Example 6.1

Disease mapping: from foundations to multidimensional modeling

Martinez-Beneito M.A. and Botella-Rocamora P.

This document reproduces the analysis made at Example 6.1 of the book: "Disease mapping: from foundations to multidimensional modeling" by Martinez-Beneito M.A. and Botella-Rocamora P., published by CRC press in 2019. You can watch the analysis made with full detail at this pdf document, or even execute it if you want with the material available at https://github.com/MigueBeneito/DMBook. Anyway, this pdf file should be enough for following most of the details of the analysis made for this example.

The statistical analysis below has been run in `R`, by additionally using the library `Rmarkdown`, so be sure that you have this software installed if you want to reproduce by yourself the content of this document. In that case we advise you to download first the annex material at https://github.com/MigueBeneito/DMBook, open with `Rstudio` the corresponding `.Rproj` file that you will find at the folder corresponding to this example and compile the corresponding `.Rmd` document. This will allow you to reproduce the whole statistical analysis below.

This document has been executed with real data that are not provided in order to preserve their confidentiality. Slightly modified data are provided instead, as described in Chapter 1 of the book. Thus, when reproducing this document you will not obtain exactly the same results, although they should be very close to those shown here.

Libraries and data loading

```
# Libraries loading
#------------------
if (!require(RColorBrewer)) {
    install.packages("RColorBrewer")
    library(RColorBrewer)
}
if (!require(rgdal)) {
    install.packages("rgdal")
    library(rgdal)
}
if (!require(INLA)) {
    install.packages("INLA", repos = c(getOption("repos"), INLA = "https://inla.r-inla-download.org/R/s
        dep = TRUE)
    library(INLA)
}
if (!require(splines)) {
    install.packages("splines")
    library(splines)
}

# Data loading
#-------------
# For reproducing the document, the following line should be changed to
# load('../Data/ObsOral-mod.Rdata') since that file contains the
# modified data making it possible to reproduce this document.
load("../Data/ObsOral.Rdata")
# load('../Data/ObsOral-mod.Rdata')
```

```r
load("../Data/ExpOral.Rdata")
load("../Data/VR.Rdata")
```

R function for calculating the DIC criterion of the models fitted

The function below computes the DIC criterion for disease mapping models fitted with WinBUGS. It returns DIC values comparable to those reported by INLA, in contrast to WinBUGS. See annex material for Example 4.3.

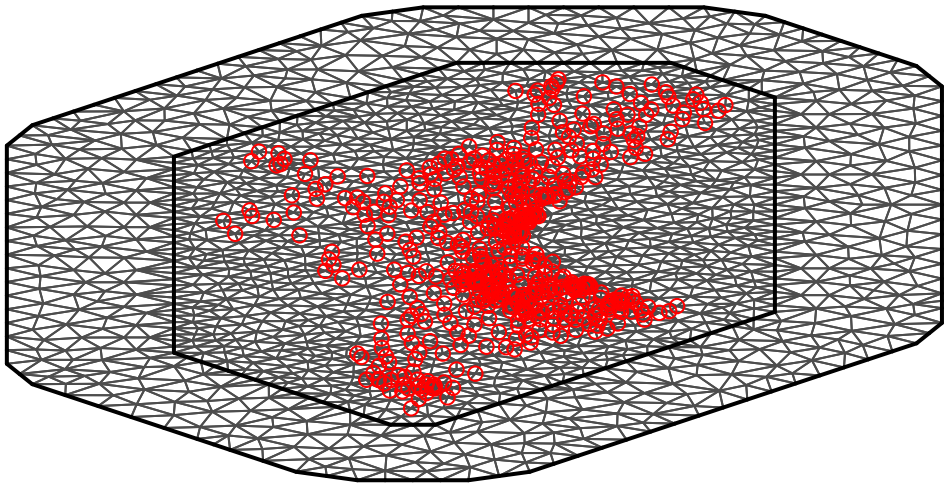
```r
# Arguments: Simu.sSMRs: matrix of dimensions n.IterXn.Units where
# n.Iter are the number of MCMC iterations saved and n.Units the number
# of spatial units in the analysis. You will typically find this as a
# submatrix of the sims.matrix element of any bugs object. O: Vector of
# length n.Units with the observed deaths per spatial unit. E: Vector
# of length n.Units with the expected deaths per spatial unit.
DICPoisson = function(Simu.sSMRs, O, E) {
    mu = t(apply(Simu.sSMRs/100, 1, function(x) {
        x * E
    }))
    D = apply(mu, 1, function(x) {
        -2 * sum(O * log(x) - x - lfactorial(O))
    })
    Dmean = mean(D)
    mumean = apply(Simu.sSMRs/100, 2, mean) * E
    DinMean = -2 * sum(O * log(mumean) - mumean - lfactorial(O))
    # if(save==TRUE){return(c(Dmedia,Dmedia-DenMedia,2*Dmedia-DenMedia))}
    cat("D=", Dmean, "pD=", Dmean - DinMean, "DIC=", 2 * Dmean - DinMean,
        "\n")
}
```

