Simulation study: Adjusting for covariate misclassification

```
library(ggplot2)
library(INLA)
library(inlamisclass)
```

```
# Number of iterations for importance sampling
niter <- 200000
```

In this file, we look at two simulated examples and a simulation study where the second example is simulated a number of times to examine the variability. This file takes several hours to run, due to the importance sampling. If you want to just compile the file using pre-computed results, you can set all the "run" parameters in the chunk above to "FALSE".

First example: Symmetric misclassification with independent exposure

In the simplest case, $\boldsymbol{x}$ is independent of any covariates and has a symmetrical misclassification matrix. In this example, we have $x_i \sim \text{Bernoulli}(0.5)$ for $1 \leq i \leq n$ with $n = 100$, and misclassification matrix

$$\mathsf{M} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$$

That means that the exposure model is

$$\text{logit}(E[\boldsymbol{x}]) = \mathbf{0} \ ,$$

and we simulate the main model of interest according to the model

$$\boldsymbol{y} = \mathbf{1} + \boldsymbol{x} + \boldsymbol{\epsilon} \ , \quad \boldsymbol{\epsilon} \sim N(\mathbf{0}, \mathbf{I}) \ .$$

In this example we will estimate β_0, β_x, τ_y .

The model for $\boldsymbol{x}$ ($\pi(\boldsymbol{x})$) is simple, and thanks to the symmetry of M and the fact that $P(x_i = 1) = P(x_i = 0)$, we have that $\pi(\boldsymbol{w} \mid \boldsymbol{x}) = \pi(\boldsymbol{x} \mid \boldsymbol{w})$. Therefore, it would be sufficient to sample $\boldsymbol{x}^{(k)}$ by adding misclassification error to $\boldsymbol{w}$ using the given misclassification matrix. This may seem counter-intuitive, but is a direct consequence of the simulation setup. The posterior distribution of $\boldsymbol{\theta}_y = (\beta_0, \beta_x)$ is obtained by

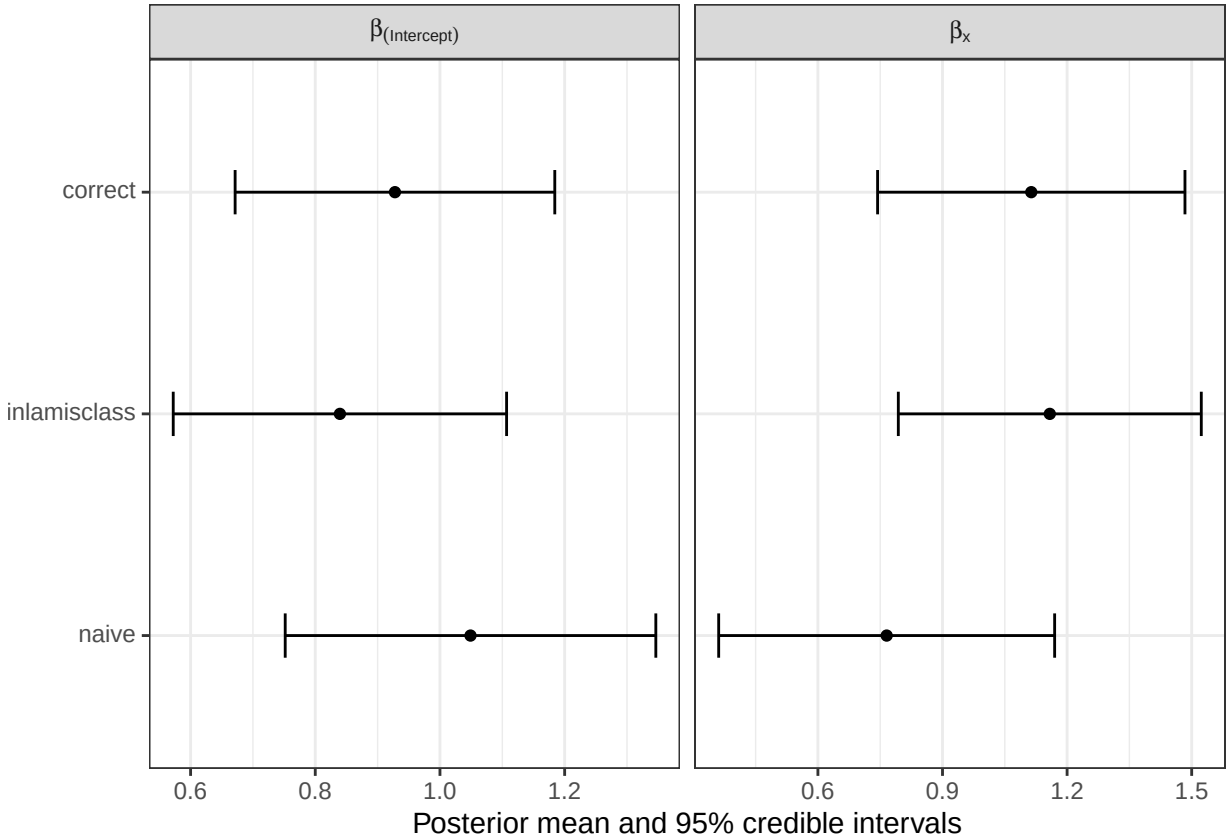
$$\pi(\boldsymbol{\theta}_y \mid \boldsymbol{y}, \boldsymbol{w}) \approx \sum_{k=1}^{K} \pi(\boldsymbol{\theta}_y \mid \boldsymbol{x}^{(k)}, \boldsymbol{y}) \cdot w_k$$

