# PRACTICAL 1 - LINEAR (MIXED) MODELS

In this practical we are going to fit linear (mixed) models in `inlabru`. We are going to to:

- Fit a simple linear regression
- Fit a linear regression with discrete covariates and interactions
- Fit a linear mixed model.

> Note
>
> You can download the R-script of this practical by clicking the button below:

Start by loading useful libraries:

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

## 1 Simple linear regression

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 $\sigma^2$ is the observation error, and the mean parameter $\mu_i$ is linked to the **linear predictor** ($\eta_i$) through an identity function:

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

Here $\mathbf{x} = (x_1, \dots, x_N)$ is a continuous covariate and $\beta_0, \beta_1$ are parameters to be estimated.

To finalize the Bayesian model we assign prior distribution as $\tau = 1/\sigma^2 \sim$ Gamma$(a, b)$ and $\beta_0, \beta_1 \sim \mathcal{N}(0, 1/\tau_\beta)$ (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 $\beta_0$ and $\beta_1$ are fixed effects, the precision parameters $\tau_{\beta_0}$ and $\tau_{\beta_1}$ 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.

# 1 Simulate example data

We fix the model parameters $\beta_0, \beta_1$ and the hyperparameter $\tau_y$ 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)

# Fix the model parameters
beta = c(2,0.5)
sd_error = 0.1

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

# create the data frame object
df = data.frame(y = y, x = x)
```

# 1 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

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")
```

**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 5: 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: 25.11.12-1
## 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 = 1.03, Running = 0.214, Post = 0.0188, Total = 1.26
## 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 $\mu$ 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.

# 2 Plot

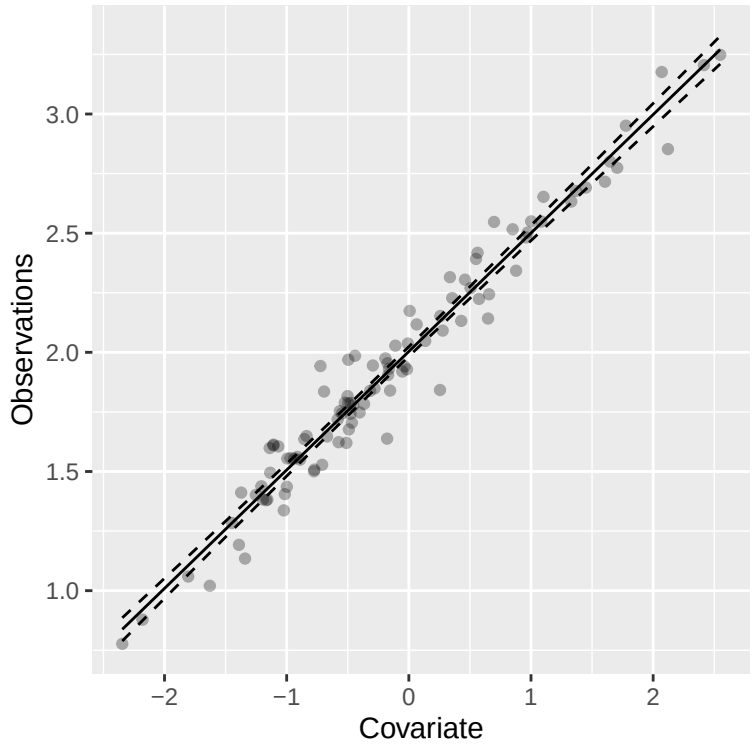



Figure 1: Data and 95% credible intervals

# 3 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 $\mu$ when the covariate has value $x_0 = 0.45$.
What is the predicted value for $\mu$? 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 2.228097 0.0116709 2.204786 2.228269 2.251065 2.228269  0.0002606852
  mean.mc_std_err
1    0.0003855534
```

You can see the predicted mean and sd by examining the produced `pred` object. In this case the mean is ca 2.22 with sd ca 0.01. This gives a 95% CI ca [2.20, 2.25].

**NOTE** Now we have produced a credible interval for the expected mean $\mu$ 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 $\tau_y$. 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.

# 4 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 specie).

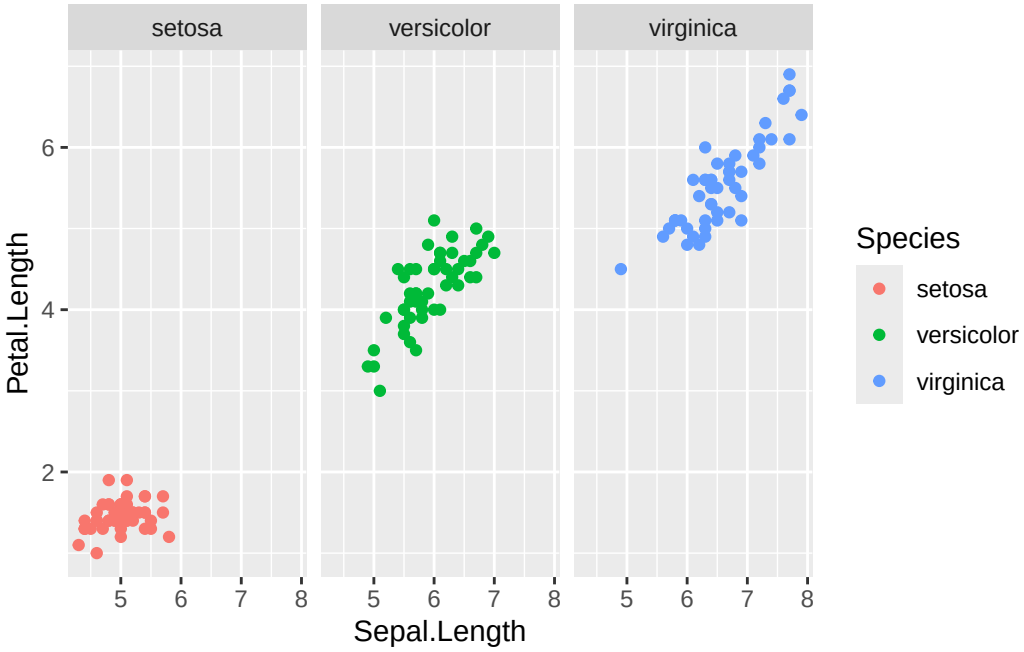
You can get more information about these data using

