Summary of work by week 9

Helene Behrens

3/3/2021

So far, we have tried to see if it is possible to fit a Lee-Carter model for mortality predicions and the Lee-Carter model extended with a birth-year cohort effect using inlabru. The first results are good, but we have only tested the model for synthetic data, which are not necessarily very close to the real-life situation.

The Lee-Carter model

We have used the follwing version of the Lee-Carter model:

$$Y_{x,t} \sim Poisson(E_{x,t} \cdot e^{\eta_{x,t}}),$$

where $Y_{x,t}$ is the expected number of deaths for people at age x in year t, $E_{x,t}$ is the number of people "at risk" (observed people) of age x at time t and

$$\eta_{x,t} = C + \alpha_x + \beta_x \cdot \kappa_t + \epsilon_{x,t}.$$

Here α_x can be interpreted as the overall mortality profile for the ages in x, κ_t is the change in mortality for different times t, relative to a zero-level and β_x can be interpreted as the sensitivity to time-related changes in mortality for different ages x. $\epsilon_{x,t}$ is the error term, assumed to be normal distributed with zero mean, and C is a constant. α_x, β_x and κ_t are subject to the following constraints:

$$\sum_x \alpha_x = 0, \quad \sum_x \beta_x = 1, \quad \sum_t \kappa_t = 0.$$

We have modeled α_x as a random walk, β_x as iid normal distributed with zero mean and κ_t as a random walk with drift:

$$\kappa_{t+1} = \phi + \kappa_t + \epsilon_\kappa, \quad \epsilon_\kappa \sim N(0, 1/\tau_\kappa).$$

As inla only handles latent effects modelled as random walks without drift, we rewrite to be able to run with inla:

$$\kappa_t = \kappa_t^* + \phi \cdot t, \quad \kappa_t^* = \kappa_{t-1}^* + \epsilon_\kappa, \quad \sum_t \kappa_t^* = 0.$$

Our model is then

$$\eta_{x,t} = C + \alpha_x + \beta_x \cdot \phi \cdot t + \beta_x \kappa_t^* + \epsilon.$$

From this point, we will refer to κ_t^* as only κ_t. We have also tried to include a cohort-effect of a persons birth-year, $t - x$, to our model:

$$\eta_{x,t} = C + \alpha_x + \beta_x \cdot \phi \cdot t + \beta_x \cdot \kappa_t + \gamma_{t-x} + \epsilon,$$

where γ_{t-x} is the cohort effect subject to the constraint $\sum_{x,t} \gamma_{t-x} = 0$ and the remaining parameters are the same as before. We have modeled γ_{t-x} as a random walk.

Using Inlabru

The multiplicative term $\beta_x \cdot \kappa_t$ makes the predictor $\eta_{x,t}$ non-linear, which means that Inla can not be used to do inference of data following this model. We have tried to use the Inlabru package in R to get around this obstacle.

Inlabru uses a Taylor approximation to linearize the predictor $\eta_{x,t}$ and then runs Inla with this linearization as the linear predictor.

To find the ideal linearization point, Inlabru runs a fix-point iteration with Inla, setting the linearization point in one step as the point that maximize the posterior distribution of the latent effects that results when running inla with the linearization point from the last step. For further explanation, see https://inlabru-org.github.io/inlabru/articles/method.html or the last part of this document.

Below are some examples of inference with Inlabru on our model. We have used synthetic data, so we have sampled values of the latent effects from different functions for α_x, β_x, κ_t and ϕ. In general, we get a good model fit for most model choices, and Inlabru fits the values of the non-linear predictor $\eta_{x,t}$ well in all cases.

For some combinations of κ_t and ϕ, it does seem like the model gets unidentifiable. We also observe that introducing the cohort effect γ_{t-x} seems to amplify this effect somewhat. However, the combination of ϕ and κ_t that causes problems are not very likely to be observed in real life, given the assmption that κ_t is a random walk without drift and ϕ is the drift. Also, we do not see any indication that γ_{t-x} itself causes any unidenitfyability problems with any of the other effects.

Below, I have included the implementation of running inlabru for four different configurations for the latent effects.

```
library(INLA)
```

```
## Loading required package: Matrix

## Loading required package: sp

## Loading required package: parallel

## Loading required package: foreach

## This is INLA_21.01.13 built 2021-01-13 16:58:38 UTC.
##  - See www.r-inla.org/contact-us for how to get help.
##  - To enable PARDISO sparse library; see inla.pardiso()
##  - Save 350.5Mb of storage running 'inla.prune()'
```

```
library(inlabru)
```

```
## Loading required package: ggplot2
```

```
library(ggplot2)
library(patchwork)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v tibble  3.0.5     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x purrr::accumulate() masks foreach::accumulate()
## x tidyr::expand()     masks Matrix::expand()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x tidyr::pack()       masks Matrix::pack()
## x tidyr::unpack()     masks Matrix::unpack()
## x purrr::when()       masks foreach::when()
```

```
seed = 324
set.seed(seed)

N = 1000   #  Number of observations

general.title = paste("N = ", N, "seed = ", seed)   #  Used to identify plots

nx = 10   #  x from 1 to nx
nt = 10   #  t from 1 to nt

at.risk = 1000   #  number of people at risk for each x-t-pair, here as a constant

x = sample(1:nx, N, replace = TRUE)
t = sample(1:nt, N, replace = TRUE)

tau.iid = 1/0.1**2   #  Standard deviation of 0.1 for the iid effects (beta)
tau.epsilon = 1/0.01**2   #  Standard deviation of 0.01 for the error term

# sample beta_x, same for all configurations of the latent effects
beta = rnorm(nx, 0, sqrt(1/tau.iid))
beta = 1/nx + beta - mean(beta)   # sum to 1
```

The following configuration of the latent effects seem to be easy for inlabru to fit. Here we have not included the cohort effect.

```
#   underlying models for latent effects:
kappa = 0.5*cos((1:nt)*pi/3)
kappa = kappa - mean(kappa)   #  shift around zero

alpha = cos(((1:nx - 3)* pi)/6)
alpha = alpha - mean(alpha)   #  shift around zero

phi = -0.5

#  sample synthetic data and arrange observations in the obs dataframe
obs = data.frame(x,t)
obs = obs %>%
  mutate(beta = beta[as.vector(obs$x)],
         kappa = kappa[as.vector(obs$t)],
```

