PRACTICAL 2 - GLMM

In this practical we are going to fit a Generalized (Mixed) Linear Model in `inlabru`.

We are going to:

- Fit a Poisson regression
- Change the prior distributions for the model hyperparameters (both for fixed and random effects)
- Compute and visualize posterior densities and summaries for marginal effects

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

We start by loading some useful libraries

```
library(dplyr)
library(INLA)
library(ggplot2)
library(patchwork)
library(inlabru)
```

In this practical we are going to analyse the dataset `grouseticks` contained in the library `lme4` or if your prefer you can download the data by clicking the button below:

The data contain the number of ticks on the heads of red grouse chicks sampled in the field. See `?grouseticks` for details.

The data are analysed in the paper

Elston et al. "Analysis of aggregation, a worked example: numbers of ticks on red grouse chicks." Parasitology (2001)

and in this practical we will follow their analysis.

```
grouseticks<- read.csv(here::here("datasets/grouseticks.csv"))
```

1 Fitting Poisson regression

We assume that the number of ticks y_{ijk} counted on chick i of brood j in year k follows a Poisson distribution with mean λ_{ijk}

$$y_{ijk}|\lambda_{ijk} \sim \text{Poisson}(\lambda_{ijk})$$

We then model log mean counts $\eta_{ijk} = \log(\lambda_{ijk})$ as a linear function of year, altitude, brood, and individual chick within brood.

We assume a fixed effect α_k of year k, a linear effect of altitude x_{ij}, two random effects e_{jk} and ϵ_{ijk} brood and individual within brood respectively. Thus:

$$\eta_{ijk} = \log(\lambda_{ijk}) = \alpha_k + \beta x_{ij} + e_{jk} + \epsilon_{ijk} \quad (1)$$

where $e_{jk} \sim \mathcal{N}(0, \tau_e^{-1})$ and $\epsilon_{jk} \sim \mathcal{N}(0, \tau_\epsilon^{-1})$.

We first fit the model using the default priors for fixed and random effects.

To fit the model we first have to create two more variables, one that indexes the combination jk of brood and year and another that indexes the combination ijk of individuals per brood per year.

```
grouseticks  = grouseticks %>%
  group_by(BROOD, YEAR) %>%
  mutate(ij = cur_group_id()) %>%
  ungroup() %>%
  mutate(ijk = seq_along(INDEX))
```

Now we can fit the model.

Task

Complete the code to fit the model in Equation 1. You need to define the components, the likelihood and then use the `bru()` function to get the results

```
bru_options_set(control.compute = list(dic = T, waic = T))

cmp= ~ -1 + year(...) +
  height(...) +
  brood_year(...) +
  brood_year_chicken(...)

lik = bru_obs(forumula = ...,
              data = ...,
              family = ...)

fit = bru(cmp, lik)
```

Click here to see the solution

```
bru_options_set(control.compute = list(dic = T, waic = T))


cmp= ~ -1 +  year(YEAR, model = "iid", initial = log(0.01), fixed = T) +
  height(cHEIGHT, model =  "linear") +
  brood_year(ij , model = "iid") +
  brood_year_chicken(ijk, model= "iid")

lik = bru_obs(TICKS ~ .,
              data = grouseticks ,
              family = "Poisson")


fit = bru(cmp, lik)
```

1 Posterior Summaries & model fit

Now we can check the results using the `summary` function directly by calling `model$summary.hyperpar`, `model$summary.fixed` and `model$summary.random`. In addition, `inlabru` objects can be passed to the `tidy()` function to produce posterior summaries of the hyperparameters and fixed effects in a `tibble` format:

```
tidy(fit)
```

```
# A tibble: 1 x 5
  term   estimate std.error conf.low conf.high
  <chr>     <dbl>     <dbl>    <dbl>     <dbl>
1 height  -0.0242   0.00306  -0.0302   -0.0182
```

Task

Use the `tidy()` function to produce posterior summaries of the hyperparameters
Click here to see the solution

```
tidy(fit, "hyperpar")
```

```
# A tibble: 2 x 5
  term                            estimate std.error conf.low conf.high
  <chr>                              <dbl>     <dbl>    <dbl>     <dbl>
1 Precision for brood_year            1.21     0.238    0.804      1.74
2 Precision for brood_year_chicken    3.62     0.666    2.49       5.11
```

We can also use the `glance()` function from `broom` to obtain our model's goodness-of-fit metrics in a tibble (provided we set `bru_options_set(control.compute = list(dic = TRUE, waic = TRUE))`).