with $w_k = \pi(\boldsymbol{y} \mid \boldsymbol{x}^{(k)})$.

```
MC_matrix <- matrix(c(0.9, 0.1, 0.1, 0.9), nrow = 2, byrow = T)

set.seed(1)
data1 <- generate_misclassified(n = 100, p = 1, MC_matrix = MC_matrix,
                                betas = c(1, 1, 0),
                                alphas = c(0, 0))
sum(data1$x)/nrow(data1)
```

```
start_time <- Sys.time()
model1 <- inla_is_misclass(formula_moi = y ~ w,
                           formula_imp = w ~ 1,
                           alpha = 0,
                           MC_matrix = MC_matrix,
                           data = data1,
                           niter = niter)
end_time <- Sys.time()

saveRDS(list(model = model1,
             data = data1,
             runtime = end_time - start_time,
             niter = niter, nburnin = 0, rundate = Sys.time()),
        file = "results/simulated1.rds")
```

```
simulated1 <- readRDS("results/simulated1.rds")
naive1 <- inla(y ~ w, data = simulated1$data)
correct2 <- inla(y ~ x, data = simulated1$data)

plot_compare_inlamisclass(is_results = simulated1$model,
                          naive_mod = naive1,
                          correct_mod = correct2,
                          niter = simulated1$niter,
                          plot_intercept = TRUE)
```



In this example, we ran the importance sampling for 2×10^5 iterations, which took 7 hours.

Second example: Asymmetric misclassification and exposure depending on covariate

In a second example we used

$$\mathsf{M} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix} ,$$

and generated the true $\boldsymbol{x}$ with a dependency on an additional continuous (and error-free) covariate $\boldsymbol{z}$. First, each component of $\boldsymbol{z}$ was generated uniformly $z_i \sim \text{Unif}(-1, 1)$ for $1 \leq i \leq 200$, and then the $\boldsymbol{x}$ variables was sampled according to an exposure model given as

$$\text{logit}[E(\boldsymbol{x} \mid \boldsymbol{z})] = \alpha_0 \mathbf{1} + \alpha_z \boldsymbol{z} ,$$

with $\boldsymbol{\alpha}^\top = (\alpha_0, \alpha_z) = (-0.5, 0.25)$. This dependency was then also appropriately reflected in the analysis, assuming that $\boldsymbol{\alpha}$ was known. The response $\boldsymbol{y}$ was simulated according to the linear model

$$\boldsymbol{y} = \mathbf{1} + \boldsymbol{x} + \boldsymbol{z} + \boldsymbol{\epsilon} , \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) .$$

