PRACTICAL 1

Aim of this practical:

1. Aim 1
2. Aim 2
3. Aim 3

0 Linear Model

In this practical we will:

- Simulate Gaussian data
- Learn how to fit a linear model with `inlabru`
- Generate predictions from the model

Start by loading useful libraries:

```
library(dplyr)
library(INLA)
library(ggplot2)
library(patchwork)
library(inlabru)
# load some libraries to generate nice plots
library(scico)
```

As our first example we consider a simple linear regression model with Gaussian observations

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2), \qquad i = 1, \dots, N$$

where σ^2 is the observation error, and the mean parameter μ_i is linked to the **linear predictor** (η_i) through an identity function:

$$\eta_i = \mu_i = \beta_0 + \beta_1 x_i$$

where x_i is a covariate and β_0, β_1 are parameters to be estimated. We assign β_0 and β_1 a vague Gaussian prior.

To finalize the Bayesian model we assign a Gamma(a, b) prior to the precision parameter $\tau = 1/\sigma^2$ and two independent Gaussian priors with mean 0 and precision τ_β to the regression parameters β_0 and β_1 (we will use the default prior settings in INLA for now).

Question

What is the dimension of the hyperparameter vector and latent Gaussian field?
Answer
The hyperparameter vector has dimension 1, $\boldsymbol{\theta} = (\tau)$ while the latent Gaussian field $\boldsymbol{u} = (\beta_0, \beta_1)$ has dimension 2, 0 mean, and sparse precision matrix:

$$\boldsymbol{Q} = \begin{bmatrix} \tau_{\beta_0} & 0 \\ 0 & \tau_{\beta_1} \end{bmatrix}$$

Note that, since β_0 and β_1 are fixed effects, the precision parameters τ_{β_0} and τ_{β_1} are fixed.

Note

We can write the linear predictor vector $\boldsymbol{\eta} = (\eta_1, \dots, \eta_N)$ as

$$\boldsymbol{\eta} = \boldsymbol{A}\boldsymbol{u} = \boldsymbol{A}_1\boldsymbol{u}_1 + \boldsymbol{A}_2\boldsymbol{u}_2 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \beta_0 + \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \beta_1$$

Our linear predictor consists then of two components: an intercept and a slope.

0.1.1 Simulate example data

First, we simulate data from the model

$$y_i \sim \mathcal{N}(\eta_i, 0.1^2),\ i = 1, \dots, 100$$

with

$$\eta_i = \beta_0 + \beta_1 x_i$$

where $\beta_0 = 2$, $\beta_1 = 0.5$ and the values of the covariate x are generated from an Uniform(0,1) distribution. The simulated response and covariate data are then saved in a `data.frame` object.

```
beta = c(2,0.5)
sd_error = 0.1

n = 100
x = rnorm(n)
y = beta[1] + beta[2] * x + rnorm(n, sd = sd_error)

df = data.frame(y = y, x = x)
```

0.1.2 Fitting a linear regression model with `inlabru`

Defining model components

The model has two parameters to be estimated β_1 and β_2. We need to define the two corresponding model components:

```
cmp =  ~ -1 + beta_0(1) + beta_1(x, model = "linear")
```

The `cmp` object is here used to define model components. We can give them any useful names we like, in this case, `beta_0` and `beta_1`.

Note

Note that we have excluded the default Intercept term in the model by typing `-1` in the model components. However, `inlabru` has automatic intercept that can be called by typing `Intercept()` , which is one of `inlabru` special names and it is used to define a global intercept, e.g.

```
cmp =  ~  Intercept(1) + beta_1(x, model = "linear")
```

Observation model construction

The next step is to construct the observation model by defining the model likelihood. The most important inputs here are the `formula`, the `family` and the `data`.

The `formula` defines how the components should be combined in order to define the model predictor.

```
formula = y ~ beta_0 + beta_1
```

Note

In this case we can also use the shortcut `formula = y ~ .`. This will tell `inlabru` that the model is linear and that it is not necessary to linearize the model and assess convergence.