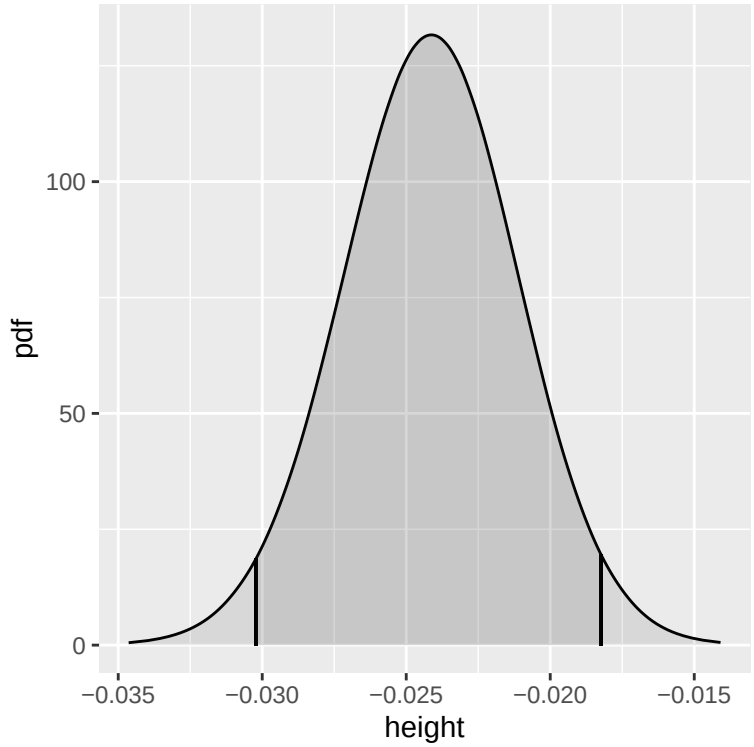
```
glance(fit)
```

```
# A tibble: 1 x 5
    dic  waic marginal_loglik  nobs elapsed
  <dbl> <dbl>           <dbl> <int>   <dbl>
1 1574. 1571.           -934.   403    3.70
```

1 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 height effect we can type:

```
plot(fit, "height")
```



Task

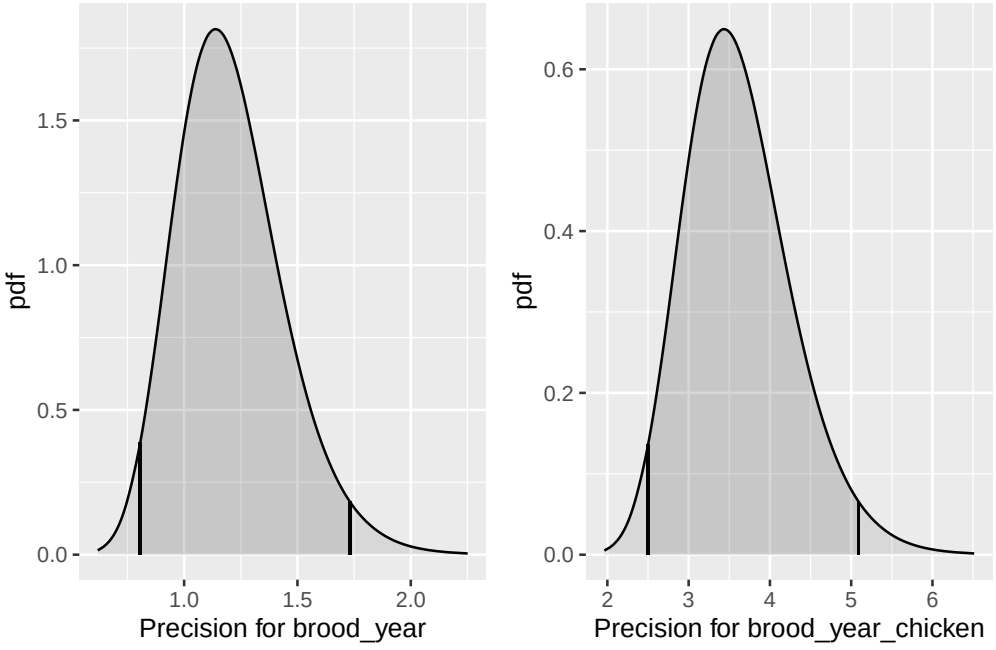
Plot the posterior marginals for the precision parameters of (i) the brood random effect τ_e^{-1} and (ii) within-brood ϵ_{ijk}

Take hint

See the `summary()` output to check the names for the different model components.

