PRACTICAL 1

Aim of this practical:

In this first practical we are going to look at some simple models

1. A Gaussian model with simulated data
2. A Linear mixed model
3. A GLM model with random effects

We are going to learn:

- How to fit some commonly used models with `inlabru`
- How to explore the results
- How to get predictions for missing data points

0 Linear Model

In this practical we will:

- Fit a simple linear regression with `inlabru`
- Fit a linear regression with discrete covariates and interactions

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

We fix the model parameters β_0, β_1 and the hyperparameter τ_y to a given value and simulate the data accordingly using the code below. The simulated response and covariate data are then saved in a `data.frame` object.

```
# set seed for reproducibility
set.seed(1234)

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`

Step1: Defining model components

The first step is to define the two model components: The intercept and the linear covariate effect.

Task

Define an object called `cmp` that includes and (i) intercept `beta_0` and (ii) a covariate `x` linear effect `beta_1`.
Take hint
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`. You can remove the automatic intercept construction by adding a `-1` in the components
Click here to see the solution

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

Note

`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")
```

Step 2: Build the observation model

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`.

Task

Define a linear predictor `eta` using the component labels you have defined on the previous task.
Take hint
The `eta` object defines how the components should be combined in order to define the model predictor.
Click here to see the solution

```
eta = y ~ beta_0 + beta_1
```

The likelihood for the observational model is defined using the `bru_obs()` function.

Task

Define the observational model likelihood in an object called `lik` using the `bru_obs()` function.
Take hint
The `bru_obs` is expecting three arguments:

- The linear predictor `eta` we defined in the previous task
- The data likelihood (this can be specified by setting `family = "gaussian"`)
- The data set `df`

Click here to see the solution

```
lik =  bru_obs(formula = eta,
               family = "gaussian",
               data = df)
```

Step 3: 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)
```

Step 4: Extract results

There are several ways to extract and examine the results of a fitted `inlabru` object.

The most natural place to start is to use the `summary()` which gives access to some basic information about model fit and estimates

```
summary(fit.lm)
## inlabru version: 2.13.0.9016
## INLA version: 25.08.21-1
## Components:
## Latent components:
## beta_0: main = linear(1)
## beta_1: main = linear(x)
## Observation models:
##   Family: 'gaussian'
##     Tag: <No tag>
##     Data class: 'data.frame'
##     Response class: 'numeric'
##     Predictor: y ~ beta_0 + beta_1
##     Additive/Linear: TRUE/TRUE
##     Used components: effects[beta_0, beta_1], latent[]
## Time used:
##     Pre = 0.494, Running = 0.262, Post = 0.218, Total = 0.974
## Fixed effects:
##         mean   sd 0.025quant 0.5quant 0.975quant  mode kld
## beta_0 2.004 0.01      1.983    2.004      2.024 2.004   0
## beta_1 0.497 0.01      0.477    0.497      0.518 0.497   0
##
## Model hyperparameters:
##                                          mean    sd 0.025quant 0.5quant
## Precision for the Gaussian observations 94.85 13.41      70.43    94.22
##                                         0.975quant  mode
## Precision for the Gaussian observations     122.92 92.96
##
## Marginal log-Likelihood:  63.27
##  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.

Another way, which gives access to more complicated (and useful) output is to use the `predict()` function. Below we 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

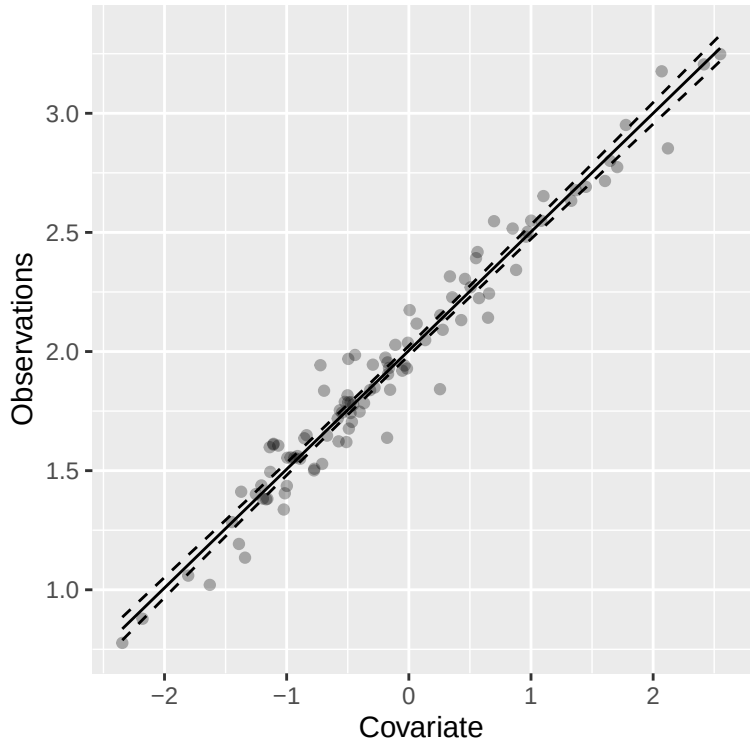


Figure 1: Data and 95% credible intervals

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 the linear predictor μ when the covariate has value $x_0 = 0.45$.
What is the predicted value for μ? And what is the uncertainty?
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)
pred
```

```
     x     mean         sd    q0.025     q0.5    q0.975   median sd.mc_std_err
1 0.45 2.227476 0.01205076 2.202807 2.227608 2.250729 2.227608  0.0002759884
  mean.mc_std_err
1    0.0003985336
```

You can see the predicted mean and sd by examining the produced `pred` object. In this case the mean is 2.23 with sd ca 0.01. This gives a 95% CI ca [2.2, 2.25].

Note

Before we have produced a credible interval for the expected mean μ if we want to produce a *prediction* interval for a new observation y we need to add the uncertainty that comes from the likelihood with precision τ_y. To do this we can again use the `predict()` function to compute a 95% prediction interval for y.

