-1}\right) + \log\left(\left(\frac{t_j^{(b_i)} - t_i}{c} + 1\right)^{1-p} - \left(\frac{t_{j+1}^{(b_i)} - t_i}{c} + 1\right)^{1-p}\right)$$

and

$$\log\lambda(t|\mathcal{H}_t) = \log\left(\mu + \sum_{(t_h,m_h)\in\mathcal{H}_t} Ke^{\alpha(m_h - M_0)}\left(\frac{t - t_h}{c} + 1\right)^{-p}\right)$$

Below we provide the code for $\Lambda_i(t_j^{(b_i)}, t_{j+1}^{(b_i)})$ and $\log\lambda(t|\mathcal{H}_t)$.

```r
# time triggering function - K exp(alpha (mh - M0)) sum_h ((t - t_h)/c + 1)^(-p)
gt <- function(th, t, ti, mi, M0){
  output <- rep(0,length(ti))
  t.diff <- t - ti
  neg <- t.diff <= 0
  if(sum(!neg) > 0){
    log.out <- log(th[2]) + th[3]*(mi[!neg] - M0)  - th[5]*log(1 + t.diff[!neg]/th[4])
    output[!neg] <- exp(log.out)
  }
  else{
    output
  }
  output
}

# Hawkes process conditional intensity
lambda_ <- function(th, t, ti.v, mi.v, M0){
  if(is.null(ti.v) | all(ti.v > t)){
    th[1]
  }
  th[1] + sum(gt(th, t, ti.v, mi.v, M0))
}

# logarithm of the number of triggered observations by point (ti, mi) in the interval [T1, T2]
log.Lambda_h <- function(th, ti, mi, M0, T1, T2){
  th <- as.numeric(th)
  T.low <- pmax(T1, ti)
  gamma.l <- (T.low - ti)/th[4]
  gamma.u <- (T2 - ti)/th[4]
  w.l <- (1 + gamma.l)^(1-th[5])
  w.u <- (1 + gamma.u)^(1-th[5])
  # output
  log(th[2]) + th[3]*(mi - M0) + log(th[4]) - log(th[5] - 1) + log1p(w.l - 1) + log1p(-w.u/w.l)
}
```

Functions for binning

The user also needs to create decide the binning strategy. The reason why the binning strategy is fundamental is that the number of bins determines (up to a certain limit) the accuracy of the approximation. Considering more bins enhances the accuracy of the approximation but increases the computational time because it increases the number of quantities to be approximated. Also, we cannot reduce the approximation error to zero, and the numerical value of the integral in each bin goes to zero increasing the number of bins which can be problematic in terms of numerical stability. We found out that for the ETAS model considered here, having around 10 bins for each observed point is usually enough, and that is best considering bins of increasing length. In fact, the function

$$g_t(t, t_i, m_i) = Ke^{\alpha(m_i - M_0)}\left(\frac{t - t_i}{c} + 1\right)^{-p}\mathbb{I}(t > t_i)$$

varies the most for value of t close to t_i and become almost constant moving away from t_i. This means that we need shorter bins close to t_i, to capture the variation, and less bins far from t_i.

We choose a binning strategy defined by three parameters $\Delta, \delta > 0$, and $n_{max} \in \mathbb{N}^+$. The bins relative to the observed point t_i are given by

$$t_i,\, t_i + \Delta,\, t_i + \Delta(1+\delta),\, t_i + \Delta(1+\delta)^2, ..., t_i + \Delta(1+\delta)^{n_i}, T_2$$

where, $n_i \leq n_{max}$ is the maximum $n \in \mathbb{N}^+$ such that $t_i + \Delta(1+\delta)^n < T_2$. The parameter Δ regulates the length of the first bin, δ regulates has the length increases in subsequent bins, the value n_{max} regulates the maximum number of bins.

This strategy presents two advantages. The first is that we have shorter bins close to the point t_i and longer bins as we move away from that point. The second is that the first (or second, or third, or any) bin has the same length for all points. This is useful because the integral in a bin is a function of the bin length and not of the position of the bin. This means that the we need to calculate the value of the integral in the first (second, third, or any) bin only one time and it would be the same for all observations. This reduces significantly the computational time.

Below the functions to implement to find the bins given a point, T_2 and the binning strategy strategy's parameters, and the function to calculate the integral in each bin efficiently.

```r
# find points defining the bins for an observed point
breaks_exp <- function(tt_, T2_, coef_ = 2, delta_, N_exp_ = 10){

  tt_breaks <- tt_ + delta_*((1 + coef_)^(0:N_exp_))
  tt_breaks <- tt_breaks[tt_breaks < T2]
  if(T2_ - tt_ < delta_){
    return(c(tt_, T2_))
  }
  if(T2 - tt_breaks[length(tt_breaks)] < delta_){
    tt_breaks[length(tt_breaks)] = T2_
  }
  if(tt_breaks[length(tt_breaks)] < T2_){
    tt_breaks <- c(tt_breaks, T2_)
  }
  return(c(tt_,tt_breaks))
}

# build the grid - this takes in input a data.frame with columns ts (times), magnitudes, and idx.p (eve

time.grid <- function(data.point, coef.t, delta.t,
                            T2., displaygrid = FALSE, N.exp.){
  tt. <- data.point$ts
  idx.p <- data.point$idx.p
  # time bins
  # find bins break points
  t_b <- breaks_exp(tt., T2., coef_ = coef.t, delta_ = delta.t, N_exp_ = N.exp.)
  time.bins <- data.frame(t.start = t_b[-length(t_b)],
                          t.end = t_b[-1]) %>%
    mutate(t.bin.name = paste0(round(t.start,3),'-',round(t.end,3)))
  if(nrow(time.bins) - 1 == 0){
    time.bins$t.ref_layer = paste0('last-',idx.p)
  }
  else{
    time.bins$t.ref_layer = c(1:(nrow(time.bins) - 1), paste0('last-',idx.p))
  }
  cbind(time.bins, data.point, row.names = NULL)
}

# calculate the value of the integral with respect to t of ((t - t_i)/c + 1)^(-p) for each row of time.
It_df <- function(param_, time.df){
  tth <- as.numeric(time.df$ts)
  T1b <- as.numeric(time.df$t.start)
  T2b <- as.numeric(time.df$t.end)
  param_c <- param_[4]
  param_p <- param_[5]
  T.l <- pmax(tth, T1b)
  fun.l <- (1 + (T.l - tth)/param_c)^(1-param_p)
  fun.u <- (1 + (T2b - tth)/param_c)^(1-param_p)
  ( param_c/ (param_p - 1) )* ( fun.l - fun.u )
}

# compute the integral of the integral in the bins represented by list.input_$time.sel and repeat them

compute.grid <- function(param., list.input_){
  It.vec <- It_df(param_ = param., time.df = list.input_$time.sel)
  It.vec[list.input_$Imapping]
}
```

Priors

We know have to set the priors for the parameters. The INLA method is designed for Latent Gaussian models, which means that all the unobservable parameters has to be Gaussian. This seems in contrast with the positivity constraint of the ETAS parameters μ, K, α, c, p, but we have a solution.

