

The A-local feature

Håvard Rue (hrue@r-inla.org)

Dec, 2025

Introduction

This is a short note to discuss the use of the new “A-local” feature. **Warning:** Some of this content is rather advanced, and we may come with a more friendlier vignette at a later stage.

The basic issue this feature address, is that the linear predictor sometimes contains more than one element from one or more model component. For example

$$\eta_i = \sum_j A_{ij}x_j + \dots$$

where x is defined in $f(\cdot)$ and A is a fixed matrix. Here, “...” represents other model components in the model that are not relevant for this discussion. Earlier, such cases have been addressed by removing the A-matrix locally and redefine it globally, by introducing $\eta^* = A\eta$ for another A-matrix. The overall model, in terms of η^* is the same.

The step with introducing η^* is often unnecessary complicated and leads to a construction which also depends on the irrelevant parts of the model given above in “...”. This can now be avoided in many cases.

First example

The general format is

$$y \sim f(k, w, \text{model} = "...", \dots, \text{A.local} = A) + \dots$$

Let x represent the model component with elements $x_1, \dots, x_n$, then the above says that

$$\eta_i = w_k x_{k_i} + \sum_j A_{ij}x_j + \dots$$

The redundancy is obvious, A represents here an alternative approach to give (k, w) , and we could represent the k -part as A as well. The *values* for which x are defined, here $1, \dots, n$, is extracted from k and not A . If we define *values* manually, then (k, w) can be just dummy and we can move to using A only. An in-between approach is also possible: Use k as normal but to set $w = 0$ (-vector), so that all information x_j ’s influence on η_i is from the i -th row $A_{i1}, \dots, A_{in}$, but the information about the *values* are taken from k .

Here is a simple example with the model

$$y_i = 1 + x_i + x_{i-1} + \epsilon_i$$

where x_i is *iid*, which is simulated using

```
n <- 100
x <- rnorm(n)
i <- 1:n
```

```
im <- c(NA, 1:(n-1))
y <- 1 + x + c(0, x[-n]) + rnorm(n)
```

The first (traditional) approach is to use *copy*

```
r1 <- inla(y ~ 1 + f(i, model="iid") + f(im, copy="i"),
  family="stdnormal",
  data = data.frame(y,i,im))
```

The second approach is to use *Alocal* to define the additional contribution

```
A <- matrix(0,n,n)
A[row(A)-col(A) == 1] <- 1
A <- inla.as.sparse(A)
r2 <- inla(y ~ 1 + f(i, model="iid", A.local=A),
  family="stdnormal",
  data = list(y=y,i=i,A=A))
```

The third approach is to use *Alocal* with all info in the *A*-matrix

```
diag(A) <- 1 ## we add also the diagonal
r3 <- inla(y ~ 1 + f(i.NA, model="iid", A.local=A, values=1:n),
  family="stdnormal",
  data = list(y=y, i.NA=rep(NA,n), A=A, n=n))
```

The forth approach use *Alocal* and also the *i*-index to define *values*. Note that *w.zero* zero-out the contribution, so in practice we only use the *A*-matrix.

```
r4 <- inla(y ~ 1 + f(i, w.zero, model="iid", A.local=A),
  family="stdnormal",
  data = list(y=y, i=i, w.zero=rep(0,n), A=A, n=n))
```

The four approaches gives the same results

```
as.vector(c(r1$mlik[1,], r2$mlik[1,], r3$mlik[1,], r4$mlik[1,]))
```

```
## [1] -199.7263 -199.7263 -199.7263 -199.7263
```

The copy-approach (first element) is slightly different as it use an *almost*-copy and not an identical copy.

Example with replicate/group

A.local is *not* replicated when *replicate* and/or *group* is used. This allows extra flexibility.

