Supporting information: "Bayesian models for missing and misclassified variables using integrated nested Laplace approximations"

Emma Skarstein, Leonardo Bastos, Håvard Rue, Stefanie Muff

This supporting information contains all the PDF-documents containing the code to reproduce the results in the examples in the paper, along with some additional examples. The subsections are:

- Simulation study: Adjusting for covariate misclassification: Two examples of covariate misclassification in linear regression, along with a simulation study repeating the second example. Described in Section 5.1.
- Covariate misclassification from a latent Gaussian variable with measurement error: A simulated example of how dichotomizing a continuous variable with error leads to misclassification, presented in Section 5.2.
- Missing observations in a binary covariate: A simulated example with missing observations, presented in Section 5.3.
- Adjusting for response misclassification with the sslogit link: A simulated example of how to deal with response misclassification in INLA, described in Section 5.4.
- Birth weight analysis: The birthweight example presented in Section 6.1
- Cervical cancer and herpes: The cervical cancer case-control study example presented in Section 6.2

Some of the files use functions and data sets from the R package `inlamisclass`, which is available to download from GitHub:

```
devtools::install_github("emmaSkarstein/inlamisclass")
```

or see https://github.com/emmaSkarstein/inlamisclass.

Simulation study: Adjusting for covariate misclassification

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

```
niter <- 150000
```

In this file, we look at two simulated examples and a simultation study where the second example is simulated a number of times to examine the variability.

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

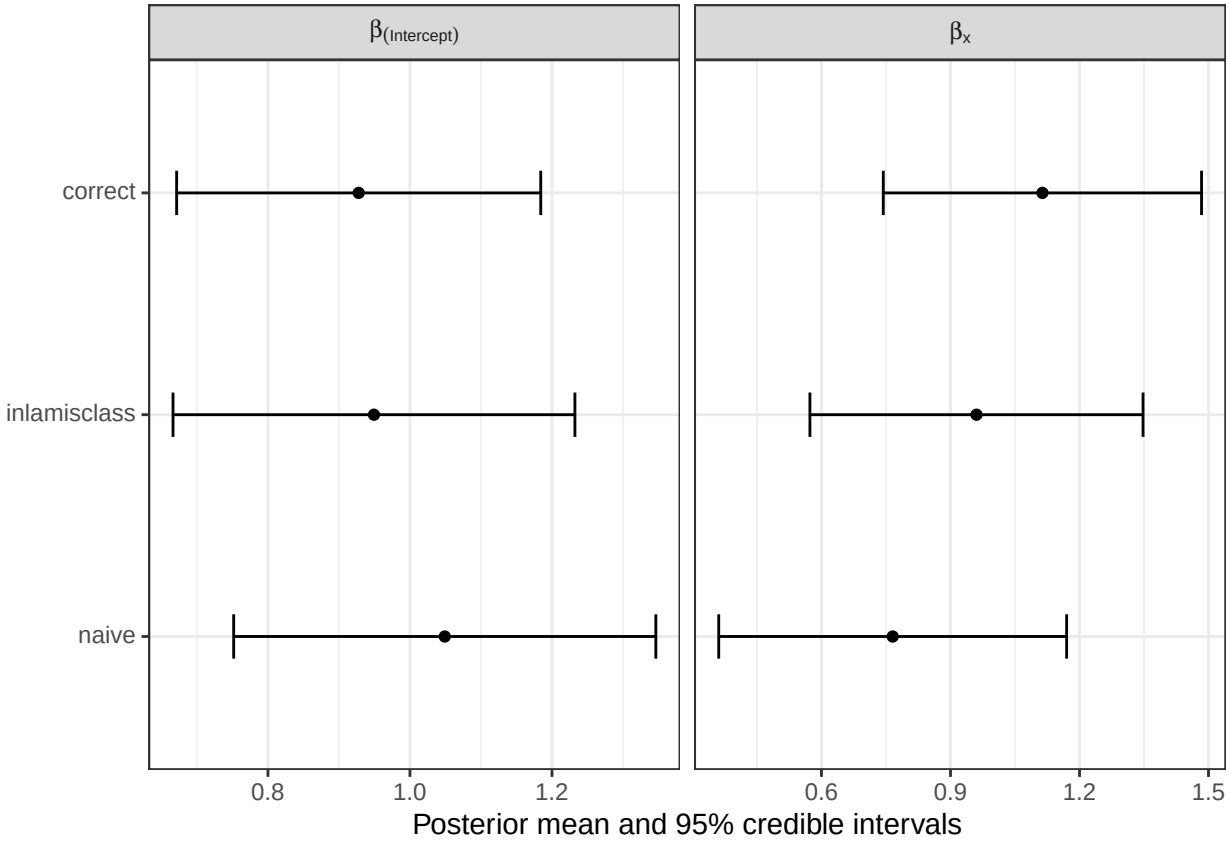
$$\pi(\boldsymbol{\theta}_y \mid \boldsymbol{y}, \boldsymbol{w}) \approx \sum_{k=1}^{K} \pi(\boldsymbol{\theta}_y \mid \boldsymbol{x}^{(k)}, \boldsymbol{y}) \cdot w_k$$