To sample from $\pi(\boldsymbol{x} \mid \boldsymbol{w}, \boldsymbol{z})$, we used that the components in $\boldsymbol{x}$ are independent, thus

$$\pi(\boldsymbol{x} \mid \boldsymbol{w}, \boldsymbol{z}) = \prod_{i=1}^{n} \pi(x_i \mid w_i, z_i) .$$

Each component can then be sampled from a Bernoulli distribution with success probability

$$\pi(x_i \mid w_i, z_i) = \frac{\pi(w_i \mid x_i) \cdot \pi(x_i \mid z_i)}{\sum_{j=0}^{1} \pi(w_i \mid x_i = j) \cdot \pi(x_i = j \mid z_i)} ,$$

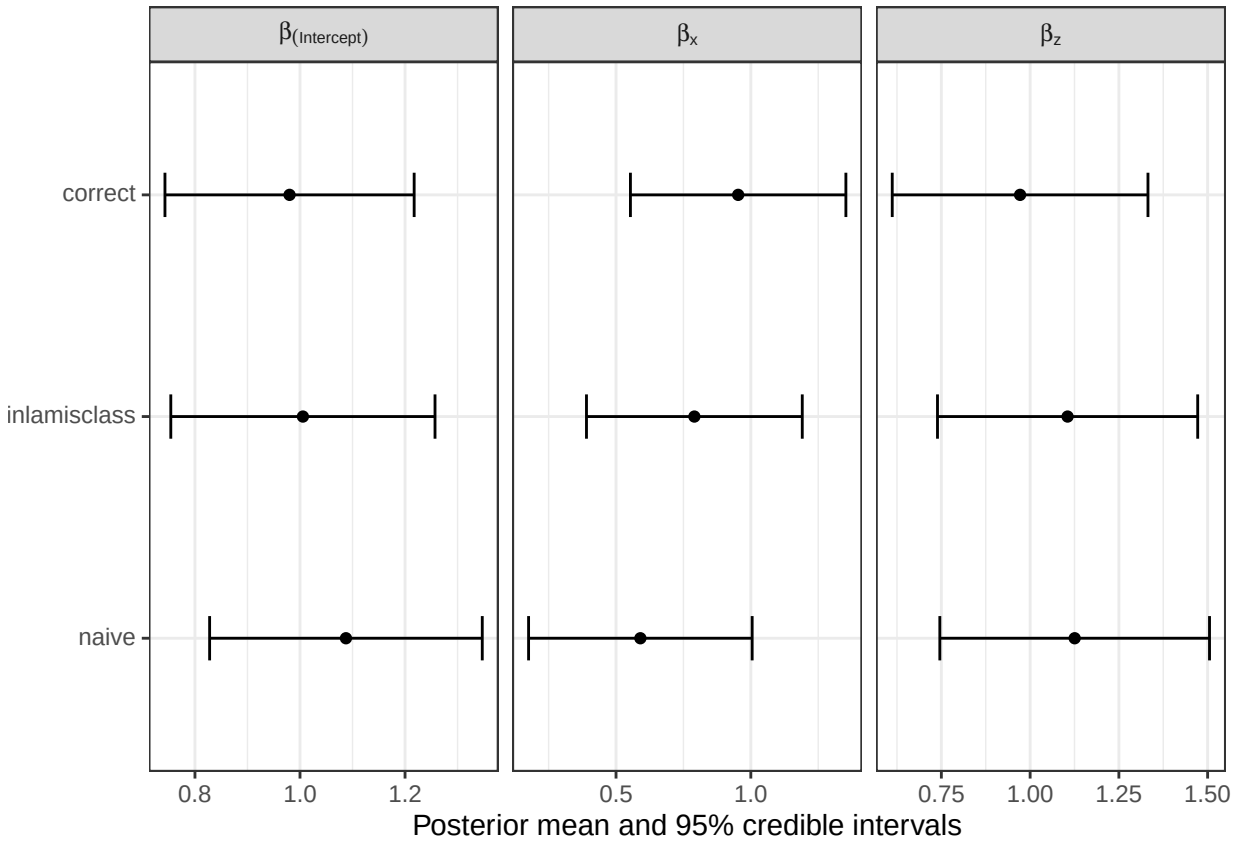
using the error model $\pi(w \mid x)$ as encoded in the MC matrix, and $\pi(x \mid z)$ from the exposure model.

