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:

- 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: 2.13.0.9011
```

```
## INLA version: 25.09.19
## 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 ~ .
##     Additive/Linear: TRUE/TRUE
##     Used components: effects[beta_0, beta_1], latent[]
## Time used:
##     Pre = 1.33, Running = 0.213, Post = 0.0199, Total = 1.56
## Fixed effects:
##          mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## beta_0 1.977 0.011      1.956    1.977      1.998 1.977   0
## beta_1 0.488 0.012      0.465    0.488      0.511 0.488   0
##
## Model hyperparameters:
##                                           mean    sd 0.025quant 0.5quant
## Precision for the Gaussian observations 91.69 12.96      68.08    91.08
##                                         0.975quant  mode
## Precision for the Gaussian observations     118.81 89.86
##
## Marginal log-Likelihood:  61.67
##  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

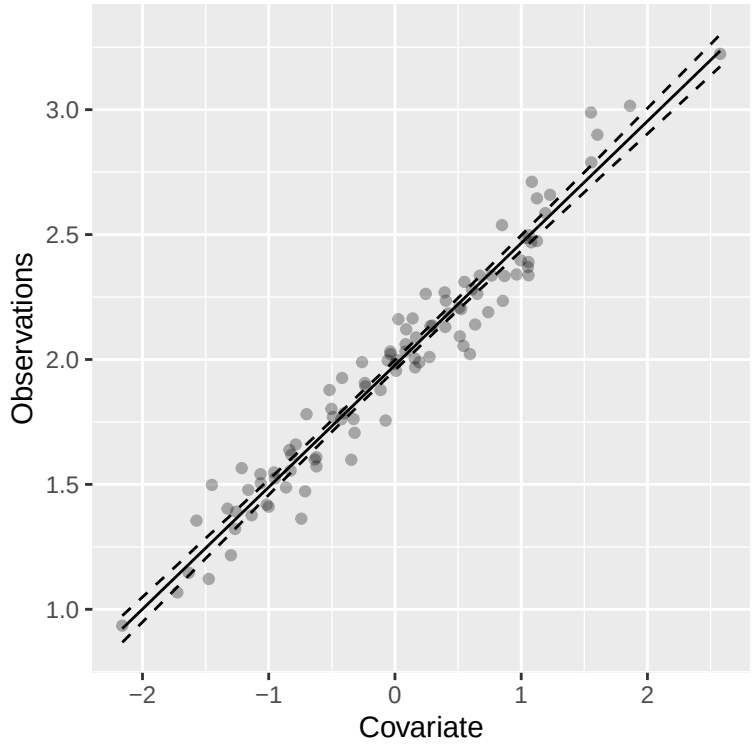


Figure 1: Data and 95% credible intervals

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

0.4.1 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")
```

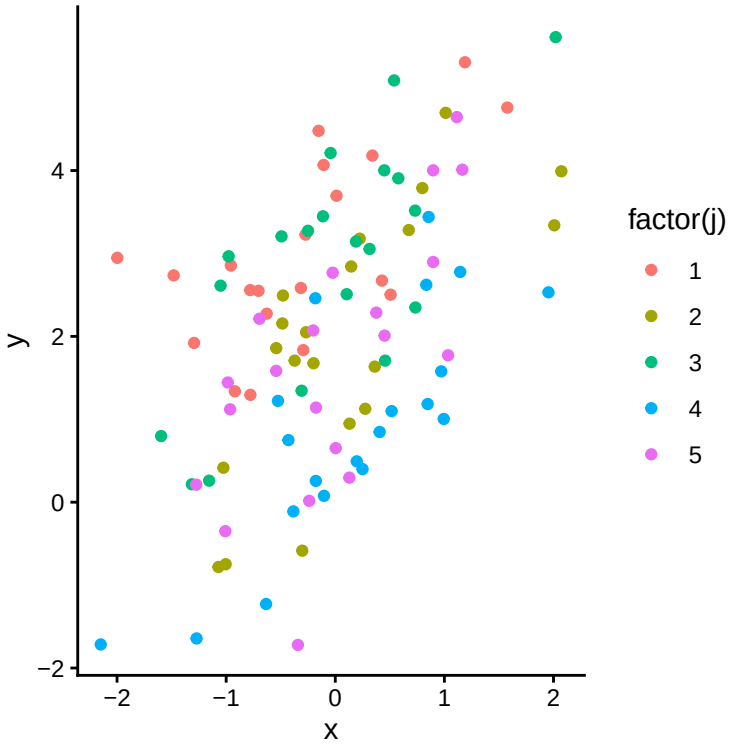


Figure 2: Data for the linear mixed model example with 5 groups

0.4.2 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 =  like(formula = y ~.,
            family = "gaussian",
            data = df)
```

```
Warning: `like()` was deprecated in inlabru 2.12.0.
i Please use `bru_obs()` instead.
```