If the dimensions of *A.local* do not match the dimension of η , it still works. Like η has length n , then *A* is normally a (sparse) matrix of size $n' \times m$ for some m and $n' = n$. If $n' < n$ then row $n' + 1, \dots, n$ are treated as empty or no *A*-matrix. If $n' > n$ then only the first n rows are used, with no warnings given. Here is an example where $n' = 2n$ as we stack the two *A*'s. The result is the same as before

```
AA <- rbind(A,A)
r5 <- inla(y ~ 1 + f(i, w.zero, model="iid", A.local=AA),
  family="stdnormal",
  data = list(y=y, i=i, w.zero=rep(0,n), AA=AA, n=n))
r5$mlik[1,]
```

```
## log marginal-likelihood (integration)
## -199.7263
```

Here is a somewhat more advanced example with two replicated AR1 processes.

```
n <- 1000
phi <- 0.9
x <- scale(arima.sim(n=n, model = list(ar = phi)))
xx <- scale(arima.sim(n=n, model = list(ar = phi)))
```

where each process is convolved with its kernel

```
h <- 10L
nh <- h + 1L
kern <- dnorm(-h:0, sd=h)
kern <- kern / sum(kern)
A <- inla.as.sparse(INLA:::inla.toeplitz(c(kern[nh], rep(0, n-nh), kern[1:h])))

h <- 30L
nh <- h + 1L
kern <- dnorm(-h:0, sd= h)
kern <- kern / sum(kern)
B <- inla.as.sparse(INLA:::inla.toeplitz(c(kern[nh], rep(0, n-nh), kern[1:h])))
```

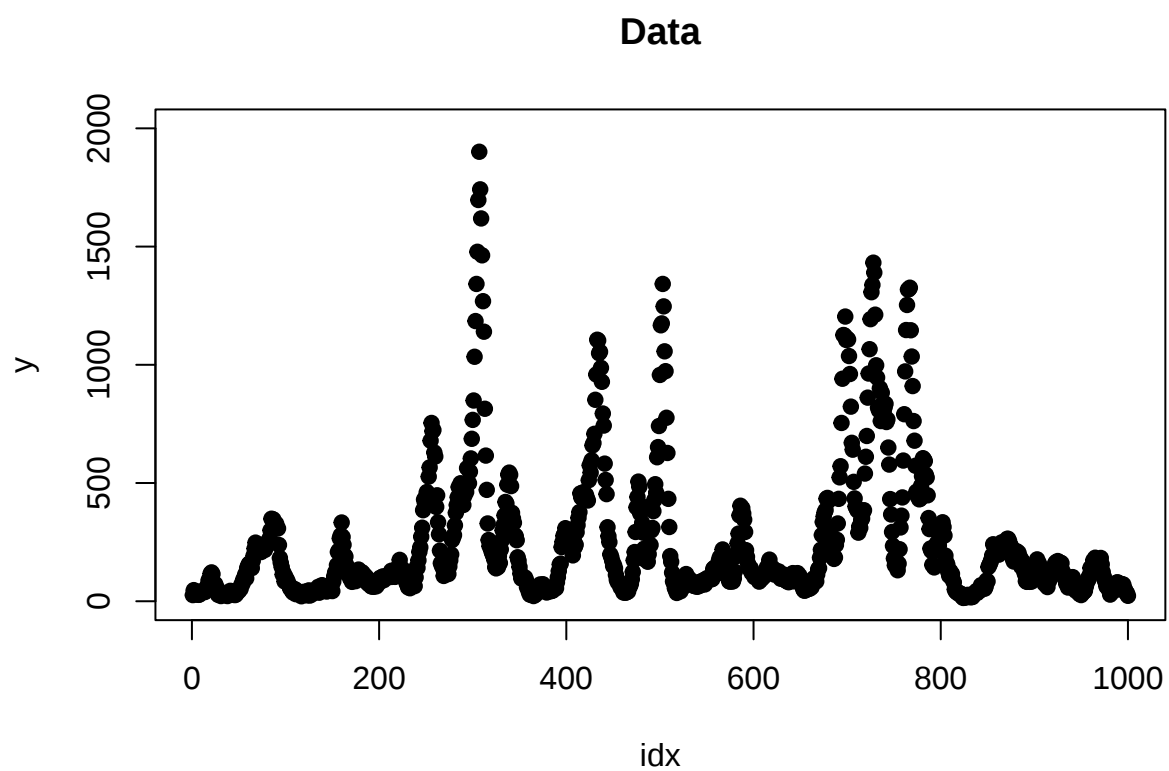
to form the lin.predictor

```
eta <- 5 + as.vector(A %*% x) + as.vector(B %*% xx)
y <- rpois(n, lambda = exp(eta))
```