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(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 = "code/results/simulated1.rds")
```

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

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



In this example, we ran the importance sampling for 10^5 iterations, which took 5 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

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

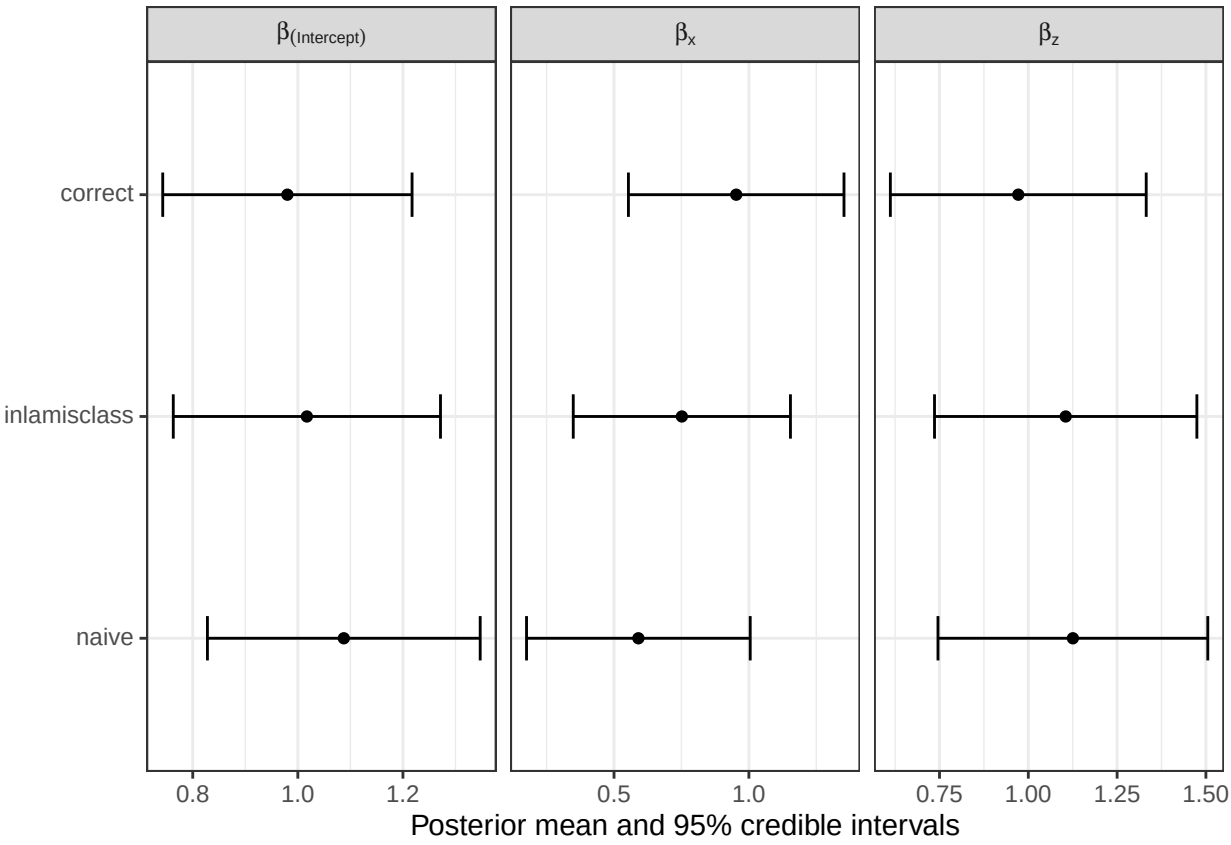
using the error model $\pi(w \mid x)$ as encoded in the MC matrix, and $\pi(x \mid z)$ from the exposure model.

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(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 = "code/results/simulated2.rds")
```

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

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



In this example, we ran the importance sampling for 10^5 iterations, which took 4.82 hours.