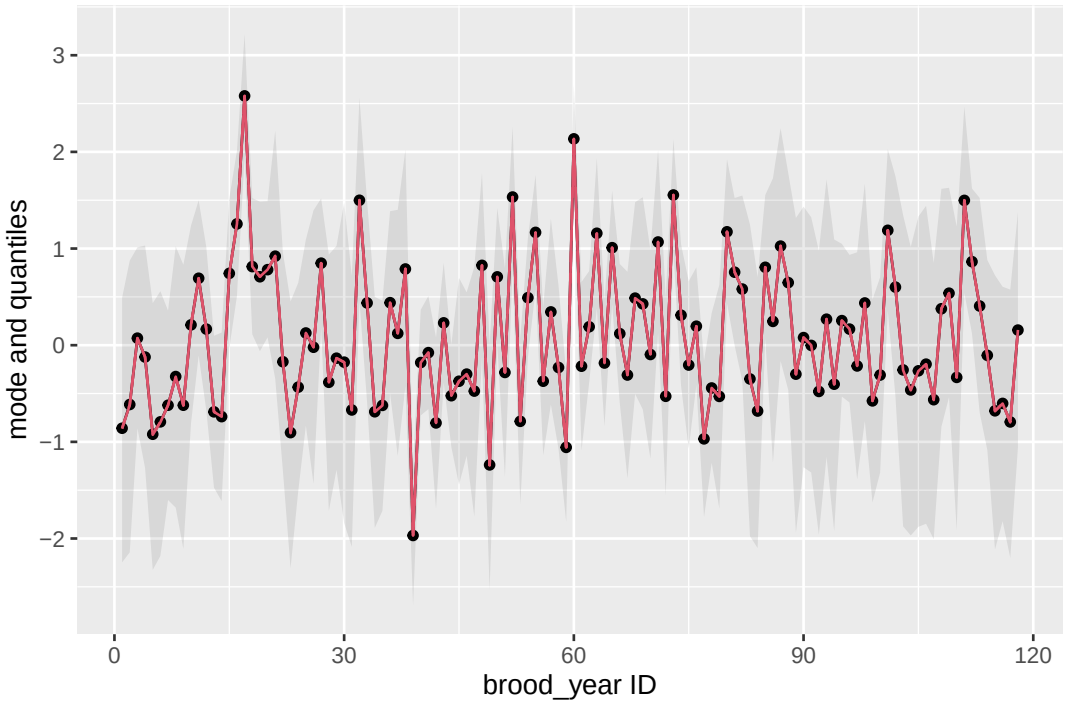
Click here to see the solution

```
plot(fit, "Precision for brood_year") +
plot(fit, "Precision for brood_year_chicken")
```



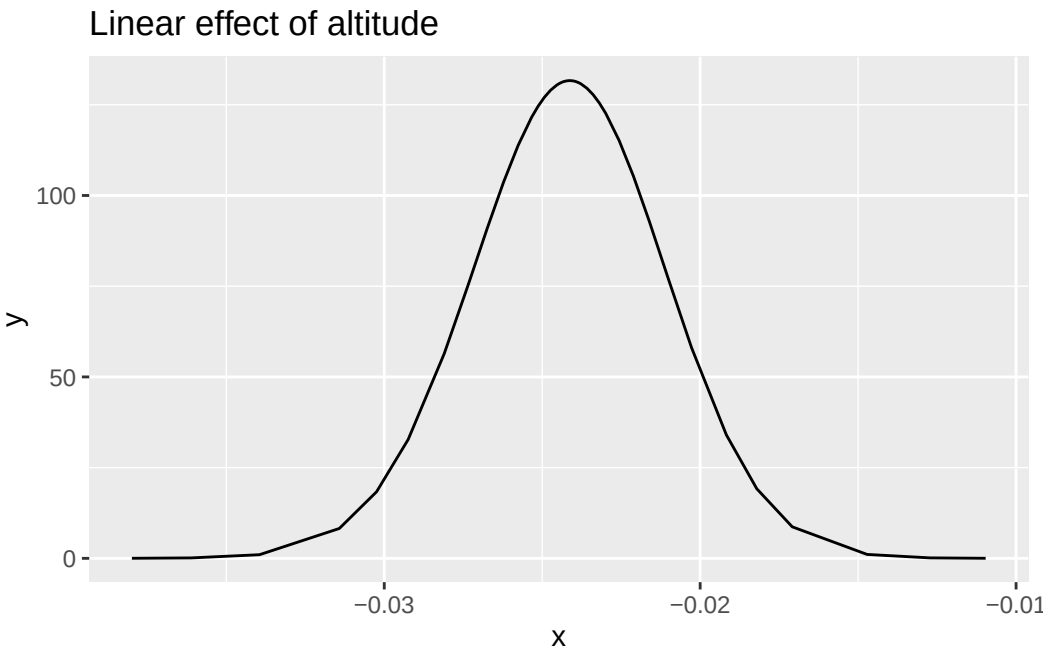
You can also use the `plot()` function to visualize the mode and quantiles for random effect. For example we can visualize the brood random effects as:

```
plot(fit, "brood_year")
```