```
pred2 = predict(fit.lm, new_data,
                formula = ~ {
                  mu = beta_0 + beta_1
                  sigma = sqrt(1/Precision_for_the_Gaussian_observations)
                  list(q1 = qnorm(0.025, mean = mu, sd = sigma),
                       q2 =  qnorm(0.975, mean = mu, sd = sigma))},
                n.samples = 1000)
round(c(pred2$q1$mean, pred2$q2$mean),2)
```

```
[1] 2.03 2.43
```

Notice that now the interval we obtain is larger.

0.3.1 Setting Priors

In `R-INLA`, the default choice of priors for each β is

$$\beta \sim \mathcal{N}(0, 10^3).$$

and the prior for the variance parameter in terms of the log precision is

$$\log(\tau) \sim \text{logGamma}(1, 5 \times 10^{-5})$$

> **Note**
>
> If your model uses the default intercept construction (i.e., `Intercept(1)` in the linear predictor) `inlabru` will assign a default $\mathcal{N}(0,0)$ prior to it.

To check which priors are used in a fitted model one can use the function `inla.prior.used()`

```
inla.priors.used(fit.lm)
```

```
section=[family]
    tag=[INLA.Data1] component=[gaussian]
        theta1:
            parameter=[log precision]
            prior=[loggamma]
            param=[1e+00, 5e-05]
section=[linear]
    tag=[beta_0] component=[beta_0]
        beta:
            parameter=[beta_0]
            prior=[normal]
            param=[0.000, 0.001]
    tag=[beta_1] component=[beta_1]
        beta:
            parameter=[beta_1]
            prior=[normal]
            param=[0.000, 0.001]
```

From the output we see that the precision for the observation $\tau \sim$ Gamma$(1e+00, 5e-05)$ while β_0 and β_1 have precision 0.001, that is variance $1/0.001$.

Changing the precision for the linear effects

The precision for linear effects is set in the component definition. For example, if we want to increase the precision to 0.01 for β_0 we define the respective components as:

```
cmp1 =  ~-1 +  beta_0(1, prec.linear = 0.01) + beta_1(x, model = "linear")
```

> **Task**
>
> Run the model again using 0.1 as default precision for both the intercept and the slope parameter.
> Click here to see the solution
>
> ```
> cmp2 =  ~ -1 +
>           beta_0(1, prec.linear = 0.1) +
>           beta_1(x, model = "linear", prec.linear = 0.1)
>
> lm.fit2 = bru(cmp2, lik)
> ```
>
> Note that we can use the same observation model as before since both the formula and the dataset are unchanged.

Changing the prior for the precision of the observation error τ

Priors on the hyperparameters of the observation model must be passed by defining argument `hyper` within `control.family` in the call to the `bru_obs()` function.

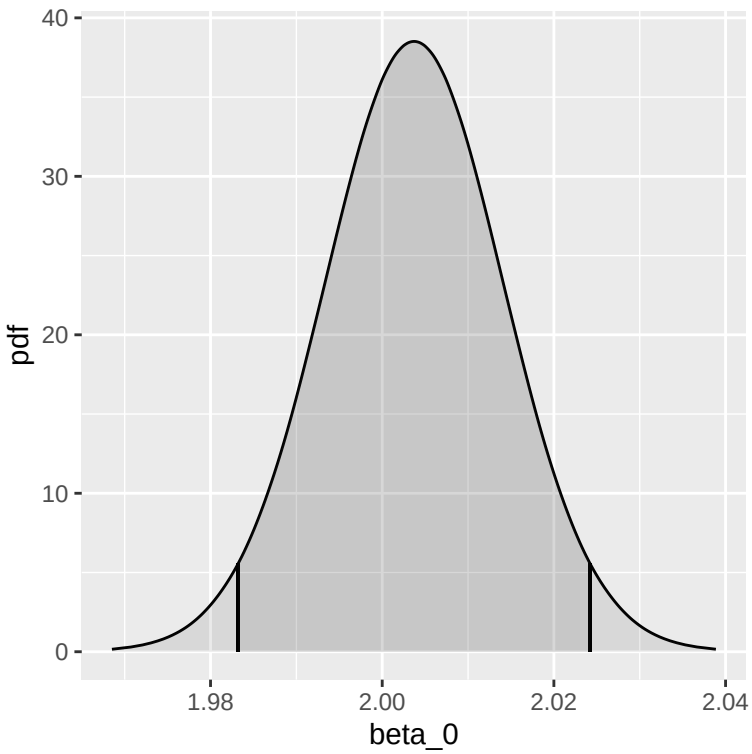
```
# First we define the logGamma (0.01,0.01) prior

prec.tau <- list(prec = list(prior = "loggamma",   # prior name
                             param = c(0.01, 0.01))) # prior values

lik2 =  bru_obs(formula = y ~.,
                family = "gaussian",
                data = df,
                control.family = list(hyper = prec.tau))

fit.lm2 = bru(cmp2, lik2)
```

The names of the priors available in `inlabru` can be seen with `names(inla.models()$prior)`

0.3.2 Visualizing the posterior marginals

Posterior marginal distributions of the fixed effects parameters and the hyperparameters can be visualized using the `plot()` function by calling the name of the component. For example, if want to visualize the posterior density of the intercept β_0 we can type:

```
plot(fit.lm, "beta_0")
```

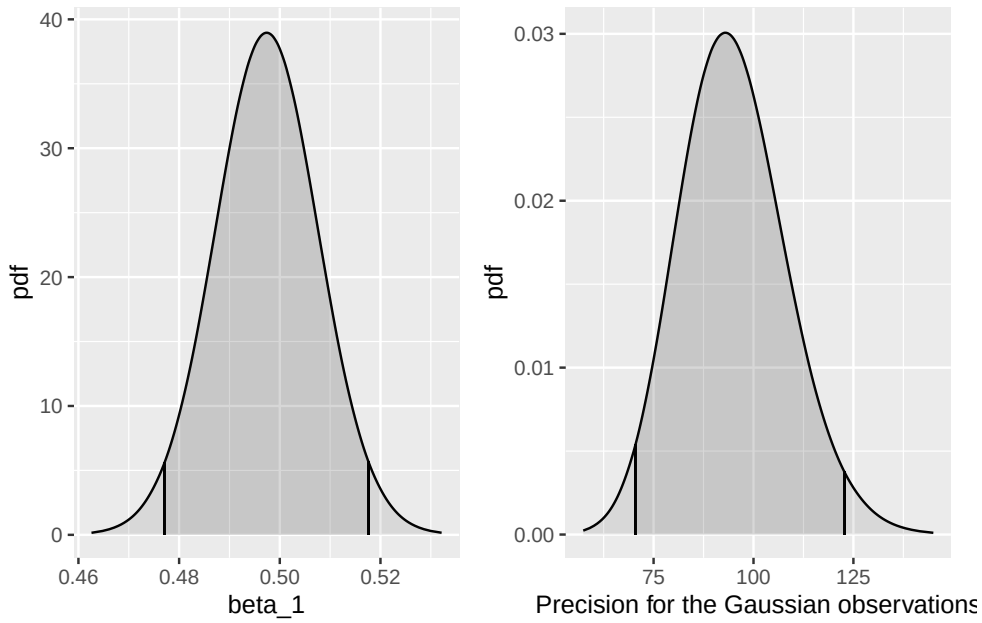


Task

Plot the posterior marginals for β_1 and for the precision of the observation error $\pi(\tau|y)$
Take hint

You can use the `bru_names(fit.lm)` function to check the names for the different model components.
Click here to see the solution

```
plot(fit.lm, "beta_1") +
plot(fit.lm, "Precision for the Gaussian observations")
```



Supplementary Material

The marginal densities for the hyper parameters can be also found by calling`inlabru_model$marginals.hyperpar`.