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:

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

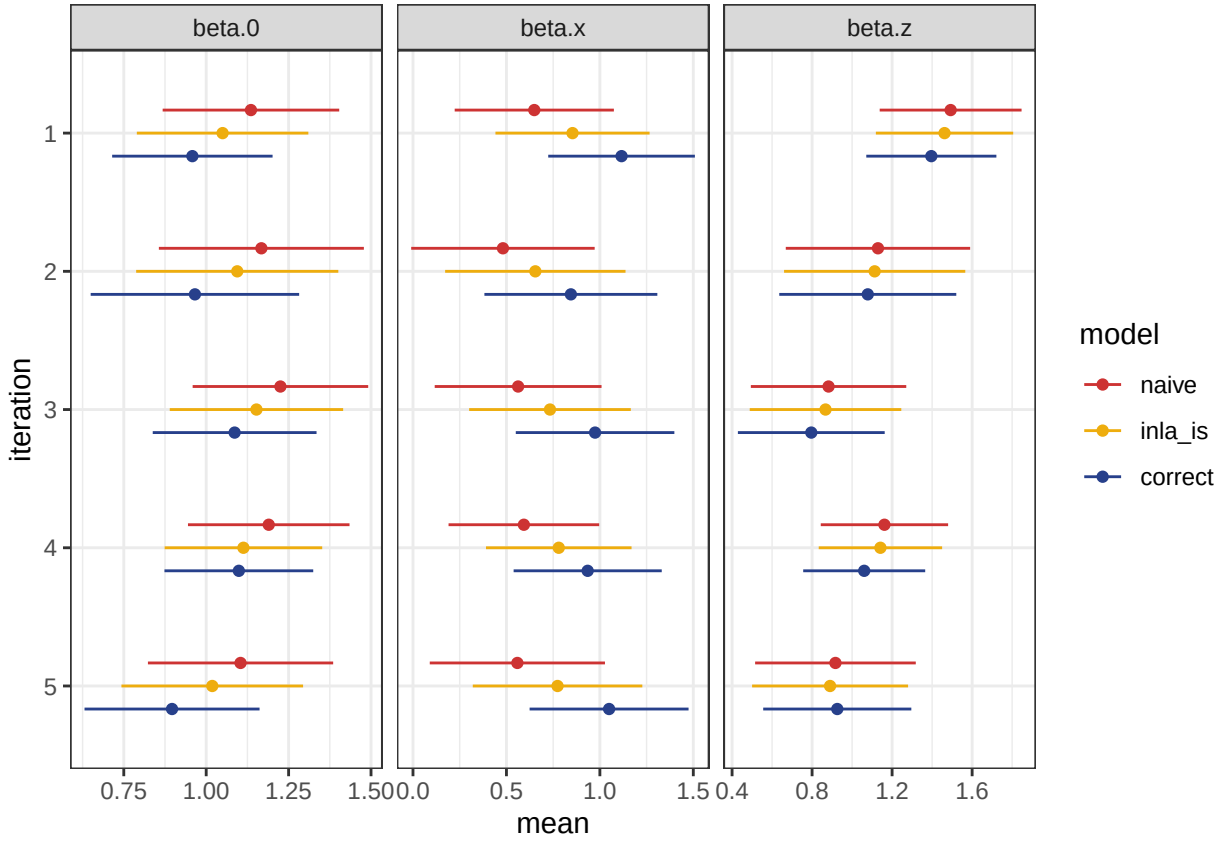
```r
n <- 100
n_runs <- 5
# Suffix giving number of iterations and sample size when saving data and models
name_append <- paste0("n", n, "_", "niter", niter)
```

```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(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("code/results/run_simulation_many_times_", name_append, ".rds"))
```

```r
all_runs <- readRDS(paste0("code/results/run_simulation_many_times_", name_append, ".rds"))
all_runs$model <- factor(all_runs$model, levels = c("naive", "inla_is", "correct"))
```

```
colors <- c("brown3", "darkgoldenrod2", "royalblue4")
```