Another useful way to retrieve marginals densities of hyperparameters, fixed and random effects is by calling `model$marginals.hyperpar`,`model$marginals.fixed` and `model$marginals.random` respectively. The output can then be plotted as follows:

```
fit$marginals.fixed$height %>%
  ggplot() +
  geom_line(aes(x,y)) +
  ggtitle("Linear effect of altitude")
```

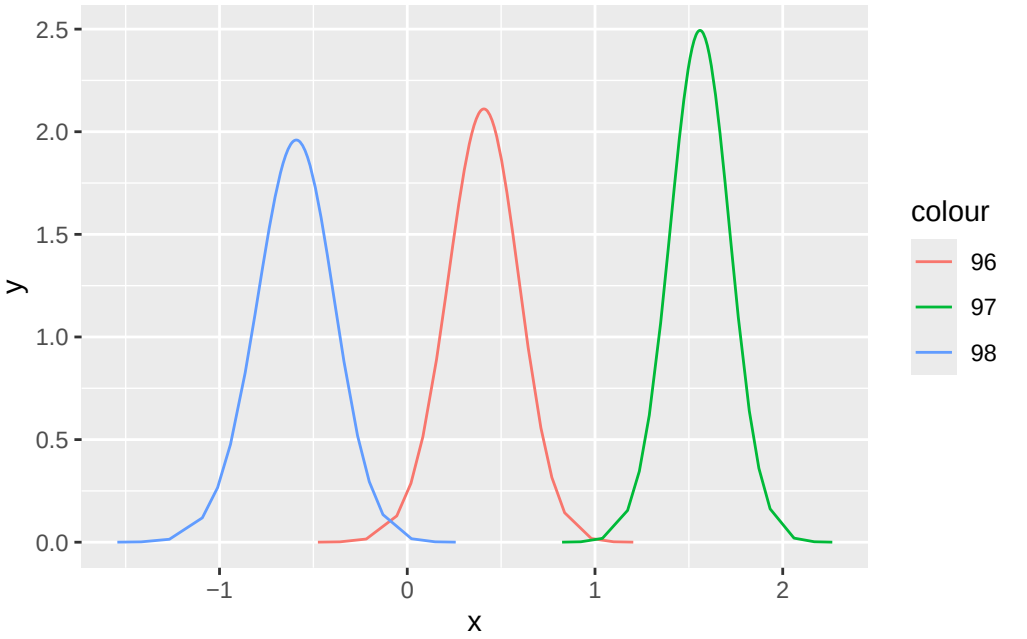


Notice that this is the same density plot we produced by using the `plot()` function above.

Task

Plot the posterior distribution of the year random effect α_k .
Click here to see the solution

```
ggplot() + geom_line(data = fit$marginals.random$year[[1]], aes(x,y, color = "96") )+
  geom_line(data = fit$marginals.random$year[[2]], aes(x,y, color = "97")) +
  geom_line(data = fit$marginals.random$year[[3]], aes(x,y, color = "98"))
```