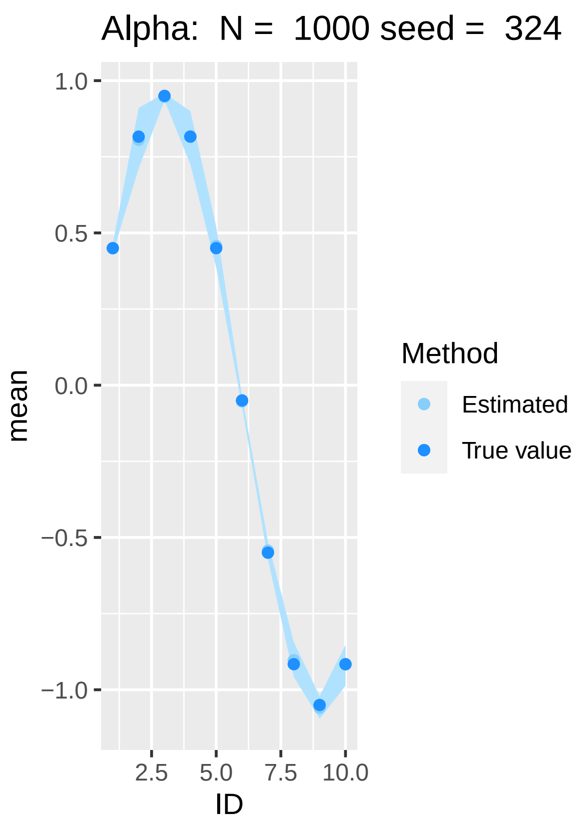
```r
         alpha = alpha[as.vector(obs$x)],
         phi.t = phi*obs$t,
         phi = phi,
         at.risk = at.risk,
         epsilon = rnorm(n = N, 0, sqrt(1/tau.epsilon))) %>%
  mutate(eta = alpha + beta*phi.t + beta*kappa + epsilon) %>%   #  linear predictor
  mutate(y.o = rpois(N, at.risk*exp(eta))) %>%   #  simulate data
  mutate(t1 = t, x1 = x)  %>%   #  add extra t and x to the observations for the sake of inlabru:
  mutate(xt = seq_along(t))

#   ----  Start defining the inlabru model components  ----

#  helper values for constraining of beta:
A.mat = matrix(1, nrow = 1, ncol = nx)
e.vec = 1

#  add more inforamative priors
pc.prior <- list(prec = list(prior = "pc.prec", param = c(0.3,0.8)))
pc.alpha <- list(prec = list(prior = "pc.prec", param = c(0.1,0.1)))
pc.prior.small <- list(prec = list(prior = "pc.prec", param = c(0.02, 0.1)))

#  define the components of the model
comp = ~ -1 +
  Int(1) +
  alpha(x, model = "rw1", constr = TRUE, hyper = pc.alpha) +
  phi(t, model = "linear", prec.linear = 1) +
  beta(x1, model = "iid", extraconstr = list(A = A.mat, e = e.vec)) +
  kappa(t1, model = "rw1", values = 1:nt, constr = TRUE, hyper = pc.prior) +
  epsilon(xt, model = "iid", hyper = pc.prior.small)

#  define the likelihood
form.1 = y.o ~ -1 + Int + alpha + beta*phi + beta*kappa + epsilon
likelihood.1 = like(formula = form.1, family = "poisson", data = obs, E = at.risk)

c.c <- list(cpo = TRUE, dic = TRUE, waic = TRUE, config = TRUE)  #  control compute

# run inlabru
res = bru(components = comp,
          likelihood.1,
          options = list(verbose = F,
                         #bru_verbose = 1,
                         num.threads = "1:1",
                         control.compute = c.c
                         ))

# would in a script only rerun if necessary
#res = bru_rerun(res)
```

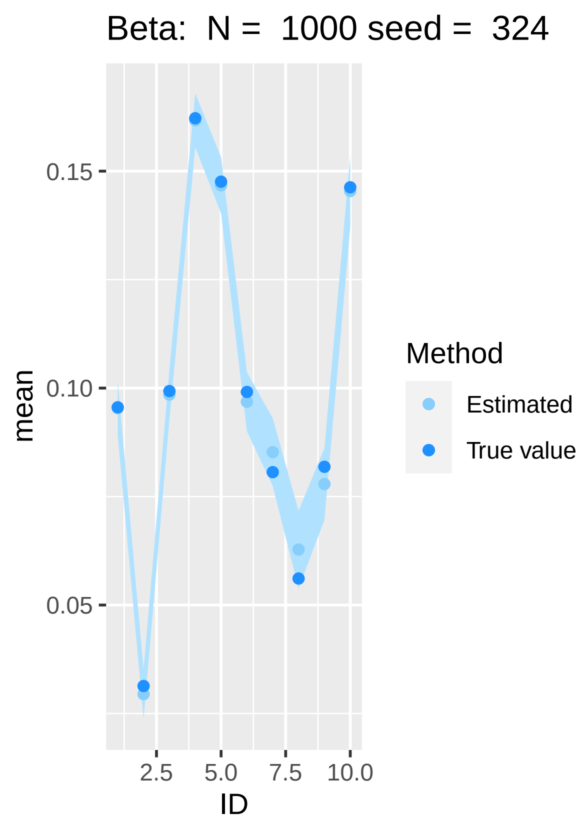
```
## N =  1000 seed =  324
```

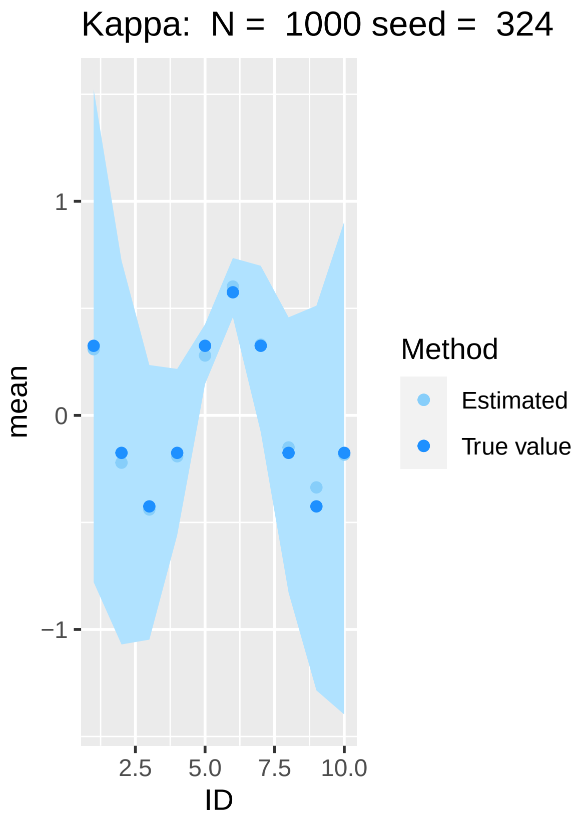
```
##           mean         sd 0.025quant   0.5quant 0.975quant       mode
## Int  1.0059154 0.07047172  0.8573339  1.0084676  1.1391249  1.0125924
## phi -0.5121967 0.12810151 -0.7543841 -0.5167904 -0.2424901 -0.5242521
```

```
##                kld
## Int 1.189714e-09
## phi 6.227557e-10
```

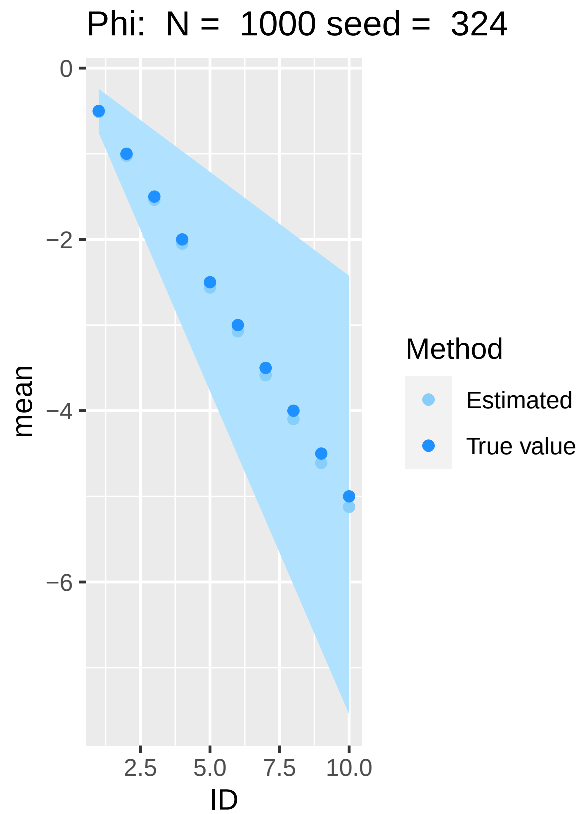
```
##                                mean          sd  0.025quant      0.5quant
## Precision for alpha       14.549384    5.011102    6.861611     13.865443
## Precision for beta        94.890084   40.480905   35.881227     88.549168
## Precision for kappa        6.794516    3.564170    1.995917      6.121774
## Precision for epsilon 13410.910028 7282.992253 5476.480401 11443.872471
##                        0.975quant        mode
## Precision for alpha      26.33345   12.531581
## Precision for beta      191.41873   75.460861
## Precision for kappa      15.60322    4.718882
## Precision for epsilon 32605.63109 8703.281870
```