In fact, internally, INLA works with parameters having a standard Gaussian distribution has prior and they are transformed to have the desired distribution. More formally, we call $\boldsymbol{\theta}$ the parameter in the internal scale and $\eta(\boldsymbol{\theta})$ the parameters in the ETAS scale. Assuming that θ has a standard Gaussian distribution with cumulative distribution function (CDF) $\Phi(\theta)$, and calling F_Y^{-1} the inverse of the CDF of the target distribution for the parameter, we have that if

$$\eta(\theta) = F_Y^{-1}(\Phi(\theta))$$

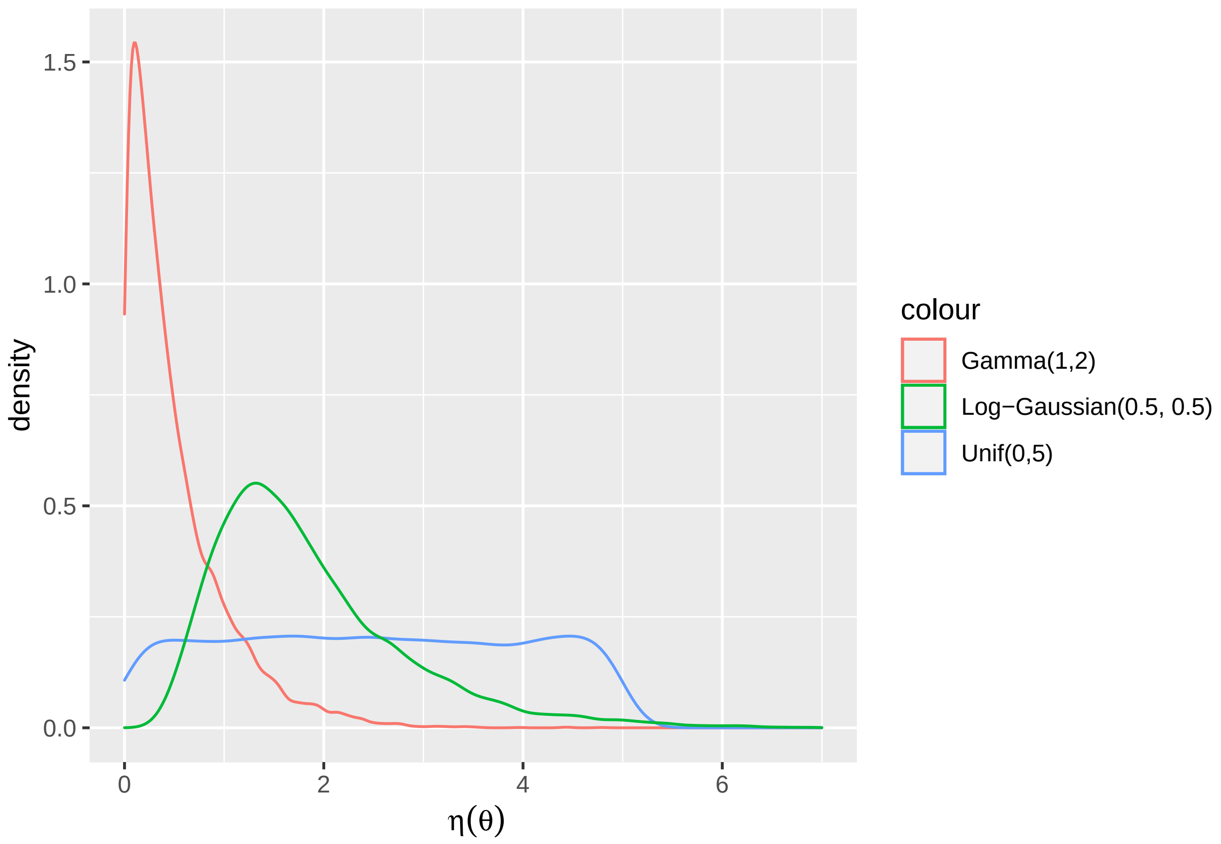
then, the quantity $\eta(\theta)$ has a distribution with CDF $F_Y(\cdot)$.

The package `inlabru` provides function to easily implement such transformation. The function is called `bru_forward_transformation` and takes in input the quantile function of the target distribution and its parameters. Below we report three example of transformations such that the parameter in the ETAS scale have a Gamma, Uniform, or Log-Gaussian distribution. We show the empirical density obtained transforming the same sample of values from a standard Gaussian distribution.

```r
# gamma copula transformation
gamma.t <- function(x, a, b){
  bru_forward_transformation(qgamma, x, a, b)
}
# uniform copula transformation
unif.t <- function(x, a, b){
  bru_forward_transformation(qunif, x, min = a, max = b)
}
# log-gaussian copula transformation
loggaus.t <- function(x, m, s){
  bru_forward_transformation(qlnorm, x, meanlog = m, sdlog = s)
}

set.seed(1)
# extract sample from standard gaussian distribution
st.gaussian.sample <- rnorm(10000)
# transform in Gamma(1,2)
gamma.values <- gamma.t(st.gaussian.sample, 1, 2)
# transform in Uniform(0,1)
unif.values <- unif.t(st.gaussian.sample, 0, 5)
# transform in LogGaussian(0.5, 1)
loggaus.values <- loggaus.t(st.gaussian.sample, 0.5, 0.5)
```

```r
ggplot() +
  geom_density(aes(x = gamma.values, color = 'Gamma(1,2)')) +
  geom_density(aes(x = unif.values, color = 'Unif(0,5)')) +
  geom_density(aes(x = loggaus.values, color = 'Log-Gaussian(0.5, 0.5)')) +
  xlim(0,7) +
  xlab(~eta(theta))
```



Import data and set up

In this section we load the data on which the ETAS model will be fitted and we set up the priors for the parameters. We consider weakly informative priors for this example, specifically,

$$\begin{aligned} \mu &\sim \text{Gamma}(0.1, 0.1) \\ K &\sim \text{LogGaus}(-1, 2) \\ \alpha &\sim \text{Unif}(0, 10) \\ c &\sim \text{Unif}(0, 10) \\ p &\sim \text{Unif}(1, 10) \end{aligned}$$

We create a list of functions in which each element of the list is the transformation function needed for the parameter to have the desired prior distribution.

```r
link.f.be <- list(mu = \(x) gamma.t(x, 0.1, 0.1),
                  K = \(x) loggaus.t(x, -1, 2),
                  alpha = \(x) unif.t(x, 0, 10),
                  c_ = \(x) unif.t(x, 0, 10),
                  p = \(x) unif.t(x, 1, 10))
```