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: 2.13.0.9011
## INLA version: 25.09.19
## 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 = 1.18, Running = 0.212, Post = 0.0344, Total = 1.43
## 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:
```

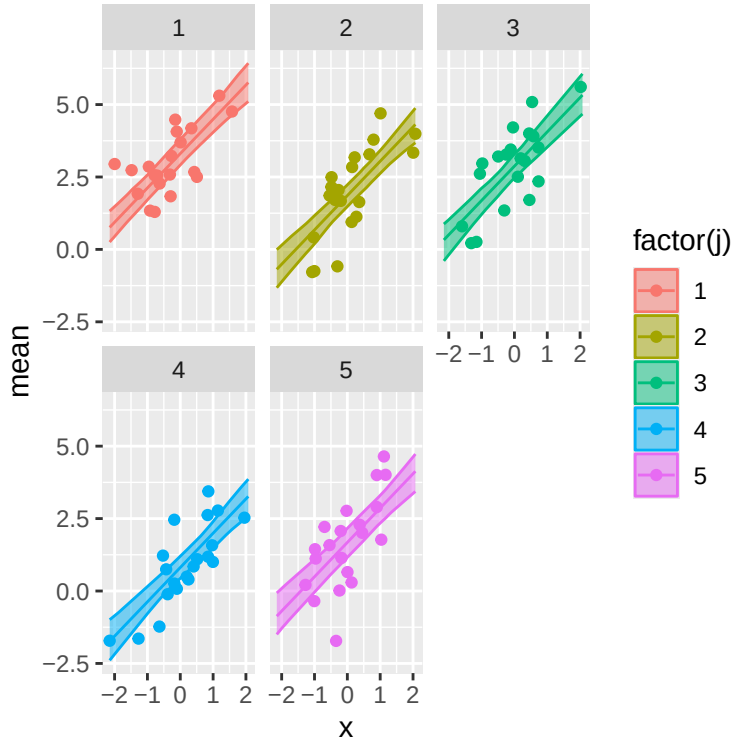
```
##                                             mean    sd 0.025quant 0.5quant
## Precision for the Gaussian observations 0.995 0.144      0.738    0.986
## Precision for u                         1.613 1.060      0.369    1.356
##                                         0.975quant  mode
## Precision 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)')
```

0.4.3 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. This is identical in form to the predictor in Section 0.1. The only difference is now we must specify a different data likelihood.

0.5.1 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)
```