```
tau_e <- fit.lm$marginals.hyperpar$`Precision for the Gaussian observations`
```

We can then apply a transformation using the `inla.tmarginal` function to transform the precision posterior distributions to a SD scale.

```
sigma_e <- tau_e %>%
  inla.tmarginal(function(x) 1/x,.)
```

Then, we can compute posterior summaries using `inla.zmarginal` function as follows:

```
post_sigma_summaries <- inla.zmarginal(sigma_e,silent = T)
cbind(post_sigma_summaries)
```

```
           post_sigma_summaries
mean         0.01075447
sd           0.001538938
quant0.025 0.008142315
quant0.25    0.00965747
```

```
quant0.5   0.01060981
quant0.75  0.01169141
quant0.975 0.01417676
```

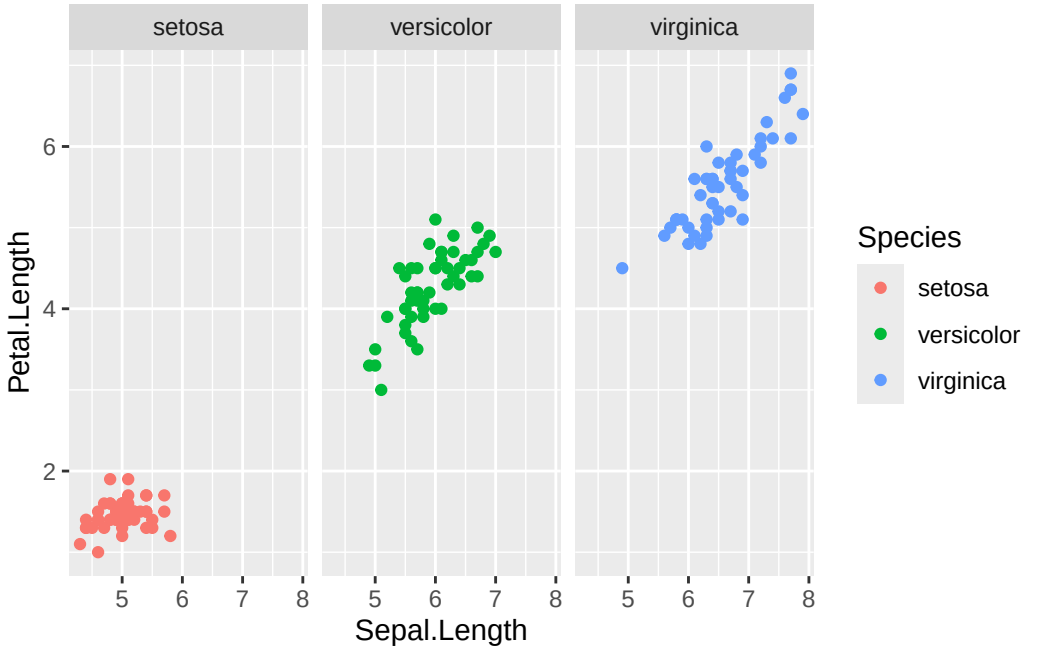
0 Linear model with discrete variables and interactions

We consider now the dataset `iris`. Here data are recorded about 150 different iris flowers belonging to 3 different species (50 for each species).

You can get more information about these data by typing `?iris`

We want to model the `Petal.length` as a function of `Sepal.length` and species.

```
data("iris")
iris %>% ggplot() +
  geom_point(aes(Sepal.Length, Petal.Length, color= Species)) +
  facet_wrap(.~Species)
```



Model 1 - Only `Species` effect Our first model assumes that the Sepal length only depends on the species, which is a categoriacal variable.

$$Y_i \sim \mathcal{N}(\mu_i, \sigma_y), \qquad i =$$
$$\mu_i = \eta_i = \beta_1 \, I(\text{obs } i\text{belongs to species } 1) + \beta_2 \, I(\text{obs } i\text{belongs to species } 2) + \beta_3 \, I(\text{obs } i\text{belongs to species } 3)$$

Using `lm()` we can fit the model as:

```
mod1 = lm(Petal.Length ~ Species, data  = iris)
summary(mod1)
```

```
Call:
lm(formula = Petal.Length ~ Species, data = iris)

Residuals:
   Min     1Q Median     3Q    Max
-1.260 -0.258  0.038  0.240  1.348

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        1.46200    0.06086   24.02   <2e-16 ***
Speciesversicolor  2.79800    0.08607   32.51   <2e-16 ***
Speciesvirginica   4.09000    0.08607   47.52   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4303 on 147 degrees of freedom
Multiple R-squared:  0.9414,    Adjusted R-squared:  0.9406
F-statistic:  1180 on 2 and 147 DF,  p-value: < 2.2e-16
```

Notice that `lm()` uses `setosa` as reference category, the parameter `Speciesversicolor` is then interpreted as the difference between the effect of the reference species and effect of `versicolor` species.

`inlabru` features a `model = 'fixed'` component type that allows users to specify linear fixed effects using a formula as input. The basic component input is a model matrix. Alternatively, one can supply a formula specification, which is then used to generate a model matrix automatically

```
cmp = ~   spp(
  main = ~ Species,
  model = "fixed"
)
```

Task

Use the component defined above to fit the linear model using `inlabru` and compare the output with that form the `lm` function.
Take hint
Realizing that fixed effects are treated as random effects with fixed precision, the fitted values can then be inspected via `fit1a$summary.random`
Click here to see the solution

```
lik = bru_obs(formula =Petal.Length ~ spp,
              data = iris)
fit1a = bru(cmp, lik)

fit1a$summary.random$spp
```

```
                ID     mean         sd 0.025quant 0.5quant 0.975quant     mode
1      (Intercept) 1.462020 0.06077711   1.342676 1.462020   1.581364 1.462020
2 Speciesversicolor 2.797970 0.08595181   2.629191 2.797970   2.966747 2.797970
3  Speciesvirginica 4.089965 0.08595181   3.921186 4.089965   4.258742 4.089965
          kld
1 3.690491e-09
```

```
2 3.690461e-09
3 3.690491e-09
```

Model 2 - Interaction between `Species` and `Sepal.Length`

Our second model is defined as

$$Y_i \sim \mathcal{N}(\mu_i, \sigma_y),$$
$$\mu_i = \eta_i = \beta_0 + \beta_1 x_i \, I(\text{obs } i\text{belongs to species } 1) + \beta_2 x_i \, I(\text{obs } i\text{belongs to species } 2) + \beta_3 x_i \, I(\text{obs } i\text{belongs to spec}$$

that is, we have a common intercept β_0 while the linear effect of the Sepal length depends on the Species. Using `lm()` we have: ::: {.cell}