The rest of the procedure is again the same as in the first example: For each iteration k, a sample $\boldsymbol{x}^{(k)}$ is employed to obtain the posterior distribution of the regression parameters $\pi(\boldsymbol{\theta}_y \mid \boldsymbol{x}^{(k)}, \boldsymbol{z}, \boldsymbol{y})$, which is weighted with the conditional marginal likelihood $\pi(\boldsymbol{y} \mid \boldsymbol{x}^{(k)}, \boldsymbol{z})$.

In this example we estimate $\beta_0, \beta_x, \beta_z, \tau_y$.

```
MC_matrix <- matrix(c(0.9, 0.1, 0.2, 0.8), nrow = 2, byrow = T)

set.seed(1)
data2 <- generate_misclassified(n = 100, p = 2, MC_matrix = MC_matrix,
                                betas = c(1, 1, 1),
                                alphas = c(-0.5, 0.25))

start_time <- Sys.time()
model2 <- inla_is_misclass(formula_moi = y ~ w + z,
                           formula_imp = w ~ z,
                           alpha = c(-0.5, 0.25),
                           MC_matrix = MC_matrix,
                           data = data2,
                           niter = niter)
end_time <- Sys.time()

saveRDS(list(model = model2,
             data = data2,
             runtime = end_time - start_time,
             niter = niter, nburnin = 0, rundate = Sys.time()),
        file = "results/simulated2.rds")
```

```
simulated2 <- readRDS("results/simulated2.rds")
naive2 <- inla(y ~ w + z, data = simulated2$data)
correct2 <- inla(y ~ x + z, data = simulated2$data)

plot_compare_inlamisclass(is_results = simulated2$model,
                          naive_mod = naive2,
                          correct_mod = correct2,
                          plot_intercept = TRUE, niter = simulated2$niter)
```