Using that the second replicate is stored at indices $n + 1, \dots, 2n$, we can just bind A and B together

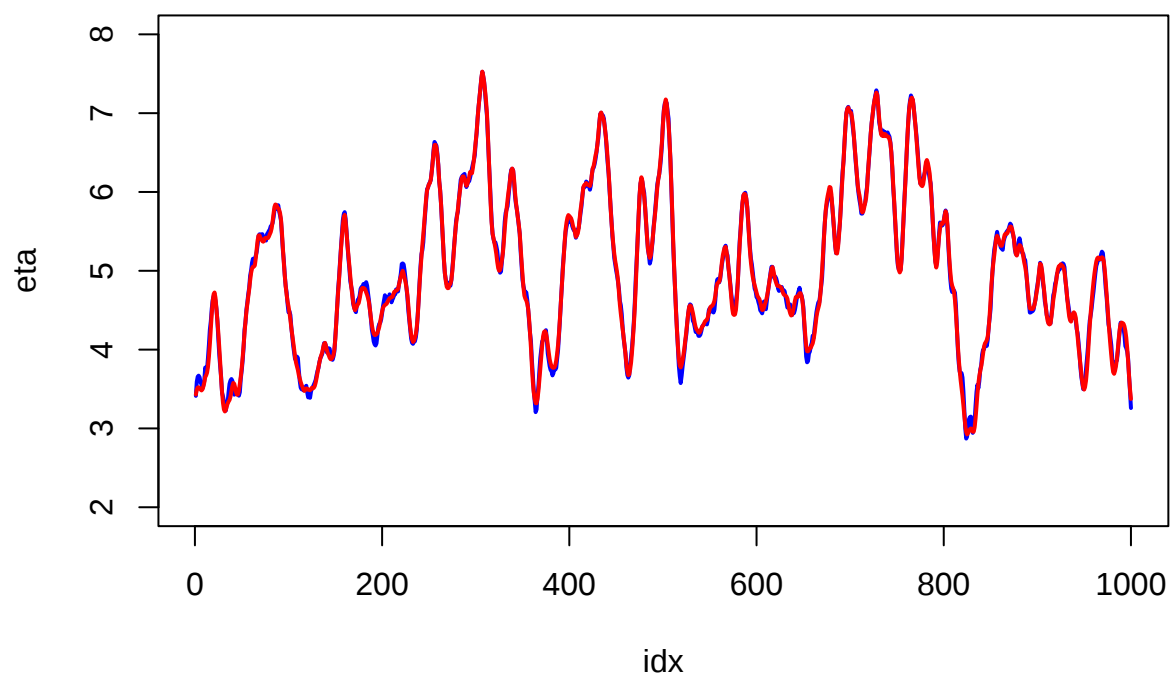
```
AB <- cbind(A, B)
r <- inla(y ~ 1 + f(idx, w, model="ar1", constr = T, A.local = AB, nrep = 2),
  data = list(y = y, idx = 1:n, AB = AB, w = rep(0, n)),
  family = "poisson")

idx <- 1:n
plot(idx, y, pch=19, ylim = range(pretty(y)), main="Data")
```



