PRACTICAL 6

Aim of this practical:

In this practical we are going to look at some model comparison and validation techniques.

0 Model Checking for Linear Models

In this exercise we will:

- Learn about some model assessments techniques available in INLA
- Conduct posterior predictive model checking

Libraries to load:

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

Recall 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 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.

0.1.1 Simulate example data

We simulate data from a simple linear regression model

```
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 the linear regression model with `inlabru`

Now we fit a simple linear regression model in `inalbru` by defining (1) the model components, (2) the linear predictor and (3) the likelihood.

```
# Model components
cmp =  ~ -1 + beta_0(1) + beta_1(x, model = "linear")
# Linear predictor
formula = y ~ Intercept + beta_1
# Observational model likelihood
lik =  bru_obs(formula = y ~.,
            family = "gaussian",
            data = df)
# Fit the Model
fit.lm = bru(cmp, lik)
```

0.1.3 Residuals analysis

A common way for model diagnostics in regression analysis is by checking residual plots. In a Bayesian setting residuals can be defined in multiple ways depending on how you account for posterior uncertainty. Here, we will adopt a Bayesian approach by generating samples from the posterior distribution of the model parameters and then draw samples from the residuals defined as:

$$r_i = y_i - x_i^T \beta$$

We can use the `predict` function to achieve this:

```
res_samples <- predict(
  fit.lm,          # the fitted model
  df,              # the original data set
  ~ data.frame(
    res = y-(beta_0 + beta_1)  # compute the residuals
  ),
  n.samples = 1000   # draw 1000 samples
)
```

The resulting data frame contains the posterior draw of the residuals mean for which we can produce some diagnostics plots , e.g.

```
ggplot(res_samples,aes(y=mean,x=1:100))+geom_point() +
ggplot(res_samples,aes(y=mean,x=x))+geom_point()
```

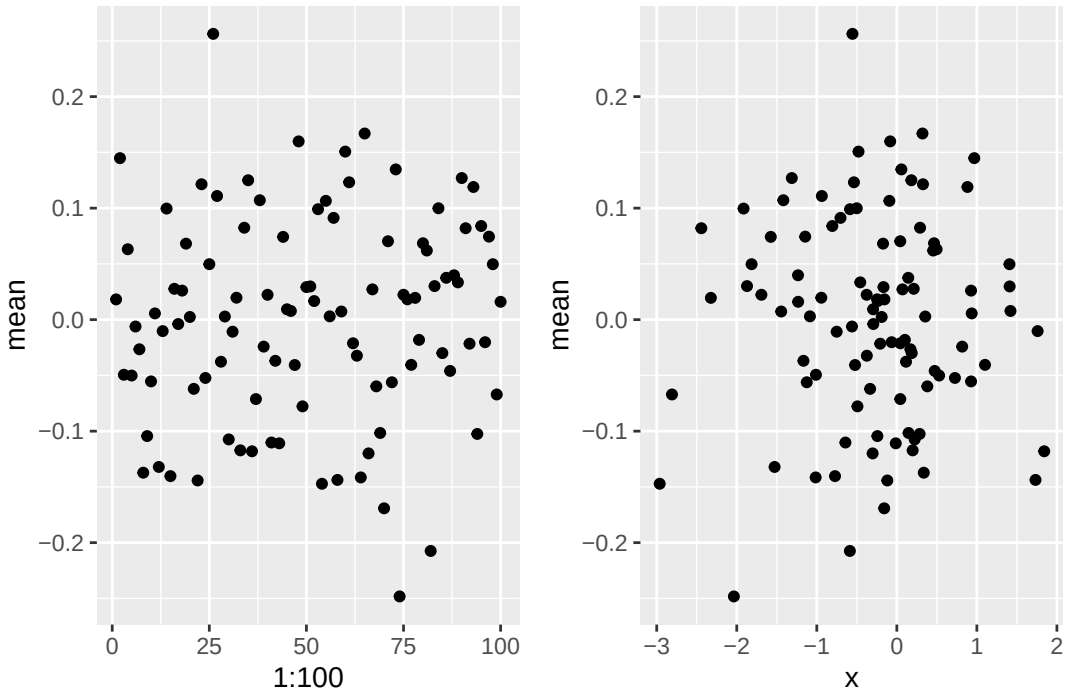
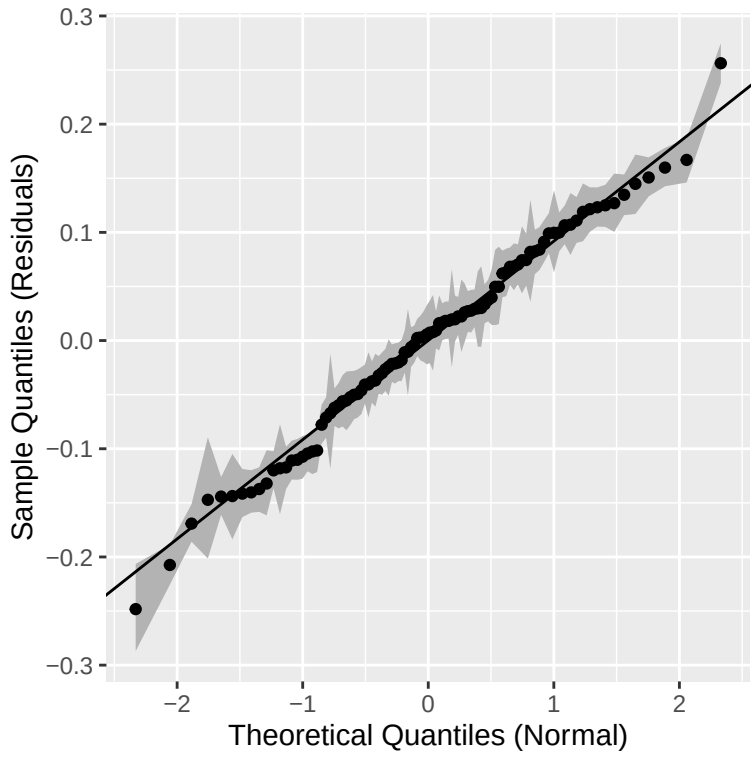


Figure 1: Bayesian residual plots: the left panel is the residual index plot; the right panel is the plot of the residual versus the covariate x

We can also compare these against the theoretical quantiles of the Normal distribution as follows:

```
arrange(res_samples, mean) %>%
  mutate(theortical_quantiles = qnorm(1:100 / (1+100))) %>%
  ggplot(aes(x=theortical_quantiles,y= mean)) +
  geom_ribbon(aes(ymin = q0.025, ymax = q0.975), fill = "grey70")+
  geom_abline(intercept = mean(res_samples$mean),
              slope = sd(res_samples$mean)) +
  geom_point() +
  labs(x = "Theoretical Quantiles (Normal)",
       y= "Sample Quantiles (Residuals)")
```



0.1.4 Posterior Predictive Checks

Now, instead of generating samples from the mean, we will account for the observational process uncertainty by:

1. Sampling $y_i^{1k} \sim \pi(y_i|\mathbf{y})\ k = 1, \dots, M;\ i = 1, \dots, 100$ using `generate()` (here we will draw $M = 500$ samples)

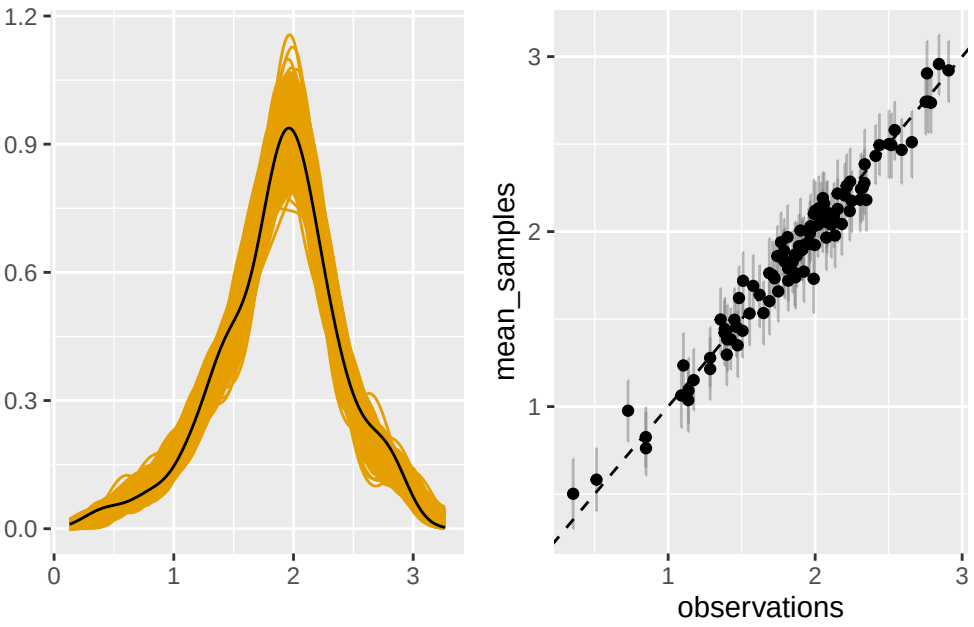
```
samples =  generate(fit.lm, df,
  formula = ~ {
    mu <- (beta_0 + beta_1)
    sd <- sqrt(1 / Precision_for_the_Gaussian_observations)
    rnorm(100, mean = mu, sd = sd)
  },
  n.samples = 500
)
```

2. Comparing some summaries of the simulated data with the one of the observed one

Here we compare (i) the estimated posterior densities $\widehat{\pi}^k(y|\mathbf{y})$ with the estimated data density and (ii) the samples means and 95% credible intervals against the observations.