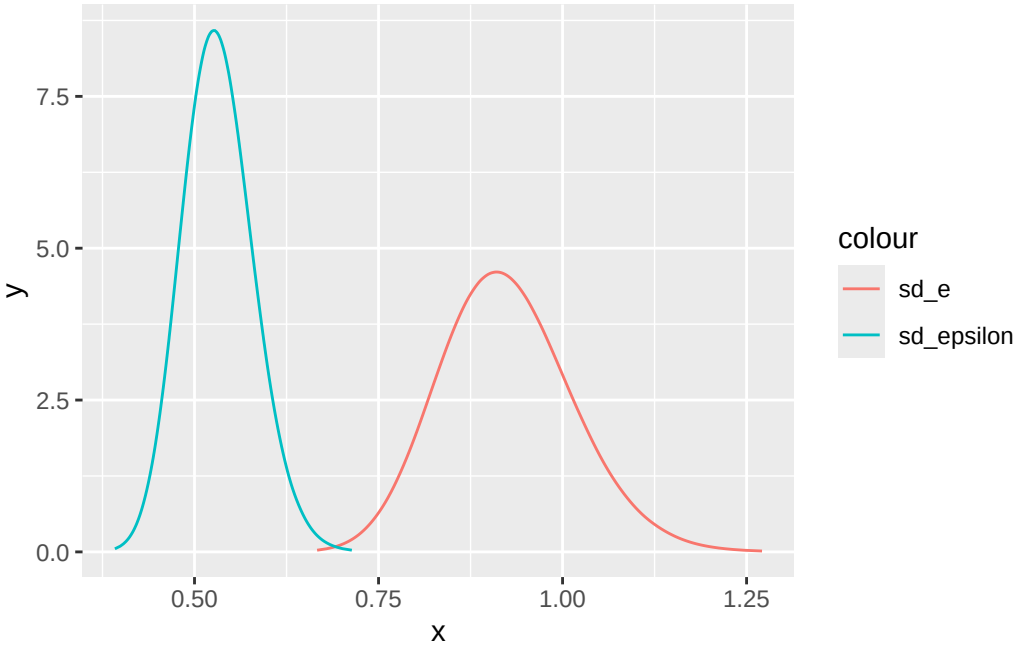
1 Applying transformations to marginal densities

For theoretical and computational purposes, INLA works with the precision which is the inverse of the variance. To obtain the posterior summaries on the SDs scale we can apply a transformation using the `inla.tmarginal` function to transform the precision posterior distributions.Transforming the samples is necessary because some quantities such as the mean and mode are not invariant to monotone transformation.

```
sd_e <- fit$marginals.hyperpar$`Precision for brood_year`%>%
  inla.tmarginal(function(x)sqrt(1/x),.)

sd_eps <- fit$marginals.hyperpar$`Precision for brood_year_chicken` %>%
  inla.tmarginal(function(x) sqrt(1/x),.)

ggplot() +
  geom_line(data = sd_e, aes(x,y, color = "sd_e")) +
  geom_line(data = sd_eps, aes(x,y, color = "sd_epsilon"))
```



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

```
post_var_summaries <- cbind( inla.zmarginal(sd_e,silent = T),
                             inla.zmarginal(sd_eps,silent = T))
colnames(post_var_summaries) <- c("sigma_e","sigma_eps")
post_var_summaries
```

```
           sigma_e    sigma_eps
mean       0.9239009  0.5320194
sd         0.09013151 0.04815832
quant0.025 0.7595966  0.4431616
quant0.25  0.8606525  0.4983449
quant0.5   0.9192956  0.5299775
quant0.75  0.9820568  0.5633522
quant0.975 1.113322   0.632172
```

Supplementary material

To obtain the posterior summaries on the SDs scale we can also sample from the posterior distribution for the precision while back-transforming the samples and then computing the summary statistics. We can use the `generate()` function to draw samples from the approximated joint posterior for the hyperparameters, then invert them to get variances and lastly compute the mean, std. dev., quantiles, etc.
To get the right name for the hyperparameters to use in the `generate()` function, you can use the function `bru_names()`.

```
bru_names(fit)
```

```
                          height                                    year
                        "height"                                  "year"
                      brood_year                        brood_year_chicken
                     "brood_year"                      "brood_year_chicken"
       Precision for brood_year         Precision for brood_year_chicken
"Precision_for_brood_year"       "Precision_for_brood_year_chicken"
```

We then obtain the samples:

```
out = generate(fit, formula = ~
                  data.frame(sd_e = sqrt(1/Precision_for_brood_year),
                             sd_eps = sqrt(1/Precision_for_brood_year_chicken)),
                n.samples = 1000)

sd_sample = data.frame(sd_e = unlist(sapply(out, function(x) x[1])),
                       sd_eps = unlist(sapply(out, function(x) x[2])))
```