```
?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.

Task

Implement the model above using `inlabru` using the `model.matrix()` function in R.
Click here to see the solution

```
#Option 1 - Use the model.matrix() function

mm = model.matrix(Petal.Length ~ Species, data  = iris)
iris1 = cbind(iris, mm)
cmp = ~ Intercept(1) +  versicolor(Speciesversicolor, model = "linear") + virginica(Specie
lik = bru_obs(formula =Petal.Length ~ .,
              data = iris1)
fit1a = bru(cmp, lik)
```

Another way to fit this model is to realize that a fixed effect can be seen as an iid effect with fixed precision, one of the `inlabru` ways to fit the model is:

```
cmp = ~ -1 +  cov(Species, model = "iid", fixed = T, initial = log(0.001))
lik = bru_obs(formula =Petal.Length ~ .,
              data = iris)
fit1b = bru(cmp, lik)
```

Notice that we fix the precision of the `iid` effect to the same value as the precision of the linear effects which is 0.001.

The fitted values can be inspected as

```
fit1b$summary.random$cov
```

```
          ID     mean         sd 0.025quant 0.5quant 0.975quant     mode
1     setosa 1.461995 0.06077759   1.342650 1.461995   1.581339 1.461995
2 versicolor 4.259984 0.06077759   4.140639 4.259984   4.379329 4.259984
3  virginica 5.551979 0.06077759   5.432634 5.551980   5.671324 5.551980
           kld
1 3.690511e-09
2 3.690694e-09
3 3.690617e-09
```

The results from the `fit1b` model is not the same we get from the `lm()` fit. What is happening? It is just a matter of parametrization.

In the following we are going to recover the `lm()` results using the fitted `fit1b` object and the `predict()` function.

Click here to see the solution

```
newdata = data.frame(Species = 0)
preds = predict(fit1b, newdata = c() , ~ data.frame(Reference = cov_latent[2],
                                                    Speciesversicolor = cov_latent[2]-cov_latent[1]
                                                    Speciesvirginica = cov_latent[3]-cov_latent[1])
```

Two things to notice here:

1. We have used a empty object as `newdata` in the `predict()` function.
2. The suffix `_latent` indicates that we are interested in the latent model efffect.

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

Out second model is defined as

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

that is, we have a common intercept $\beta_0$ while the linear effect of the Sepal length depends on the Species. Using `lm()` we have:

```
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` using the `model.matrix()`.
Click here to see the solution

```
mm = model.matrix(Petal.Length ~ Species:Sepal.Length, data = iris)

# change the names to make the R friendly (the : makes problems when using them to identif

colnames(mm) = c("Int", "Setosa_Sepal",  "Versicolor_Sepal", "Virginica_Sepal")

iris2 = cbind(iris, mm)

cmp = ~ Intercept(1) + int1(Setosa_Sepal, model = "linear") +
  int2(Versicolor_Sepal, model = "linear") +
  int3(Virginica_Sepal, model = "linear")

lik = bru_obs(formula =Petal.Length ~ .,
              data = iris2)
fit2 = bru(cmp, lik)
```

A second option is to use a *weighted* iid random effect with fixed precision as:

```
cmp = ~Intercept(1)  +
  slope(Species, Sepal.Length, model = "iid", fixed = T, initial = log(0.001))

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

```
fit2b = bru(cmp, lik)
```

Notice that this time, the parametrization of `lm()` and `inlabru` is the same, so we get the same results from the two models.