The likelihood is defined using the `bru_obs()` function as follows:

```
lik =  bru_obs(formula = y ~.,
           family = "gaussian",
           data = df)
```

Fit the model

We fit the model using the `bru()` functions which takes as input the components and the observation model:

```
fit.lm = bru(cmp, lik)
```

Extract results

The `summary()` function will give access to some basic information about model fit and estimates

```
summary(fit.lm)
## inlabru version: 2.12.0
## INLA version: 24.06.27
## Components:
## beta_0: main = linear(1), group = exchangeable(1L), replicate = iid(1L), NULL
## beta_1: main = linear(x), group = exchangeable(1L), replicate = iid(1L), NULL
## Likelihoods:
##   Family: 'gaussian'
##     Tag: ''
##     Data class: 'data.frame'
##     Response class: 'numeric'
```

```
##     Predictor: y ~ .
##     Used components: effects[beta_0, beta_1], latent[]
## Time used:
##     Pre = 0.576, Running = 0.327, Post = 0.103, Total = 1.01
## Fixed effects:
##         mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## beta_0 2.003 0.010      1.985    2.003      2.022 2.003   0
## beta_1 0.508 0.011      0.487    0.508      0.528 0.508   0
##
## Model hyperparameters:
##                                           mean    sd 0.025quant 0.5quant
## Precision for the Gaussian observations 112.39 15.89      83.45   111.65
##                                         0.975quant   mode
## Precision for the Gaussian observations     145.65 110.15
##
## Deviance Information Criterion (DIC) ...............: -182.44
## Deviance Information Criterion (DIC, saturated) ....: 105.34
## Effective number of parameters .....................: 2.99
##
## Watanabe-Akaike information criterion (WAIC) ...: -182.69
## Effective number of parameters .................: 2.65
##
## Marginal log-Likelihood:  71.86
##  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)')
```

We can see that both the intercept and slope and the error precision are correctly estimated.

0.1.3 Generate model predictions

Now we can take the fitted `bru` object and use the `predict` function to produce predictions for μ given a new set of values for the model covariates or the original values used for the model fit

```
new_data = data.frame(x = c(df$x, runif(10)),
                      y = c(df$y, rep(NA,10)))
pred = predict(fit.lm, new_data, ~ beta_0 + beta_1,
               n.samples = 1000)
```

The `predict` function generate samples from the fitted model. In this case we set the number of samples to 1000.