Then, we need to import and prepare the data. We are going to consider data relative to the 2016 Amatrice seismic sequence. The data goes from the 24th of August 2016, date of the 6.01 earthquake that started the sequence to the 15th of August 2017. We count the time as difference in days between the first event in the sequence and the subsequent ones for a total of 357 days. Therefore we set $T_1 = 0$ and $T_2 = 357$. The minimum magnitude is set to be $M_0 = 2.99$. The data needed by `inlabru` to fit the model consists of three columns, the time in day passed from the first event (`ts`), the magnitude of the event (`magnitudes`) and an event identifier (`idx.p`). If we include also space there will be two more columns representing the 2D coordinates of the event's epicentre.

```r
# set domain parameters
M0 = 2.99
T1 = 0; T2 = 357

# import data
dd.ama <- read.csv2(file = 'data_M3.0.csv', header = TRUE, sep = ',') %>%
  mutate(time_date = as.POSIXct(paste0(year,'-',month,'-',day,' ',hr,':',min,':',sec)),
         time.diff = as.numeric(difftime(time_date, min(time_date) - 1, units = 'days')),
         Mw = as.numeric(Mw),
         Lon = as.numeric(Lon),
         Lat = as.numeric(Lat),
         Mw.class = cut(Mw, breaks = c(M0, 5, 7))) %>%
  arrange(time_date)

# data for Inlabru
data.bru <- data.frame(ts = dd.ama$time.diff,
                       magnitudes = dd.ama$Mw) %>%
  mutate(idx.p = 1:nrow(dd.ama))
```

Fitting the model

To fit a Hawkes process model, we need a method to construct the approximated likelihood starting from the factors illustrate in the decomposition. For point process models, we can leverage on a special feature of the `INLA` R-package: it allows us to fit improper Poisson models with exposure equal zero. More in details, having observed a sequence of counts $c_1, ..., c_n$, at times $t_1, ..., t_n$, with exposure $E_1, ..., E_n$, the log-likelihood used internally by the `INLA` package for a model with intensity $\log \lambda_P$ is given by:

$$\mathcal{L}_{Pois} = -\sum_{i=1}^{n} \exp\{\overline{\log \lambda_P}(t_i)\}E_i + \sum_{i=1}^{n} \overline{\log \lambda_p(t_i)}c_i$$

We can represent the different parts of the Hawkes process log-likelihood using the right combination of counts and exposures. For example, in order to represent the expected number of background events $\Lambda_0(T_1, T_2)$ it is sufficient to consider a Poisson counts model with one observation with exposure $E = 1$, counts $c = 0$, with time-indipendent log-intensity given by $\log \lambda_P = \log \Lambda_0(T_1, T_2)$. To represent the sum of log-intensity at the observed points, it is sufficient to consider a Poisson Count model with exposures $E_i = 0$, counts $c_i = 1$, and $\log \lambda_P(t_i) = \log \lambda(t_i|\mathcal{H}_t)$.

Therefore, to represent the whole likelihood we need to consider a Poisson Counts model with $1 + \sum_{i=1}^{|\mathcal{H}|} B_i + 1$ observations, we call the set of indexes representing the expected number of background events, the expected number of triggered events and the sum of log-intensities, I_0, I_{tr}, I_{sl} respectively. For this case, I_0 has only one element, I_{tr} has $\sum_i B_i$ elements and I_{sl} has 1 element. The counts and exposures are defined as follows:

$$c_i = \begin{cases} 0, & i \in I_0 \cup I_{tr} \\ |\mathcal{H}|, & i \in I_{sl} \end{cases} \qquad E_i = \begin{cases} 1, & i \in I_0 \cup I_{tr} \\ 0, & i \in I_{sl} \end{cases}$$

and the log-intensity should be

$$\log \lambda_P(t_i) = \begin{cases} \log \Lambda_0(T_1, T_2), & i \in I_0 \\ \log \Lambda_i(t_j^{(b_i)}, t_{j+1}^{(b_i)}), & i \in I_{tr} \\ \frac{1}{\mathcal{H}} \sum_{i=1}^{|\mathcal{H}|} \log \lambda(t_i|\mathcal{H}) & i \in I_{sl} \end{cases}$$

We consider only one element for the sum of log-intensities. This is done because it gives the same results numerically and provides advantages in terms of computational time and memory requirements.

The first step is to create two functions. The first calculates $\log \Lambda_i(t_j^{(b_i)}, t_{j+1}^{(b_i)})$ efficiently leveraging on the binning strategy. The second calculates the Hawkes process log-intensity for a set of observations given the history of the process.

```
# the function takes in input the parameters in the internal scale (th.K, th.alpha, th.c, th.p) as vect

logLambda.i.inla <- function(th.K, th.alpha, th.c, th.p, list.input_, link.functions = NULL){
    # tranform the parameters in the ETAS scale
    theta_ <- c(0,
                link.functions$K(th.K[1]),
                link.functions$alpha(th.alpha[1]),
                link.functions$c_(th.c[1]),
                link.functions$p(th.p[1]))

    # compute the integral efficiently for each bin of each observation
    comp. <- compute.grid(param. = theta_, list.input_ = list.input_)
    # compute the final output
    out <- theta_[3]*(list.input_$df_grid$magnitudes - list.input_$M0) + log(theta_[2] + 1e-100) + log(
    out
}

# function to calculate the Hawkes process conditional log-intensity for a set of observations (tt) giv

loglambda.inla <- function(th.mu, th.K, th.alpha, th.c, th.p, tt, th, mh, M0,
                           link.functions = NULL){
    # if no link.functions are provided
    if(is.null(link.functions)){
      th.p <- c(th.mu[1], th.K[1], th.alpha[1], th.c[1], th.p[1])
    }
    else{
      th.p <- c(link.functions$mu(th.mu[1]),
                link.functions$K(th.K[1]),
                link.functions$alpha(th.alpha[1]),
                link.functions$c_(th.c[1]),
                link.functions$p(th.p[1]))
    }

    # calculate the value of the log-intensities for each observation in parallel and take the mean
    out <- mean(unlist(mclapply(tt, \(x) {
      th_x <- th < x
      log(lambda_(th = th.p, t = x, ti.v = th[th_x],
                  mi.v = mh[th_x], M0 = M0))
    },
    mc.cores = 5))) # number of cores used in the parallelization
    out
}
```

Then, we need to create a predictor function `predictor.fun` representing $\log \lambda_P$ of the surrogate Poisson Count model. It uses an indicator variable which defines which function should be used to calculate $\log \lambda_P$

```
# predictor function for the surrogate Poisson model, it takes as input the parameters in the internal
predictor.fun <- function(th.mu, th.K, th.alpha, th.c, th.p,
                            list.input, T1, T2, M0,
                          link.functions = NULL){

    out <- rep(0, list.input$n)
    out[list.input$idx.bkg] <- log(link.functions$mu(th.mu[1])) + log(T2 - T1)
    out[list.input$idx.trig] <- logLambda.i.inla(th.K = th.K, th.alpha = th.alpha,
                                                 th.c = th.c, th.p = th.p,
                                                 list.input_ = list.input,
                                                 link.functions = link.functions)
    out[list.input$idx.sl] <- loglambda.inla(th.mu = th.mu, th.K = th.K,
                                             th.alpha = th.alpha,
                                             th.c = th.c, th.p = th.p,
                                             tt = list.input$sample.s$ts,
                                             th = list.input$sample.s$ts,
                                             mh = list.input$sample.s$magnitudes,
                                             M0 = M0,
                                             link.functions = link.functions)
    out
  }
```