```
mod2 = lm(Petal.Length ~ Species:Sepal.Length, data = iris)
mod2
```

```
Call:
lm(formula = Petal.Length ~ Species:Sepal.Length, data = iris)

Coefficients:
                   (Intercept)      Speciessetosa:Sepal.Length
                        0.5070                          0.1905
Speciesversicolor:Sepal.Length   Speciesvirginica:Sepal.Length
                        0.6326                          0.7656
```

:::

Task

Fit the same model in `inlabru`.
Click here to see the solution

```
cmp = ~   spp_sepal(
  main = ~ Species:Sepal.Length,
  model = "fixed"
)

lik = bru_obs(formula =Petal.Length ~ spp_sepal,
              data = iris)

fit1b = bru(cmp, lik)

fit1b$summary.random$spp_sepal
```

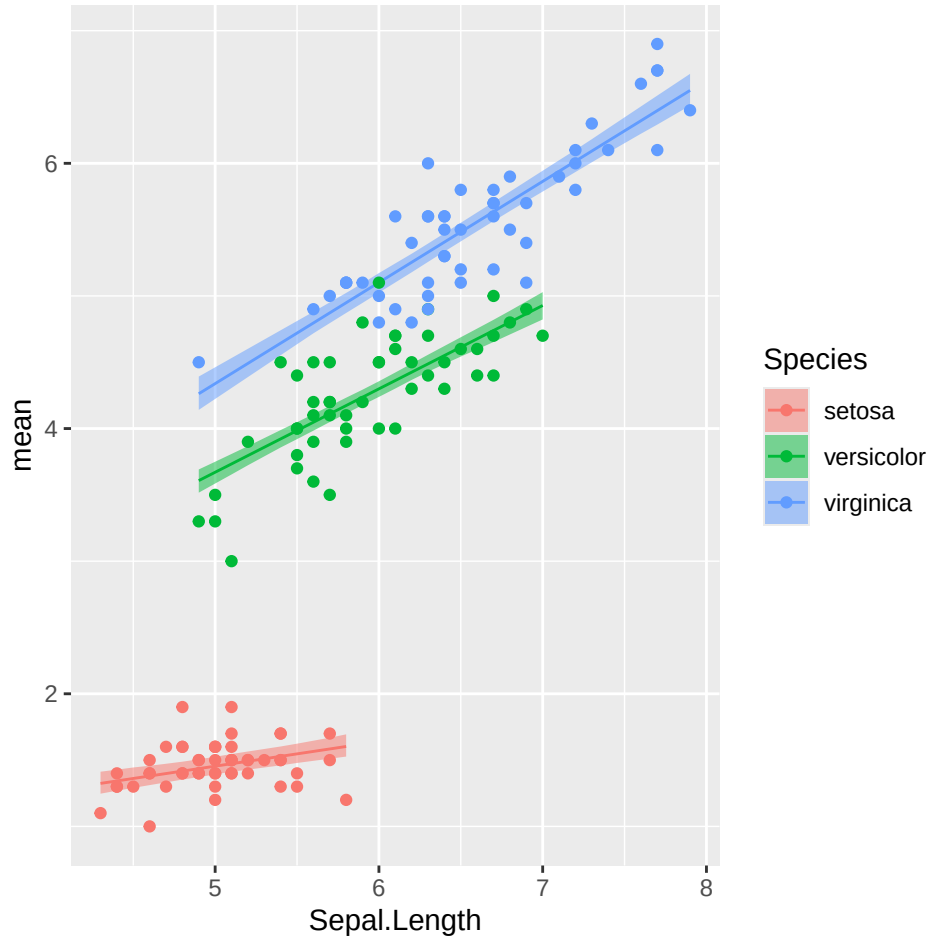
```
                            ID      mean         sd 0.025quant  0.5quant
1                   (Intercept) 0.5069832 0.25212309 0.01190078 0.5069833
2     Speciessetosa:Sepal.Length 0.1904884 0.05065269 0.09102422 0.1904884
3 Speciesversicolor:Sepal.Length 0.6326456 0.04260982 0.54897474 0.6326456
4  Speciesvirginica:Sepal.Length 0.7656467 0.03832808 0.69038371 0.7656467
  0.975quant      mode          kld
1  1.0020651 0.5069833 3.742099e-09
2  0.2899528 0.1904884 3.742120e-09
3  0.7163165 0.6326456 3.742124e-09
```

```
4   0.8409098 0.7656467 3.742119e-09
```

Task

Plot the estimated regression lines for the three species using model `fit1b`
Click here to see the solution

```
preds = predict(fit1b, iris, ~ spp_sepal)

pp = preds %>% ggplot() + geom_line(aes(Sepal.Length, mean, group = Species, color = Speci
  geom_ribbon(aes(Sepal.Length, ymin = q0.025, ymax = q0.975,
                  group = Species, fill = Species), alpha = 0.5) +
  geom_point(aes(Sepal.Length, Petal.Length,color = Species))

pp
```



0 Linear Mixed Model

In this practical we will:

- Understand the basic structure of a Linear Mixed Model (LLM)
- Simulate data from a LMM
- Learn how to fit a LMM with `inlabru` and predict from the model.

Consider the a simple linear regression model except with the addition that the data that comes in groups. Suppose that we want to include a random effect for each group j (equivalent to adding a group random intercept). The model is then:

$$y_{ij} = \beta_0 + \beta_1 x_i + u_j + \epsilon_{ij} \quad \text{for } i = 1, \dots, N \text{ and } j = 1, \dots, m.$$

Here the random group effect is given by the variable $u_j \sim \mathcal{N}(0, \tau_u^{-1})$ with $\tau_u = 1/\sigma_u^2$ describing the variability between groups (i.e., how much the group means differ from the overall mean). Then, $\epsilon_j \sim \mathcal{N}(0, \tau_\epsilon^{-1})$ denotes the residuals of the model and $\tau_\epsilon = 1/\sigma_\epsilon^2$ captures how much individual observations deviate from their group mean (i.e., variability within group).

The model design matrix for the random effect has one row for each observation (this is equivalent to a random intercept model). The row of the design matrix associated with the ij-th observation consists of zeros except for the element associated with u_j, which has a one.

$$\boldsymbol{\eta} = \boldsymbol{A}\boldsymbol{u} = \boldsymbol{A}_1\boldsymbol{u}_1 + \boldsymbol{A}_2\boldsymbol{u}_2 + \boldsymbol{A}_3\boldsymbol{u}_3$$

Supplementary material: LMM as a LGM

In matrix form, the linear mixed model for the *j*-th group can be written as:

$$\overset{N\times 1}{\hat{\mathbf{y}}_j} = \overset{N\times 2}{\widehat{X_j}} \underbrace{\beta}_{1\times 1} + \overset{n_j\times 1}{\widehat{Z_j}} \underbrace{u_j}_{1\times 1} + \overset{n_j\times 1}{\hat{\epsilon}_j},$$

In a latent Gaussian model (LGM) formulation the mixed model predictor for the *i*-th observation can be written as :

$$\eta_i = \beta_0 + \beta_1 x_i + \sum_k^K f_k(u_j)$$

where $f_k(u_j) = u_j$ since there's only one random effect per group (i.e., a random intercept for group j). The fixed effects (β_0, β_1) are assigned Gaussian priors (e.g., $\beta \sim \mathcal{N}(0, \tau_\beta^{-1})$). The random effects $\mathbf{u} = (u_1, \dots, u_m)^T$ follow a Gaussian density $\mathcal{N}(0, \mathbf{Q}_u^{-1})$ where $\mathbf{Q}_u = \tau_u \mathbf{I}_m$ is the precision matrix for the random intercepts. Then, the components for the LGM are the following:

- Latent field given by

$$\begin{bmatrix} \beta \\ \mathbf{u} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \tau_\beta^{-1}\mathbf{I}_2 & \mathbf{0} \\ \mathbf{0} & \tau_u^{-1}\mathbf{I}_m \end{bmatrix}\right)$$

- Likelihood:

$$y_i \sim \mathcal{N}(\eta_i, \tau_\epsilon^{-1})$$

- Hyperparameters:
  - $\tau_u \sim \text{Gamma}(a, b)$
  - $\tau_\epsilon \sim \text{Gamma}(c, d)$

0.5.1 Simulate example data