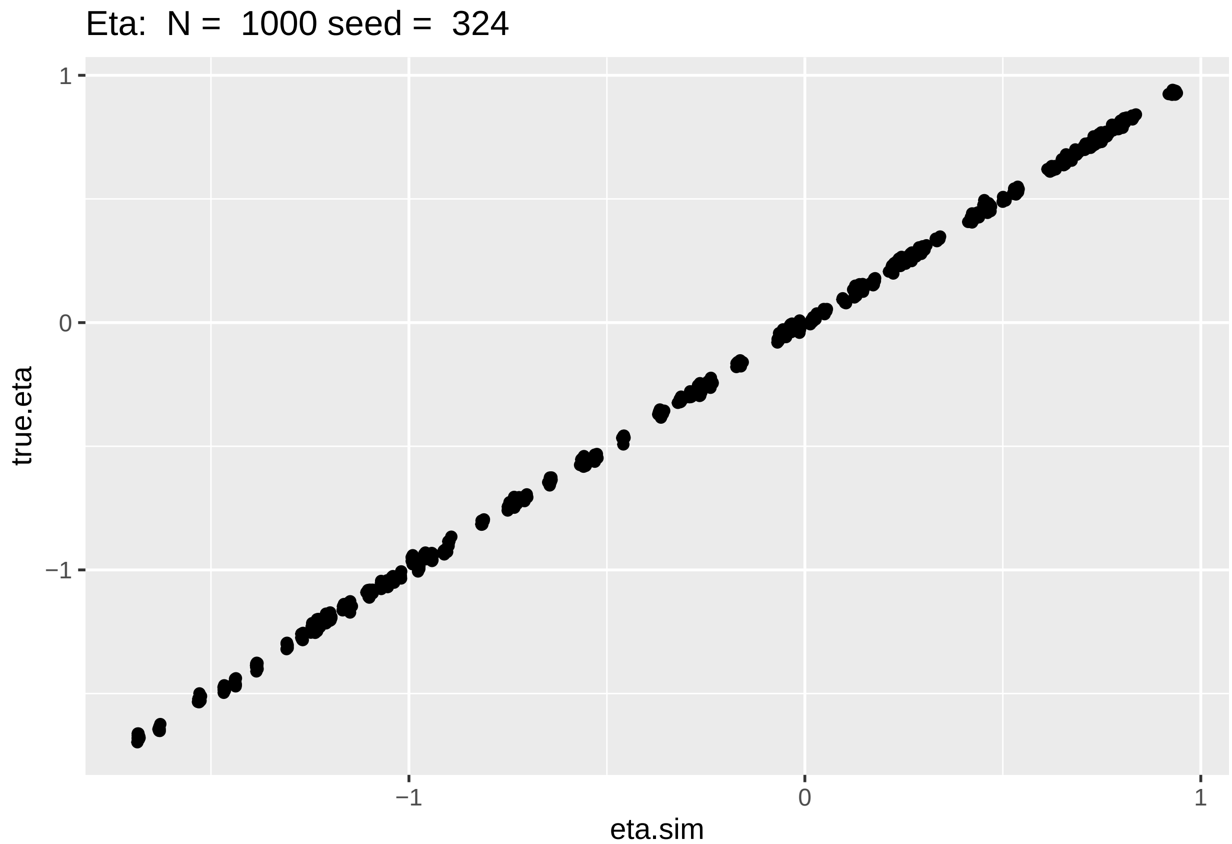




Kappa: N = 1000 seed = 324
mean
ID
Method
Estimated
True value



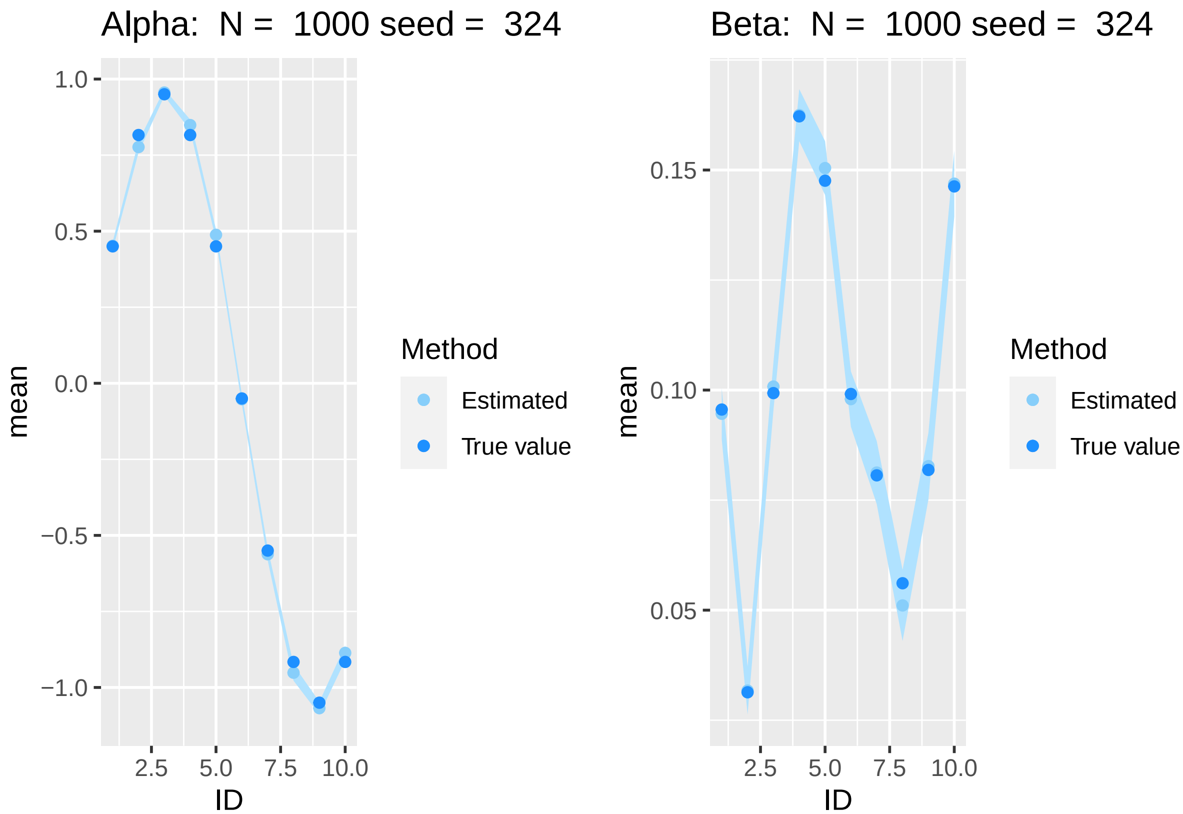
Phi: N = 1000 seed = 324
mean
ID
Method
Estimated
True value