Now, we need a function that takes the data as input and actually build and fit the model. The function will take as input a `data.frame` representing the observations, the columns must be : `ts` observed time differences from a starting event, `magnitudes` observed magnitudes, `idx.p` event identifier; the time domain defined by `T1` and `T2`; the minimum magnitude `M0`; the link functions to transform the parameters to have the desired prior distribution `link.functions`; the parameters of the binning strategy δ (`coef.t`), Δ (`delta.t`), and n_{max} (`N.max`), and a list containing the `inlabru` options. The function creates the `data.frame` representing the observations of the surrogate Poisson Count model, it defines the formula and components objects needed to `inlabru` to fit the model and fits the model.

```
Hawkes.bru <- function(sample.s, M0, T1, T2, link.functions = NULL,
                        coef.t., delta.t., N.max., bru.opt){
  # create data.frame for the number of expected background points part.
  df.0 <- data.frame(counts = 0, exposures = 1, part = 'background')
  # create the time bins for each observations and merge them together in a data.frame
  cat('Start creating grid...', '\n')
  time.g.st <- Sys.time()
  df.j <- foreach(idx = 1:nrow(sample.s), .combine = rbind) %do% {
    time.grid(data.point = sample.s[idx,],
              coef.t = coef.t.,
              delta.t = delta.t.,
              T2. = T2, N.exp. = N.max.
    )
  }
  # set counts, exposure and identifier for the predictor function
  df.j$counts <- 0
  df.j$exposures <- 1
  df.j$part = 'triggered'

  # input that will be stored in list.input and needed for the efficient calculation of the expected nu
  t.names <- unique(df.j$t.ref_layer)
  time.sel <- df.j[vapply(t.names, \(bname) match(TRUE, df.j$t.ref_layer == bname), 0L), , drop = FALSE
  Imapping <- match(df.j$t.ref_layer, t.names)
  cat('Finished creating grid, time ', Sys.time() - time.g.st, '\n')

  # creating data.frame representing the sum of log-intensities part.
  df.s <- data.frame(counts = nrow(sample.s), exposures = 0, part = 'SL')

  # bind data.frames together
  data.input = bind_rows(df.0, df.s, df.j)
  # create list.input
  list.input <- list(n = nrow(data.input),
                     df_grid = df.j,
                     M0 = M0,
                     Imapping = Imapping,
                     time.sel = time.sel,
                     sample.s = sample.s,
                     idx.bkg = data.input$part == 'background',
                     idx.trig = data.input$part == 'triggered',
                     idx.sl = data.input$part == 'SL')
  # create formula representing the logintensity of the surrogate Poisson counts model
  merged.form <- counts ~ predictor.fun(th.mu = th.mu, th.K = th.K,
                                        th.alpha = th.alpha,
                                        th.c = th.c, th.p = th.p,
                                        list.input = list.input,
                                        T1= T1, T2 = T2, M0 = M0,
                                        link.functions = link.functions)
  # create components representing the parameters in the internal scale
  cmp.part <- counts ~ -1 +
    th.mu(1, model = 'linear', mean.linear = 0 , prec.linear = 1) +
    th.K(1, model = 'linear', mean.linear = 0, prec.linear = 1) +
    th.alpha(1, model = 'linear', mean.linear = 0, prec.linear = 1) +
    th.c(1, model = 'linear', mean.linear = 0, prec.linear = 1) +
    th.p(1, model = 'linear', mean.linear = 0, prec.linear = 1)
  # fit the model as a Poisson Count model
  bru(formula = merged.form, components = cmp.part, data = data.input, family = 'Poisson',
      options = append(bru.opt, list(E = data.input$exposure)))

}
```

The last thing before fitting the model is to specify the starting value for the parameters and the `inlabru` options. Setting the starting value is crucial, because in the first step the linear approximations will be calculated with respect to the provided initial value. If this is not done carefully may prevent the algorithm to converge to the true mode. This is mainly due to the large plateau at the maximum of the log-likelihood which usually characterizes Hawkes process models for seismicity. It also helps in reducing the number of steps needed for convergence. The most relevant `inlabru` options that may need to be changed with the problem at hand are the `bru_max_iter` argument stating the maximum number of iterations before the algorithm stops and the `bru_method`. The latter is not used in the present example but may be needed for different problems. Specifically it sets parameters of the internal optimization algorithm. The two most important parameters for the algorithm are `max_step`, which regulates *how far* the parameters can jump from iteration to iteration (default is 2) and the relative tolerance used to check convergence (defualt is 0.01

meaning that the algorithm reaches converges is the maximum difference between parameter values between consecutive steps is less than 1% of the parameter standard deviation). If the algorithm start jumping in a loop between values, lowering the `max_step` argument may help.

```r
# set initial values for the parameters in the internal scale to get reasonable initial values (default
th.init <- list(th.mu = 0.5,
                th.K = 0.5,
                th.alpha = -2,
                th.c = -2,
                th.p = -2)

# options for inlabru
bru.opt.list <- list(bru_verbose = 4, # type of visual output
                     bru_max_iter = 100, # maximum number of iterations
                     #bru_method = list(max_step = 0.5, rel_tol = 0.01),
                     inla.mode = 'experimental', # type of inla algorithm
                     bru_initial = th.init) # parameters initial values
```