```
set.seed(12)
beta = c(1.5,1)
sd_error = 1
tau_group = 1

n = 100
n.groups = 5
x = rnorm(n)
v = rnorm(n.groups, sd = tau_group^{-1/2})
y = beta[1] + beta[2] * x + rnorm(n, sd = sd_error) +
  rep(v, each = 20)

df = data.frame(y = y, x = x, j = rep(1:5, each = 20))
```

Note that `inlabru` expects an integer indexing variable to label the groups.

```
ggplot(df) +
  geom_point(aes(x = x, colour = factor(j), y = y)) +
  theme_classic() +
  scale_colour_discrete("Group")
```

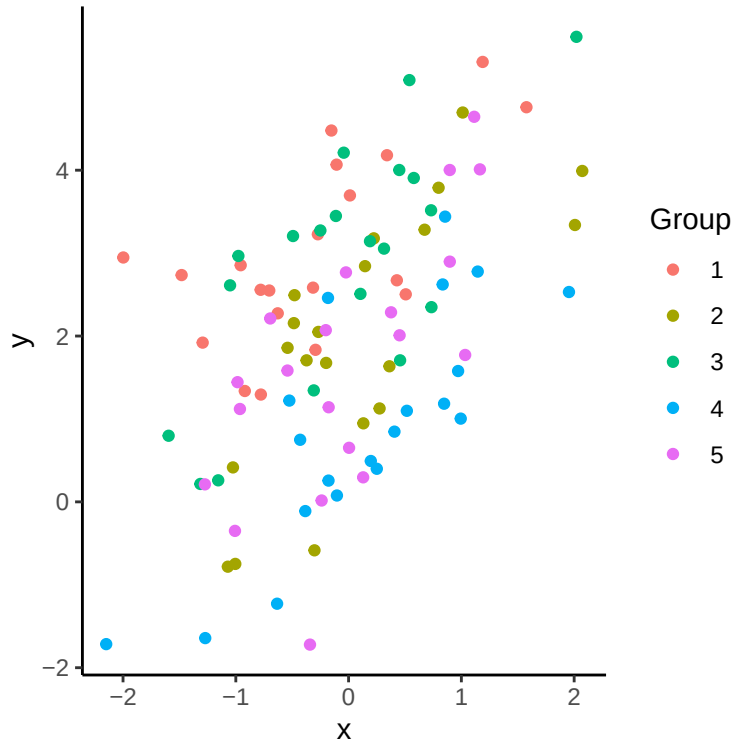


Figure 2: Data for the linear mixed model example with 5 groups

0.5.2 Fitting a LMM in `inlabru`

Defining model components and observational model

In order to specify this model we must use the `group` argument to tell `inlabru` which variable indexes the groups. The `model = "iid"` tells INLA that the groups are independent from one another.

```
# Define model components
cmp =  ~ -1 + beta_0(1) + beta_1(x, model = "linear") +
  u(j, model = "iid")
```

The group variable is indexed by column `j` in the dataset. We have chosen to name this component `v()` to connect with the mathematical notation that we used above.

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

Fitting the model

The model can be fitted exactly as in the previous examples by using the `bru` function with the components and likelihood objects.

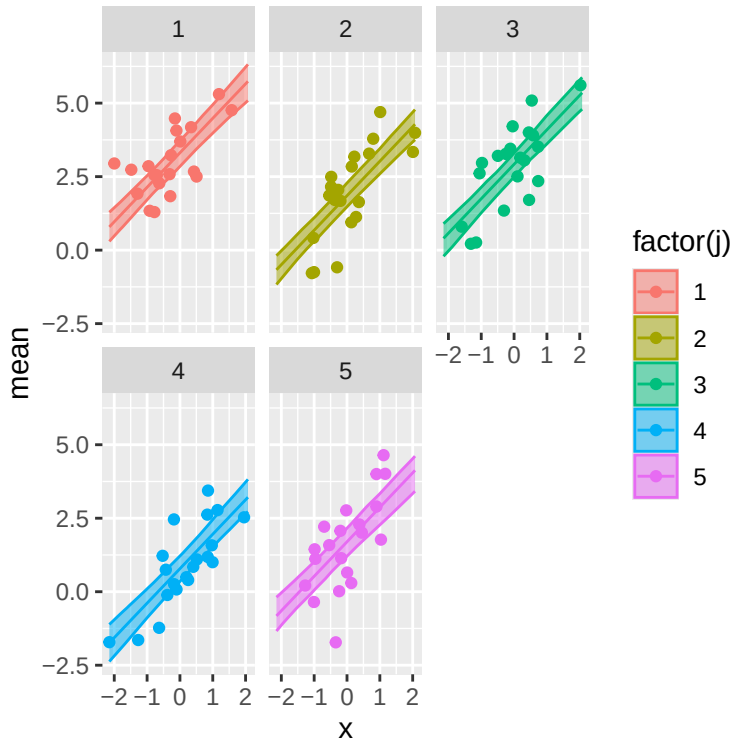
```
fit = bru(cmp, lik)
summary(fit)
## inlabru version: 2.13.0.9016
## INLA version: 25.08.21-1
## Components:
## Latent components:
## beta_0: main = linear(1)
## beta_1: main = linear(x)
## u: main = iid(j)
## Observation models:
##   Family: 'gaussian'
##     Tag: <No tag>
##     Data class: 'data.frame'
##     Response class: 'numeric'
##     Predictor: y ~ .
##     Additive/Linear: TRUE/TRUE
##     Used components: effects[beta_0, beta_1, u], latent[]
## Time used:
##     Pre = 0.325, Running = 0.303, Post = 0.16, Total = 0.787
## Fixed effects:
##          mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## beta_0 2.108 0.438      1.229    2.108      2.986 2.108   0
## beta_1 1.172 0.120      0.936    1.172      1.407 1.172   0
##
## Random effects:
##   Name     Model
##     u IID model
##
## Model hyperparameters:
##                                             mean    sd 0.025quant 0.5quant
## Precision for the Gaussian observations 0.995 0.144      0.738    0.986
```

