# Big Data in Environmental Research

# Pre-World Congress Meeting of New Researchers in Statistics and Probability

# Problem sheet

Website: `http://www.stat.ubc.ca/~gavin/STEPIBookNewStyle/course_toronto.html`

In the first part of the lab sheet we will look at some very simple examples that will help you become familiar with the R-INLA software. Initially we will look at two very simple non-spatial models that you may have fitted before with other software packages. We will then look at simple temporal and spatial GMRF models.

The aims of this session are to get a general understanding of how to implement simple models in R-INLA, how to specify (unstructured) random effects, and how to fit simple GMRF models in time and space, i.e. structured effects.

We will need the INLA package:

```
> library(INLA)
```

1. For this example we use the data set *Seeds*. Check the INLA help file for information on the data set. The data set looks at the number of germinated seeds on each of 21 plates.

   We are using a binomial model
   $$r_i \sim \text{Binomial}(p_i, n_i),$$

   where $p_i$ is the probability of germinating on plate $i$.

   We will first fit a very simple model with two fixed effects and an interaction between them

   $$\text{logit}(p_i) = \alpha_0 + \alpha_1 x_{1i} + \alpha_2 x_{2i} + \alpha_{12} x_{1i} x_{2i}$$

   and will then include a random effect to account for differences between plates

   $$\text{logit}(p_i) = \alpha_0 + \alpha_1 x_{1i} + \alpha_2 x_{2i} + \alpha_{12} x_{1i} x_{2i} + b_i$$

   We will see later that when we include spatial effects in a model the code we need for this is very similar to the code for the simple random effect here.

   (a) **Only fixed effects**

   Load the data

```
> data(Seeds)
```

The response variable is $r$ (we rename it as $Y$ just to be consistent with the notation we use later)

```
> Y <- Seeds$r
```

There are two fixed effects $x_1$ (seed type) and $x_2$ (root extract), $n$ gives the total number of seeds on the plate (germinated plus not germinated)

```
> Z1 <- Seeds$x1
> Z2 <- Seeds$x2
> n <- Seeds$n
```

Specifying the simple formula is done in the same way as specifying a model for `lm` or `glm` in R; we are considering a model with two main effects and an interaction here

```
> formula = Y ~ 1 + Z1 * Z2
```

The data are stored in the data frame data (again, renaming is not necessary here but we do this for consistency with later examples):

```
> data = data.frame(Y, Z1, Z2, n)
```

We can now call INLA to fit the model (this should be very quick)

```
> result=inla(formula,data=data,family="binomial", Ntrials=n, verbose = TRUE,
   control.compute=list(dic=TRUE))
```

With the option `verbose = TRUE` you get a lot of output while INLA is running. Set it to `FALSE` if you find this annoying.
To see a summary of the results

```
> summary(result)
```

You can also plot the results:

```
> plot(result)
```

(b) **Fixed and random effects**

In order to include a random effect for differences between plates we have to include a variable that indicates the plate

```
> plate=Seeds$plate
> data=data.frame(Y, Z1, Z2, n, plate)
```

The random effect is specified as a function of the plate index and as iid in the formula

```
> formula = Y ~ Z1*Z2+f(plate,model="iid")
```

We can now call INLA to fit the model with the same call as above

```
> result=inla(formula,data=data,family="binomial",Ntrials=n,
    verbose = TRUE, control.compute=list(dic=TRUE))
```

Did including the random effect improve the model?

2. A model with non-linear effects

   For this example we use the data set `trees` available in R. This data set comprises the *Volume*, *Girth* and *Height* of 31 felled cherry trees and there is an interest in predicting Volume from the other two variables. However, the relationship between the variables might not be linear and hence we fit smooth functions of the explanatory variables. This is done with a structured effect in 1 dimension.

```
> data(trees)
```

The response variable is *Volume* (log transformed since it is rather skew)

```
> Y <- log(trees$Volume)
```

There are two explanatory variables *Girth* and *Height*

```
> Z1 <- trees$Girth
> Z2 <- trees$Height
```

We also need to choose prior values for the smooth functions. We choose the following for now but will change these later

```
> prior.c=c(1,0.05)
> hyper=list(theta=list(param=prior.c))
```

We now need to specify in the formula that we are fitting a smooth function to the two explanatory variables modelled as a one-dimensional CAR2 process (random walk 2).

```
> formula = Y ~ 1 + f(Z1, model = "rw2", hyper=hyper)
   + f(Z2, model = "rw2", hyper=hyper)
```

The data are stored in the data frame data again

```
> data=data.frame(Y, Z1, Z2)
```