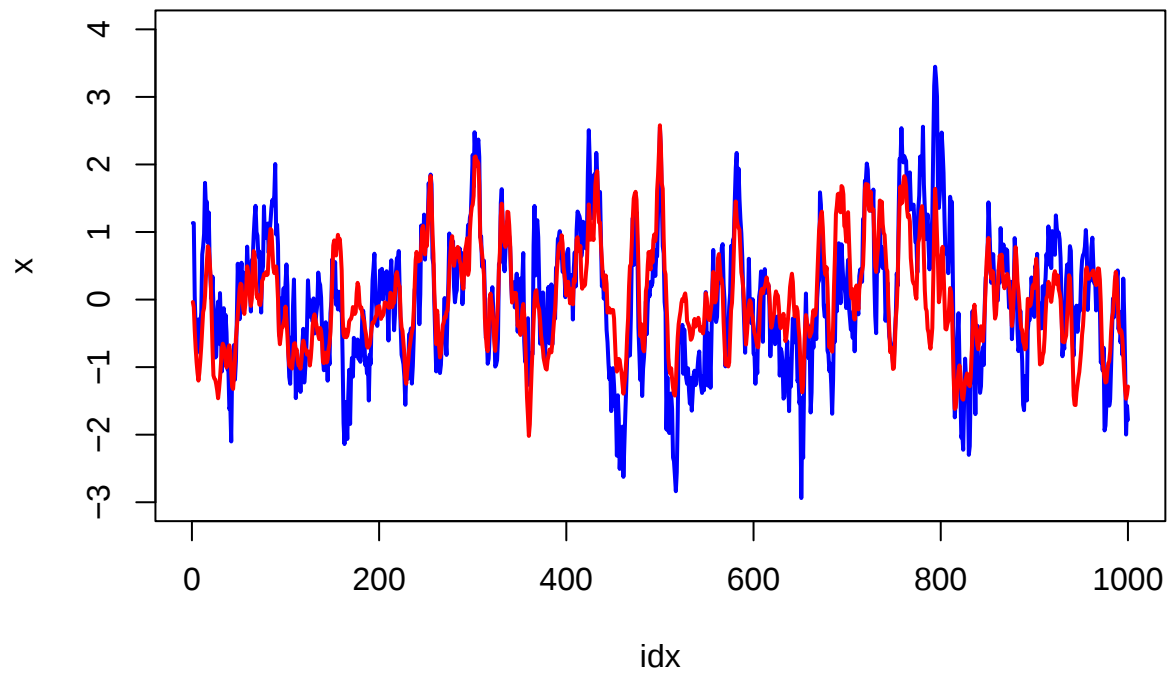
```
plot(idx, eta, col="blue", lwd=2, type="l", ylim = range(pretty(eta)))  
lines(idx, r$summary.linear.predictor$mean, col="red", lwd=2)  
title("eta.true and E(eta|...)")
```

eta.true and E(eta|...)