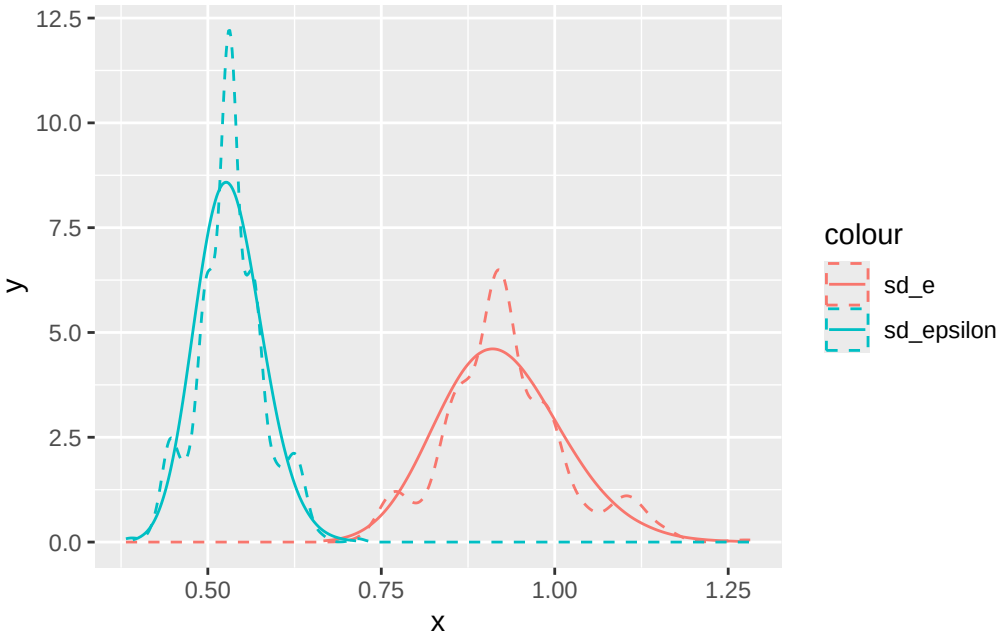
Compute posterior summaries:

```
sd_sample %>%
  summarise(across(
    everything(),
    list(
      mean = ~ mean(.x),
      q025 = ~ quantile(.x, 0.025),
      q975 = ~ quantile(.x, 0.975)
    )
  ))
```

```
  sd_e_mean sd_e_q025 sd_e_q975 sd_eps_mean sd_eps_q025 sd_eps_q975
1  0.929162 0.7652468  1.127279    0.5332328     0.437908    0.6268977
```

And finally, overlay the densities obtained with `inla.tmarginal` and `generate`:

```
ggplot() + geom_line(data = sd_e, aes(x,y, color = "sd_e")) +
  geom_density(data = sd_sample, aes(x = sd_e, color = "sd_e"), linetype = "dashed") +
  geom_line(data = sd_eps, aes(x,y, color = "sd_epsilon"))+
  geom_density(data = sd_sample, aes(x = sd_eps, color = "sd_epsilon"), linetype = "dashed
```



Note that, if one is interested in hyperparameters, sampling is not always the best choice!

1 Model fitted values and predictions

An easy way to obtain fitted or predicted values is through the recently added compatibility with the broom-style `augment()` function, which extends the original data with posterior fitted values, or adds posterior summaries of the linear predictor to a prediction grid.

First, lets have a look at the expected counts $\mathbb{E}(\hat{\lambda}_{ijk}) = \exp \hat{\eta}_{ijk}$ using our existing data:

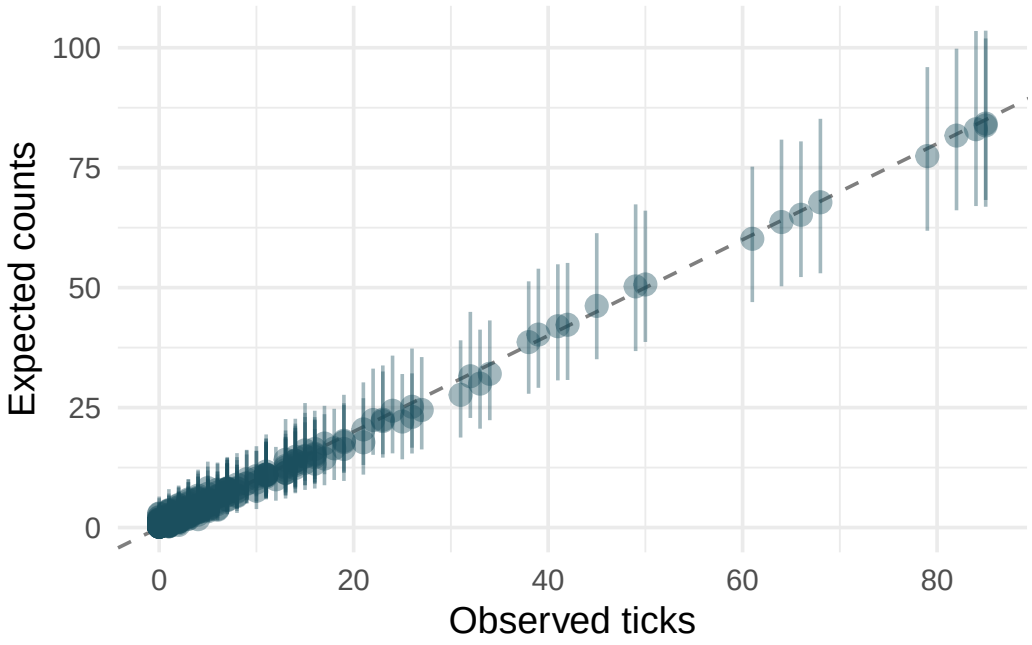
```
fitted_values <- augment(
  fit,
  data = grouseticks,
  pred_formula = ~ exp(year + height + brood_year + brood_year_chicken),
  n_samples = 500L,
  seed = 1L
)

head(fitted_values)
```

```
# A tibble: 6 x 13
  INDEX TICKS BROOD HEIGHT  YEAR LOCATION cHEIGHT    ij   ijk .fitted
  <int> <int> <int>  <int> <int>    <int>   <dbl> <int> <int>   <dbl>
1     1     0   501    465    95       32    2.76     1     1   0.701
2     2     0   501    465    95       32    2.76     1     2   0.682
3     3     0   502    472    95       36    9.76     2     3   0.753
4     4     0   503    475    95       37   12.8      3     4   1.07
5     5     0   503    475    95       37   12.8      3     5   1.13
6     6     3   503    475    95       37   12.8      3     6   1.90
# i 3 more variables: .fitted_low <dbl>, .fitted_high <dbl>, .fitted_sd <dbl>
```