We can now call INLA to fit the model; note that we now specify a normal outcome (*family*) and hence do **not** specify *Ntrials*

```
> result=inla(formula,data=data,family="normal", verbose = TRUE,
   control.compute=list(dic=TRUE))
```

To see a summary of the results

```
> summary(result)
> plot(result)
```

Change the values of the prior to change the smoothness of the function and choose `rw1` as an alternative to `rw2`. How are these different?

```
> prior.c=c(20,0.05)
> hyper=list(theta=list(param=prior.c))
> result=inla(formula,data=data,family="normal", verbose = TRUE,
    control.compute=list(dic=TRUE))
> plot(result)
```

3. A simple temporal model

   The previous data example was a bit small- and you might agree that it was not really necessary to use the smooth effects for that particular example. We look at different example now which uses almost the same code. It is data of giving a series of measurements (over time) of head acceleration in a simulated motorcycle accident, used to test crash helmets.

   ```
   > library(MASS)
   > data(mcycle)
   ```

   The response variable is *accel*

   ```
   > Y <- mcycle$accel
   ```

   The explanatory variable is times

   ```
   > Z1 <- mcycle$times
   ```

   The acceleration clearly doesnt change linearly over time

   ```
   > plot(Z1, Y)
   ```

   We again choose prior values for the temporal effect.

   ```
   > prior.c=c(1,0.05)
   > hyper=list(theta=list(param=prior.c))
   ```

   We now need to specify in the formula that we are fitting a smooth function to the two explanatory variables modelled as a CAR1 process (random walk 1).

```
> formula = Y ~ -1 + f(Z1, model = "rw2", hyper=hyper)
```

The data are stored in the data frame data again

```
> data=data.frame(Y, Z1)
```

We can now call INLA to fit the model.

```
> result=inla(formula,data=data,family="normal", verbose = TRUE,
    control.compute=list(dic=TRUE))
> plot(result)
```

4. 
```
> require(lattice)
> require(INLA)
```

This spatial example uses data on the total Larynx cancer mortality in 544 districts of Germany between 1986 and 1990.

This dataset is typical of many simple spatial examples and has the following properties:

- Data is given as the total number of counts in fixed areas across the spatial domain of interest.
- We know which areas neighbour each other.
- There is a covariate that is also spatial.

We will use the standard Poisson likelihood to model the counts

$$y_i \sim Po\left(E_i e^{\eta_i}\right),$$

where $E_i$ is the "population at risk" in region $i$. Using a different $E_i$ in each region can help account for spatial heterogeneity in the underlying population.

In this example, we will fit a collection of models to this data and compare them.

**Load the data** We begin by loading the data, the graph showing how the regions of Germany are connected, and a function to help us plot.

```
> data(Germany)
> g = system.file("demodata/germany.graph", package="INLA")
> source(system.file("demodata/Bym-map.R", package="INLA"))
> summary(Germany)
```

**The zeroth model: no space** Here we will fit the model without spatial dependence

$$\eta_i = \text{intercept} + (\text{covariate})_i + (\text{random effect})_i$$

where the random effect is i.i.d. $N(0, \sigma^2)$.

The formula is then

```
> formula0 = Y ~ 1 + x + f(region, model="iid")
```

and INLA can be run by calling

```
> result0 = inla(formula0, family="poisson", data=Germany, E=E)
```

However, as we are looking towards comparing multiple models, we need to tell `R-INLA` to compute a few extra things. To do this, we need to use the `control.compute` argument in the `inla` call.

*Important note:* All of the `control.xxx` arguments take a *list* as their value. If you don't give them a list you'll get an odd error!

To run the model and compute some model-checking diagnostics (DIC and CPO scores)

```
> result0  =  inla(formula0,family="poisson",data=Germany,E=E,
                              control.compute= list(dic=TRUE, cpo = TRUE))
```

To compute the expected number of counts, we need to transform the marginals for $\eta_i$ into the marginals for $\exp(\eta_i)$. This can be done using `inla.tmarginal`, but a cheap and nasty approximation is to just take the linear predictor and exponentiate it

```
> expected = Germany$E*exp(result0$summary.linear.predictor$mean)
```

- Look at the PIT plot. Is it ok?
- Compare the observed counts to the expected counts. (do a `hist` of `Y/expected`).

**The first model** The first model we will fit is a BYM model.

The simple code for fitting the model is

```
> formula1 = Y ~ 1+ f(region,model="bym",graph=g)
> result1  =  inla(formula1,family="poisson",data=Germany,E=E)
```

However, as we are looking towards comparing multiple models, we need to tell `R-INLA` to compute a few extra things. To do this, we need to use the `control.compute` argument in the `inla` call.