```
## Precision for u                              1.613 1.060      0.369    1.356
##                                             0.975quant  mode
## Precision for the Gaussian observations       1.30 0.971
## Precision for u                               4.35 0.918
##
## Marginal log-Likelihood:  -179.93
##  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)')
```

0.5.3 Model predictions

To compute model predictions we can create a `data.frame` containing a range of values of covariate where we want the response to be predicted for each group. Then we simply call the predict function while spe

```
# New data
xpred = seq(range(x)[1], range(x)[2], length.out = 100)
j = 1:n.groups
pred_data = expand.grid(x = xpred, j = j)
pred = predict(fit, pred_data, formula = ~ beta_0 + beta_1 + u)


pred %>%
  ggplot(aes(x=x,y=mean,color=factor(j)))+
  geom_line()+
  geom_ribbon(aes(x,ymin = q0.025, ymax= q0.975,fill=factor(j)), alpha = 0.5) +
  geom_point(data=df,aes(x=x,y=y,colour=factor(j)))+
  facet_wrap(~j)
```



Question

Suppose that we are also interested in including random slopes into our model. Assuming intercept and slopes are independent, can your write down the linear predictor and the components of this model as a LGM?
Give me a hint
In general, the mixed model predictor can decomposed as:

$$\boldsymbol{\eta} = X\beta + Z\mathbf{u}$$

Where X is a $n \times p$ design matrix and β the corresponding p-dimensional vector of fixed effects. Then Z is a $n \times q_J$ design matrix for the q_J random effects and J groups; **v** is then a $q_J \times 1$ vector of q random effects for the J groups. In a latent Gaussian model (LGM) formulation this can be written as:

$$\eta_i = \beta_0 + \sum \beta_j x_{ij} + \sum_k f(k)(u_{ij})$$

See Solution

- The linear predictor is given by

$$\eta_i = \beta_0 + \beta_1 x_i + u_{0j} + u_{1j} x_i$$

- Latent field defined by:
  - $\beta \sim \mathcal{N}(0, \tau_\beta^{-1})$
  - $\mathbf{u}_j = \begin{bmatrix} u_{0j} \\ u_{1j} \end{bmatrix}, \mathbf{u}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_u^{-1})$ where the precision matrix is a block-diagonal matrix with entries $\mathbf{Q}_u = \begin{bmatrix} \tau_{u_0} & 0 \\ 0 & \tau_{u_1} \end{bmatrix}$
- The hyperparameters are then:
  - τ_{u_0}, τ_{u_1} and τ_ϵ

To fit this model in `inlabru` we can simply modify the model components as follows:

```
cmp =  ~ -1 + beta_0(1) + beta_1(x, model = "linear") +
  u0(j, model = "iid") + u1(j,x, model = "iid")
```

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.

0.6.1 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)
```

0.6.2 Fitting a GLM in `inlabru`

Define model components

The predictor here only contains only 2 components (Intercept and Slope).

> Task
>
> Define an object called `cmp` that includes and (i) intercept `beta_0` and (ii) a covariate `x` linear effect `beta_1`.
> Click here to see the solution
>
> ```
> cmp =  ~ -1 + beta_0(1) + beta_1(x, model = "linear")
> ```

Define linear predictor

> Task
>
> Define a linear predictor `eta` using the component labels you have defined on the previous task.
> Click here to see the solution
>
> ```
> eta = y ~ beta_0 + beta_1
> ```

Build observational model

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 = eta,
               family = "poisson",
               data = df)
```

Fit the model

Once the likelihood object is constructed, fitting the model is exactly the same process, we just need to specify the model components and the observational model, and pass this on to the `bru` function:

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

And model summaries can be viewed using

```
summary(fit_glm)
```

```
inlabru version: 2.13.0.9016
INLA version: 25.08.21-1
Components:
Latent components:
beta_0: main = linear(1)
beta_1: main = linear(x)
Observation models:
  Family: 'poisson'
    Tag: <No tag>
    Data class: 'data.frame'
    Response class: 'integer'
    Predictor: y ~ beta_0 + beta_1
    Additive/Linear: TRUE/TRUE
    Used components: effects[beta_0, beta_1], latent[]
Time used:
    Pre = 0.275, Running = 0.232, Post = 0.0428, Total = 0.55
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

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)')
```

0.6.3 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

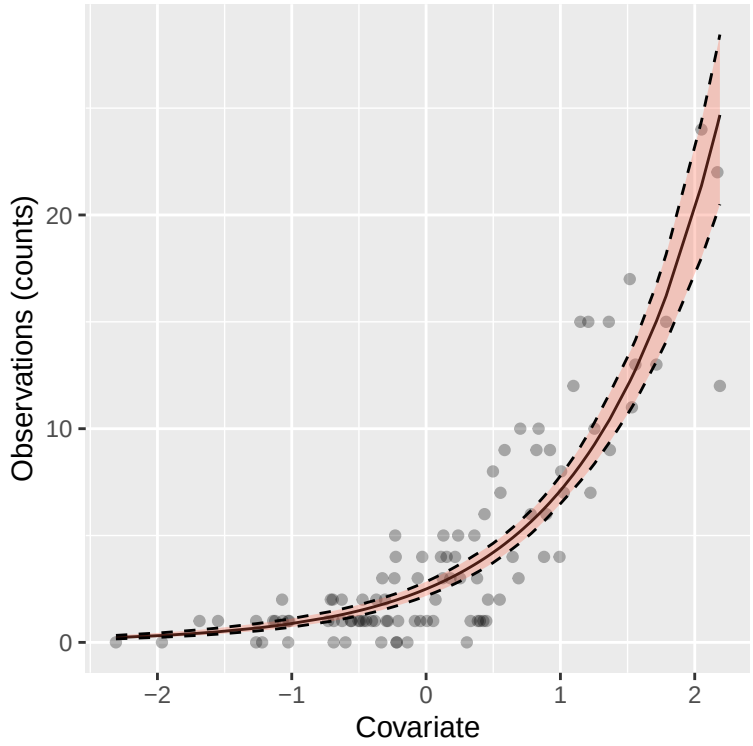


Figure 3: Data and 95% credible intervals

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)
```