Geostatistical model

```r
# centroids of each municipality in the Valencian Region
centroids = t(sapply(VR.cart@polygons, function(x) {
    apply(x@Polygons[[1]]@coords, 2, mean)
}))

# data for smoothing the SMRs and their location
data = data.frame(list(x = centroids[, 1], y = centroids[, 2], O = Obs.muni,
    E = Exp.muni))

# mesh for defining the SPDE approach on our region of study
mesh = inla.mesh.2d(loc = centroids, max.edge = c(1, 10), cutoff = 1, max.n = c(1500,
    500))
plot(mesh)
points(centroids, col = "red")
```




```r
# SPDE stuff
spde = inla.spde2.pcmatern(mesh, prior.range = c(0.3, 0.5), prior.sigma = c(1,
    0.01))
indexs = inla.spde.make.index("s", spde$n.spde)
A = inla.spde.make.A(mesh = mesh, loc = centroids)

# Definition of a grid of points for predicting the Gaussian process to
# be fitted
VR.all = aggregate(VR.cart, FUN = length)
plot(VR.all)
aux = VR.all@bbox
gridx = seq(from = aux[1, 1], to = aux[1, 2], length.out = 100)
gridy = seq(from = aux[2, 1], to = aux[2, 2], length.out = 200)
grid = matrix(nrow = 0, ncol = 2)
for (i in 1:length(gridx)) {
    aux2 = apply(cbind(gridx[i], gridy), 1, function(x) {
        over(VR.all, SpatialPoints(matrix(x, ncol = 2)))
    })
    if (length(which(!is.na(aux2))) > 0) {
        grid = rbind(grid, cbind(gridx[i], gridy[which(!is.na(aux2))]))
    }
}
points(grid, pch = ".")
```

```r
# 7391 points to predict
dim(grid)[1]
```

```
## [1] 7391
```

```r
Ap = inla.spde.make.A(mesh = mesh, loc = grid)

# stack for estimating the geostatistical process
stk.e = inla.stack(tag = "dat", data = list(O = Obs.muni, E = Exp.muni),
    A = A, effects = list(s = indexs))
# stack for predicting the geostatistical process
stk.p = inla.stack(tag = "pred", data = list(O = NA, E = NA), A = Ap, effects = list(s = indexs))
# stk.full has stk.e and stk.p
stk.full = inla.stack(stk.e, stk.p)

formula = O ~ 1 + f(s, model = spde)
# SPDE model
resulGeo = inla(formula, family = "poisson", E = E, control.family = list(link = "log"),
    data = inla.stack.data(stk.e), control.compute = list(dic = TRUE),
    control.predictor = list(compute = TRUE, cdf = c(log(1)), link = 1,
        A = inla.stack.A(stk.e)))
resulGeo$cpu.used
```

```
##  Pre-processing    Running inla Post-processing           Total
##        6.004120       78.276929        0.222404       84.503453
```

```r
# SPDE model with predictions
resulGeo.pred = inla(formula, family = "poisson", E = E, control.family = list(link = "log"),
```

```
    data = inla.stack.data(stk.full), control.compute = list(dic = TRUE),
    control.predictor = list(compute = TRUE, cdf = c(log(1)), link = 1,
        A = inla.stack.A(stk.full)))
resulGeo.pred$cpu.used
```

```
##  Pre-processing    Running inla Post-processing           Total
##        5.756564      663.021528        1.098110      669.876202
```

```
# summary for the main parameters in the model
summary(resulGeo)
```