0.5.2 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: 2.13.0.9011
INLA version: 25.09.19
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 ~ .
    Additive/Linear: TRUE/TRUE
    Used components: effects[beta_0, beta_1], latent[]
Time used:
    Pre = 1.17, Running = 0.214, Post = 0.0066, Total = 1.39
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)')
```

0.5.3 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

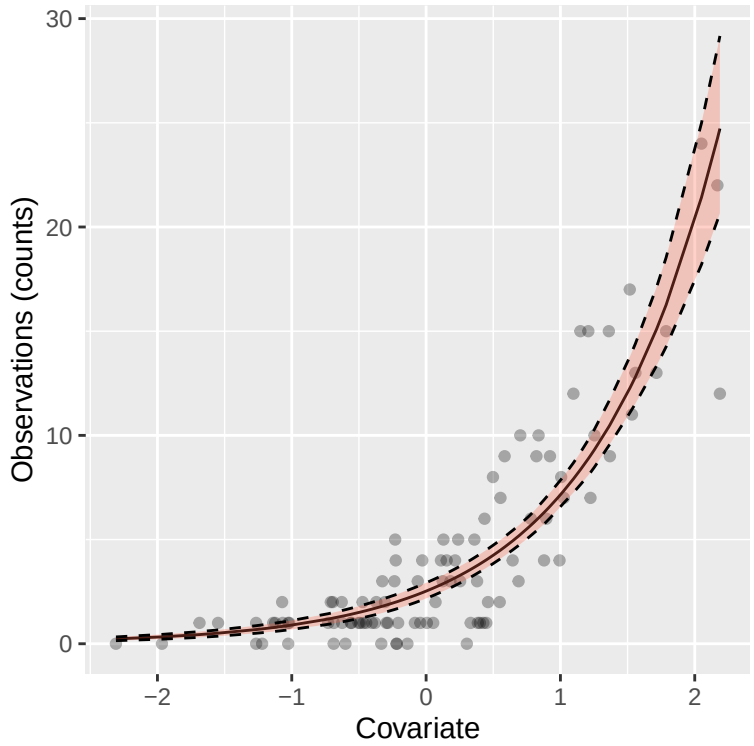


Figure 3: Data and 95% credible intervals

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).

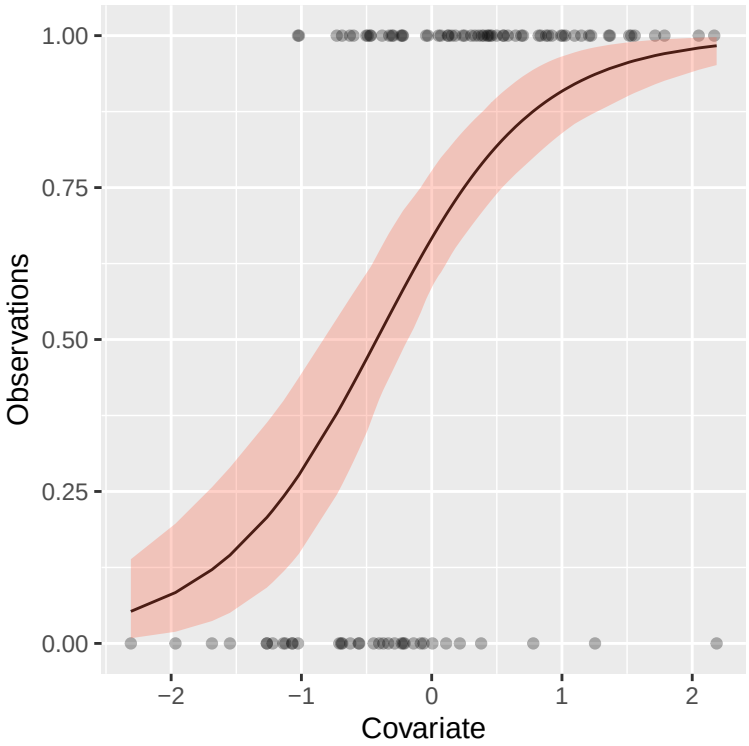
Click here to see the solution

```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 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).

0.8.1 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:

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

where $\sigma_\epsilon^2 = 0.25$

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

0.8.2 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"`)

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

Now we define the observational model:

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

We then can fit the model:

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

```
              mean          sd 0.025quant 0.5quant 0.975quant     mode
Intercept 1.298799 0.06506724    1.17164 1.298566   1.427294 1.298569
                   kld
Intercept 9.965732e-09
```