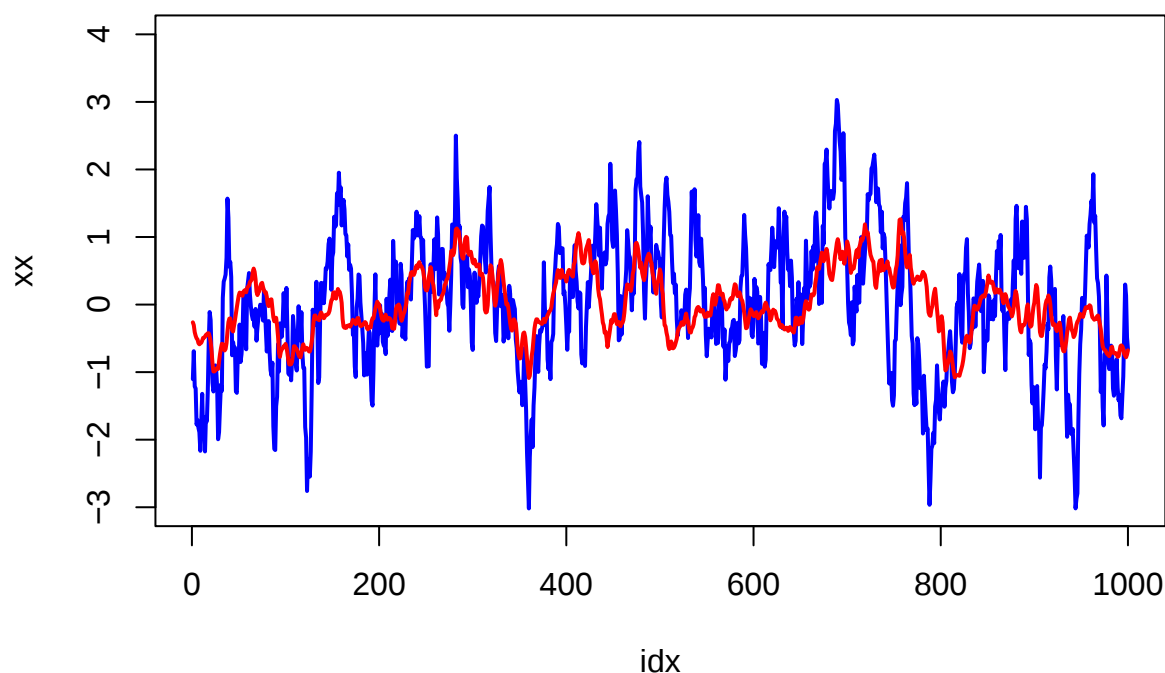
```
plot(idx, x, col="blue", lwd=2, type="l", ylim = range(pretty(x)))  
lines(idx, r$summary.random$idx$mean[1:n], col="red", lwd=2)  
title("AR1 and E(AR1|...)")
```

AR1 and $E(\text{AR1}|\dots)$



```
plot(idx, xx, col="blue", lwd=2, type="l", ylim = range(pretty(x)))  
lines(idx, r$summary.random$idx$mean[n + 1:n], col="red", lwd=2)  
title("AR1 and  $E(\text{AR1}|\dots)$ ")
```

AR1 and E(AR1|...)



and the summary is

```
summary(r)
```

```
## Time used:
##   Pre = 0.135, Running = 4.88, Post = 0.0488, Total = 5.07
## Fixed effects:
##           mean      sd 0.025quant 0.5quant 0.975quant  mode kld
## (Intercept) 4.997 0.003      4.991   4.997      5.003 4.997   0
##
## Random effects:
##   Name      Model
##   idx AR1 model
##
## Model hyperparameters:
##           mean      sd 0.025quant 0.5quant 0.975quant  mode
## Precision for idx 1.150 0.141      0.890   1.144      1.445 1.137
## Rho for idx       0.882 0.017      0.845   0.883      0.914 0.884
##
## Marginal log-Likelihood: -4499.59
##   is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

The z model

The new feature gives an easy way to implement the z-model, which is the classic mixed-effect model formulation

$$\eta = \dots + Zz$$

where Z is a fixed matrix and z is zero mean Normal with precision τQ . This is implemented in model “z” as follows.

```
n <- 300
m <- 30
Z <- matrix(rnorm(n*m), n, m)
rho <- 0.8
Qz <- toeplitz(rho^(0:(m-1)))
z <- inla.qsample(1, Q=Qz)
eta <- Z %*% z
s <- 0.1
y <- eta + rnorm(n, sd = s)
```

Using the z-model we get

```
r <- inla(y ~ -1 + f(idx, model="z", Z=Z, Cmatrix=Qz, precision=exp(20),
      hyper = list(prec = list(initial = 0, fixed = TRUE))),
      data = list(y=y, idx=1:n, n=n),
      control.family = list(hyper = list(prec = list(initial = log(1/s^2), fixed= TRUE))))
```

Using A.local we get