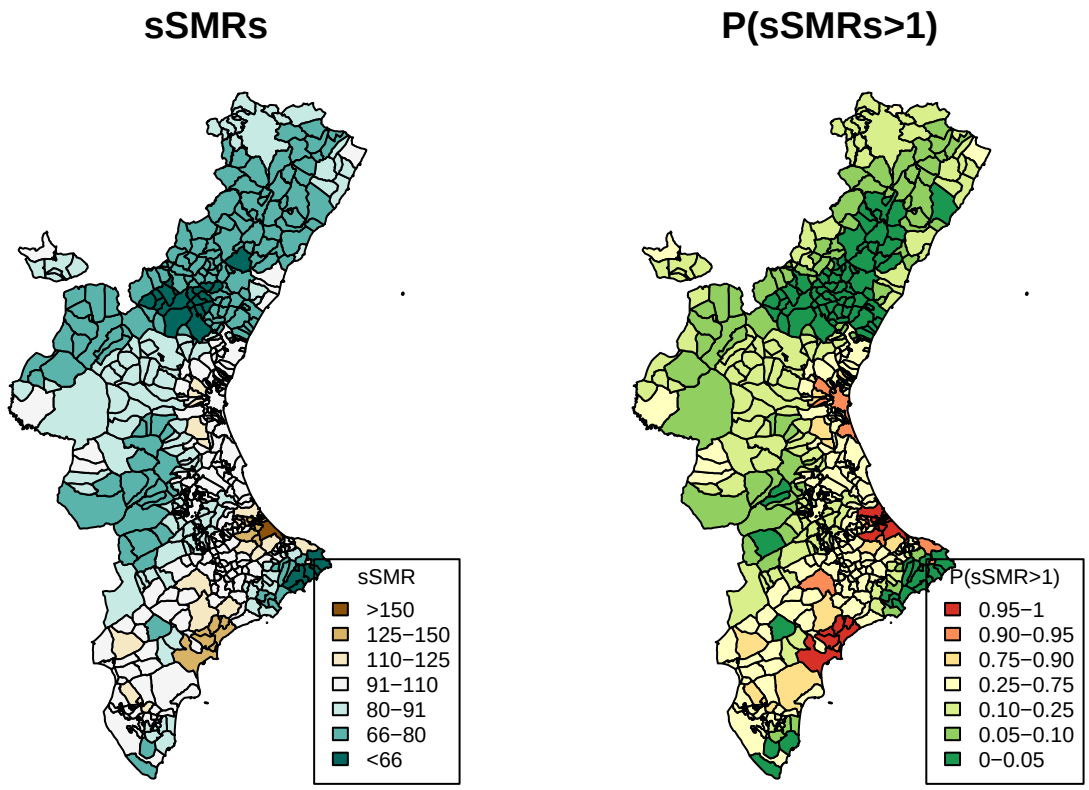
```
##
## Call:
## c("inla(formula = formula, family = \"poisson\", data = inla.stack.data(stk.e), ",  "    E = E, cont
##
## Time used:
##  Pre-processing    Running inla Post-processing           Total
##          6.2099         95.1149          0.2313        101.5562
##
## Fixed effects:
##                mean     sd 0.025quant 0.5quant 0.975quant   mode kld
## (Intercept) -0.1816 0.0871    -0.4091  -0.1713    -0.0421 -0.162   0
##
## Random effects:
## Name   Model
##  s   SPDE2 model
##
## Model hyperparameters:
##                mean     sd 0.025quant 0.5quant 0.975quant   mode
## Range for s 0.6476 1.2329     0.0615   0.3151     3.2971 0.1324
## Stdev for s 0.2020 0.1993     0.0137   0.1432     0.7291 0.0374
##
## Expected number of effective parameters(std dev): 50.25(11.43)
## Number of equivalent replicates : 10.75
##
## Deviance Information Criterion (DIC) ...............: 1783.69
## Deviance Information Criterion (DIC, saturated) ....: 651.66
## Effective number of parameters .....................: 53.02
##
## Marginal log-Likelihood:  -961.98
## Posterior marginals for linear predictor and fitted values computed
```

```
# Choropleth maps with the sSMRs and P(sSMRs>1) at the centroid of each
# municipality
colors = brewer.pal(7, "BrBG")[7:1]
colorsProb = brewer.pal(7, "RdYlGn")[7:1]
cuts = c(-0.1, 1/1.5, 1/1.25, 1/1.1, 1.1, 1.25, 1.5, 100)
cuts.Prob = c(-0.1, 0.05, 0.1, 0.25, 0.75, 0.9, 0.95, 100)

sSMRs = 100 * resulGeo.pred$summary.fitted.values[1:540, "mean"]
sSMRs.P = 1 - resulGeo.pred$summary.fitted.values[1:540, "1 cdf"]

par(mfrow = c(1, 2))
par(mar = c(1, 1, 2, 1) + 0.1)
plot(VR.cart, col = colors[as.numeric(cut(sSMRs, 100 * cuts))])
```

```
title("sSMRs", cex = 0.75)
legend(x = "bottomright", fill = colors[7:1], legend = c(">150", "125-150",
    "110-125", "91-110", "80-91", "66-80", "<66"), cex = 0.65, inset = 0.03,
    title = "sSMR")
plot(VR.cart, col = colorsProb[as.numeric(cut(sSMRs.P, cuts.Prob))])
title("P(sSMRs>1)", cex = 0.75)
legend(x = "bottomright", fill = colorsProb[7:1], legend = c("0.95-1",
    "0.90-0.95", "0.75-0.90", "0.25-0.75", "0.10-0.25", "0.05-0.10", "0-0.05"),
    cex = 0.65, inset = 0.03, title = "P(sSMR>1)")
```