```
# Tidy format for plotting
samples_long = data.frame(samples) %>%
  mutate(id = 1:100) %>% # i-th observation
  pivot_longer(-id)

# compute the mean and quantiles for the samples
draws_summaries = data.frame(mean_samples = apply(samples,1,mean),
q25 = apply(samples,1,function(x)quantile(x,0.025)),
q975 = apply(samples,1,function(x)quantile(x,0.975)),
```

```
observations = df$y)

p1 = ggplot() + geom_density(data = samples_long,
                             aes(value, group = name),  color = "#E69F00") +
  geom_density(data = df, aes(y))  +
  xlab("") + ylab("")

p2 = ggplot(draws_summaries,aes(y=mean_samples,x=observations))+
  geom_errorbar(aes(ymin = q25,
                    ymax = q975),
                alpha = 0.5, color = "grey50")+
geom_point()+geom_abline(slope = 1,intercept = 0,lty=2)+labs()

p1 +p2
```



0 GLM model checking

In this exercise we will:

- Learn about some model assessments techniques available in INLA
- Conduct posterior predictive model checking using CPO and PIT

Libraries to load:

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

In this exercise, we will use data on horseshoe crabs (*Limulus polyphemus*) where the number of satellites males surrounding a breeding female are counted along with the female's color and carapace width.

A possible model to study the factors that affect the number of satellites for female crabs is

$$y_i \sim \text{Poisson}(\mu_i), \qquad i = 1, \dots, N$$
$$\eta_i = \mu_i = \beta_0 + \beta_1 x_i + \dots$$

We can explore the conditional means and variances given the female's color:

```
crabs <- read.csv("datasets/crabs.csv")

# conditional means and variances
crabs %>%
  summarise( Mean = mean(satell ),
             Variance = var(satell),
                     .by = color)
```

```
   color     Mean  Variance
1 medium 3.294737 10.273908
2   dark 2.227273  6.737844
3  light 4.083333  9.719697
4 darker 2.045455 13.093074
```

The mean of the number of satellites vary by color which gives a good indication that color might be useful for predicting satellites numbers. However, notice that the mean is lower than its variance suggesting that overdispersion might be present and that a negative binomial model would be more appropriate for the data (we will cover this later).

Fitting the model

First, lets begin fitting the Poisson model above using the carapace's color and width as predictors. Since, color is a categorical variable in our model we need to create a dummy variable for it. We can use the `model.matrix` function to help us constructing the design matrix and then append this to our data:

```
crabs_df = model.matrix( ~  color , crabs) %>%
  as.data.frame() %>%
  select(-1) %>%         # drop intercept
  bind_cols(crabs) %>%  # append to original data
  select(-color)         # remove original color categorical variable
```

The new data set `crabs_df` contains a dummy variable for the different color categories (`dark` being the reference category). Then we can fit the model in `inlabru` as follows:

```
cmp =  ~ -1 + beta0(1) +  colordarker +
       colorlight + colormedium +
       w(weight, model = "linear")

lik =  bru_obs(formula = satell ~.,
            family = "poisson",
            data = crabs_df)

fit_pois = bru(cmp, lik)
```

```
summary(fit_pois)
```

```
inlabru version: 2.13.0.9011
INLA version: 25.09.19
Components:
Latent components:
beta0: main = linear(1)
colordarker: main = linear(colordarker)
colorlight: main = linear(colorlight)
colormedium: main = linear(colormedium)
w: main = linear(weight)
Observation models:
  Family: 'poisson'
    Tag: <No tag>
    Data class: 'data.frame'
    Response class: 'integer'
    Predictor: satell ~ .
    Additive/Linear: TRUE/TRUE
    Used components: effects[beta0, colordarker, colorlight, colormedium, w], latent[]
Time used:
    Pre = 0.848, Running = 0.189, Post = 0.00686, Total = 1.04
Fixed effects:
              mean    sd 0.025quant 0.5quant 0.975quant   mode kld
beta0       -0.501 0.196     -0.885   -0.501     -0.117 -0.501   0
colordarker -0.008 0.180     -0.362   -0.008      0.345 -0.008   0
colorlight   0.445 0.176      0.101    0.445      0.790  0.445   0
colormedium  0.248 0.118      0.017    0.248      0.479  0.248   0
w            0.001 0.000      0.000    0.001      0.001  0.001   0

Marginal log-Likelihood:  -489.43
 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)')
```

0.2.1 Model assessment and model choice

Now that we have fitted the model we would like to carry some model assessments. In a Bayesian setting, this is often based on posterior predictive checks. To do so, we will use the CPO and PIT - two commonly used Bayesian model assessment criteria based on the **posterior predictive distribution**.

> **Posterior predictive model checking**
>
> The posterior predictive distribution for a predicted value $\hat{y}$ is
>
> $$\pi(\hat{y}|\mathbf{y}) = \int_{\theta} \pi(\hat{y}|\theta)\pi(\theta|\mathbf{y})d\theta.$$
>
> The probability integral transform (PIT) introduced by Dawid (1984) is defined for each observation as:
>
> $$\text{PIT}_i = \pi(\hat{y}_i \leq y_i|\mathbf{y}_{-i})$$

The PIT evaluates how well a model's predicted values match the observed data distribution. It is computed as the cumulative distribution function (CDF) of the observed data evaluated at each predicted value. If the model is well-calibrated, the PIT values should be *approximately uniformly distributed.* Deviations from this uniform distribution may indicate issues with model calibration or overfitting.
Another metric we could used to asses the model fit is the conditional predictive ordinate (CPO) introduced by Pettit (1990), and defined as:

$$\text{CPO}_i = \pi(y_i|\mathbf{y}{-}i)$$

The CPO measures the density of the observed value of y_i when model is fit using all data but y_i. CPO provides a measure of how well the model predicts each individual observation while taking into account the rest of the data and the model. *Large values indicate a better fit* of the model to the data, while small values indicate a bad fitting of the model

To compute PIT and CPO we can either:

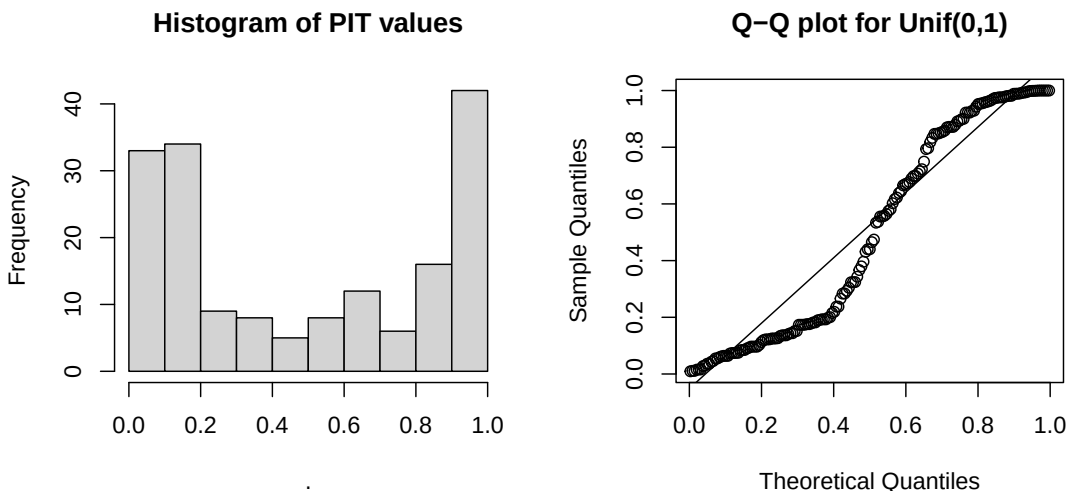
1. ask `inlabru` to compute them by set `options = list(control.compute = list(cpo = TRUE))` in the `bru()` function arguments.
2. set this as default in `inlabru` global option using the `bru_options_set` function.

Here we will do the later and re-run the model