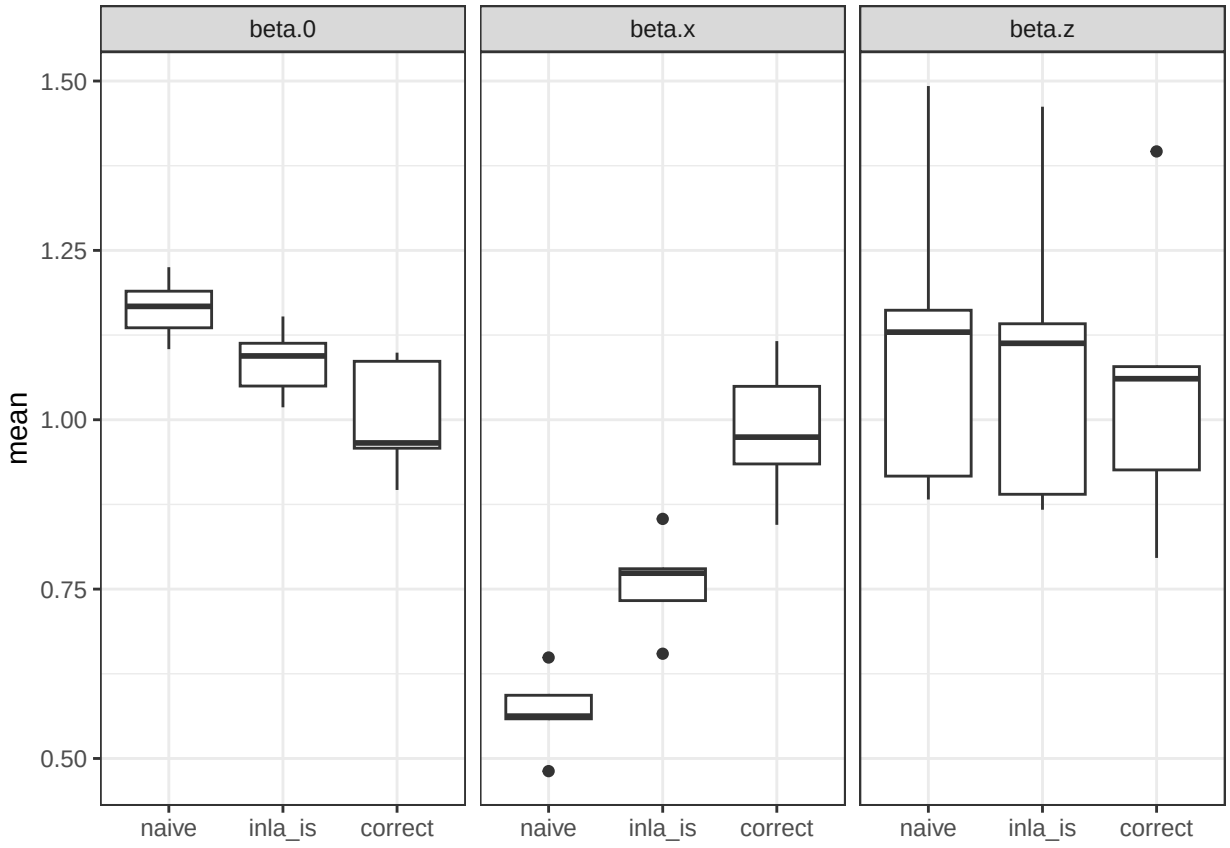
```
ggplot(all_runs, aes(x = mean, y = iteration, color = model)) +
  geom_point(position = position_dodge2(width = 0.5, reverse = TRUE)) +
  geom_linerange(aes(xmin = X0.025quant, xmax = X0.975quant),
                 position = position_dodge2(width = 0.5, reverse = TRUE)) +
  scale_y_discrete(limits = rev) +
  scale_color_manual(values = colors) +
  facet_grid(cols = vars(variable), scales = "free") +
  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
```

Missing observations in a binary covariate

Covariate misclassification from a latent Gaussian variable with measurement error

```
library(INLA)
library(dplyr)
library(ggplot2)
```

Models adjusting for misclassification arising from a dichotomized variable with error

In this example, we simulate a continuous latent variable with measurement error, which is then dichotomized.

```
set.seed(1)
n <- 200

x_c <- rnorm(n, 0, 1) # Continuous, not observed
x_d <- ifelse(x_c > 0, 1, 0) # Dichotomized, not observed
w_c <- x_c + rnorm(n) # Continuous with cont. ME, may be observed
w_d <- ifelse(w_c > 0, 1, 0) # Dichotomized, observed
```

From the variables we have simulated, we can generate a sample misclassification matrix:

```
pix0 <- sum(x_d==0)
pix1 <- sum(x_d==1)
table(unobserved = x_d, observed = w_d)/matrix(c(pix0, pix0, pix1, pix1),
                                               byrow = TRUE, nrow = 2)
```

```
##           observed
## unobserved         0         1
##          0 0.6603774 0.3396226
##          1 0.2872340 0.7127660
```

Lastly, we generate the response.

```
y <- 1 + x_c + rnorm(n)
```

Now, the observed data is `y` and `w_d` (often, researchers choose to use a dichotomized variable even though the continuous version is available, so perhaps `w_c` is also observed, but simply ignored, or it may have been available at some point but was not sent to the statistician). For the purposes of this example, we assume that we are interested in the relationship between `y` and `x_c`.