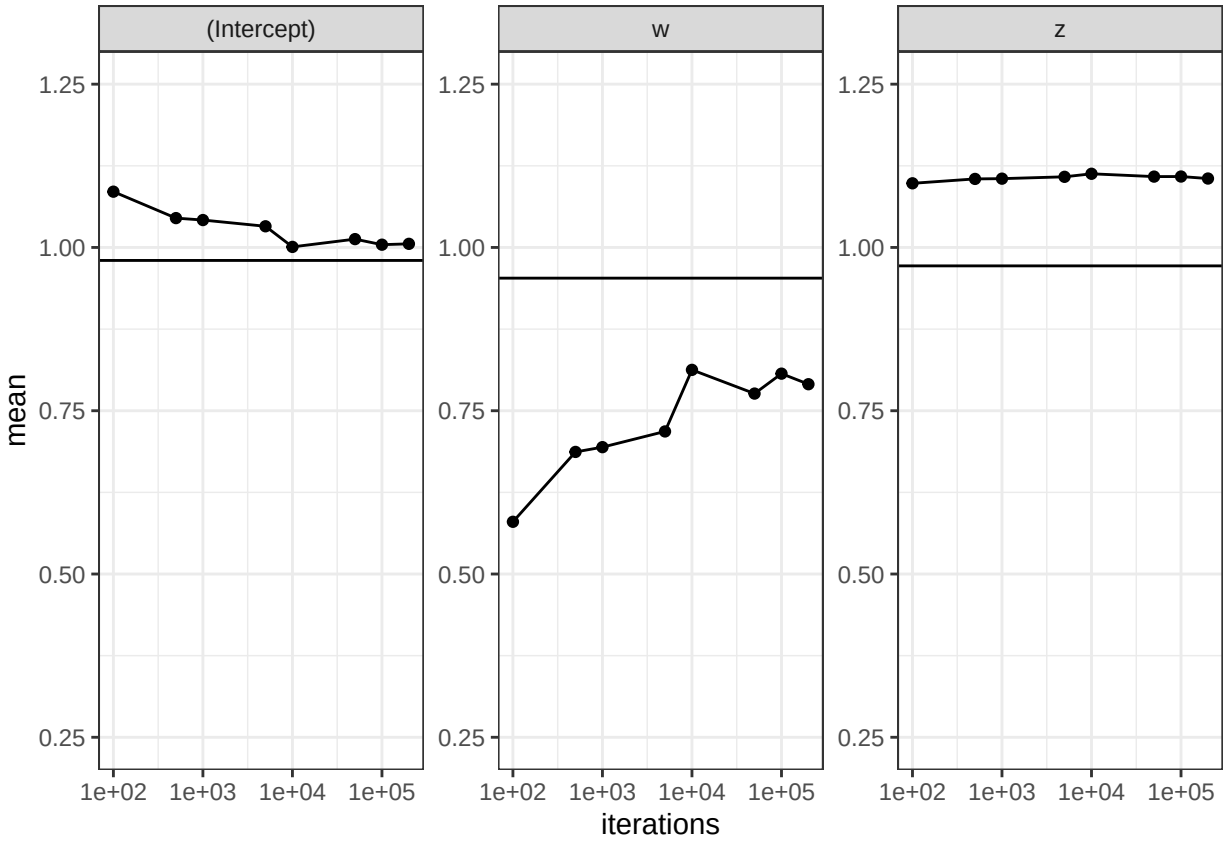
In this example, we ran the importance sampling for 2×10^5 iterations, which took 6.77 hours.

Examining how the estimate changes if we run the importance sampling longer

We are curious how the estimate changes if we use more samples for the importance sampling procedure. To examine this, we create several different subsets of the samples generated in the second example.

```
is_samples <- readRDS("results/simulated2.rds")

cutoffs <- c(100, 500, 1000, 5000, 10000, 50000, 100000, 200000)

subset_list <- list()
for(i in 1:length(cutoffs)){
  cut <- cutoffs[i]
  subset_list[[i]] <- lapply(1:cut, function(i) is_samples$model[[i]])
}
```

```r
summary_list <- lapply(subset_list, make_results_df)
moi_list <- lapply(summary_list, "[[", "moi")
moi_df <- dplyr::bind_rows(moi_list, .id = "iterations")
moi_df$iterations <- rep(cutoffs, each = 3)

correct <- inla(y ~ x + z, data = is_samples$data)
correct_mean <- data.frame(mean = correct$summary.fixed$mean)
correct_mean$variable <- c("(Intercept)", "w", "z")

ggplot(moi_df, aes(y = mean, x = iterations)) +
  geom_point() +
  geom_line() +
  geom_hline(data = correct_mean, aes(yintercept = mean)) +
  ylim(0.25,1.25) +
  scale_x_log10() +
  facet_wrap(~ variable, scales = "free") +
  theme_bw()
```



Note that the x-axis is on a log10 scale here. This shows that after 10000 iterations, the estimate seems to stabilize. Note that this may be specific to this particular case and might not generalize.