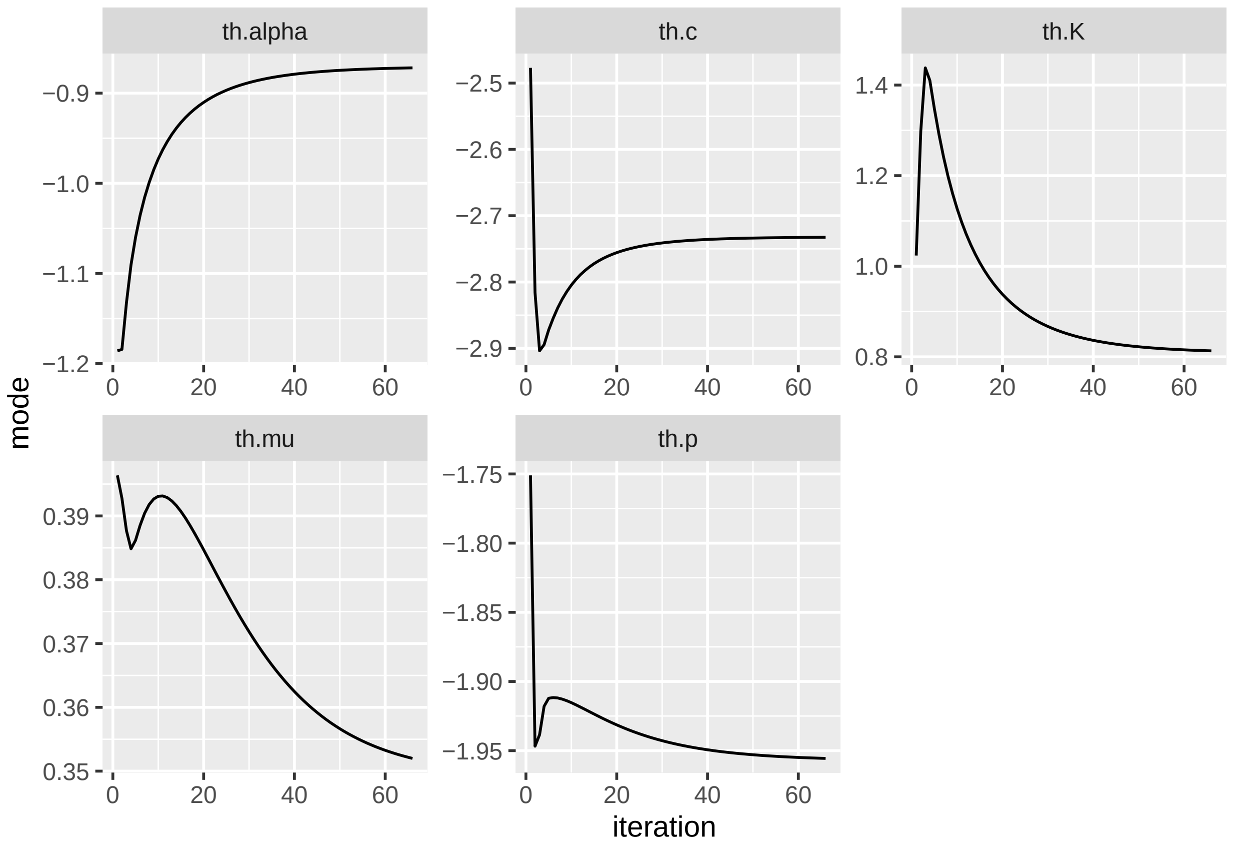
Now, we can fit the model using the function created before. For the binning we use $\delta = 1$, $\Delta = 0.1$ and $n_{max} = 3$.

```r
fit_etas <- Hawkes.bru(sample.s = data.bru, # data
                       M0 = M0, # magnitude of completeness
                       T1 = 0, T2 = T2, # time domain
                       link.functions = link.f.be, # link functions
                       coef.t. = 1, # binning parameter (delta)
                       delta.t. = 0.1, # binning parameter (Delta)
                       N.max. = 3, # binning parameter (n.max)
                       bru.opt = bru.opt.list)
```

```
## Start creating grid...
## Finished creating grid, time  1.993999
```

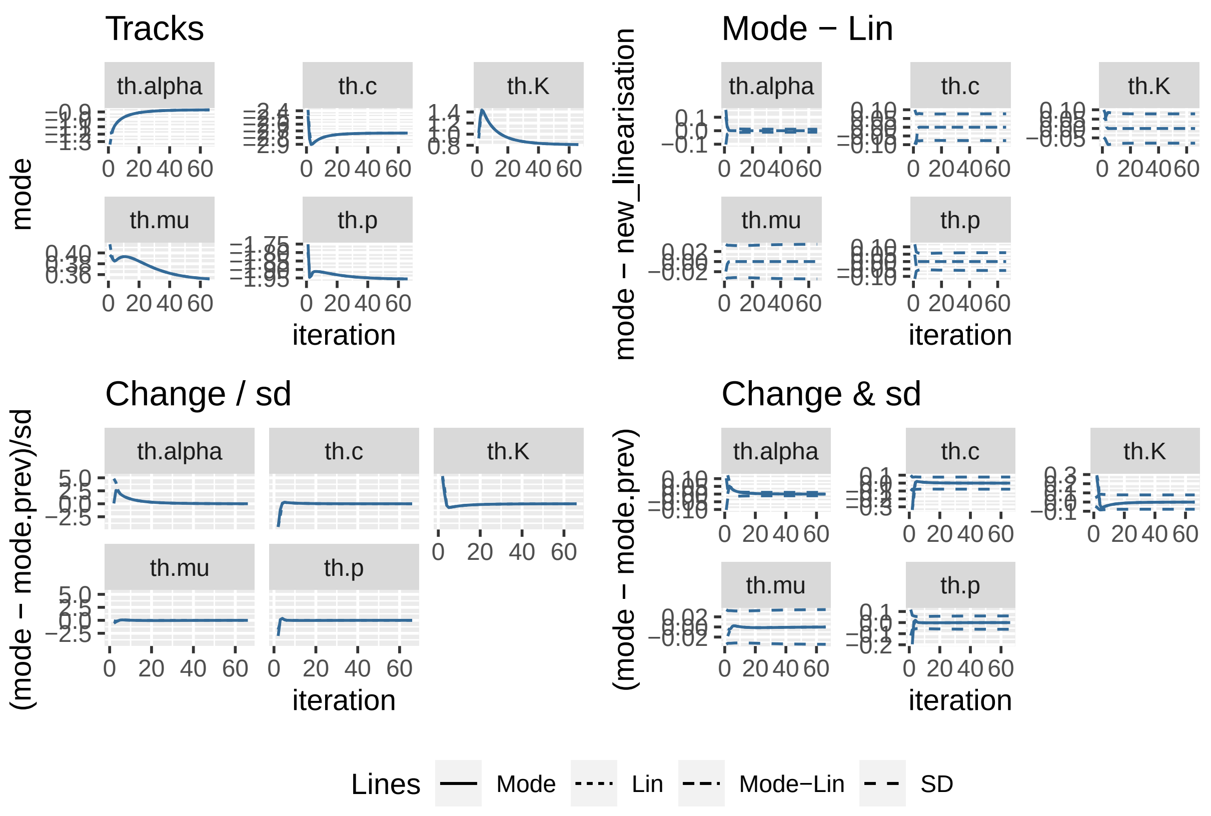
If the model did not reach the maximum number of iterations it means that it probably correctly converged. If the model reached the maximum number of iteration, a simple way to visually check if the algorithm reached convergence is to plot the parameters value at each iteration. The `inlabru` package provide these values which are stored in the `track` element of the `bru_iinla` element of the output.

```r
ggplot(fit_etas$bru_iinla$track, aes(x = iteration, y = mode)) +
  geom_line() +
  facet_wrap(facets = vars(effect), scales = 'free')
```



The package `inlabru` provides also a function to automatically retrieve this plot and three more. These plots are useful to perform diagnostic operations in case the algorithm did not converge and can be easily retrieve with the code below. The track plot as we have shown above is in the top-left corner of the plot below.