As described in Section 3.2.2, we then have the model

$$
\begin{aligned}
\boldsymbol{y} &= \beta_0 \mathbf{1} + \beta_{x_c} \boldsymbol{x}_c + \boldsymbol{Z}^\top \boldsymbol{\beta} + \boldsymbol{\varepsilon} \ , & \boldsymbol{\varepsilon} &\sim \mathcal{N}(\mathbf{0}, \sigma_\varepsilon^2 \boldsymbol{I}) \ , \\
\boldsymbol{w}_c &= \boldsymbol{x}_c + \boldsymbol{u}_{w_c} \ , & \boldsymbol{u}_{w_c} &\sim \mathcal{N}(\mathbf{0}, \sigma_{w_c}^2 \boldsymbol{I}) \ , \\
\boldsymbol{w}_d &= \mathrm{I}(\boldsymbol{w}_c > 0) \ , & & \\
\boldsymbol{x}_c &= \alpha_0 \mathbf{1} + \widetilde{\boldsymbol{Z}}^\top \boldsymbol{\alpha} + \boldsymbol{\varepsilon}_{x_c} \ , & \boldsymbol{\varepsilon}_{x_c} &\sim \mathcal{N}(\mathbf{0}, \sigma_{x_c}^2 \boldsymbol{I}) \ ,
\end{aligned}
$$

where $\boldsymbol{Z}$ and $\widetilde{\boldsymbol{Z}}$ are both covariate matrices with whichever covariates are relevant to include, these are

assumed to be without error. Now, we have that

$$\begin{aligned}\Pr(w_{d,i} = 1 \mid w_{c,i}) &= \Pr(w_{c,i} > 0) \\ &= \Pr(x_{c,i} + u_{w_c,i} > 0) \\ &= \Pr\left(\frac{u_{w_c,i}}{\sigma_{w_c}} < \frac{x_{c,i}}{\sigma_{w_c}}\right) \\ &= \Phi\left(\frac{x_{c,i}}{\sigma_{w_c}}\right) \ ,\end{aligned}$$

since $u_{w_c,i}/\sigma_{w_c} \sim \mathcal{N}(0,1)$. This relation means that we can model $\boldsymbol{w}_d$ as a Bernoulli variable with a probit link function. We can re-write the hierarchical model from above:

$$\begin{aligned}\boldsymbol{y} &= \beta_0\mathbf{1} + \beta_{x_c}\boldsymbol{x}_c + \boldsymbol{Z}^\top\boldsymbol{\beta} + \boldsymbol{\varepsilon} \ , \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2_\varepsilon \boldsymbol{I}) \ , \\ \boldsymbol{w}_d &\sim \text{Bernoulli}(\Phi(\boldsymbol{x_c}/\sigma_{w_c})) \ , \\ \boldsymbol{x}_c &= \alpha_0\mathbf{1} + \widetilde{\boldsymbol{Z}}^\top\boldsymbol{\alpha} + \boldsymbol{\varepsilon}_{x_c} \ , \qquad \boldsymbol{\varepsilon}_{x_c} \sim \mathcal{N}(\mathbf{0}, \sigma^2_{x_c} \boldsymbol{I}) \ ,\end{aligned}$$

We use stacks in R-INLA to structure the hierarchical model:

```
stk_y <- inla.stack(data = list(y = y),
                    A = list(1),
                    effects = list(
                      list(beta.0 = rep(1, n),
                           beta.x_c = 1:n)),
                    tag = "y")

stk_w_d <- inla.stack(data = list(w_d = w_d),
                    A = list(1),
                    effects = list(
                      list(id.x_c = 1:n,
                           weight.x_c = 1)),
                    tag = "w_d")

stk_x_c <- inla.stack(data = list(x_c = rep(0, n)),
                    A = list(1),
                    effects = list(
                      list(id.x_c = 1:n,
                           weight.x_c = -1,
                           alpha.0 = rep(1, n))),
                    tag = "x_c")

stk_full <- inla.stack(stk_y, stk_w_d, stk_x_c)

formula <- list(y, w_d, x_c) ~ -1 + beta.0 + alpha.0 +
  f(beta.x_c, copy = "id.x_c",
    hyper = list(beta = list(param = c(0, 1/1000), fixed = FALSE))) +
  f(id.x_c, weight.x_c, model = "iid", values = 1:n,
    hyper = list(prec = list(initial = -15, fixed = TRUE)))

res <- inla(formula, data = inla.stack.data(stk_full), Ntrials = rep(1, n),
            family = c("gaussian", "binomial", "gaussian"),
            control.family = list(
              list(hyper = list(prec = list(initial = log(1),
                                            param = c(10, 9),
                                            fixed = FALSE))),
```

```
              list(link = "probit"),
              list(hyper = list(prec = list(initial = log(1),
                                            param = c(10, 9),
                                            fixed = FALSE)))
            ),
            control.predictor = list(compute = TRUE))
```

Looking at the results

```
data <- data.frame(y = y, x_c = x_c, w_c = w_c, w_d = w_d, x_d = x_d)
```

Results from using correct continuous variable:

```
correct_model <- inla(y ~ x_c, data = data)
correct_model$summary.fixed
```

```
##                   mean         sd 0.025quant  0.5quant 0.975quant      mode
## (Intercept) 0.9553833 0.07565846  0.8068701 0.9553833   1.103896 0.9553833
## x_c         1.0785024 0.08157634  0.9183726 1.0785024   1.238632 1.0785024
##                       kld
## (Intercept) 1.943440e-09
## x_c         1.943381e-09
```