Then we can plot the observed counts vs. the expected counts as follows:

```
ggplot(fitted_values, aes(x = TICKS, y = .fitted)) +
  geom_abline(slope = 1, intercept = 0, linetype = 2, colour = "grey50") +
  geom_pointrange(aes(ymin = .fitted_low, ymax = .fitted_high),
                  alpha = 0.4, colour = "#1B4F5E") +
  labs(x = "Observed ticks", y = "Expected counts") +
  theme_minimal(base_size = 14)
```



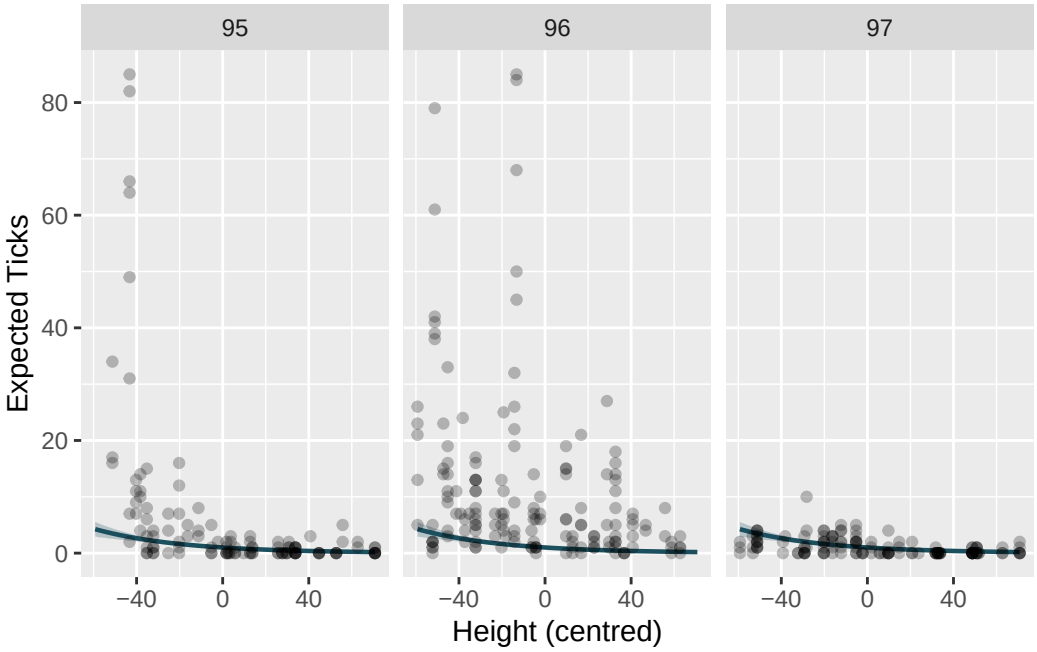
The `augment()` function in `inlabru` calls `predict()` (which in turn calls `generate()` to draw posterior samples) so we can use `predict()` directly to look at the conditional height effect per year. To compute model predictions we can create a **prediction grid** containing a range of values of the covariate (height) where we want the response to be predicted for each year. Then we simply call the `predict` function while specifying the model components.

```
# grid across the observed height range
yh_grid <- expand.grid(YEAR= c("95","96","97"),
  cHEIGHT = seq(min(grouseticks$cHEIGHT),
                max(grouseticks$cHEIGHT),
                length.out = 100))

# predict the height and year components only

pred_yh <- predict(fit, yh_grid, ~ exp(height+year))

ggplot(pred_yh, aes(cHEIGHT, mean)) +
  geom_ribbon(aes(ymin = q0.025, ymax = q0.975), alpha = 0.25, fill = "#1B4F5E") +
  geom_line(colour = "#1B4F5E", linewidth = 0.8) +
  labs(x = "Height (centred)", y = "Expected Ticks") +
  geom_point(data=grouseticks,aes(cHEIGHT,TICKS),alpha=0.25)+
  facet_wrap(~YEAR)
```