0 Plot

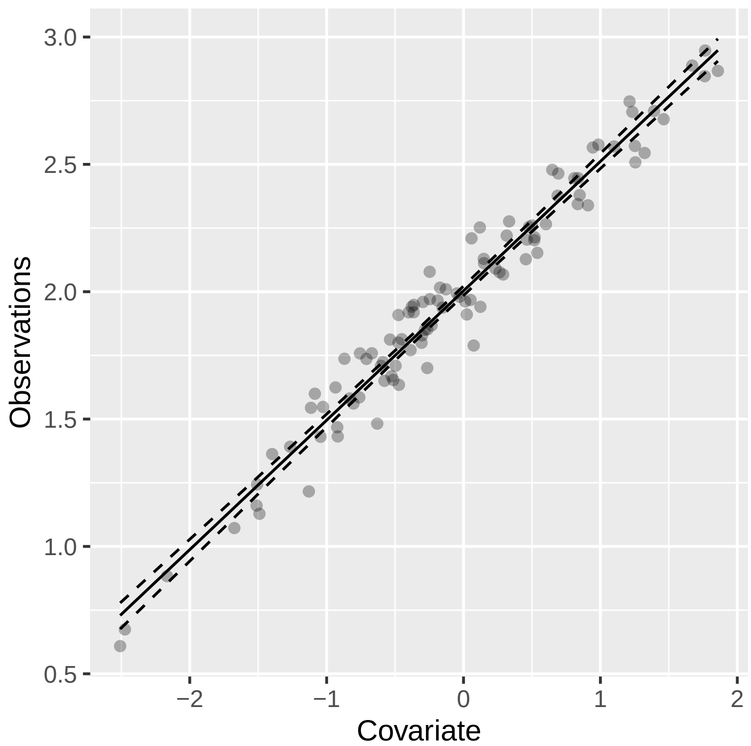


Figure 1: Data and 95% credible intervals

0 R Code

```
pred %>% ggplot() +
  geom_point(aes(x,y), alpha = 0.3) +
  geom_line(aes(x,mean)) +
  geom_line(aes(x, q0.025), linetype = "dashed")+
  geom_line(aes(x, q0.975), linetype = "dashed")+
  xlab("Covariate") + ylab("Observations")
```

Task

Generate predictions for a new observation with $x_0 = 0.45$
Take hint
You can create a new data frame containing the new observation x_0 and then use the `predict` function.
Click here to see the solution

```
new_data = data.frame(x = 0.45)
pred = predict(fit.lm, new_data, ~ beta_0 + beta_1,
               n.samples = 1000)
```

0 Generalized Linear Model

In this practical we will:

- Simulate non-Gaussian data
- Learn how to fit a generalised linear model with `inlabru`
- Generate predictions from the model

A generalised linear model allows for the data likelihood to be non-Gaussian. In this example we have a discrete response variable which we model using a Poisson distribution. Thus, we assume that our data

$$y_i \sim \text{Poisson}(\lambda_i)$$

with rate parameter λ_i which, using a log link, has associated predictor

$$\eta_i = \log \lambda_i = \beta_0 + \beta_1 x_i$$

with parameters β_0 and β_1, and covariate x. This is identical in form to the predictor in Section 0.1. The only difference is now we must specify a different data likelihood.

0.4.1 Simulate example data

This code generates 100 samples of covariate `x` and data `y`.

```
set.seed(123)
n = 100
beta = c(1,1)
x = rnorm(n)
lambda = exp(beta[1] + beta[2] * x)
y = rpois(n, lambda  = lambda)
df = data.frame(y = y, x = x)
```

0.4.2 Fitting a GLM in `inlabru`

Define model components and likelihood

Since the predictor is the same as Section 0.1, we can use the same component definition:

```
cmp =  ~ -1 + beta_0(1) + beta_1(x, model = "linear")
```

However, when building the observation model likelihood we must now specify the Poisson likelihood using the `family` argument (the default link function for this family is the log link).

```
lik =  bru_obs(formula = y ~.,
            family = "poisson",
            data = df)
```

Fit the model

Once the likelihood object is constructed, fitting the model is exactly the same process as in Section 0.1.

```
fit_glm = bru(cmp, lik)
```

And model summaries can be viewed using

```
summary(fit_glm)
```

```
inlabru version: 2.12.0
INLA version: 24.06.27
Components:
beta_0: main = linear(1), group = exchangeable(1L), replicate = iid(1L), NULL
beta_1: main = linear(x), group = exchangeable(1L), replicate = iid(1L), NULL
Likelihoods:
  Family: 'poisson'
    Tag: ''
    Data class: 'data.frame'
    Response class: 'integer'
    Predictor: y ~ .
    Used components: effects[beta_0, beta_1], latent[]
Time used:
    Pre = 0.403, Running = 0.276, Post = 0.0469, Total = 0.726
Fixed effects:
        mean    sd 0.025quant 0.5quant 0.975quant  mode kld
beta_0 0.915 0.071      0.775    0.915      1.054 0.915   0
beta_1 1.048 0.056      0.938    1.048      1.157 1.048   0

Deviance Information Criterion (DIC) ...............: 386.39
Deviance Information Criterion (DIC, saturated) ....: 120.67
Effective number of parameters .....................: 2.00

Watanabe-Akaike information criterion (WAIC) ...: 387.33
Effective number of parameters .................: 2.73

Marginal log-Likelihood:  -204.02
 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)')
```

0.4.3 Generate model predictions

To generate new predictions we must provide a data frame that contains the covariate values for x at which we want to predict.