Results from using dichotomized version but continuous latent variable (our proposed model):

```
res$summary.fixed["beta.0",]
```

```
##             mean        sd 0.025quant  0.5quant 0.975quant      mode
## beta.0 0.9374249 0.1386488  0.6599709 0.9387588    1.20726 0.9387339
##                 kld
## beta.0 7.089973e-08
```

```
res$summary.hyperpar["Beta for beta.x_c",]
```

```
##                        mean       sd 0.025quant 0.5quant 0.975quant    mode
## Beta for beta.x_c 1.157823 0.296819  0.5858487 1.153674   1.754414 1.13589
```

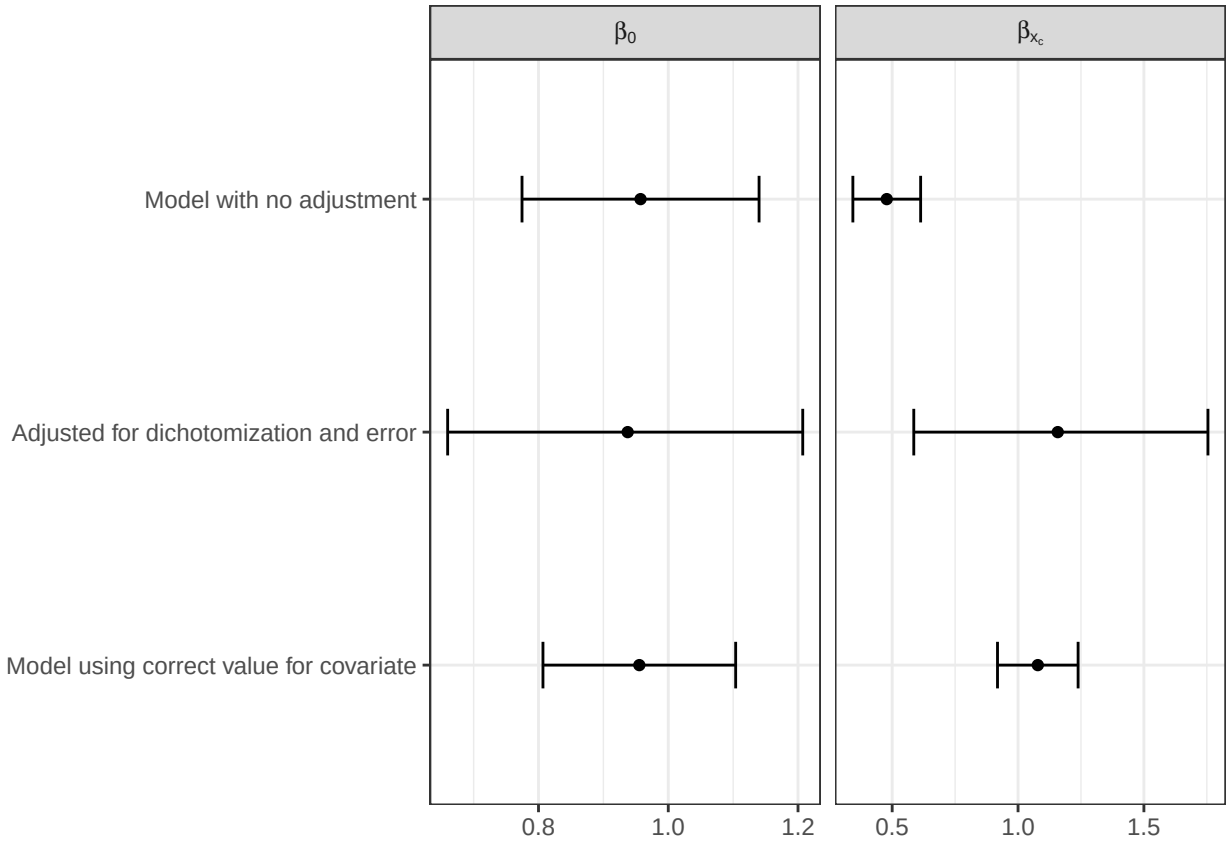
Results from using continuous variable with measurement error with no correction:

```
naive_model <- inla(y ~ w_c, data = data)
naive_model$summary.fixed
```

```
##                   mean         sd 0.025quant  0.5quant 0.975quant      mode
## (Intercept) 0.9572666 0.09306920  0.7745771 0.9572666  1.1399560 0.9572666
## w_c         0.4784398 0.06858374  0.3438138 0.4784398  0.6130657 0.4784398
##                       kld
## (Intercept) 1.942837e-09
## w_c         1.942792e-09
```

```
adjusted_results <- rbind("(Intercept)" = res$summary.fixed["beta.0",1:5],
                          "x_c" = res$summary.hyperpar["Beta for beta.x_c",1:5])
adjusted_results$variable <- rownames(adjusted_results)

naive_results <- naive_model$summary.fixed
rownames(naive_results) <- c("(Intercept)", "x_c")
naive_results$variable <- rownames(naive_results)
```

```r
correct_results <- correct_model$summary.fixed
correct_results$variable <- rownames(correct_results)

all_res <- dplyr::bind_rows(adjusted = adjusted_results,
                            naive = naive_results,
                            correct = correct_results, .id = "Model")

all_res$labels <- paste0("beta", "[", c(0, "x[c]"), "]")
all_res$Model <- factor(all_res$Model, levels = c("naive", "adjusted", "correct"))
all_res$Model <- plyr::revalue(all_res$Model,
                               c("naive" = "Model with no adjustment",
                                 "adjusted" = "Adjusted for dichotomization and error",
                                 "correct" = "Model using correct value for covariate"))

ggplot(all_res, aes(y = Model)) +
  geom_point(aes(x = mean)) +
  geom_errorbarh(aes(xmin = .data$"0.025quant", xmax = .data$"0.975quant"), height = .2) +
  scale_y_discrete(limits = rev) +
  facet_wrap(vars(labels), scales = "free_x", labeller = label_parsed) +
  theme_bw() +
  theme(axis.title = element_blank())
```




```r
ggsave("figures/cont_to_binary.pdf", height = 2.3, width = 7)
```