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

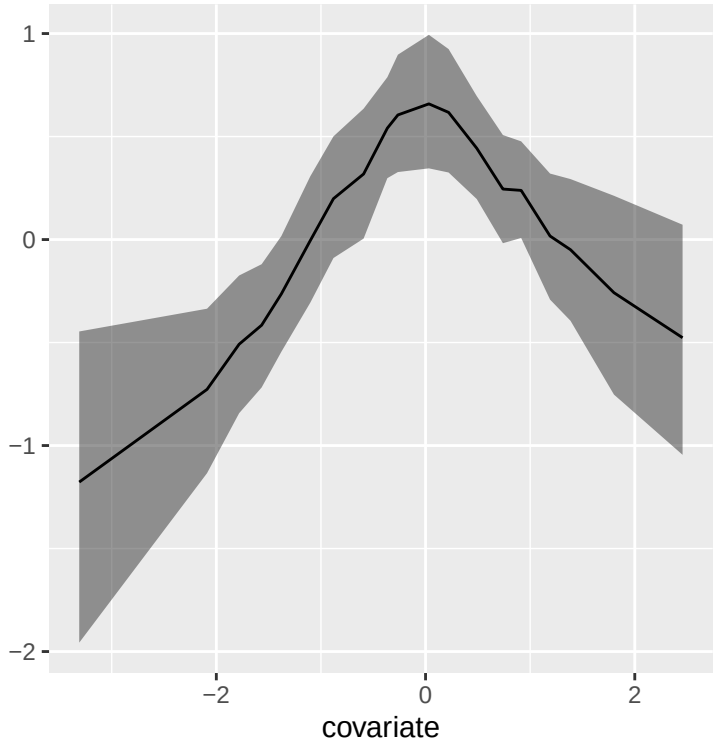


Figure 4: Smooth effect of the covariate

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")
```

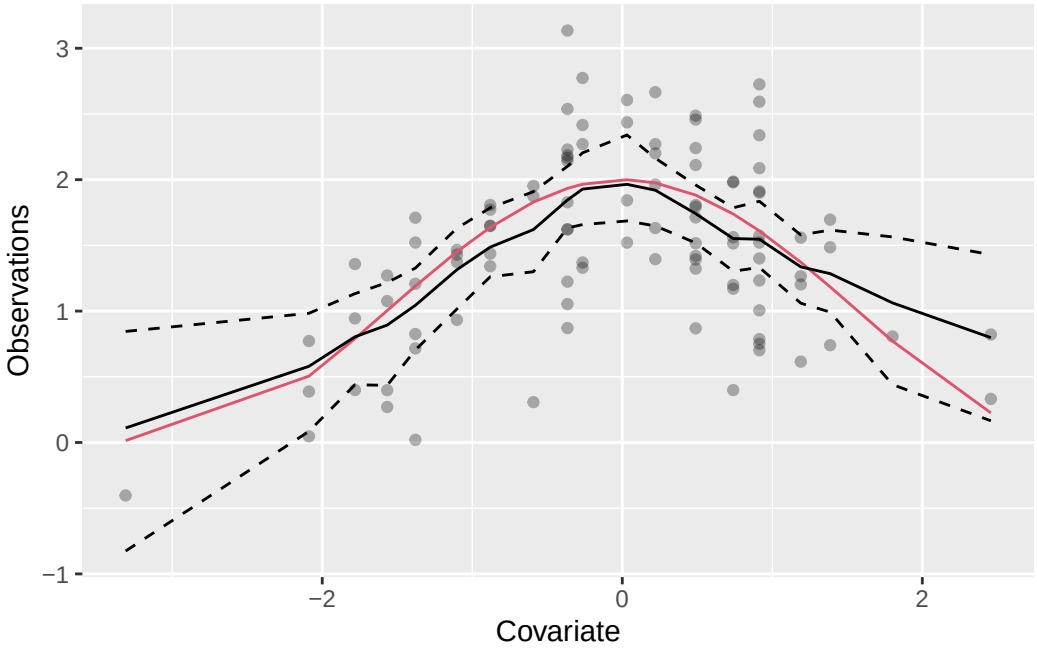


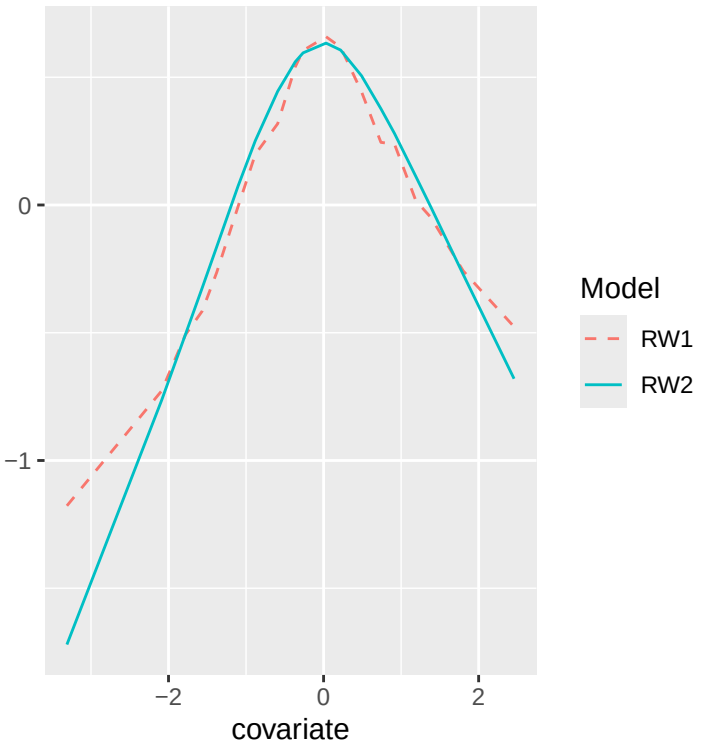
Figure 5: Data and 95% credible intervals

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.