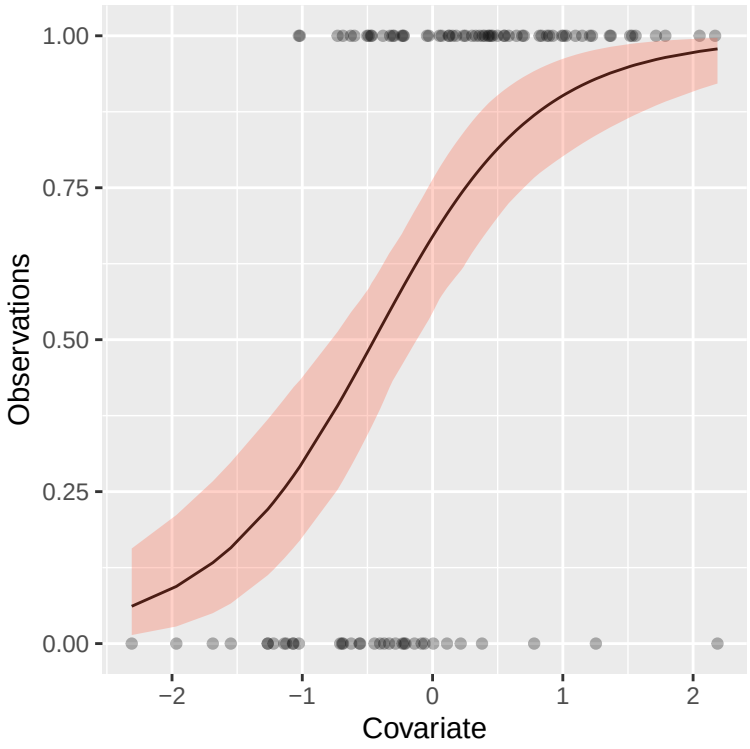
Here we use the logit link function $\text{logit}(x) = \log\left(\frac{x}{1-x}\right)$ (`plogis()` function in R) to link the linear predictor to the probabilities ψ.
Take hint
You can set `family = "binomial"` for binary responses and the `plogis()` function for computing the predicted values.

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

```
# 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 Generalised Additive Model

In this practical we will:

- Learn how to fit a GAM with `inlabru`
- Generate predictions from the model

Generalised Additive Models (GAMs) are very similar to linear models, but with an additional basis set that provides flexibility.

Additive models are a general form of statistical model which allows us to incorporate smooth functions alongside linear terms. A general expression for the linear predictor of a GAM is given by

$$\eta_i = g(\mu_i) = \beta_0 + \sum_{j=1}^{L} f_j(x_{ij})$$

where the mean $\boldsymbol{\mu} = E(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_L)$ and $g()$ is a link function (notice that the distribution of the response and the link between the predictors and this distribution can be quite general). The term $f_j()$ is a smooth function for the *j*-th explanatory variable that can be represented as

$$f(x_i) = \sum_{k=0}^{q} \beta_k b_k(x_i)$$

where b_k denote the basis functions and β_K are their coefficients.

Increasing the number of basis functions leads to a more *wiggly* line. Too few basis functions might make the line too smooth, too many might lead to overfitting.To avoid this, we place further constraints on the spline coefficients which leads to constrained optimization problem where the objective function to be minimized is given by:

$$\text{min} \sum_i (y_i - f(x_i))^2 + \lambda(\sum_k b_k^2)$$

The first term measures how close the function $f()$ is to the data while the second term $\lambda(\sum_k b_k^2)$, penalizes the roughness in the function. Here, $\lambda > 0$ is known as the smoothing parameter because it controls the degree of smoothing (i.e. the trade-off between the two terms). In a Bayesian setting,including the penalty term is equivalent to setting a specific prior on the coefficients of the covariates.

In this exercise we will set a random walk prior of order 1 on f, i.e. $f(x_i) - f(x_i - 1) \sim \mathcal{N}(0, \sigma_f^2)$ where σ_f^2 is the smoothing parameter such that small values give large smoothing. Notice that we will assume x_i's are equally spaced for now (we will cover a stochastic differential equation approach that relaxes this assumption later on in the course).

0.9.1 Simulate Data

Lets generate some data so evaluate how RW models perform when estimating a smooth curve. The data are simulated from the following model:

$$y_i = 1 + \text{cos}(x) + \epsilon_i,\ \epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

where $\sigma_\epsilon^2 = 0.25$

```
n = 100
x = rnorm(n)
```

```r
eta = (1 + cos(x))
y = rnorm(n, mean =  eta, sd = 0.5)

df = data.frame(y = y,
                x_smooth = inla.group(x)) # equidistant x's
```

0.9.2 Fitting a GAM in `inlabru`

Now lets fit a flexible model by setting a random walk of order 1 prior on the coefficients. This can be done bye specifying `model = "rw1"` in the model component (similarly,a random walk of order 2 can be placed by setting `model = "rw2"`)

```r
cmp =  ~ Intercept(1) +
  smooth(x_smooth, model = "rw1")
```

Now we define the observational model:

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

We then can fit the model:

```r
fit = bru(cmp, lik)
fit$summary.fixed
```

```
              mean          sd 0.025quant 0.5quant 0.975quant     mode
Intercept 1.370604 0.05953714   1.254405 1.370335   1.488349 1.370339
                   kld
Intercept 1.207209e-08
```

The posterior summary regarding the estimated function using RW1 can be accessed through `fit$summary.random$smooth`, the output includes the value of x_i (`ID`) as well as the posterior mean, standard deviation, quantiles and mode of each $f(x_i)$. We can use this information to plot the posterior mean and associated 95% credible intervals.

1 R plot

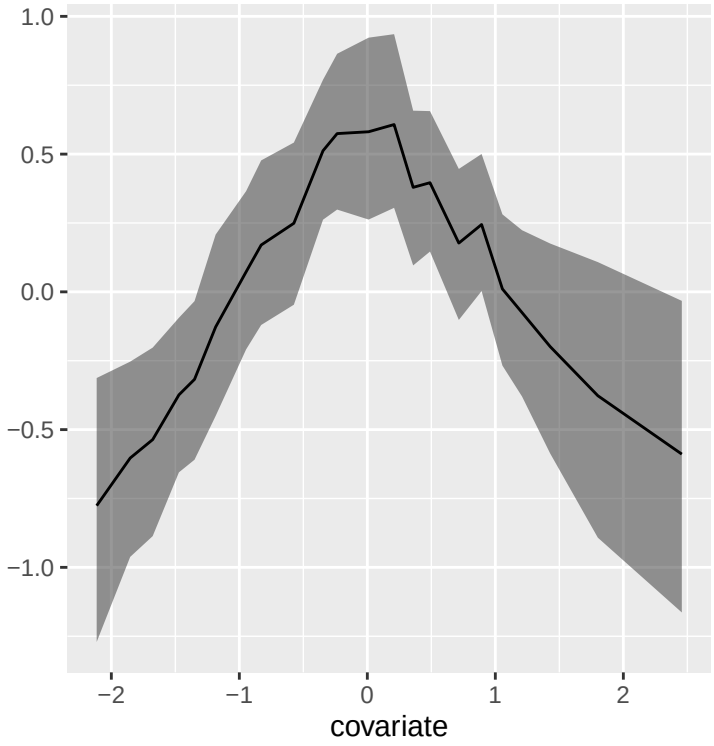


Figure 4: Smooth effect of the covariate

2 R Code