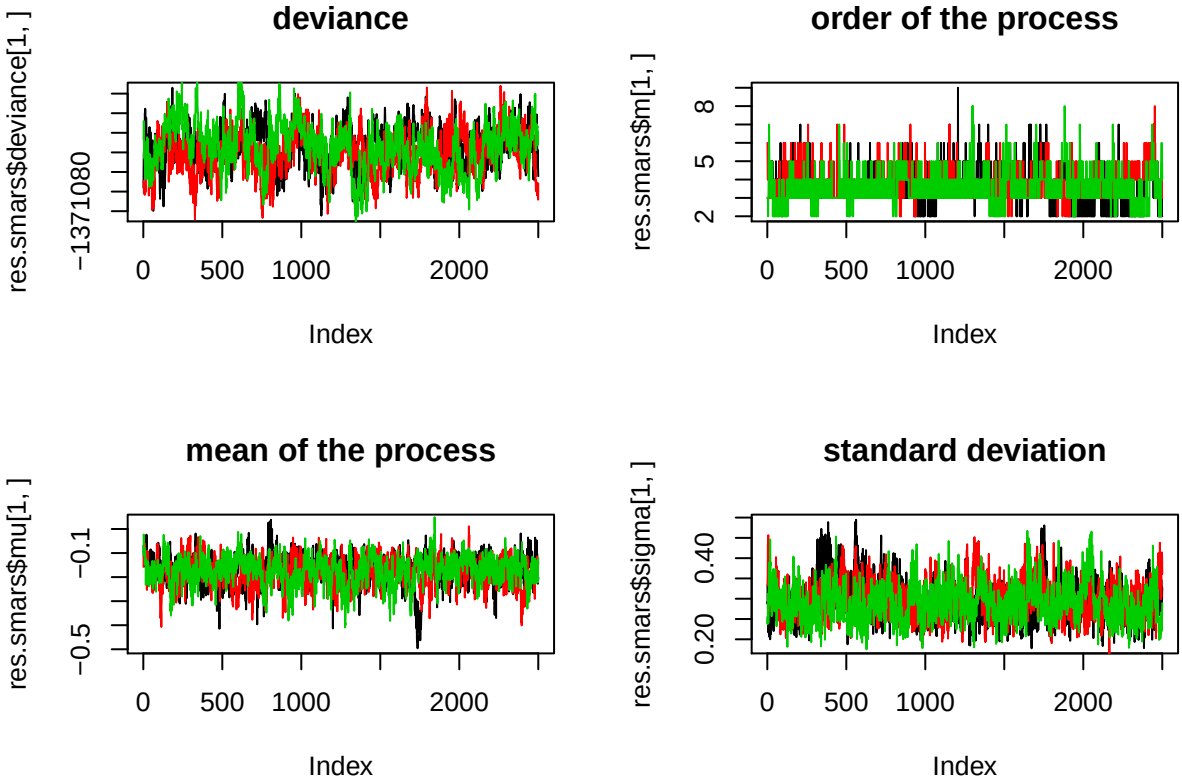
Moving average (SMARS) model

```
source("SMARS.r")
tpo.SMARS = system.time(res.smars <- MARS(neigh = VR.nb, O = Obs.muni,
    E = Exp.muni, n.burnin = 5000, n.chains = 3, n.iter = 25000, n.thin = 8,
    jump = 10, par.h = 0.04))
# Computing time
tpo.SMARS
```

```
##    user  system elapsed
##  415.40    2.39  419.44
```

```
# Some plots to assess convergence deviance
par(mfrow = c(2, 2))
plot(res.smars$deviance[1, ], type = "l", main = "deviance")
```

```
lines(res.smars$deviance[2, ], col = 2)
lines(res.smars$deviance[3, ], col = 3)
# order of the spatial moving average process
plot(res.smars$m[1, ], type = "l", main = "order of the process")
lines(res.smars$m[2, ], col = 2)
lines(res.smars$m[3, ], col = 3)
# mean of the underlying process
plot(res.smars$mu[1, ], type = "l", main = "mean of the process")
lines(res.smars$mu[2, ], col = 2)
lines(res.smars$mu[3, ], col = 3)
# standard deviation of the underlying process
plot(res.smars$sigma[1, ], type = "l", main = "standard deviation")
lines(res.smars$sigma[2, ], col = 2)
lines(res.smars$sigma[3, ], col = 3)
```




```
# posterior distribution for m, the order of the spatial moving average
# process
table(res.smars$m)/prod(dim(res.smars$m))
```

```
##
##           2           3           4           5           6
## 0.0924000000 0.4965333333 0.2881333333 0.0984000000 0.0198666667
##           7           8           9
## 0.0040000000 0.0005333333 0.0001333333
```

```
# DIC
DICPoisson(rbind(res.smars$RR[1, , ], res.smars$RR[2, , ], res.smars$RR[3,
```

```
    , ]) * 100, Obs.muni, Exp.muni)
```

```
## D= 1714.535 pD= 72.83322 DIC= 1787.368
```