How do different measurement error variances correspond to different misclassification probabilities?

Using simulated data as well as the exact sensitivity and specificity calculated given our simulation setup, we can take a look at how the sensitivity and specificity of the classification of the continuous variable change for increasing measurement error variance.

First, we simulate data for different levels of measurement error:

```
simulate_dichotomized <- function(me_variance = 1, n = 200){
  x_c <- rnorm(n, 0, 1) # Continuous, not observed
  x_d <- ifelse(x_c > 0, 1, 0) # Dichotomized, not observed
  w_c <- x_c + rnorm(n, 0, sqrt(me_variance)) # Continuous with cont. ME, may be observed
  w_d <- ifelse(w_c > 0, 1, 0) # Dichotomized, observed

  return(list(x_d = x_d, w_d = w_d))
}

calculate_mc <- function(x_d, w_d){
  pix0 <- sum(x_d==0)
  pix1 <- sum(x_d==1)
  mc_tab <- table(unobserved = x_d, observed = w_d)/
    matrix(c(pix0, pix0, pix1, pix1), byrow = TRUE, nrow = 2)
  return(list(spec = mc_tab[1,1], sens = mc_tab[2,2]))
}

me_variances <- seq(0.01, 2, by = 0.05)
result_df <- data.frame(me_var = NA, sens = NA, spec = NA)
for(i in 1:length(me_variances)){
  var_list <- simulate_dichotomized(me_variance = me_variances[i])
  sens_spec <- calculate_mc(var_list$x_d, var_list$w_d)
  result_df[i,] <- c(me_variances[i], sens_spec$sens, sens_spec$spec)
}
```