```
data.frame(fit$summary.random$smooth) %>%
  ggplot() +
  geom_ribbon(aes(ID,ymin = X0.025quant, ymax= X0.975quant), alpha = 0.5) +
  geom_line(aes(ID,mean)) +
  xlab("covariate") + ylab("")
```

2.0.1 Model Predictions

We can obtain the model predictions using the `predict` function.

```
pred = predict(fit, df, ~ (Intercept + smooth))
```

The we can plot them together with the true curve and data points:

```
pred %>% ggplot() +
  geom_point(aes(x_smooth,y), alpha = 0.3) +
  geom_line(aes(x_smooth,1+cos(x_smooth)),col=2)+
  geom_line(aes(x_smooth,mean)) +
  geom_line(aes(x_smooth, q0.025), linetype = "dashed")+
  geom_line(aes(x_smooth, q0.975), linetype = "dashed")+
  xlab("Covariate") + ylab("Observations")
```

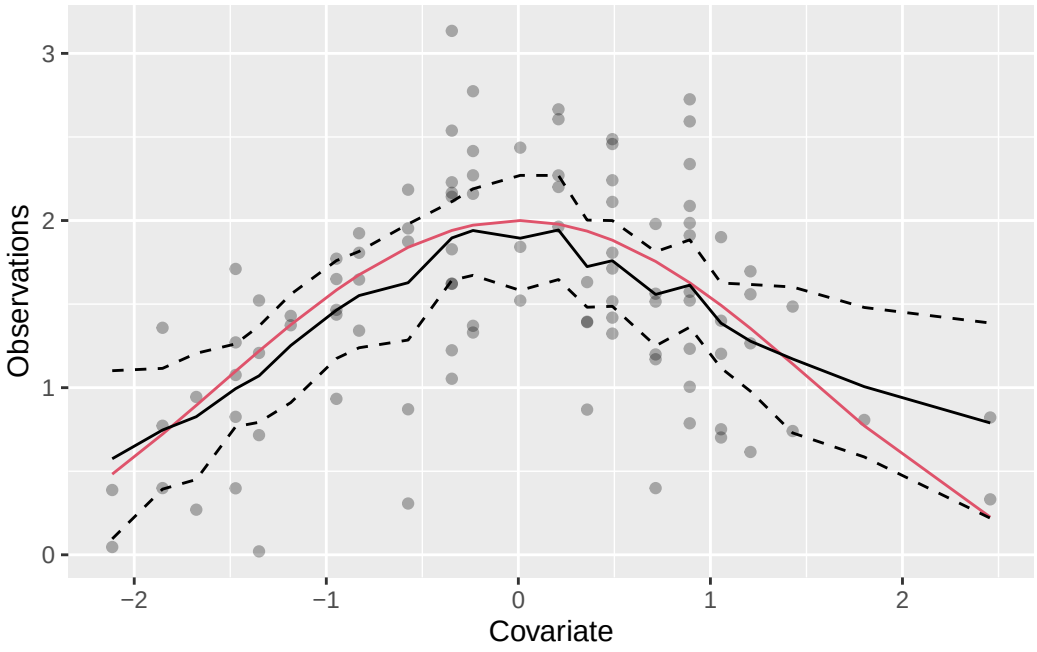
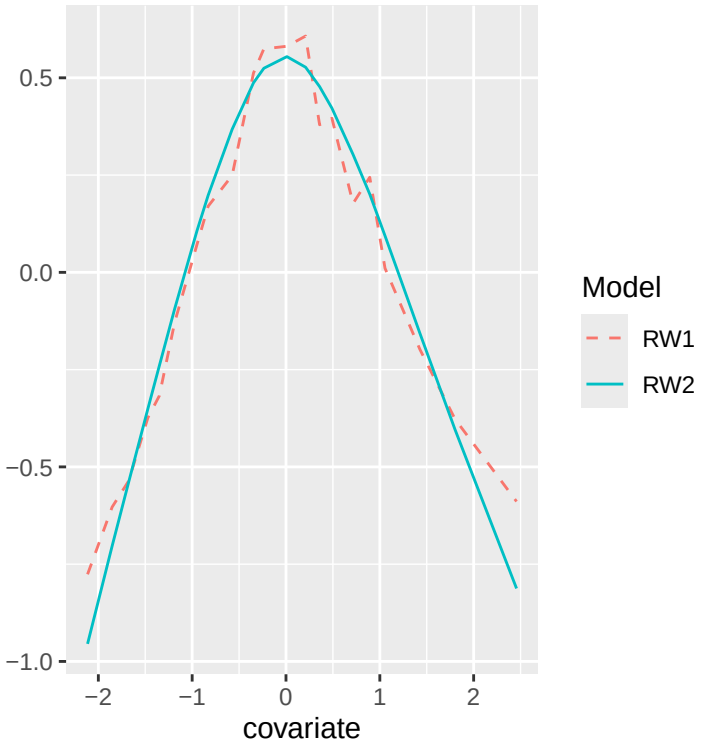


Figure 5: Data and 95% credible intervals

Task

Fit a flexible model using a random walk of order 2 (RW2) and compare the results with the ones above.
Take hint
You can set `model = "rw2"` for assigning a random walk 2 prior.
Click here to see the solution

```
cmp_rw2 =  ~ Intercept(1) +
  smooth(x_smooth, model = "rw2")
lik_rw2 =  bru_obs(formula = y ~.,
            family = "gaussian",
            data = df)
fit_rw2 = bru(cmp_rw2, lik_rw2)

# Plot the fitted functions
ggplot() +
  geom_line(data= fit$summary.random$smooth,aes(ID,mean,colour="RW1"),lty=2) +
  geom_line(data= fit_rw2$summary.random$smooth,aes(ID,mean,colour="RW2")) +
  xlab("covariate") + ylab("") + scale_color_discrete(name="Model")
```



We see that the RW1 fit is too wiggly while the RW2 is smoother and seems to have better fit.