*Important note:* All of the `control.xxx` arguments take a *list* as their value. If you don't give them a list you'll get an odd error!

To run the model and compute some model-checking diagnostics (DIC and CPO scores)

```
> result1  =  inla(formula1,family="poisson",data=Germany,E=E,
                                control.compute= list(dic=TRUE, cpo = TRUE))
```

Finally, `formula1` has two components: an intercept and a spatial component. If you need the marginal distributions for the sum, you need to ask INLA to compute them. This is controlled using `control.predictor`.

*Aside*: If we were predicting (`y[i] = NA`), then we need to do one more thing in order to get the predictions on the correct scale. In order for these marginal distributions to be on the correct scale, we need to tell them the link function, which in this case is the exponential. A list of link functions supported for each different likelihood can be found in the documentation (e.g. `inla.doc("poisson")`). *This is not relevant for this example!*

```
> result1  =  inla(formula1,family="poisson",data=Germany,E=E,
                                control.compute= list(dic=TRUE, cpo = TRUE),
                                control.predictor=list(compute=TRUE, link=1))
```

We can now plot the results. The log-risk can be plotted using `Bym.map`

```
> Bym.map(result1$summary.linear.predictor$mean)
```

For everything else, we can use INLA's built in plot command.

```
> plot(result1, plot.random.effects=FALSE)
```

- Look at the PIT plot. What does that tell you about the model fit?
- Compare observed and expected.

**The second model: A covariate** If you look again at the `Germany` data, you can see that we have another entry `x`, which is a spatially varying covariate that is a proxy for the number of smokers in the region. Let's add that to the model!

```
> formula2 = Y ~ 1 + x +  f(region,model="bym",graph=g)
```

- Fit this model (`result2 = inla(...)`), computing the DIC and the PIT.
- Look at the PIT plot. Is it better or worse?
- Look at the DIC (`result2$dic$dic`). Is it better or worse?
- Compare observed and expected.

**The third model: A non-linear covariate** When modelling risk, it is often the case that the covariate does not effect the risk in a linear manner. For example, a covariate may have almost no effect on the risk until its value is above a certain level. To take this into account, let's fit a nonlinear covariate.

In INLA, this is done by replacing `... + x +...` with a new random effect.

```
> formula3 = Y ~ 1 + f(x, model="rw2") +  f(region,model="bym",graph=g)
```

- Fit this model (`result3 = inla(...)`), computing the DIC and the PIT.
- Look at the PIT plot. Is it better or worse?
- Look at the DIC (`result3$dic$dic`). Is it better or
- Compare observed and expected worse?
- Look at the plot of the random effect for covariate. Is it non-linear??
- What happens if you replace the `rw2` model with an `rw1` model? (call this `formula4` and `result4`)

The next four questions consider the data on hospital admissions for COPD. On the website you will find the data on the shape file (*englandlocalauthority.sph*), data on details of the areas (*englandlocalauthority.dbf*) along with the observed and expected numbers of hospital admissions (*copdmortalityobserved.csv*, *copdmortalityexpected.csv*).

5. Fit a Poisson log-normal model with spatial effects to the data using the R package `CARBayes`. Note, that the spatial effects come from the ICAR model, which requires a set of observed and expected values together with an adjacency matrix. (The adjacency matrix can be created using the `spdep` and `shapefiles` packages.) After fitting the model find the risks of hospital admission for COPD.
6. In this question we will implement the Poisson log-normal model with spatial effects using R-INLA. Use the data and the adjacency matrix from the previous exercise. The adjacency matrix can be converted into a file in the INLA format using the code `nb2INLA`.
7. Analyse the data of hospital admissions in 2001 using the CARBayes package and R-INLA and produce a map of the smoothed risks.
   (a) Compare the results of the two approaches.
   (b) Investigate the sensitivity of the results to the choice of priors for the variance terms.
8. Choose two other years to reproduce your analysis from the previous question. Is there any evidence of any changes in the patterns of risk over time?

9. Recall the first question from the previous problem sheet (`http://www.stat.ubc.ca/~gavin/STEPIBookNewStyle/course/cimat/ProbSheet1/ProbSheet1.pdf`). There the areas were considered independently, and so the estimates of risk may be largely influenced by the underlying population of each area. Now estimate risks for each area using a suitable smoothing model (e.g. CAR) to produce estimates that preserve the spatial structure of the underlying geography. Compare the results of the smooth model with the independent model.

10.
```
> require(lattice)
> require(INLA)
```

The data shows the difference in rainfall in 1962 from the long term average at 7352 sites across the US. The data was used by Kaufman, Schervish and Nychka (2008) and is available at `http://www.image.ucar.edu/Data/Taper/`.