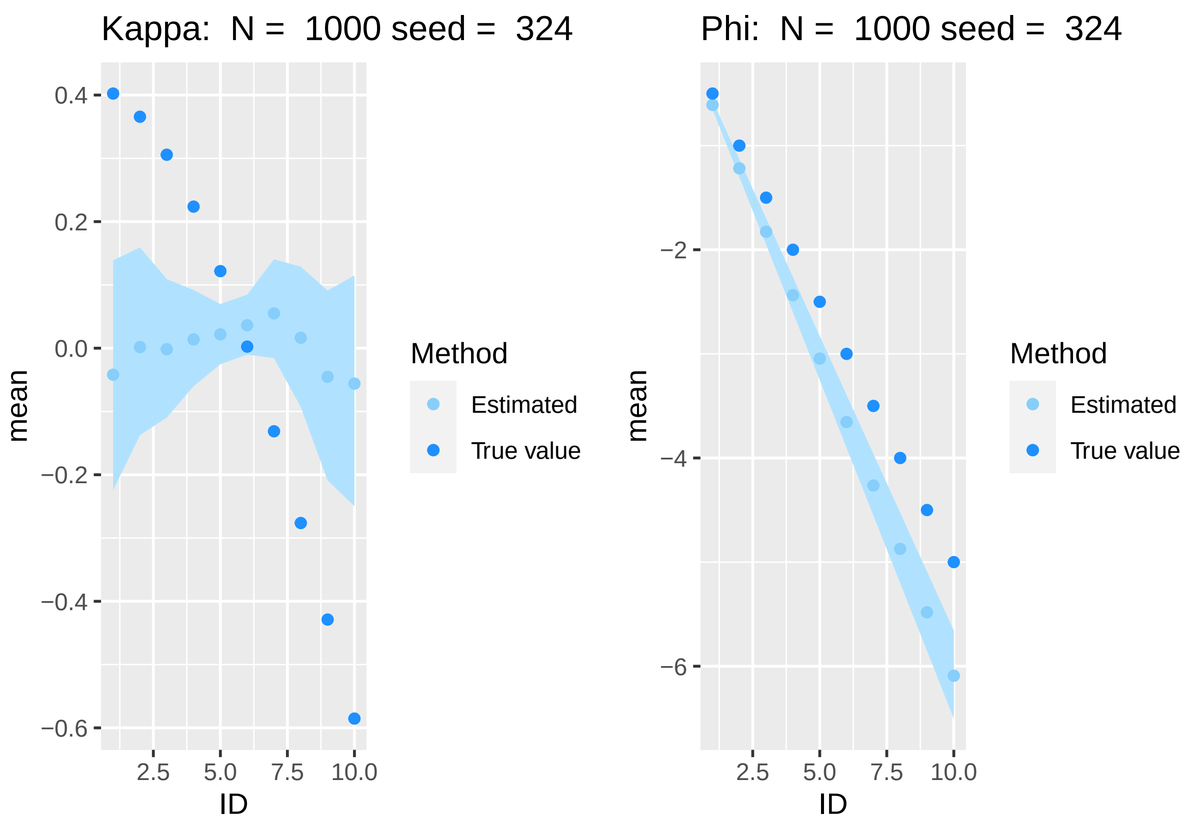
The next configuration shows good results for the estimation of the predictor η, but a mix-up in the estimation of κ_t and ϕ. This model is also without the cohort effect. We have used the same code as above, only changing the function used for κ_t.

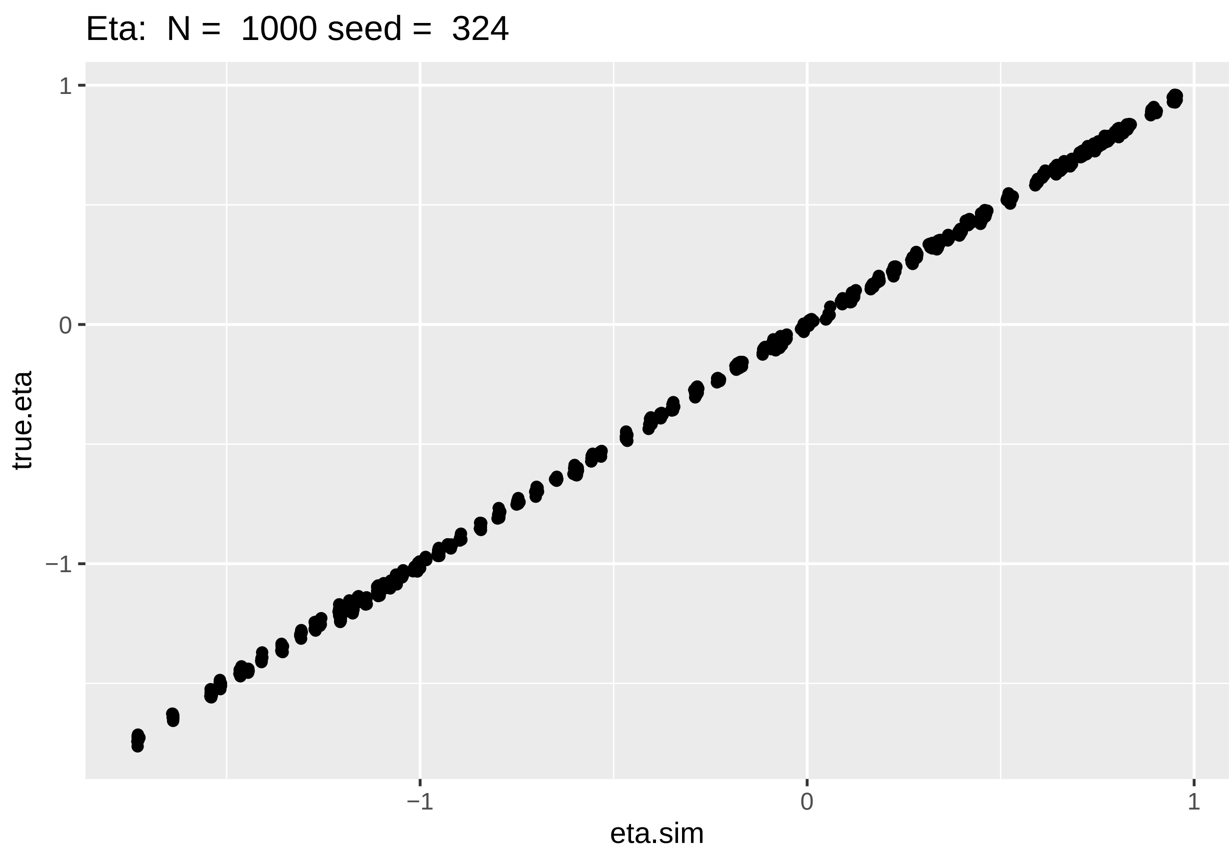
```
#   underlying models for latent effects:
kappa = cos((1:nt)*pi/20)
kappa = kappa - mean(kappa)    #  shift around zero
```

```
## N =  1000 seed =  324

##           mean         sd 0.025quant   0.5quant 0.975quant       mode
## Int  1.0602031 0.01133901  1.0364243  1.0604003  1.0827027  1.0606267
## phi -0.6091293 0.02069865 -0.6502816 -0.6094643 -0.5658577 -0.6098307
##              kld
## Int 4.703355e-08
## phi 2.626510e-08

##                               mean           sd  0.025quant    0.5quant
## Precision for alpha       13.94096     4.746833    6.630367    13.30221
## Precision for beta        93.64760    39.965321   35.306777    87.43123
## Precision for kappa      624.06952   661.535035   82.152292   427.90864
## Precision for epsilon 19821.57082 13959.026821 6717.908641 15718.98924
##                        0.975quant         mode
## Precision for alpha       25.08254     12.05489
## Precision for beta       188.79127     74.50529
## Precision for kappa     2360.67194    209.55766
## Precision for epsilon  56924.11323  10876.90593
```