This code block generates predictions for the data we used to fit the model (contained in `df$x`) as well as 10 new covariate values sampled from a uniform distribution `runif(10)`.

```
# Define new data, set to NA the values for prediction

new_data = data.frame(x = c(df$x, runif(10)),
                      y = c(df$y, rep(NA,10)))
```

```
# Define predictor formula
pred_fml <- ~ exp(beta_0 + beta_1)

# Generate predictions
pred_glm <- predict(fit_glm, new_data, pred_fml)
```

Since we used a log link (which is the default for `family = "poisson"`), we want to predict the exponential of the predictor. We specify this using a general `R` expression using the formula syntax.

Note

Note that the `predict` function will call the component names (i.e. the "labels") that were decided when defining the model.

Since the component definition is looking for a covariate named x, all we need to provide is a data frame that contains one, and the software does the rest.

0 Plot

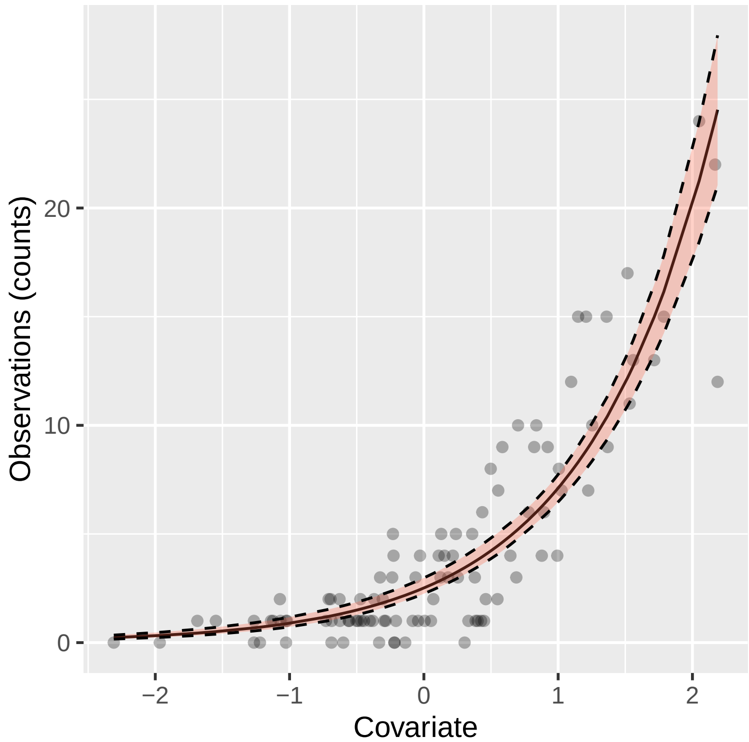


Figure 2: Data and 95% credible intervals

0 R Code

```
pred_glm %>% ggplot() +
  geom_point(aes(x,y), alpha = 0.3) +
  geom_line(aes(x,mean)) +
```

```
  geom_ribbon(aes(x = x, ymax = q0.975, ymin = q0.025),fill = "tomato", alpha = 0.3)+
xlab("Covariate") + ylab("Observations (counts)")
```

Task

Suppose a binary response such that

$$y_i \sim \text{Bernoulli}(\psi_i)$$
$$\eta_i = \text{logit}(\psi_i) = \alpha_0 + \alpha_1 \times w_i$$

Using the following simulated data, use `inlabru` to fit the logistic regression above. Then, plot the predictions for the data used to fit the model along with 10 new covariate values

```
set.seed(123)
n = 100
alpha = c(0.5,1.5)
w = rnorm(n)
psi = plogis(alpha[1] + alpha[2] * w)
y = rbinom(n = n, size = 1, prob =  psi) # set size = 1 to draw binary observations
df_logis = data.frame(y = y, w = w)
```

Here we use the logit link function $\text{logit}(x) = \log\left(\frac{x}{1-x}\right)$ (`plogis()` function in R) to link the linear predictor to the probabilities ψ.