This is a standard setup, in which precipitation was measured at irregular locations across the United States and we wish to have an interpolated rainfall covariate. For simplicity, we look across the whole year and consider only the deviation from the long running average precipitation field. In doing this, we remove some of the less pleasant features of the dataset (nonstationarity, non-Gaussian responses) and are left with something useful. From preliminary analysis, the data appears to be Gaussian, have zero mean and be stationary. Therefore it satisfies all of the assumptions of classical statistics and so we expect our models to work. Needless to say, many new models and methods have been tested on this dataset!

The dataset used are a tree column *data.frame* simulated on the previous section and provided on `INLA` package. It can be called by (change the data directory as appropriate!)

```
> load("/Users/daniesi/Dropbox/BYM/St_Andrews_course/rainfall.RData")
> head(rainfall)
> loc = cbind(rainfall$lon, rainfall$lat)
```

this is a *data.frame* where the two first columns are the coordinates and the third is the anomaly at those locations. We have also stored the locations in the variable `loc` for future reference.

We consider the $n$ observations $y_i$ on locations the $s_i$, $i = 1, ..., n$, and we define the model

$$\begin{array}{c} y_i|\beta_0, x_i, \sigma_e^2 \sim N(\beta_0 + x_i, \sigma_e^2) \\ \mathbf{x} \sim N(0, Q^{-1}) \end{array} \quad (1)$$

We consider that $x$ is a realisation of an SPDE model with unknown range and precision parameters.

**The mesh** The first step is to build the mesh using `inla.mesh.create.helper`

```
> mesh=  inla.mesh.create.helper(points.domain=cbind(rainfall$lon,rainfall$lat),
                   max.edge=c(2,100))
> plot(mesh)
```

- Play around with max.edge. What happens if we set it larger? What happens if we set it (a little!) smaller?

**The spde model** With that mesh, we define the SPDE model using the function inla.spde2.matern()

```
> args(inla.spde2.matern)
```

```
> function (mesh, alpha = 2, param = NULL, constr = FALSE, extraconstr.int = NULL,
    extraconstr = NULL, fractional.method = c("parsimonious",
        "null"), B.tau = matrix(c(0, 1, 0), 1, 3), B.kappa = matrix(c(0,
        0, 1), 1, 3), prior.variance.nominal = 1, prior.range.nominal = NULL,
    prior.tau = NULL, prior.kappa = NULL, theta.prior.mean = NULL,
    theta.prior.prec = 0.1)
  NULL
```

The principal arguments are the mesh object and the $\alpha$ parameter, related to the smoothness parameter of the process.

A good default value is $\alpha = 2$ and we use this value here

```
> spde <- inla.spde2.matern(mesh, alpha=2)
```

**The A Matrix** In order to link the observation locations to the mesh, we need to construct an $A$ matrix, which we can do as follows.

```
> A = inla.spde.make.A(mesh=mesh, loc =loc)
```

**The formula** We're now ready to build up the model. Although this data is precipitation anomaly (and therefore has mean zero), we will fit a mean. Our model will, therefore, be an intercept + an SPDE model. This is implemented as

```
> formula = z ~ intercept + f(nodes, model = spde) - 1
```

We are using an explicit intercept rather than the default one *on purpose.* As these models get more complicated the default intercept can lead to bad things.

**The stack functions** So we have the mesh, the model, the A matrix and the formula. We still need to put all of these elements together. This can be done by hand (akin to Sigrunn's examples), but it get's a bit messy. There is a group of functions in R-INLA that will help us.

The stack functions are very useful function that allow us to fit SPDE models. They are very powerful and in the next few days we will show how these functions allow us to construct and solve quite complex models in R-INLA. For now, the primary use of the stack functions is to avoid indexing and logic errors when combining different model components.

The challenge here is that we have a linear predictor that looks like

$$\eta = \beta_0 \mathbf{1} + \mathbf{A}\mathbf{x},$$

where $\beta_0$ is the intercept, $\mathbf{1}$ is a vector of 1s the same size as the data, $\mathbf{A}$ is the A-matrix defined above, and $\mathbf{x}$ are the vertex values of the SPDE model. The important thing is that $\mathbf{A}$ should only apply to the SPDE model component and shouldn't touch the intercept (or, if we have them, any other covariates). In fact, applying $\mathbf{A}$ to the intercept of the covariates doesn't make sense: they are the wrong dimension.

The *inla.stack* function allow us to work with predictors that includes terms with different dimensions. The three main `inla.stack()` arguments are the `data` vectors list, a list of A-matrices (each one related to one block effect) and the effects.