Task

Another useful quantity we can compute is the intraclass correlation coefficient (ICC) which help us determine how much the response varies within groups compared to between groups. The intraclass correlation coefficient is defined as:

$$\text{ICC} = \frac{\sigma_u^2}{\sigma_u^2 + \sigma_e^2}$$

Compute the mean, median, and quantiles for the ICC by drawing posterior samples for σ_e^2 and $\sigma_e^2 ps$ using the `predict` function.
Click here to see the solution

```
ICC <-  predict(fit,grouseticks, ~ {
   tau_e <- Precision_for_brood_year
   tau_eps <- Precision_for_brood_year_chicken
   sigma_e = 1/tau_e
   sigma_eps = 1/tau_eps
   list(ICC = sigma_e/ (sigma_e+sigma_eps))
   },
   n.samples = 1000
  )

ICC
```

```
$ICC
       mean         sd    q0.025      q0.5    q0.975    median mean.mc_std_err
1 0.7469051 0.04745802 0.6513059 0.7498761 0.8285273 0.7498761     0.001574834
  sd.mc_std_err
1   0.001171295

attr(,"class")
[1] "bru_prediction" "list"
```

2 Change prior distributions

Before trying to change the priors for the hyperparameters we can check which priors are actually used in the model

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

```
section=[family]
    tag=[INLA.Data1] component=[poisson]
section=[random]
    tag=[year] component=[year]
        group.theta1:
            parameter=[logit correlation]
            prior=[normal]
            param=[0.0, 0.2]
    tag=[] component=[]
    tag=[brood_year] component=[brood_year]
        theta1:
            parameter=[log precision]
            prior=[loggamma]
            param=[1e+00, 5e-05]
        group.theta1:
            parameter=[logit correlation]
            prior=[normal]
            param=[0.0, 0.2]
    tag=[brood_year_chicken] component=[brood_year_chicken]
        theta1:
            parameter=[log precision]
            prior=[loggamma]
            param=[1e+00, 5e-05]
        group.theta1:
            parameter=[logit correlation]
            prior=[normal]
            param=[0.0, 0.2]
section=[linear]
    tag=[height] component=[height]
        beta:
            parameter=[height]
            prior=[normal]
            param=[0.000, 0.001]
```

From the output we see that the precision for the linear effect of altitude is 0.001 (which means the sd is $1/\sqrt{0.001} = 31.62$)

The precisions for the random effects have a Gamma prior with parameters 1 and 5e-05.

2 Change 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 0.1 for we define the relative components as:

```
cmp= ~ -1 + year(YEAR, ... ) +
  height(cHEIGHT, model =  "linear", prec.linear = 0.1) + ...
```

Task

Run the model again using 0.1 as default precision for the altitude slope parameter and for the `year` fixed effect
Click here to see the solution

```
cmp2 =  ~ -1 + year(YEAR, model = "iid", initial = log(0.1), fixed = T) +
  height(cHEIGHT, model =  "linear", prec.linear = 0.1) +
  brood_year(ij , model = "iid") +
  brood_year_chicken(ijk, model= "iid")

fit2 = bru(cmp2, lik)
```

Note that we can use the same observation model as before since both the formula and the dataset are unchanged.

2 Change the precision for random effects

Priors on the hyperparameters of the random effects model must be passed by defining argument `hyper` within component of interest

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

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


cmp3 =  ~ -1 + year(YEAR, model = "iid", initial = log(0.1), fixed = T) +
  height(cHEIGHT, model =  "linear", prec.linear = 0.1) +
  brood_year(ij , model = "iid", hyper = prec.prior) +
  brood_year_chicken(ijk, model= "iid", hyper = prec.prior)


fit3 = bru(cmp3, lik)
```

Note that for `model = "linear"` the value for the precision is expressed in natural scale while for `model = "iid'` (and all other random effects) the value is expressed in log scale.