Template for 2D data analysis with INLABru

J Matthiopoulos

2022-05-27

I. Model fitting

I.1. Load libraries

```
# Essential
library(inlabru)
library(INLA)
library(ggplot2)

# Loaded dependencies
#library(sp)
#library(Matrix)
#library(foreach)
#library(parallel)

# Optional
library(mgcv) # For independent model performance comparisons, used as an exact method
```

I.2. Load data

In the example below, it is assumed that the data reside in a package, such as 'inlabru'. The 'try' option explores the list of available datasets. The second line loads the particular one. Other ways of importing the data, assuming they are not in a package ('?data') with option 'lib.loc' for pathname.

```
try(data(package="inlabru"))
data(gorillas, package = "inlabru")
```

I.3. Ensure data formatting

The overall structure of the data can be explored by 'str()'. The point locations (here, 'nests') need to be a 'SpatialPointsDataFrame'. If the point data are not in this form then, they will need to be converted by providing an appropriate spatial projection ('?SpatialPointsDataFrame').

```
str(gorillas)
str(gorillas$nests)
myPoints<-gorillas$nests # assign to shorthand
```

If the data set comes with in-built mesh and boundary components, then proceed to I.5, otherwise specify the mesh in the next section.

```
str(gorillas$mesh)
str(gorillas$boundary)
myMesh<-gorillas$mesh # assign to shorthand
myBoundary<-gorillas$boundary # assign to shorthand
```

I.4. Build the mesh

—-> HERE ADD MESH-BUILDING PSEUDOCODE

Plot the points (the nests(. (The `ggplot2` function `coord_fixed()` sets the aspect ratio, which defaults to 1.)

```
ggplot() +
  gg(myMesh) +
  gg(myPoints) +
  gg(myBoundary) +
  coord_fixed() +
  ggtitle("Points")
```

I.5. Specify spatial correlation structure

The following is an example using a Matern correlation structure with a PC prior.

```
myCorrelation<-inla.spde2.pcmatern(myMesh, prior.range = c(5, 0.01), prior.sigma = c(0.1, 0.01))
```

I.6. Define model

The model formula requires the explicit name 'coordinates' to recognise the mesh information that it will receive later, but can use the user-defined 'mySmooth()' to specify the spatial error term.

```
myModel<-coordinates~mySmooth(coordinates, model=myCorrelation) + Intercept(1)
```

I.7 Fit the model

```
myFit<-lgcp(myModel, data=myPoints, samplers=myBoundary, domain=list(coordinates=myMesh))
```

II. Model results

II.1 Summary statistics

```
summary(myFit)
```

III.2 Plotting fixed effect parameters

```
plot(myFit, "Intercept")
```

III.3 Plotting spatial random effects

Plots of the individual parameters

```
spde.range <- spde.posterior(myFit, "mySmooth", what = "range")
spde.logvar <- spde.posterior(myFit, "mySmooth", what = "log.variance")
range.plot <- plot(spde.range)
var.plot <- plot(spde.logvar)
multiplot(range.plot, var.plot)
```

Plots of the correlation and covariance functions

```
corplot <- plot(spde.posterior(myFit, "mySmooth", what = "matern.correlation"))
covplot <- plot(spde.posterior(myFit, "mySmooth", what = "matern.covariance"))
multiplot(covplot, corplot)
```

III. Model selection and evaluation

IV. Model predictions

IV.1 Generate the prediction data frame

The 'pixels()' command generates a regular grid of points which can be used for the prediction. This is stored as a spatial data frame in the user-defined 'myPredFrame'.

```
myPredFrame<-pixels(myMesh, nx = 50, ny = 50, mask = FALSE)
```

To constrain the predictions to a particular region (e.g. the boundary of the mesh), set the mask option in the 'pixels()' command to 'mask=myBoundary'.

IV.2 Generate predictions

```
myPreds<-predict(myFit, myPredFrame,~ exp(mySmooth + Intercept))
```

Note that multiple functions and linear predictors can be predicted simultaneously, under different names.

```
myPreds<-predict(myFit, myPredFrame,
                 ~ data.frame(myLambda = exp(mySmooth + Intercept),
                              myLoglambda = mySmooth + Intercept)
                )
```

IV.3 Visualise the predictions

Plotting intensity and log-intensity surfaces

```
pl1 <- ggplot() +
  gg(myPreds$myLambda) +
  gg(myBoundary) +
  ggtitle("LGCP fit to Points", subtitle = "(Response Scale)") +
  coord_fixed()
pl2 <- ggplot() +
  gg(myPreds$myLoglambda) +
  gg(myBoundary) +
  ggtitle("LGCP fit to Points", subtitle = "(Linear Predictor Scale)") +
  coord_fixed()
multiplot(pl1, pl2, cols = 2)
```

Alternatively, plotting maps of median, lower 95% and upper 95% density surfaces as follows (assuming that the predicted intensity is in object `myLambda`).

```
ggplot() +
  gg(cbind(myPreds$myLambda, data.frame(property = "q0.500")), aes(fill = median)) +
  gg(cbind(myPreds$myLambda, data.frame(property = "q0.025")), aes(fill = q0.025)) +
  gg(cbind(myPreds$myLambda, data.frame(property = "q0.975")), aes(fill = q0.975)) +
  coord_equal() +
  facet_wrap(~property)
```

IV.4 Estimating abundance

Estimating abundance uses the `predict` function. As a first step we need an estimate for the integrated lambda (denoted 'Lambda' with an upper case L). The integration `weight` values (the quadrature points) are contained in the `ipoints` output.

```
Lambda <- predict(
  myFit,
  ipoints(myBoundary, myMesh),
  ~ sum(weight * exp(mySmooth + Intercept))
)
Lambda
```

Use the median and 95%iles of this to determine interval boundaries for estimating the posterior abundance distribution (prediction, not credible interval).

```
abundance <- predict(
  myFit, ipoints(myBoundary, myMesh),
  ~ data.frame(
    N = 500:800,
    dpois(500:800,
      lambda = sum(weight * exp(mySmooth + Intercept))
    )
  )
)
```