```
rr <- inla(y ~ -1 + f(idx, model="generic", A.local=Z, Cmatrix=Qz, values=1:m,
      hyper = list(prec = list(initial = 0, fixed = TRUE))),
      data = list(y=y, idx=rep(NA,n), n=n, m=m),
      control.family = list(hyper = list(prec = list(initial = log(1/s^2), fixed= TRUE))))
```

and we can compare (noting that “generic” model does not include the normalization constant wrt to the precision matrix; see the documentation)

```
r$mlik - (rr$mlik + 0.5 * determinant(Qz, log=TRUE)$modulus)
```

```
## [1]
## log marginal-likelihood (integration) 0.001482028
## log marginal-likelihood (Gaussian) 0.001482028
mean(abs(r$summary.random$idx$mean[n + 1:m] - rr$summary.random$idx$mean))
## [1] 4.741727e-09
mean(abs(r$summary.random$idx$sd[n + 1:m] / rr$summary.random$idx$sd - 1))
## [1] 1.031437e-07
```

Using replicate, again

This is motivated from an example from the discussion-list, where

$$y = \dots + f(\text{idx}, x1, \text{model}=\text{"besag"}, \dots, \text{replicate}=\text{r1}) + f(\text{idx2}, x2, \text{copy}=\text{"idx"}, \text{replicate}=\text{r2})$$

where the second term using a replication hence sharing the same hyperparameter. Here its written out for $m = 2$ terms, but the aim was several terms like $m \approx 15$.

This can now be done very efficient

```
g <- inla.read.graph(system.file("demodata/germany.graph", package="INLA"))
source(system.file("demodata/Bym-map.R", package="INLA"))
n <- g$n
m <- 4
s <- 0.1
X <- matrix(rnorm(n*m), n, m)
y <- rowSums(X) + rnorm(n, sd = s)
A <- inla.as.sparse(diag(X[, 1]))
for(i in 2:m) {
  A <- cbind(A, inla.as.sparse(diag(X[, i])))
}

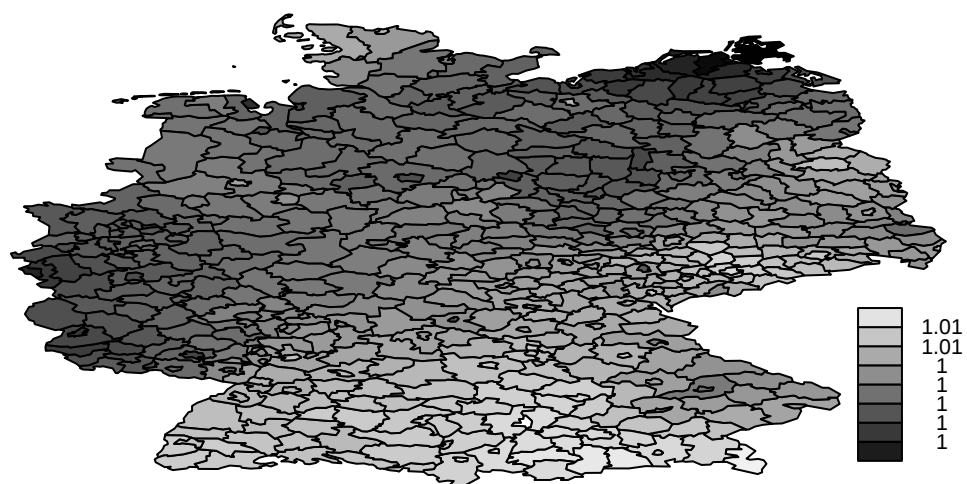
r <- inla(y ~ -1 +
  f(idx.na,
    model = "besag",
    scale.model = TRUE,
    constr = FALSE,
    nrep = m,
    graph = g,
    A.local = A,
    values = 1:n,
    hyper = list(prec = list(prior = "pc.prec",
                             param = c(0.5, 0.01)))),
  family = "normal",
  control.family = list(hyper = list(prec = list(initial = log(1/s^2),
                                                fixed = TRUE))),
  data = list(n = n, m = m, y = y, X = X,
    idx.na = rep(NA, n)))