Take hint

You can set `family = "binomial"` for binary responses and the `plogis()` function for computing the predicted values.

Note

The Bernoulli distribution is equivalent to a Binomial$(1,\psi)$ pmf. If you have proportional data (e.g. no. successes/no. trials) you can specify the number of events as your response and then the number of trials via the `Ntrials = n` argument of the `bru_obs` function (where `n` is the known vector of trials in your data set).

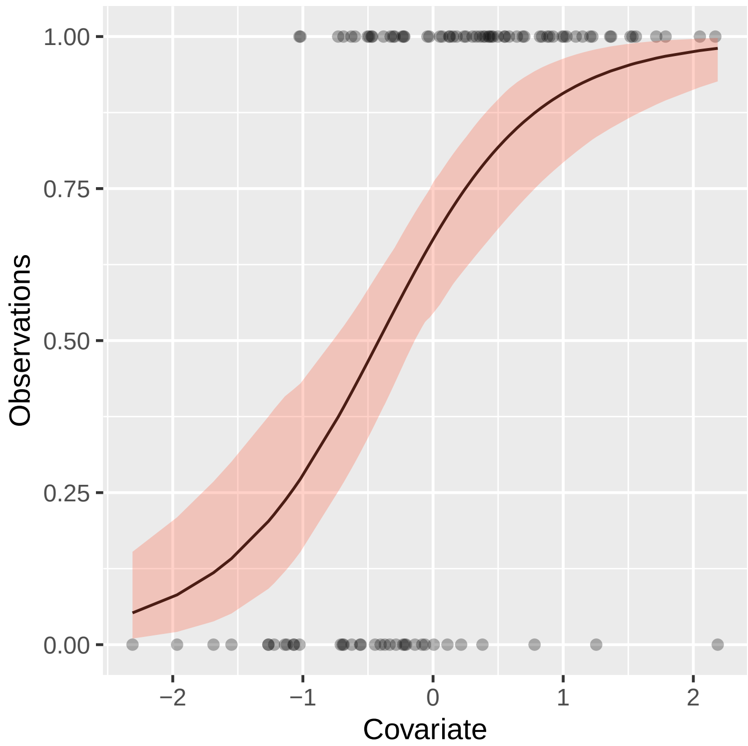
Click here to see the solution

```r
# Model components
cmp_logis =  ~ -1 + alpha_0(1) + alpha_1(w, model = "linear")
# Model likelihood
lik_logis =  bru_obs(formula = y ~.,
            family = "binomial",
            data = df_logis)
# fit the model
fit_logis <- bru(cmp_logis,lik_logis)

# Define data for prediction
new_data = data.frame(w = c(df_logis$w, runif(10)),
                      y = c(df_logis$y, rep(NA,10)))
# Define predictor formula
pred_fml <- ~ plogis(alpha_0 + alpha_1)

# Generate predictions
pred_logis <- predict(fit_logis, new_data, pred_fml)

# Plot predictions
pred_logis %>% ggplot() +
  geom_point(aes(w,y), alpha = 0.3) +
  geom_line(aes(w,mean)) +
    geom_ribbon(aes(x = w, ymax = q0.975, ymin = q0.025),fill = "tomato", alpha = 0.3)+
  xlab("Covariate") + ylab("Observations")
```



0 Spatial Data

In this practical we will:

- specific ILO1
- specific ILO2

Start by loading useful libraries:

```r
library(dplyr)
library(INLA)
library(ggplot2)
library(patchwork)
library(inlabru)
# load some libraries to generate nice plots
library(scico)
```

As our first example we consider a simple linear regression model with Gaussian observations

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2), \qquad i = 1, \dots, N$$

Question

What are the three spatial data type?
Answer
Geostatistical data, Areal data and Point processes .

Task

Task description
hint
This is a hint
Click here to see the solution

```r
2+2
```

```
[1] 4
```