Alpha: N = 1000 seed = 324
mean
ID
Method
Estimated
True value
Beta: N = 1000 seed = 324
mean
ID
Method
Estimated
True value


Kappa: N = 1000 seed = 324
0.4
0.2
0.0
−0.2
−0.4
−0.6
mean
2.5
5.0
7.5
10.0
ID
Method
Estimated
True value
Phi: N = 1000 seed = 324
−2
−4
−6
mean
2.5
5.0
7.5
10.0
ID
Method
Estimated
True value



In the following configuration, we have included the cohort effect. We observe how the introduction of the cohort effect amplifies the error from the last example.

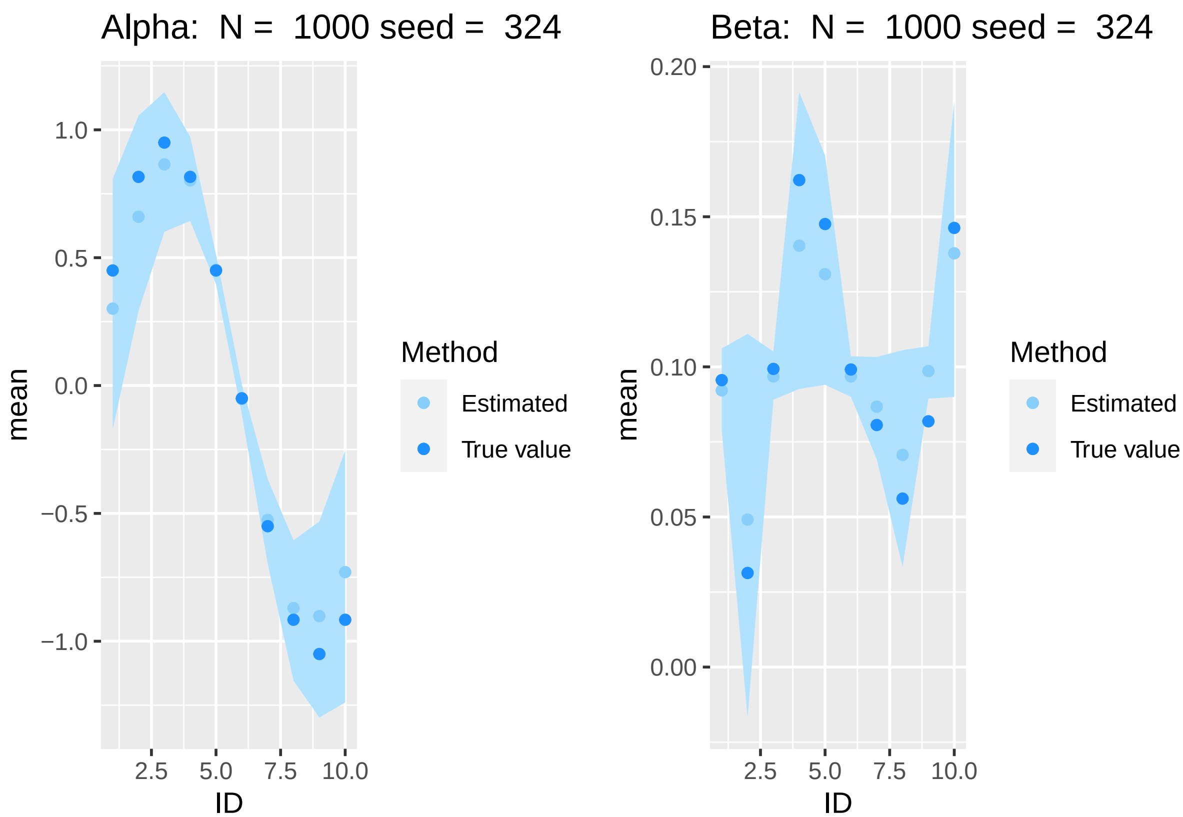
```
#   underlying models for latent effects:
kappa = cos((1:nt)*pi/20)
kappa = kappa - mean(kappa)   #  shift around zero

alpha = cos(((1:nx - 3)* pi)/6)
alpha = alpha - mean(alpha)   #  shift around zero

# define max and min of the values for the cohort
n.cohort = (nt - 1) + abs(1-nx) + 1
cohort.min = 1-nx
cohort.max = nt-1

# get samples for the cohorts
cohort = t-x

gamma = 0.2*(cohort.min:cohort.max) + sin(cohort.min:cohort.max)
gamma = gamma - mean(gamma)  #center around zero

phi = -0.5

#  sample synthetic data and arrange observations in the obs dataframe
obs = data.frame(x,t, cohort)
obs = obs %>%
```