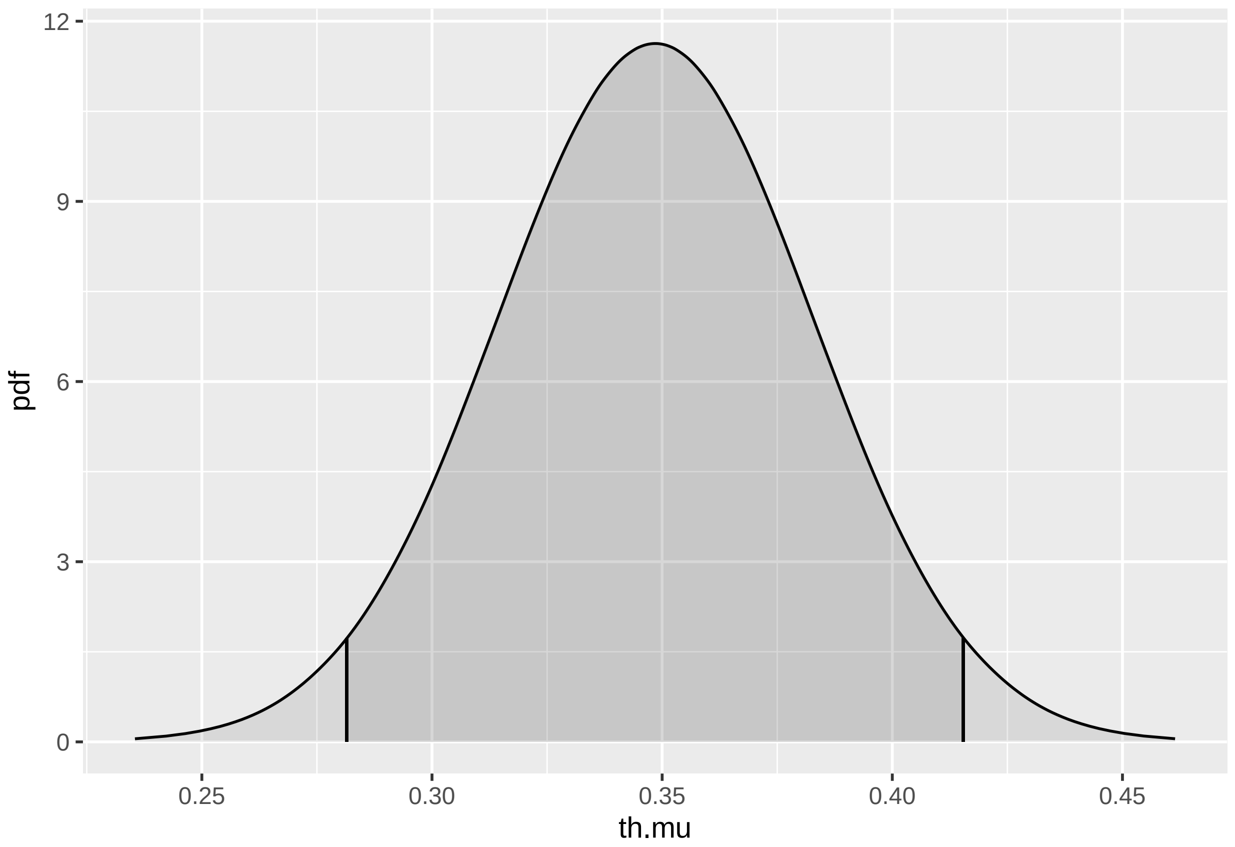
```
inlabru:::make_track_plots(fit_etas)$default
```



Parameters posterior distributions

The `inlabru` output provides the posterior of the parameters in the internal scale which can be easily accessed using

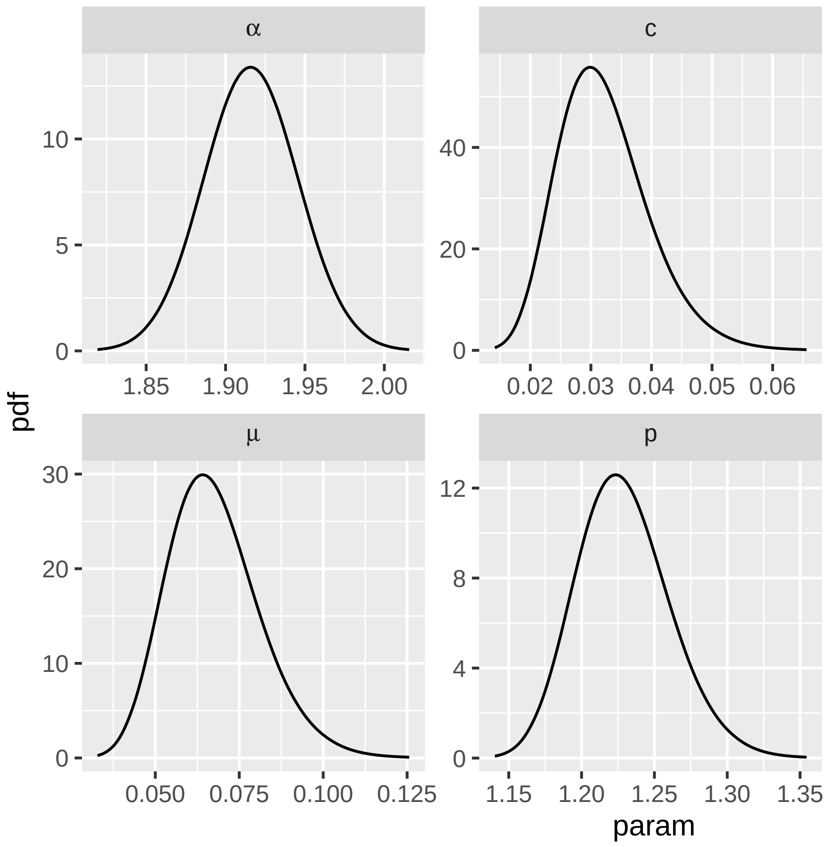
```
# substitute 'th.mu' with 'th.K', 'th.alpha', 'th.c' or 'th.p' to explore the posterior of the others p
plot(fit_etas, 'th.mu')
```



To retrieve the posterior distribution of the parameters in the ETAS scale, we need to tranform the posterior distribution of the parameters in the internal scale. The `inla.tmarginal` function of the `INLA` package computes the transformation for us. However, this function can be used for functions of one parameter, in the next section we show how to retrieve the posterior distribution of functions of multiple parameters.

```
# posterior of parameter mu in ETAS scale
post.mu <- data.frame(inla.tmarginal(link.f.be$mu,
                                     fit_etas$marginals.fixed$th.mu),
                      param = 'mu')
# posterior of parameter mu in ETAS scale
post.K <- data.frame(inla.tmarginal(link.f.be$K,
                                     fit_etas$marginals.fixed$th.K),
                      param = 'K')
# posterior of parameter mu in ETAS scale
post.alpha <- data.frame(inla.tmarginal(link.f.be$alpha,
                                     fit_etas$marginals.fixed$th.alpha),
                      param = 'alpha')
# posterior of parameter mu in ETAS scale
post.c <- data.frame(inla.tmarginal(link.f.be$c_,
                                     fit_etas$marginals.fixed$th.c),
                      param = 'c')
# posterior of parameter mu in ETAS scale
post.p <- data.frame(inla.tmarginal(link.f.be$p,
                                     fit_etas$marginals.fixed$th.p),
                      param = 'p')

ggplot(rbind(post.mu, post.K, post.alpha, post.c, post.p), aes(x,y)) +
  geom_line() +
  facet_wrap(facets = vars(param), scales = 'free', labeller = label_parsed) +
  xlab('param') +
  ylab('pdf')
```



We can use a similar syntax also to sample from the posterior distribution of the parameters or function of a single parameter. To perform this task we can use the function `inla.rmarginal` which takes as input the number of sample and a posterior distribution. The code below sample 10 times from the posterior distribution of the parameter μ in the ETAS scale.

```
inla.rmarginal(10, inla.tmarginal(link.f.be$mu,
                                  fit_etas$marginals.fixed$th.mu))
```

```
##  [1] 0.06854042 0.06273602 0.09023430 0.08240287 0.07002579 0.04843721
##  [7] 0.07204353 0.06942327 0.05349197 0.10037423
```

Posterior distribution of functions of multiple parameters

We may be interested in the posterior distribution of functions of multiple parameters, such as the expected number of points in a subset of the domain. For these cases there is the function `predict` from the `inlabru` package. We are going to use the expected number of points as an example.