```
bru_options_set(control.compute = list(cpo = TRUE))

fit_pois = bru(cmp, lik)
```

Now we can produce histograms and QQ plots to assess for uniformity in the PIT values which can be accessed through `inlabru_model$cpo$pit` :

1 Plot



.

2 R Code

```
fit_pois$cpo$pit %>%
  hist(main = "Histogram of PIT values")

qqplot(qunif(ppoints(length(fit_pois$cpo$pit))),
       fit_pois$cpo$pit,
       main = "Q-Q plot for Unif(0,1)",
       xlab = "Theoretical Quantiles",
       ylab = "Sample Quantiles")

qqline(fit_pois$cpo$pit,
       distribution = function(p) qunif(p),
       prob = c(0.1, 0.9))
```

Both Q-Q plots and histogram of the PIT values suggest a not so great model fit. For the CPO values, usually the following summary of the CPO is often used:

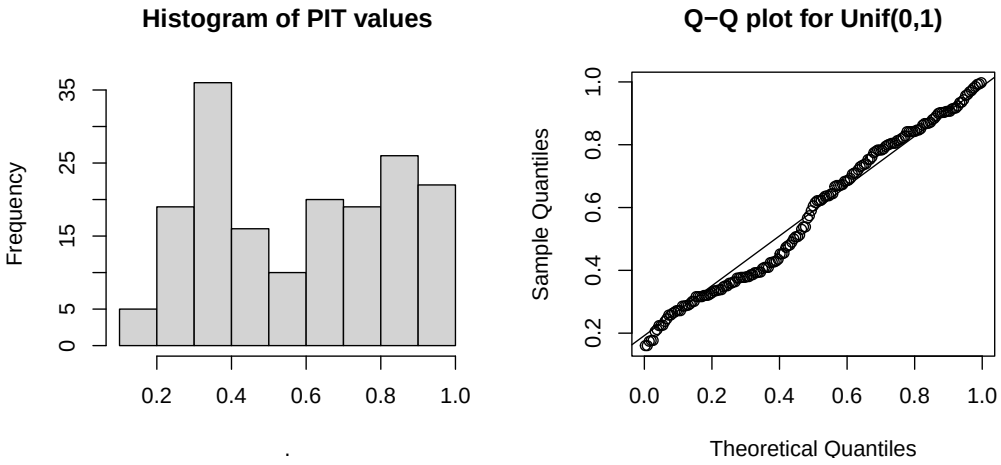
$$-\sum_{i=1}^{n} \log(\text{CPO\_}i)$$

This quantities is useful when comparing different models - a smaller values indicate a better model fit. CPO values can be accessed by typing `inlabru_model$cpo$cpo`.

Task

The model assessment above suggests that a Poisson model might not be the most appropriate model, likely due to the overdispersion we detected previously. Fit a Negative binomial to relax the Poisson model assumption that the conditional mean and variance are equal. Then, compute the CPO summary statistic and PIT QQ plot to decide which model gives the better fit.
Take hint
To specify a negative binomial model you only need to change the family distribution to `family = "nbinomial"`.
Click here to see the solution

```r
par(mfrow=c(1,2))

# Fit the negative binomial model

lik_nbinom =  bru_obs(formula = satell ~.,
            family = "nbinomial",
            data = crabs_df)

fit_nbinom = bru(cmp, lik_nbinom)

# PIT checks

fit_nbinom$cpo$pit %>%
  hist(main = "Histogram of PIT values")

qqplot(qunif(ppoints(length(fit_nbinom$cpo$pit))),
       fit_nbinom$cpo$pit,
       main = "Q-Q plot for Unif(0,1)",
       xlab = "Theoretical Quantiles",
       ylab = "Sample Quantiles")

qqline(fit_nbinom$cpo$pit,
       distribution = function(p) qunif(p),
       prob = c(0.1, 0.9))
```



.

```r
# CPO comparison

data.frame( CPO = c(-sum(log(fit_pois$cpo$cpo)),
                    -sum(log(fit_nbinom$cpo$cpo))),
            Model = c("Poisson","Negative Binomial"))
```

```
       CPO             Model
1 465.4061           Poisson
2 379.3340 Negative Binomial
```

```r
# Overall, we can see that the negative binomial model provides a better fit to the data.
```