Repeating the simulation many times

When running the first two examples, we have noticed that the model sometimes does not seem to adjust completely for the misclassification, when different simulated datasets are used. To examine this further, we simulate 10 different datasets using the same simulation setup, and fit the model separately to each of these.

We use the simulation setup from the second example, that is:

MC matrix:

$$\mathsf{M} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix} ,$$

exposure model:

$$\text{logit}(E[\boldsymbol{x} \mid \boldsymbol{z}]) = -0.5 \cdot \mathbf{1} + 0.25\boldsymbol{z} ,$$

and model of interest:

$$\boldsymbol{y} = \mathbf{1} + \boldsymbol{x} + \boldsymbol{z} + \boldsymbol{\epsilon} , \quad \boldsymbol{\epsilon} \sim N(\mathbf{0}, \mathbf{I}) .$$

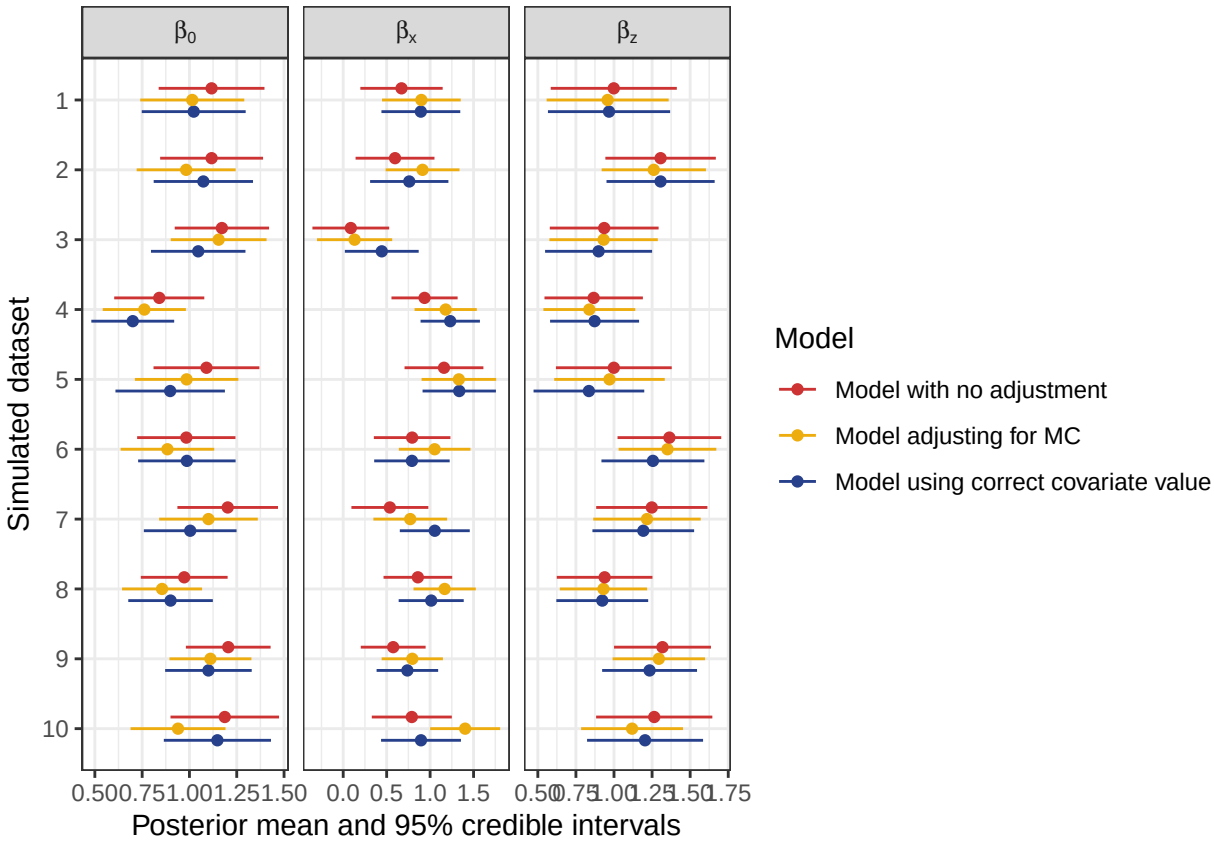
```r
n <- 100
n_runs <- 10
# Suffix giving number of iterations and sample size when saving data and models
name_append <- paste0("n", n, "_", "niter", niter)
```

```r
MC_matrix <- matrix(c(0.9, 0.1, 0.2, 0.8), nrow = 2, byrow = T)

all_runs <- list()

for(i in 1:n_runs){
  # Generate data
  data_mc <- generate_misclassified(n = n, p = 2, MC_matrix = MC_matrix,
                                    betas = c(1, 1, 1),
                                    alphas = c(-0.5, 0.25))

  # Check correct model
  correct_coef <- inla(y ~ x + z, data = data_mc)$summary.fixed
  correct_coef

  # attenuated version
  naive_coef <- inla(y ~ w + z, data = data_mc)$summary.fixed
  naive_coef

  inla_mod <- inla_is_misclass(formula_moi = y ~ w + z,
                               formula_imp = w ~ z,
                               alpha = c(-0.5, 0.25),