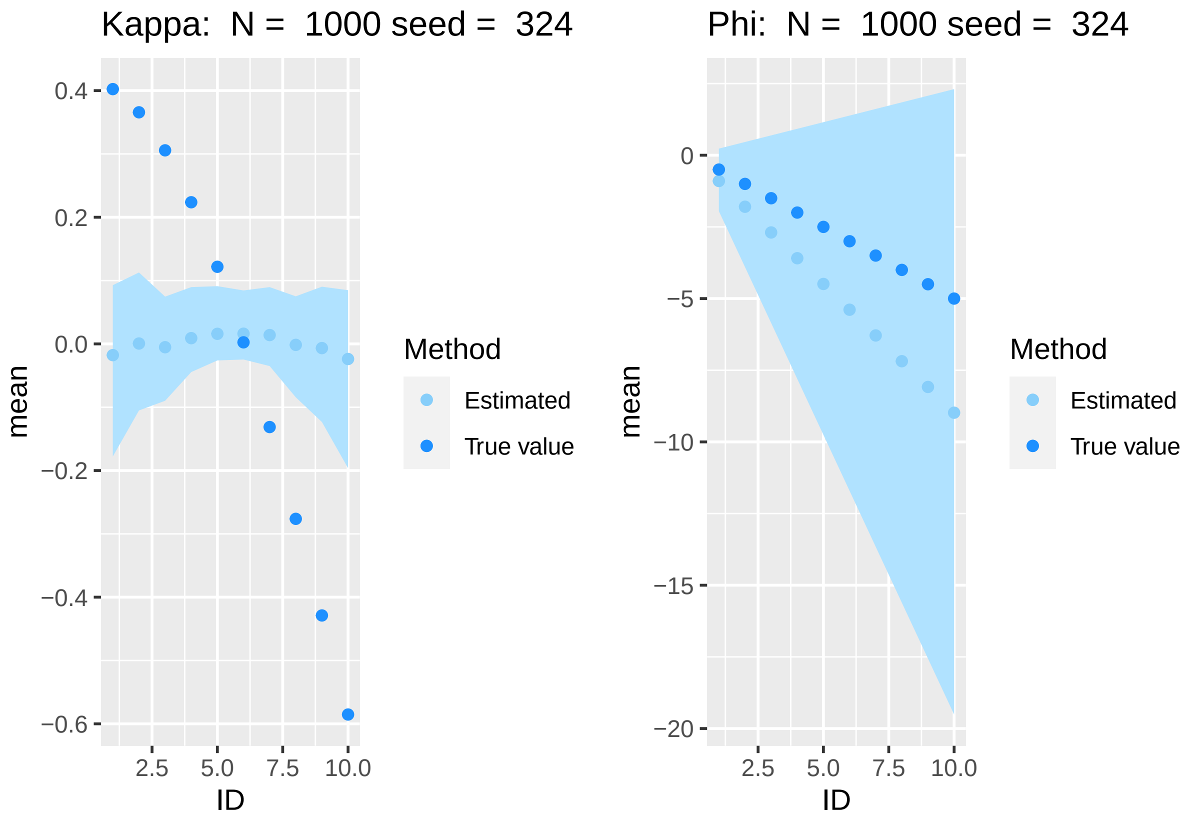
```r
  mutate(beta = beta[as.vector(obs$x)],
         kappa = kappa[as.vector(obs$t)],
         alpha = alpha[as.vector(obs$x)],
         gamma = gamma[as.vector(obs$cohort - cohort.min + 1)],
         phi.t = phi*obs$t,
         phi = phi,
         at.risk = at.risk,
         epsilon = rnorm(n = N, 0, sqrt(1/tau.epsilon))) %>%
  mutate(eta = alpha + beta*phi.t + beta*kappa + gamma + epsilon) %>%   #  linear predictor
  mutate(y.o = rpois(N, at.risk*exp(eta))) %>%   #  simulate data
  mutate(t1 = t, x1 = x)  %>%   #  add extra t and x to the observations for the sake of inlabru:
  mutate(xt = seq_along(t))

#   ----  Start defining the inlabru model components  ----

#  helper values for constraining of beta:
A.mat = matrix(1, nrow = 1, ncol = nx)
e.vec = 1

#  add more inforamative priors
pc.prior <- list(prec = list(prior = "pc.prec", param = c(0.3,0.8)))
pc.alpha <- list(prec = list(prior = "pc.prec", param = c(0.1,0.1)))
pc.prior.small <- list(prec = list(prior = "pc.prec", param = c(0.02, 0.1)))
pc.prior.gamma <- list(prec = list(prior = "pc.prec", param = c(0.8, 0.8)))


#  define the components of the model
comp = ~ -1 +
  Int(1) +
  alpha(x, model = "rw1", constr = TRUE, hyper = pc.alpha) +
  phi(t, model = "linear", prec.linear = 1) +
  beta(x1, model = "iid", extraconstr = list(A = A.mat, e = e.vec)) +
  kappa(t1, model = "rw1", values = 1:nt, constr = TRUE, hyper = pc.prior) +
  gamma(cohort, model = "rw1", values = cohort.min:cohort.max, constr = TRUE, hyper = pc.prior.gamma) +
  epsilon(xt, model = "iid", hyper = pc.prior.small)

#  define the likelihood
form.1 = y.o ~ -1 + Int + alpha + beta*phi + beta*kappa + gamma + epsilon
likelihood.1 = like(formula = form.1, family = "poisson", data = obs, E = at.risk)

c.c <- list(cpo = TRUE, dic = TRUE, waic = TRUE, config = TRUE)  #  control compute

# run inlabru
res = bru(components = comp,
          likelihood.1,
          options = list(verbose = F,
                         #bru_verbose = 1,
                         num.threads = "1:1",
                         control.compute = c.c
                         ))

# would in a script only rerun if necessary
#res = bru_rerun(res)
```

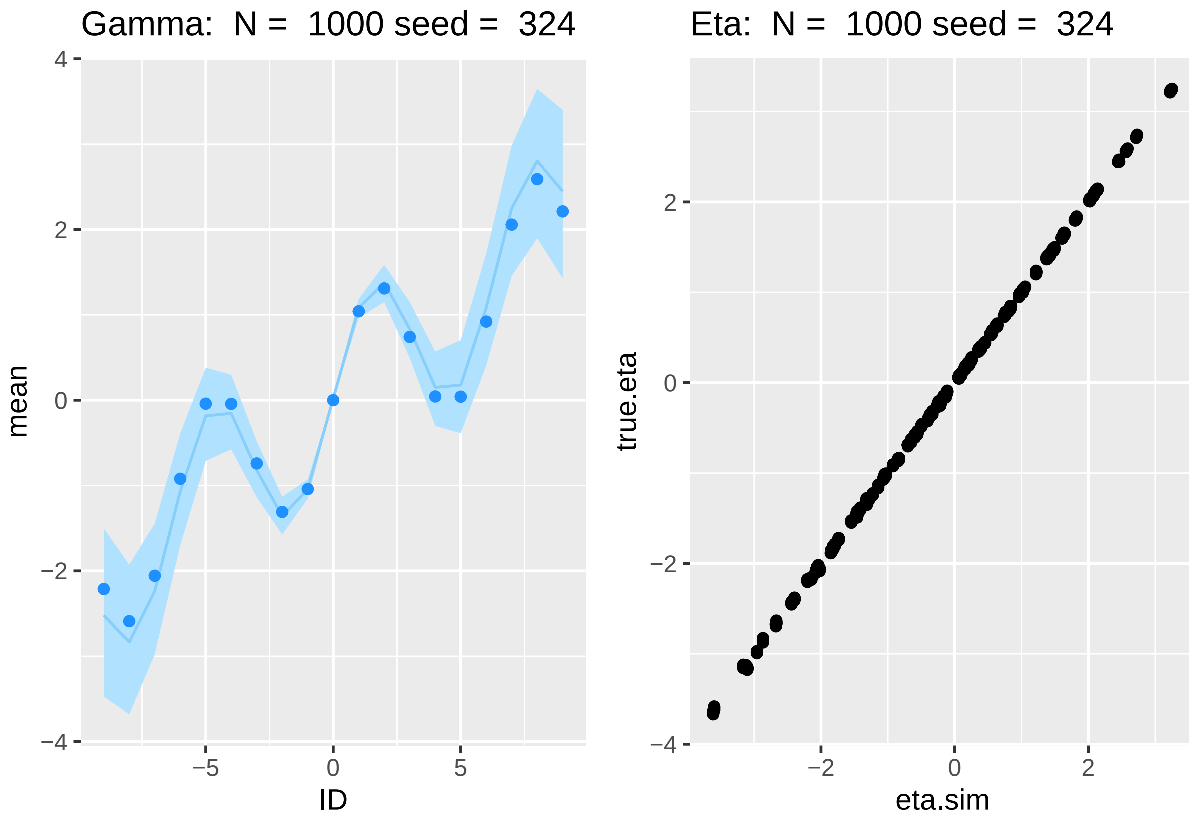
```
## N =  1000 seed =  324
```

```
##           mean        sd 0.025quant   0.5quant 0.975quant       mode
## Int  1.2132630 0.3054352  0.5918282  1.2205061  1.7936570  1.2348075
## phi -0.8982021 0.5547307 -1.9510654 -0.9113896  0.2306521 -0.9375656
##              kld
## Int 0.0001401816
## phi 0.0001372969
```

```
##                               mean           sd   0.025quant     0.5quant
## Precision for alpha       15.175284 5.511624e+00    6.9512497    14.338567
## Precision for beta        99.615773 4.368197e+01   36.8089219    92.479754
## Precision for kappa    18917.519012 1.643999e+05  109.1372891  2559.456595
## Precision for gamma        1.919699 6.710178e-01    0.9049796     1.822438
## Precision for epsilon 12817.389727 3.325650e+03 7775.2809689 12304.954954
##                         0.975quant         mode
## Precision for alpha   2.827088e+01    12.752618
## Precision for beta    2.048659e+02    77.964713
## Precision for kappa   1.255608e+05   202.678642
## Precision for gamma   3.502959e+00     1.636961
## Precision for epsilon 2.070242e+04 11317.968523
```