The code looks like this

```
> stack <- inla.stack(data=list(z=rainfall$z), A=list(1,A),
                      effects=list(intercept=rep(1,length(rainfall$z)),
                                             nodes=1:spde$n.spde ))
```

There are some things to look at here. Every argument takes a list, which will be useful tomorrow when we have more complicated models.

- We only have one source of data, so the `data` list only has one element.
- The formula has two parts (the intercept and the spde model) so the A argument and the effects argument need to have two components.
- *These need to be in the same order as the formula!*
- If nothing interesting is happening (i.e. a term has no A-matrix), we can just put 1.
- The names in the data and effects lists *must match* the names in the formula!

**Fitting the model** Finally, we are ready to fit the model. In order to do this, we need two more "stack" functions, which we can use to extract the information from the stack.

The first is `inla.stack.data`, which constructs an appropriate data.frame (actually a list) from the information in the stack. This data.frame will look a lot like your original data, but everything will be in the appropriate places and there will be a lot of `NA`s added in order to make sure INLA is computing the correct thing.

The second function is `inla.stack.A`, which constructs an appropriate A-matrix from the information in the stack. This A-matrix can then be given to INLA as part of the `control.predictor` argument.

*Note*: If you should have and A-matrix and you don't specify this argument, you will get a really weird error message. (Something like "Error in eval(expr, envir, enclos) : object 'x' not found")

The code to run the model is then

```
> result <- inla(formula,
                data=inla.stack.data(stack),
                control.predictor=list(A=inla.stack.A(stack)))
```

An object from `inla()` function has a set of several results. These includes summaries, marginal posterior densities of each parameter on the model: the regression parameters, each element of the latent field and all the hyperparameters.

The summary of $\beta_0$ is obtained by

```
> result$summary.fixed
```

The summary of $1/\sigma_e^2$ is obtained by

```
> result$summary.hyperpar[1,]
```

A marginal distribution on `inla()` output is just two vector, where one represents the parameter values and another the density. Any posterior marginal can be transformed. If we want the posterior marginal for $\sigma_e$, the square root of the $\sigma_e^2$, we use

```
> post.se <- inla.tmarginal(function(x) sqrt(1/x),
                            result$marginals.hyperpar[[1]])
```

and now we are able to summarise this distribution

```
> inla.emarginal(function(x) x, post.se)
> inla.qmarginal(c(0.025, 0.5, 0.975), post.se)
> inla.hpdmarginal(0.95, post.se)
> inla.pmarginal(c(0.5, 0.7), post.se)
```

and, of course, we can visualise it.

The parameters of the latent field is parametrised as $\log(\kappa)$ and $\log(\tau)$, where $\tau$ is the local variance parameter. We have the posterior marginals for $\kappa$, $\sigma_x^2$ and for the nominal range (the distance that we have correlation equals 0.1). This can be done with the `inla.spde2.result` function

```
> result.field <- inla.spde2.result(result, 'nodes', spde, do.transf=TRUE)
```

and we get the posterior mean of each of these parameters by

```
> inla.emarginal(function(x) x, result.field$marginals.kappa[[1]])
> inla.emarginal(function(x) x, result.field$marginals.variance.nominal[[1]])
> inla.emarginal(function(x) x, result.field$marginals.range.nominal[[1]])
```

also we can get other summary statistics, HPD interval and visualise it.

**Plotting the field** Finally, we can plot the field using the following code

```
require(maps) #install.packages(c("maps"))
require(maptools)
## If you don't have these, just remove mm=map("usa") below


## Calculate mapping between triangulation vertices and grid points:

proj = inla.mesh.projector(mesh, dims=c(200,200))
## Construct greyscale palette function:
my.grey.palette = function (n,...) { return (grey.colors(n,0.05,0.95,...))}
## Use it:
my.palette = my.grey.palette
## Construct map data appropriate for easy plotting:

levelplotmap = function(..., mm) {
      panel.levelplot(...)
      panel.lines(mm$x, mm$y, col="black")
  }
#####################
## Plot results:


## Map resulting posterior mean field to a grid:
plotdata = inla.mesh.project(proj, result.field$summary.values$mean)
## Plot PM contours:
dev.new()
bbb = (levelplot(row.values=proj$x, column.values=proj$y, x=plotdata,
                   mm=map("usa"), panel=levelplotmap,
                   col.regions=my.palette,
                   xlim=range(proj$x), ylim=range(proj$y), aspect="iso",
                   contour=TRUE, cuts=11, labels=FALSE, pretty=TRUE,
                   xlab="Easting",ylab="Northing"))
print(bbb)
```