Splines model

```
library(splines)
# Construction of the spatial B-spline basis Cubic B-splines
p = 3
# Number of internal intervals horizontal (x1)
qh = 7
# Number of internal intervals vertical (x2)
qv = 14

# Marginal basis for longitude
x1 = coordinates(VR.cart)[, 1]
x1 = (x1 - min(x1))/(max(x1) - min(x1))
dist1 = (max(x1) - min(x1))/qh
x1l = min(x1) - dist1 * 0.05
x1r = max(x1) + dist1 * 0.05
dx1 = (x1r - x1l)/qh
knots1 = seq(x1l - p * dx1, x1r + p * dx1, by = dx1)
# The Basis for the latitudes
B1 = splineDesign(knots1, x1, p + 1)
k1 = ncol(B1)

# Marginal basis for latitude
x2 = coordinates(VR.cart)[, 2]
x2 = (x2 - min(x2))/(max(x2) - min(x2))
dist2 = (max(x2) - min(x2))/qv
x2l = min(x2) - dist2 * 0.05
x2r = max(x2) + dist2 * 0.05
dx2 = (x2r - x2l)/qv
knots2 = seq(x2l - p * dx2, x2r + p * dx2, by = dx2)
# The basis for the longitudes
B2 = splineDesign(knots2, x2, p + 1)
k2 = ncol(B2)

# Row-wise Kronecker product of the former two bases
Rten = function(X1, X2) {
    one1 = matrix(1, 1, ncol(X1))
    one2 = matrix(1, 1, ncol(X2))
    kronecker(X1, one2) * kronecker(one1, X2)
}
# The bidimensional basis
Bs = Rten(B2, B1)
ks = ncol(Bs)


# Spatial structure matrices
#---------------------------
order = 1
```

```
D1 = diff(diag(k1), differences = order)
P1 = t(D1) %*% D1

D2 = diff(diag(k2), differences = order)
P2 = t(D2) %*% D2

R1 = kronecker(diag(k2), P1)
R2 = kronecker(P2, diag(k1))

Cmat.s = list(inla.as.sparse(R1), inla.as.sparse(R2))


# Define appropriate hyperprior distributions using the 'expression()'
# function - Unif(0,Inf) for standard deviations
#------------------------------------------------------------------------------
sdunif = "expression:
    logdens = -log_precision/2;
    return(logdens)"

carto.nb = VR.nb
adj = unlist(carto.nb)
# n = length(Obs.muni)

data = list(O = Obs.muni, E = Exp.muni, intercept = c(1, rep(NA, ks)),
    ID.area = c(NA, 1:ks))

formula = O ~ -1 + intercept + f(ID.area, model = "generic3", Cmatrix = Cmat.s,
    constr = TRUE, diagonal = 1e-06, hyper = list(prec1 = list(prior = sdunif),
        prec2 = list(prior = sdunif)))

resulSplines = inla(formula, family = "poisson", data = data, E = Exp.muni,
    control.predictor = list(compute = TRUE, A = cbind(rep(1, length(Obs.muni)),
        Bs), link = 1), control.compute = list(dic = TRUE), control.inla = list(strategy = "laplace"))
```

```
## Warning in inla.model.properties.generic(inla.trim.family(model), (mm[names(mm) == : Model 'generic3
##   Use this model with extra care!!! Further warnings are disabled.
```

```
summary(resulSplines)
```