                               MC_matrix = MC_matrix,
                               data = data_mc,
                               niter = niter)

  # Extracting relevant stuff
  naive_summ <- data.frame(naive_coef[, c(1,2,3,5)])
  naive_summ$variable <- c("beta.0", "beta.x", "beta.z")

  correct_summ <- data.frame(correct_coef[, c(1,2,3,5)])
  correct_summ$variable <- c("beta.0", "beta.x", "beta.z")

  inla_summ <- make_results_df(inla_mod)$moi
  inla_summ$variable <- c("beta.0", "beta.x", "beta.z")
  colnames(inla_summ) <- c("variable", "mean", "X0.025quant", "X0.975quant")

  all_mods <- dplyr::bind_rows(naive = naive_summ,
                               inla_is = inla_summ,
                               correct = correct_summ,
```

```
                                .id = "model")

  all_mods$iteration <- as.factor(i)
  all_runs <- rbind(all_runs, all_mods)
}

saveRDS(all_runs, file = paste0("results/run_simulation_many_times_", name_append, ".rds"))
```

```
all_runs <- readRDS(paste0("results/run_simulation_many_times_", name_append, ".rds"))
#all_runs <- readRDS(paste0("results/run_simulation_many_times_n100_niter150000.rds"))
all_runs$Model <- factor(all_runs$model, levels = c("naive", "inla_is", "correct"))
all_runs$Model <- plyr::revalue(all_runs$Model,
                                c("naive" = "Model with no adjustment",
                                  "inla_is" = "Model adjusting for MC",
                                  "correct" = "Model using correct covariate value"))
all_runs$labels <- paste0(gsub("\\.", "[", all_runs$variable), "]")
colors <- c("brown3", "darkgoldenrod2", "royalblue4")
```