Kappa: N = 1000 seed = 324
Phi: N = 1000 seed = 324
mean
ID
Method
Estimated
True value
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
−5
−10
−15
−20
2.5
5.0
7.5
10.0



The last example also includes the cohort effect. It shows that given a well-defined combination of κ_t and ϕ, Inlabru is also able to fit the latent effects, including the cohort effect. For this example, we have run the previous code, only changing the function used for κ_t.

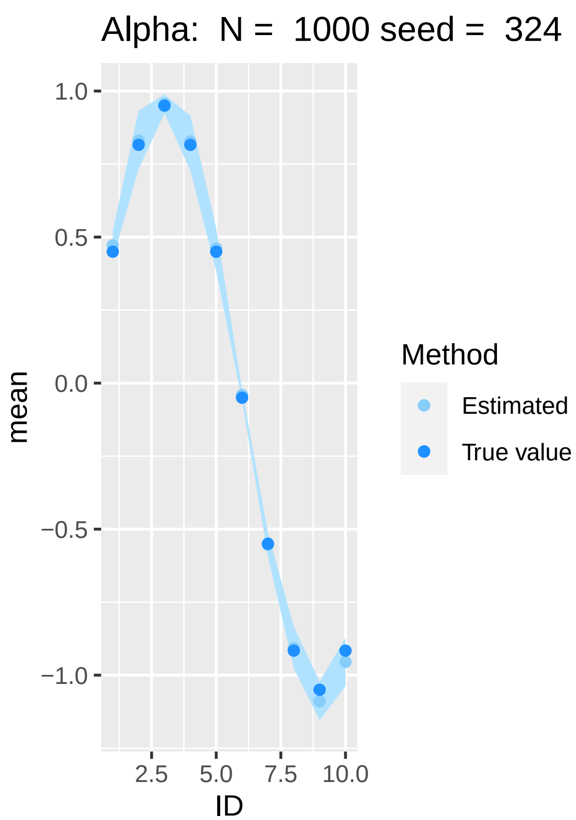
```
#   underlying models for latent effects:
kappa = 0.5*cos((1:nt)*pi/3)
kappa = kappa - mean(kappa)    #  shift around zero
```

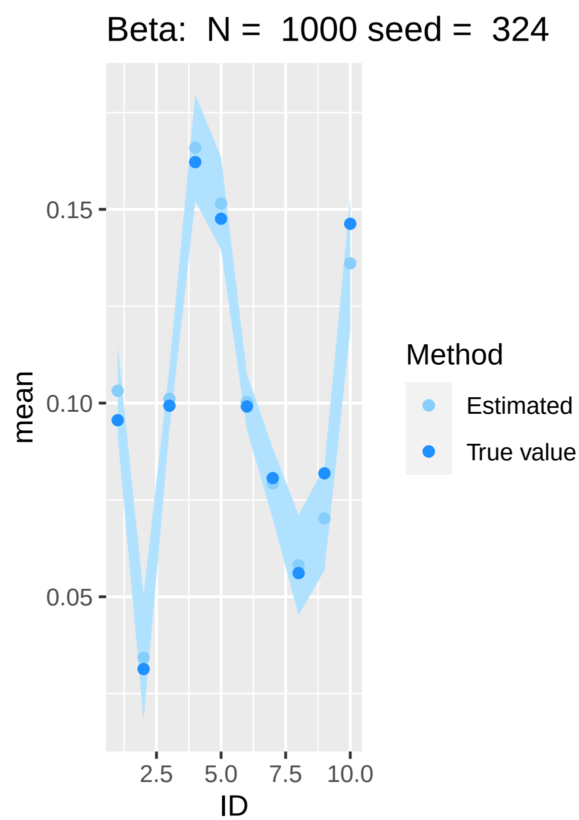
```
## N =  1000 seed =  324
```

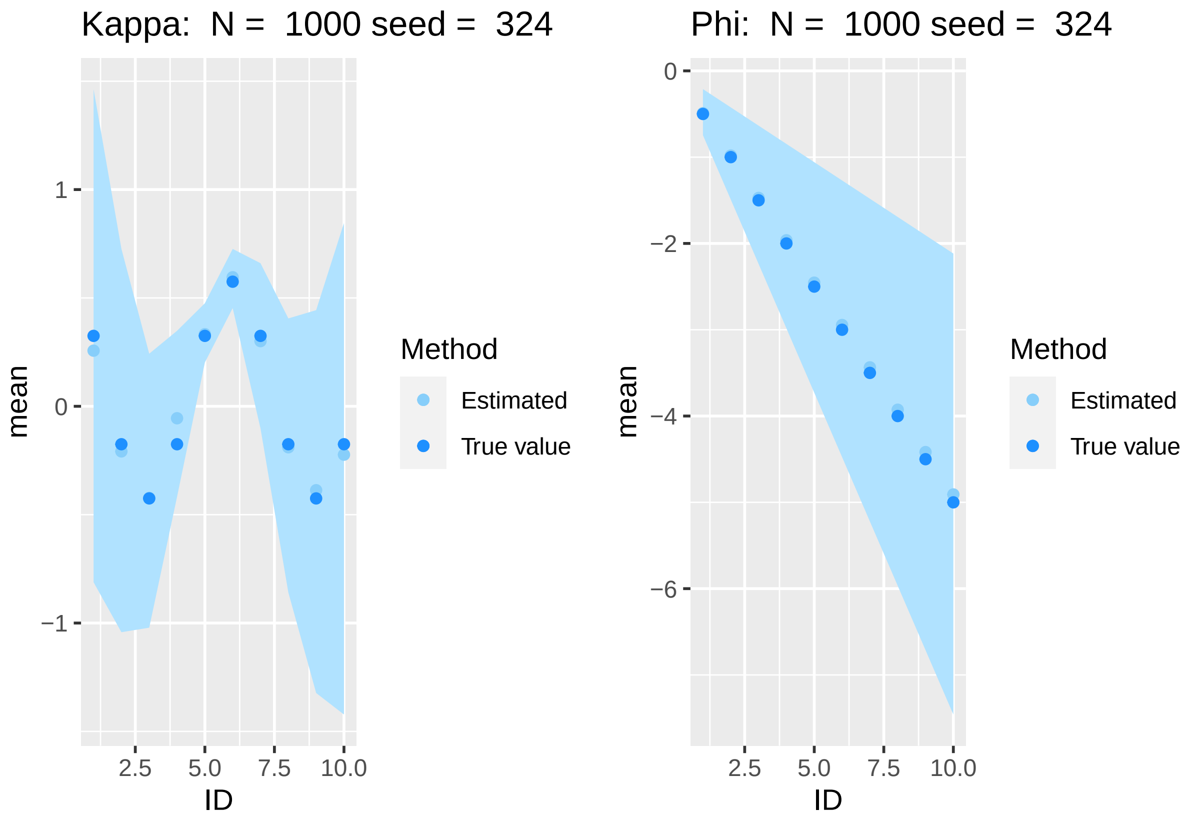
```
##            mean         sd 0.025quant   0.5quant 0.975quant       mode
## Int  1.0035890 0.07422461  0.8491594  1.0058228  1.1444290  1.009296
## phi -0.4908407 0.13432273 -0.7458678 -0.4947697 -0.2116967 -0.500851
##              kld
## Int 3.776155e-09
## phi 1.491852e-08
```

```
##                                mean           sd   0.025quant     0.5quant
## Precision for alpha       14.313876    4.9284225    6.7601798    13.638706
## Precision for beta        94.028341   40.2306329   35.3956417    87.738441
## Precision for kappa        7.258060    3.8469486    2.1122661     6.522049
## Precision for gamma        1.923056    0.6549112    0.9014636     1.839955
## Precision for epsilon 14967.372255 4226.9011223 8729.9613761 14257.941787
##                          0.975quant         mode
## Precision for alpha       25.906478    12.324135
```