beta <- matrix(r$summary.random$idx.na$mean, n,m)
summary(beta)
```

##	V1	V2	V3	V4
## Min.	:1.000	Min. :0.9938	Min. :0.9982	Min. :1.001
## 1st Qu.	:1.003	1st Qu.:0.9970	1st Qu.:1.0006	1st Qu.:1.004
## Median	:1.004	Median :0.9982	Median :1.0011	Median :1.005
## Mean	:1.004	Mean :0.9980	Mean :1.0012	Mean :1.005
## 3rd Qu.	:1.004	3rd Qu.:0.9988	3rd Qu.:1.0017	3rd Qu.:1.006
## Max.	:1.006	Max. :1.0013	Max. :1.0088	Max. :1.009

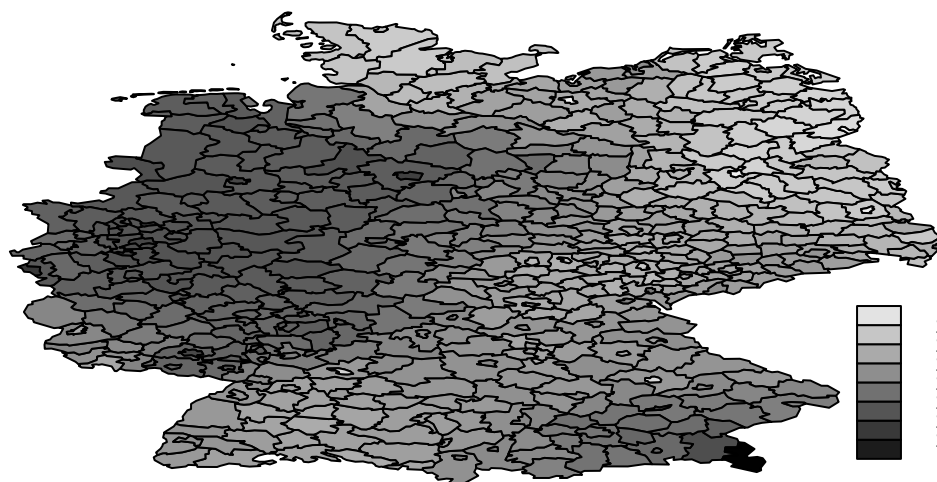
```
i=1; Bym.map(r$summary.random$idx.na$mean[ (i-1) * n + 1:n])
title(paste("coeff ", i))
```

coeff 1



```
i=2; Bym.map(r$summary.random$idx.na$mean[ (i-1) * n + 1:n])  
title(paste("coeff ", i))
```

coeff 2



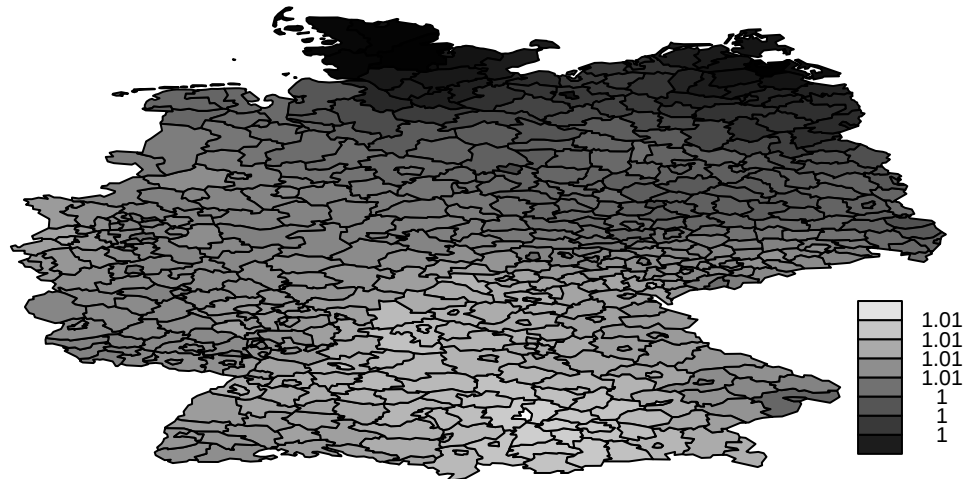
```
i=3; Bym.map(r$summary.random$idx.na$mean[ (i-1) * n + 1:n])  
title(paste("coeff ", i))
```

coeff 3