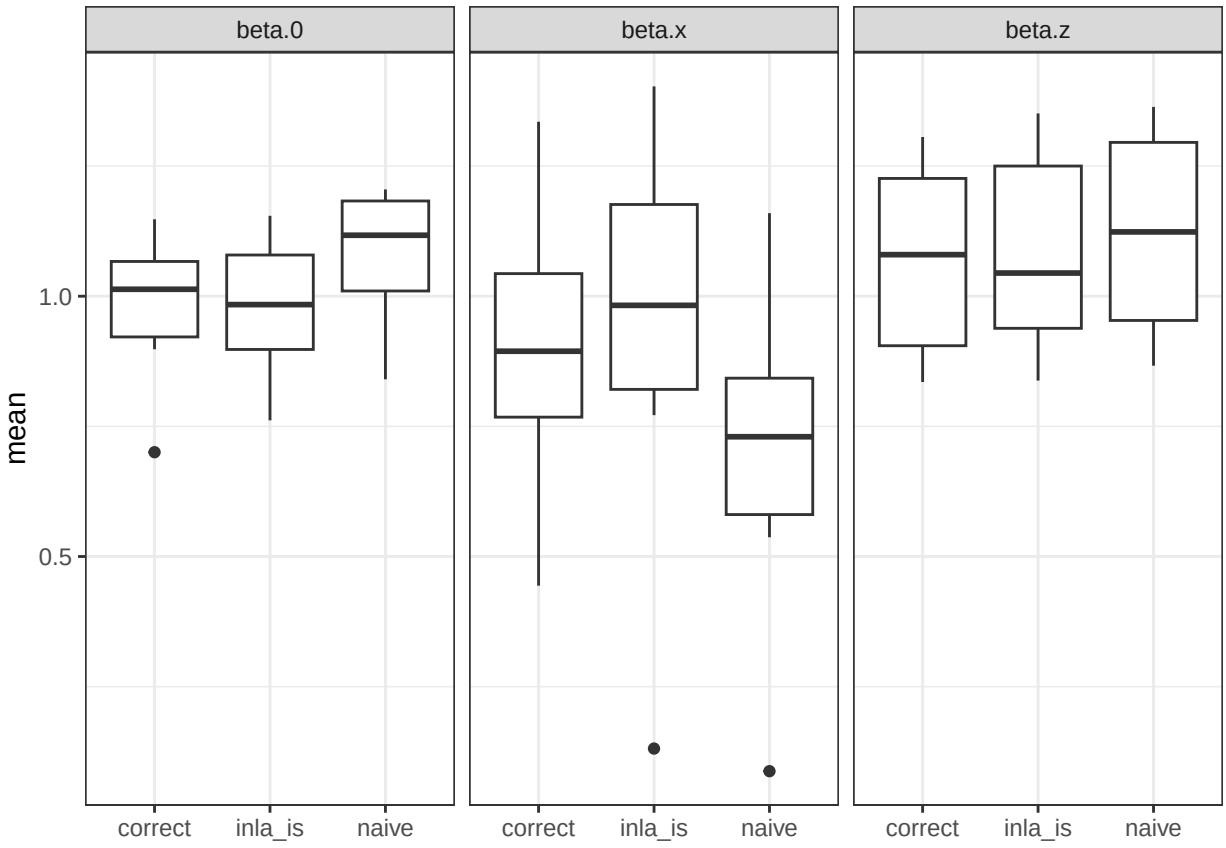
```
ggplot(all_runs, aes(x = mean, y = iteration, color = Model)) +
  geom_point(position = position_dodge2(width = 0.5, reverse = TRUE)) +
  geom_linerange(aes(xmin = X0.025quant, xmax = X0.975quant),
                 position = position_dodge2(width = 0.5, reverse = TRUE)) +
  scale_y_discrete(limits = rev) +
  scale_color_manual(values = colors) +
  facet_grid(cols = vars(labels), scales = "free", labeller = label_parsed) +
  xlab("Posterior mean and 95% credible intervals") +
  ylab("Simulated dataset") +
  theme_bw()
```




```
ggsave(paste0("figures/all_realizations_simulated_", name_append, ".pdf"))
```

```
## Saving 6.5 x 4.5 in image
```

```
ggplot(dplyr::filter(all_runs, !(variable %in% c("alpha.0", "alpha.z"))),
       aes(y = mean, x = model)) +
  geom_boxplot() +
  facet_grid(cols = vars(variable), scales = "free") +
  theme_bw() +
  theme(axis.title.x = element_blank())
```




```
ggsave(paste0("figures/boxplots_simulated", name_append, ".pdf"))
```

```
## Saving 6.5 x 4.5 in image
```

```
## Time difference of 1.279532 mins
```

The code in this file took 1.27953190008799 hours to run.