We know that the expected number of point in a time interval (T_1, T_2) of a point process model with intensity $\lambda(t)$ has a Poisson distribution with parameter λ_N given by

$$\lambda_N = \int_{T_1}^{T_2} \lambda(t|\mathcal{H}_t)dt$$

The quantity λ_N is a function of all the parameters. We can retrieve its distribution using the `predict` function. The `predict` function takes as input a model, a data and a function and returns posterior summaries of the target function using the data (if needed) and the parameters posterior distribution provided in the model. We first need to create the function calculating the integral of the intensity over the domain, and then we can pass it to the `predict` function. The function needs to have the parameters separated because they are separeted in the model.

```
# function to calculate lambdaN as function of the parameters
lambda.N <- function(th.mu, th.K, th.alpha, th.c, th.p, T1, T2, M0, Ht,
                     link.functions){
  theta_etas <- c(link.functions$mu(th.mu[1]),
                  link.functions$K(th.K[1]),
                  link.functions$alpha(th.alpha[1]),
                  link.functions$c_(th.c[1]),
                  link.functions$p(th.p[1]))

  theta_etas[1]*(T2 - T1) + sum(exp(log.Lambda_h(th = theta_etas,
                                                 ti = Ht$ts,
                                                 mi = Ht$magnitudes,
                                                 M0 = M0,
                                                 T1 = T1, T2 = T2)))
}

# predict function
lambda.N.post <- predict(fit_etas, # model fit
                         data.frame(), # data (empty because the history of the process is passed to th
                         ~ lambda.N(th.mu, th.K, th.alpha, th.c, th.p,
                                    T1, T2, M0,
                                    data.bru,
                                    link.f.be)) # target function
```

We can check if the posterior of λ_N is close to the observed number of points.

```
c(lambda.N.post[1:5], true = nrow(data.bru))
```

```
## $mean
## [1] 968.622
##
## $sd
## [1] 33.91668
##
## $q0.025
## [1] 913.6102
##
## $q0.5
## [1] 965.327
##
## $q0.975
## [1] 1035.836
##
## $true
## [1] 974
```

After that, we can work on the exected number of point. We remark that λ_N is the Poisson rate determining the distribution of the expected number of points in a time interval. The quantity λ_N has a distribution itself which induces a distribution on the space of the distributions of the number of points. The posterior summaries of this distribution of distributions can be also retrieved using the `predict` function. Specifically, the first two arguments are the same, while the third is a `data.frame` with two columns: `N` and `pdf`. The first column contains the values of the number of events for which we want to calculate the distribution, the second is the target distribution, a Poisson distribution in this case, with parameter given by the λ_N function used before.

```
N.post <- predict(fit_etas, data.frame(),
                         ~ data.frame(N = 800:1200,
                                      pdf = dpois(800:1200,
                                                  lambda.N(th.mu, th.K,
                                                           th.alpha, th.c, th.p,
                                                           T1, T2, M0,
                                                           data.bru,
                                                           link.f.be)) ))
```

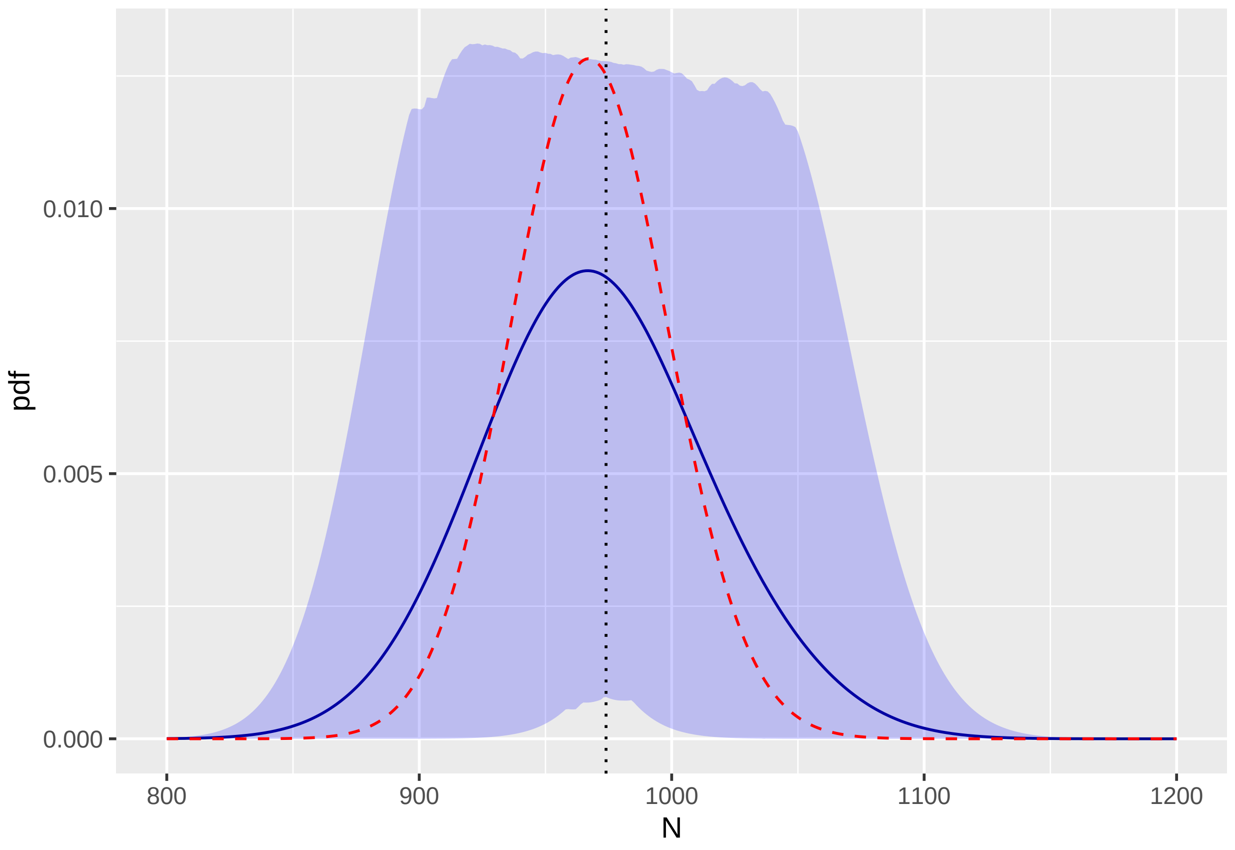
This returns a `data.frame` with the following columns

```
head(N.post)
```

```
##     N         mean           sd       q0.025         q0.5       q0.975
## 1 800 3.515230e-06 2.193653e-05 2.393874e-15 2.023568e-09 1.354197e-05
## 2 801 3.916149e-06 2.429799e-05 3.095864e-15 2.449129e-09 1.537558e-05
## 3 802 4.358082e-06 2.688078e-05 3.998742e-15 2.960490e-09 1.743590e-05
## 4 803 4.844680e-06 2.970187e-05 5.158537e-15 3.574164e-09 1.974790e-05
## 5 804 5.379861e-06 3.277910e-05 6.646486e-15 4.309678e-09 2.233890e-05
## 6 805 5.967825e-06 3.613121e-05 8.553043e-15 5.190096e-09 2.523875e-05
##         median mean.mc_std_err sd.mc_std_err
## 1 2.023568e-09    2.193653e-06  8.646194e-06
## 2 2.449129e-09    2.429799e-06  9.551909e-06
## 3 2.960490e-09    2.688078e-06  1.053927e-05
## 4 3.574164e-09    2.970187e-06  1.161413e-05
## 5 4.309678e-09    3.277910e-06  1.278259e-05
## 6 5.190096e-09    3.613121e-06  1.405102e-05
```