```
i=4; Bym.map(r$summary.random$idx.na$mean[ (i-1) * n + 1:n])  
title(paste("coeff ", i))
```

coeff 4



Using ‘Alocal’ in copy

This is to verify that Alocal-feature works as expected with copy. Consider this simple example

```
n <- 10
y <- 1:n
A <- matrix(0, n, n)
A[lower.tri(A, diag = TRUE)] <- 1
```

where $\eta_i = 1, i = 1, \dots, n$ match data y as

```
cbind(y, A %*% rep(1,n))
```

```
##      y
## [1,] 1  1
## [2,] 2  2
## [3,] 3  3
## [4,] 4  4
## [5,] 5  5
## [6,] 6  6
## [7,] 7  7
## [8,] 8  8
## [9,] 9  9
## [10,] 10 10
```

We can verify that this works as expected with

```

idx <- 1:n
iidx <- rep(NA, n)
w0 <- rep(0, n)
r <- inla(y ~ -1 +
          f(idx,
            w0,
            model = "iid",
            values = 1:n,
            hyper = list(prec = list(initial = -20,
                                      fixed = TRUE))) +
          f(iidx,
            copy = "idx",
            values = 1:n,
            A.local = A),
          data = list(idx = idx,
                      iidx = iidx,
                      w0 = w0,
                      A = A,
                      n = n),
          family = "stdnormal")

```

The `idx` part only defines the iid model, but is not used, as the weights `w0` are all zero. The distribution is essentially uniform due to the small fixed precision.

The `iidx` part copy `idx`, and then apply matrix `A` in `A.local`. This mean that both `idx` and `iidx` should have mean 1, and the `A`-matrix will make this fit the data.

```

round(dig=3, cbind(r$summary.random$idx$mean,
                   r$summary.random$iidx$mean))

```

```

##      [,1] [,2]
## [1,]    1    1
## [2,]    1    1
## [3,]    1    1
## [4,]    1    1
## [5,]    1    1
## [6,]    1    1
## [7,]    1    1
## [8,]    1    1
## [9,]    1    1
## [10,]   1    1

```

```

round(dig=3, cbind(y, A %*% r$summary.random$iidx$mean))

```

```

##      y
## [1,]  1  1
## [2,]  2  2
## [3,]  3  3
## [4,]  4  4
## [5,]  5  5
## [6,]  6  6
## [7,]  7  7
## [8,]  8  8
## [9,]  9  9
## [10,] 10 10

```

If we add scaling this still works

```
r <- inla(y ~ -1 +
  f(idx,
    w0,
    model = "iid",
    values = 1:n,
    hyper = list(prec = list(initial = -20,
                              fixed = TRUE))) +
  f(iidx,
    copy = "idx",
    values = 1:n,
    A.local = A,
    hyper = list(beta = list(initial = 0.5))),
data = list(idx = idx,
  iidx = iidx,
  w0 = w0,
  A = A,
  n = n),
family = "stdnormal")
```

Due to the scaling, then idx has to bump to 2, but iidx is still 1 to fit data

```
round(dig=3, cbind(r$summary.random$idx$mean,
  r$summary.random$iidx$mean))
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    2    1
## [3,]    2    1
## [4,]    2    1
## [5,]    2    1
## [6,]    2    1
## [7,]    2    1
## [8,]    2    1
## [9,]    2    1
## [10,]   2    1
```