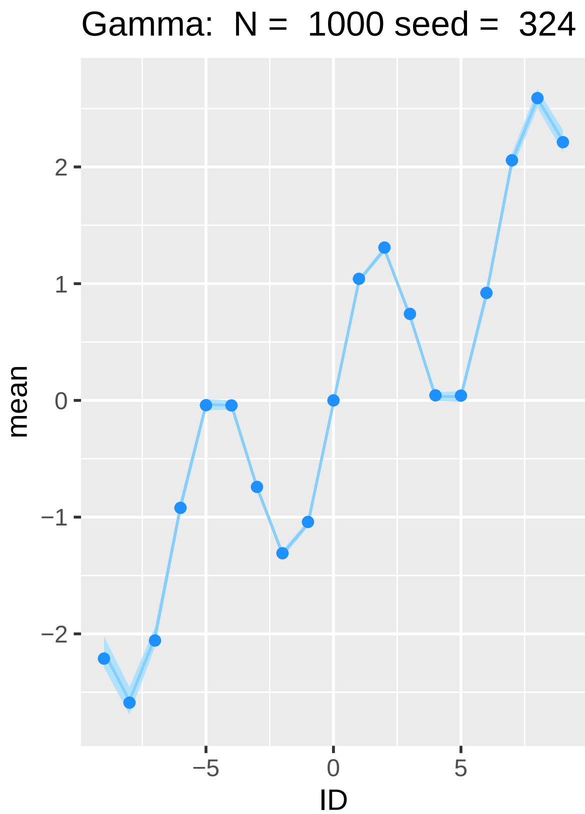
```
## Precision for beta       189.839476    74.690789
## Precision for kappa       16.777590     5.000508
## Precision for gamma        3.447572     1.673523
## Precision for epsilon 25172.812642 12924.212719
```

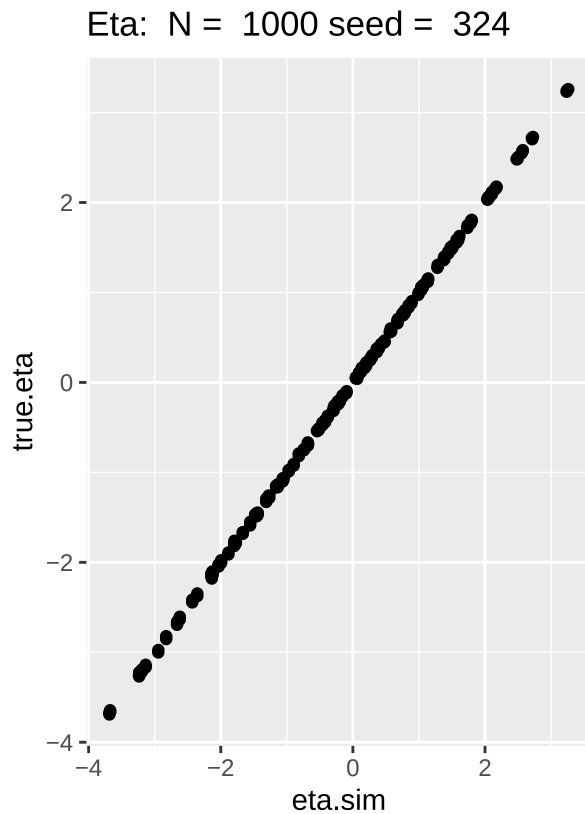






Kappa: N = 1000 seed = 324
mean
1
0
−1
2.5
5.0
7.5
10.0
ID
Method
Estimated
True value
Phi: N = 1000 seed = 324
mean
0
−2
−4
−6
2.5
5.0
7.5
10.0
ID
Method
Estimated
True value





Inlabru - further explanation.

We have a non-linear predictor $\eta_{x,t}$ that is a function of some latent effects, $\mathbf{u}$:

$$\eta_{x,t} = C + \alpha_x, \phi \cdot t \cdot \beta_x + \beta_x \cdot \kappa_t + \epsilon, \quad \mathbf{u} = [\alpha_1, ..., \alpha_{N_x}, \beta_1, ..., \beta_{N_x}, \phi, \kappa_1, ..., \kappa_{N_t}].$$

Using a Taylor approximation around some point $\mathbf{u}_0$ one gets

$$\bar{\boldsymbol{\eta}} = \boldsymbol{\eta}_{x,t}(\mathbf{u}_0) + B(\mathbf{u} - \mathbf{u}_0),$$

where B is the derivative matrix of $\boldsymbol{\eta}$ evaluated at $\mathbf{u}_0$. This linearized predictor is then used to run inla, and we obtain approximate marginal posterior distributions for the latent effects, hyperparameters and the predictor $\eta_{x,t}$. Inlabru finds the optimal linearization point $\mathbf{u}_0$ through a fixed-point iteration with inla. For each step s, the next linearization point $\mathbf{u}_s$ is set to the point that maximizes the posterior distribution for the latent effects that resulted from the inla approximation using the linearization from the last step $\mathbf{u}_{s-1}$:

$$\mathbf{u}_s = \text{argmax}_{\mathbf{u}} \; \bar{p}_{\mathbf{u}_{s-1}}(\mathbf{u} \mid \mathbf{y}, \hat{\boldsymbol{\theta}}) =: f(\bar{p}_{\mathbf{u}_{s-1}}), \quad \hat{\boldsymbol{\theta}} = \text{argmax}_{\boldsymbol{\theta}} \; \bar{p}_{\mathbf{u}_{s-1}}(\boldsymbol{\theta} \mid \mathbf{y}).$$

Inlabru runs these iterations until approximate convergence:

$$\mathbf{u}_s \approx f(\bar{p}_{\mathbf{u}_{s-1}}).$$