where the `mean` is the posterior mean of the value of the distribution for the specified value of `N`, the quantity `sd` is the posterior standard deviation and so on. We remark that the mean distribution in this case is different from a Poisson distribution with parameter equal to the posterior mean of λ_N. To illustrate the difference, we plot the mean distribution (solid blue line) and the posterior 95% credibility interval of the distribution of the number of points (shaded blue) and compare them with a Poisson distribution with parameter equal to the posterior mean of λ_N (dashed red) and with the observed number of points (dotted black).

```
ggplot(N.post, aes(x = N, y = mean)) +
  geom_line(color = 'darkblue') +
  geom_ribbon(aes(xmax = N, xmin = N, ymin = q0.025, ymax = q0.975), alpha = 0.2,
              fill = 'blue') +
  geom_vline(xintercept = nrow(data.bru), linetype = 3) +
  geom_line(data = data.frame(x = 800:1200,
                              y = dpois(800:1200, 967.6161)),
            aes(x,y), color = 'red', linetype = 2) +
  ylab('pdf')
```

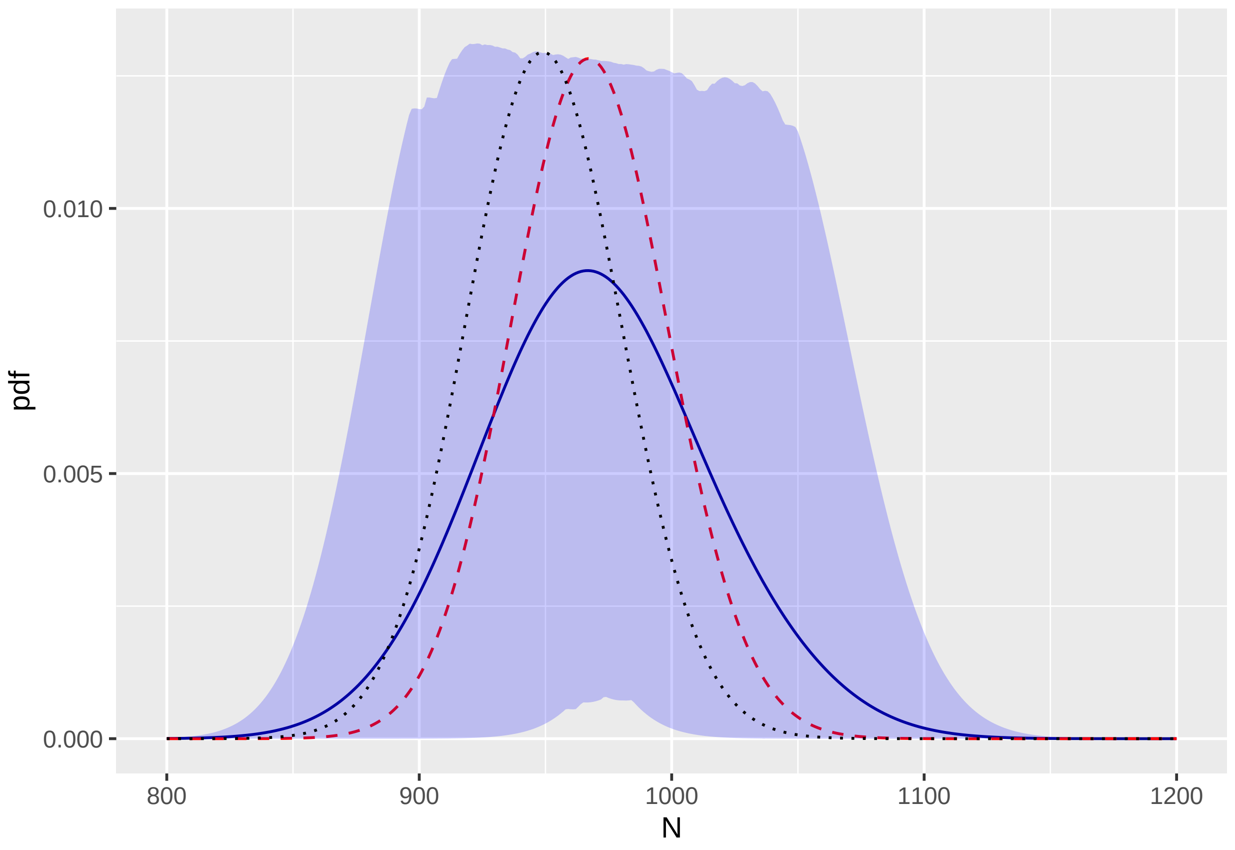


Sample from the posterior distribution of functions of multiple parameters

If we are interested in sampling from the posterior distribution of functions of multiple parameters we can use the function `generate` from the `inlabru` package. The function uses a syntax very close to the predict function. For example, the code below extract a sample from the posterior distribution of the distributions of the number of events (dotted in the plot below)

```
N.samp <- generate(fit_etas, data.frame(),
                   ~ data.frame(N = 800:1200,
                                pdf = dpois(800:1200,
                                            lambda.N(th.mu, th.K,
                                                     th.alpha, th.c, th.p,
                                                     T1, T2, M0,
                                                     data.bru,
                                                     link.f.be)) ),
                   n.samples = 1, seed = 1)

ggplot(N.post, aes(x = N, y = mean)) +
  geom_line(color = 'darkblue') +
  geom_line(data = data.frame(x = 800:1200,
                              y = dpois(800:1200, 967.6161)),
            aes(x,y), color = 'red', linetype = 2) +
  geom_ribbon(aes(xmax = N, xmin = N, ymin = q0.025, ymax = q0.975), alpha = 0.2,
              fill = 'blue') +
  geom_line(data = N.samp[[1]],
            aes(x = N, y = pdf), linetype = 3) +
  ylab('pdf')
```


pdf
N