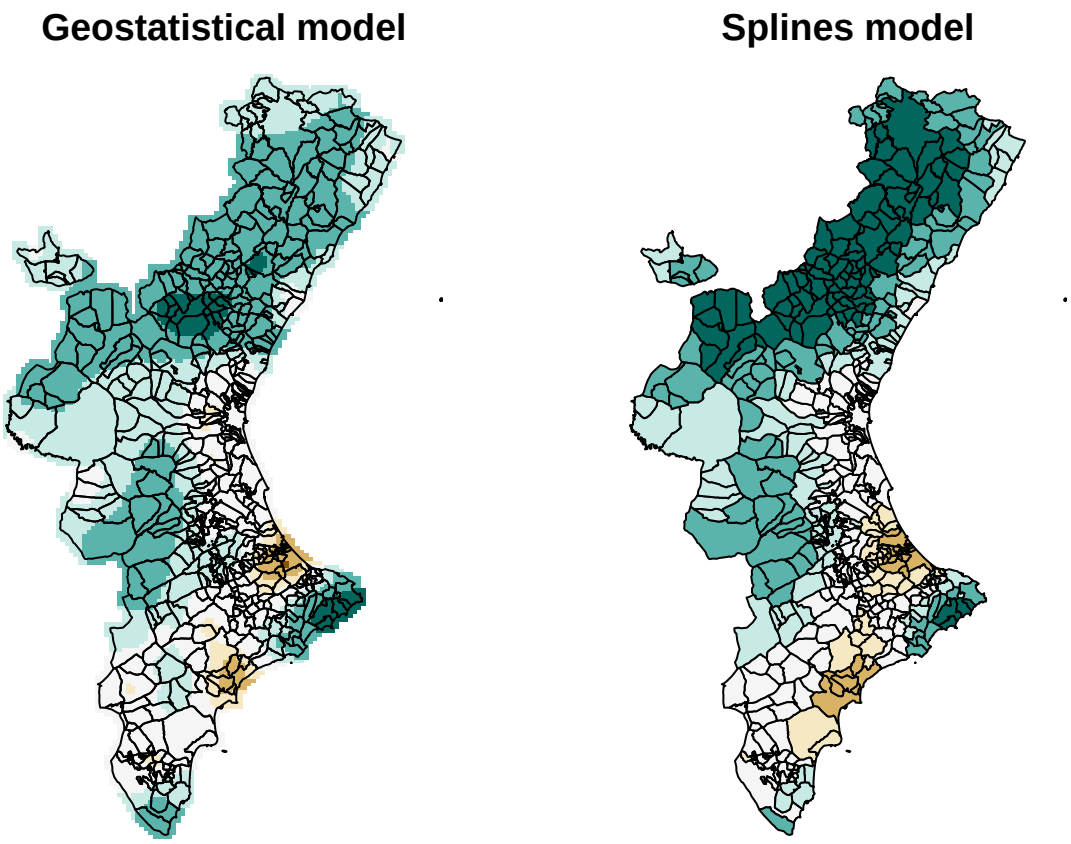
```
##
## Call:
## c("inla(formula = formula, family = \"poisson\", data = data, E = Exp.muni, ",  "    control.compute
##
## Time used:
##  Pre-processing    Running inla Post-processing           Total
##          1.6012         41.1322          0.1350         42.8685
##
## Fixed effects:
##               mean     sd 0.025quant 0.5quant 0.975quant    mode kld
## intercept -0.2215 0.1235    -0.4703  -0.2203     0.0206 -0.2179   0
##
## Random effects:
## Name   Model
##  ID.area   Generic3 model
```

```
##
## Model hyperparameters:
##                                          mean    sd 0.025quant 0.5quant
## Precision for Cmatrix[[1]] for ID.area 1.554 1.306     0.2721    1.191
## Precision for Cmatrix[[2]] for ID.area 2.238 1.978     0.3877    1.674
##                                        0.975quant   mode
## Precision for Cmatrix[[1]] for ID.area      5.006 0.6873
## Precision for Cmatrix[[2]] for ID.area      7.480 0.9564
##
## Expected number of effective parameters(std dev): 25.79(3.389)
## Number of equivalent replicates : 20.94
##
## Deviance Information Criterion (DIC) ...............: 1802.01
## Deviance Information Criterion (DIC, saturated) ....: 669.98
## Effective number of parameters .....................: 26.63
##
## Marginal log-Likelihood:  -921.76
## Posterior marginals for linear predictor and fitted values computed
```