Task

Plot the estimated regression lines for the three species using model `fit2b`
Click here to see the solution

```
preds = predict(fit2b, iris, ~ Intercept + slope)

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))
```

# 5 Linear Mixed Model

Finally we are going to consider a simple linear mixed model, that is 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:

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

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 $(\beta_0, \beta_1)$ 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:
  - $\tau_u \sim \text{Gamma}(a, b)$
  - $\tau_\epsilon \sim \text{Gamma}(c, d)$

## 5 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")
```

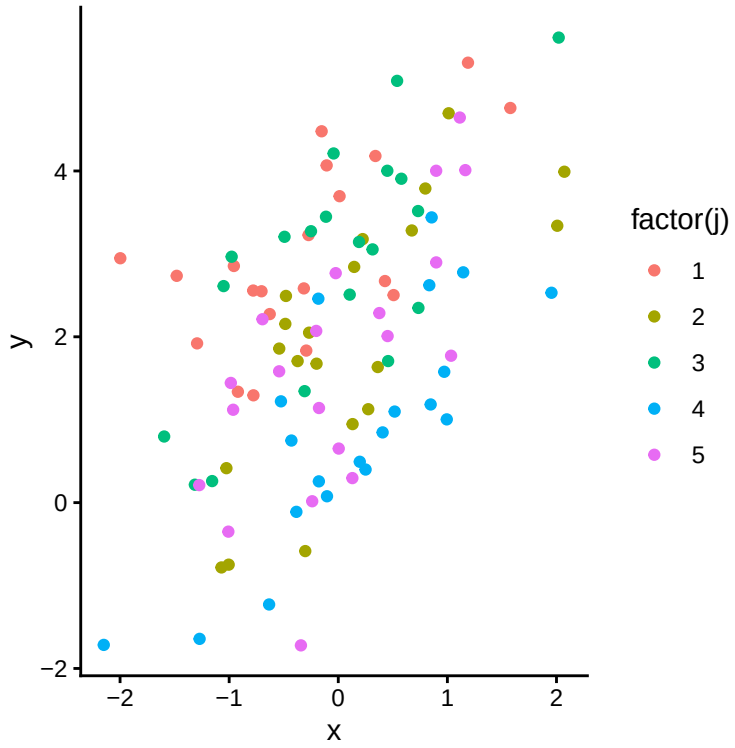



Figure 2: Data for the linear mixed model example with 5 groups

## 5 Fitting a LMM in `inlabru`

This is done in three steps:

1. Define the model components
2. Define the formula and the observation model (likelihood) using `bru_obs()`
3. Run the model using `bru()`

After that you can collect and inspect results.

**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.

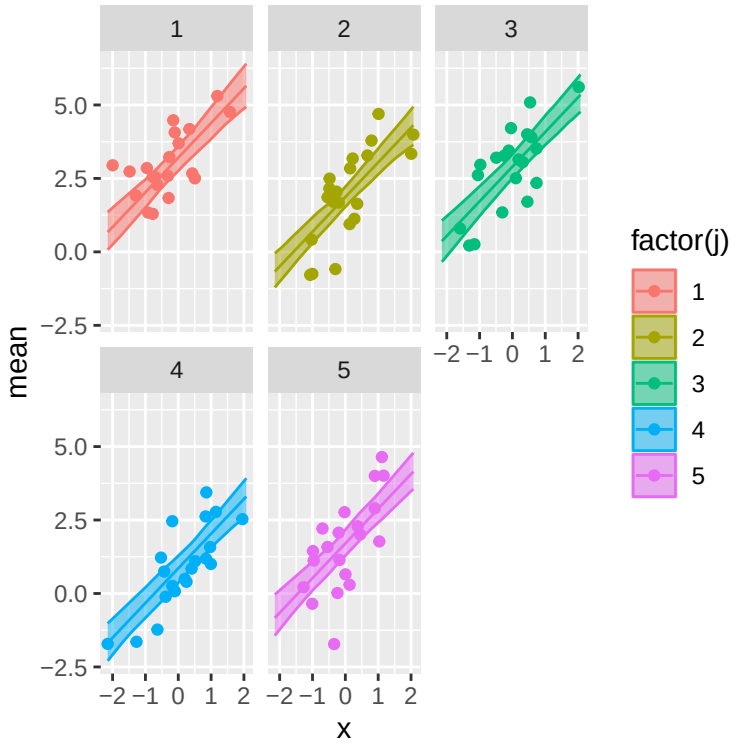
```
fit = bru(cmp, lik)
summary(fit)
## inlabru version: 2.13.0.9016
## INLA version: 25.11.12-1
## Components:
## Latent components:
## beta_0: main = linear(1)
## beta_1: main = linear(x)
## u: main = iid(j)
## Observation models:
##   Family: 'gaussian'
##     Tag: <No tag>
##     Data class: 'data.frame'
##     Response class: 'numeric'
##     Predictor: y ~ .
##     Additive/Linear: TRUE/TRUE
##     Used components: effects[beta_0, beta_1, u], latent[]
## Time used:
##     Pre = 1.03, Running = 0.213, Post = 0.0338, Total = 1.27
## 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)')
```

## 5 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 specifying the model components.

```
# 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 $\beta$ 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:
  - $\tau_{u_0}, \tau_{u_1}$ and $\tau_\epsilon$

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")
```