Figure for Example 6.1

```
par(mfrow = c(1, 2))
par(mar = c(0, 0, 1, 0) + 0.1)
sSMRs.pred = 100 * resulGeo.pred$summary.fitted.values[540 + 1:7391, "mean"]
plot(VR.cart, main = "Geostatistical model")
points(grid, col = colors[as.numeric(cut(sSMRs.pred, 100 * cuts))], pch = 15)
plot(VR.cart, add = TRUE)

rmes.spl = (resulSplines$summary.fitted.values[1:length(Obs.muni), 1])
plot(VR.cart, col = colors[as.numeric(cut(rmes.spl, cuts))], main = "Splines model")
```



BYM model

```
data = data.frame(O = Obs.muni, E = Exp.muni, id.node = 1:540)
# See the annex material of Example 3.6 for details on how the VR.graph
# file is generated
form = O ~ f(id.node, model = "bym", hyper = list(prec.spatial = list(prior = sdunif),
    prec.unstruct = list(prior = sdunif)), graph = "../Data/VR.graph")
resul.BYM.inla = inla(form, family = "poisson", data = data, E = E, control.compute = list(dic = TRUE))
```

Correlations between the sSMRs of the different models

```
sSMRs.Geo = sSMRs
sSMRs.SMARS = apply(res.smars$RR, 3, mean)
sSMRs.splines = (resulSplines$summary.fitted.values[1:length(Obs.muni),
    1])
sSMRs.BYM = resul.BYM.inla$summary.fitted.values[, 1]
cor(cbind(sSMRs.Geo, sSMRs.SMARS, sSMRs.splines, sSMRs.BYM))
```

```
##                sSMRs.Geo sSMRs.SMARS sSMRs.splines sSMRs.BYM
## sSMRs.Geo      1.0000000   0.9298012     0.9108162 0.9331293
## sSMRs.SMARS    0.9298012   1.0000000     0.8050285 0.9258731
## sSMRs.splines 0.9108162   0.8050285     1.0000000 0.8768230
## sSMRs.BYM      0.9331293   0.9258731     0.8768230 1.0000000
```