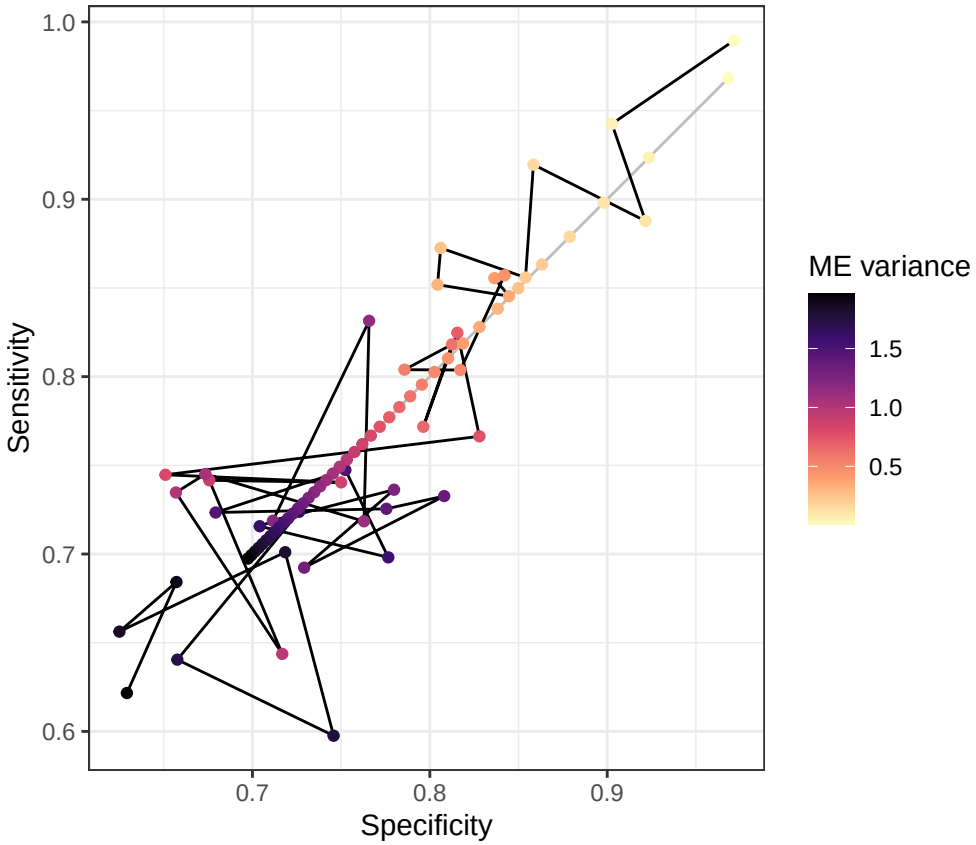
Next, we also calculate the expected sensitivity and specificity for different values of measurement error variance given the simulation setup:

```
sens_spec_exact <- function(sigma_u2 = 1){
  mu_x <- 0
  mu_v <- mu_x
  sigma_x2 <- 1
  sigma_v2 <- sigma_x2 + sigma_u2
  cov_v_x <- sigma_x2+ 2*mu_x^2
  sigma_mat <- matrix(c(sigma_v2, cov_v_x, sigma_x2, cov_v_x), byrow = TRUE, nrow = 2)

  # P(V>0, X>0)
  p_v_x <- mvtnorm::pmvnorm(c(0, 0), mean = c(mu_v, mu_x), sigma = sigma_mat)
  # P(X>0)
  p_x <- pnorm(0, mean = mu_x, sd = sqrt(sigma_x2), lower.tail = FALSE)
  # P(V>0)
  p_v <- pnorm(0, mean = mu_v, sd = sqrt(sigma_v2), lower.tail = FALSE)

  # Sensitivity
  sens <- p_v_x/p_x
  # Specificity
```

```r
  spec <- (1-(p_v + p_x - p_v_x))/(1-p_x)

  return(list(sens_calc = sens, spec_calc = spec, me_var = sigma_u2))
}

sens_spec_list <- lapply(me_variances, sens_spec_exact)
exact_sens_spec <- as.data.frame(do.call(rbind, sens_spec_list)) %>%
  mutate(across(everything(), as.numeric))
```

We combine the results and plot the sensitivity and specificity for different levels of measurement error variance:

```r
sim_and_calc <- inner_join(exact_sens_spec, result_df, by = "me_var")

ggplot(sim_and_calc) +
  geom_path(aes(x = spec_calc, y = sens_calc), color = "grey") +
  geom_path(aes(x = spec, y = sens)) +
  geom_point(aes(x = spec, y = sens, color = me_var)) +
  geom_point(aes(x = spec_calc, y = sens_calc, color = me_var)) +
  scale_color_viridis_c(option = "magma", direction = -1) +
  coord_equal() +
  labs(x = "Specificity", y = "Sensitivity", color = "ME variance") +
  theme_bw()
```



This plot shows the resulting sensitivity and specificity when the measurement error variance goes from 0.01 to 2.

Adjusting for response misclassification with the sslogit link

Accounting for imperfect sensitivity and specificity in the response

To use the `sslogit` link in INLA this needs to be enabled as follows:

```
library(INLA)
```

```
inla.models <- INLA:::inla.get.inlaEnv()$inla.models
inla.models$link$sslogit$status <- NULL
assign("inla.models", inla.models, env = INLA:::inla.get.inlaEnv())
rm(inla.models)
```

Be aware that the reason this link is not available by default is that it struggles numerically if the sensitivity and specificity is large. SO USE WITH CAUTION.

```
set.seed(1)
```

We simulate a misclassified response. `p_y` is the success probability for the correct version of the response, `y`. `s` is the misclassified response, and `p_s` is the success probability for `s`.

```
n <- 1000

p_y <- 0.1 # Success probability for y
sens <- 0.95 # P(y = 1 | s = 1)
spec <- 0.90 # P(y = 0 | s = 0)
p_s <- p_y*sens + (1-p_y)*(1-spec) # Success probability for s

df <- data.frame(s = rbinom(n = n, size = 1, prob = p_s))
```

Given this data, we can fit the naive model `r0`, which uses `s` without adjusting for misclassification, and the adjusted model `r1`, which adjusts for the misclassification by using and adjusted link function, as described in Section 4.1.

```
formula <- s ~ 1

# Model 0 (ignoring sens and spec)
r0 <- inla(formula = formula, data = df, family = "binomial",
           Ntrials = 1)

# Model 1 (adding sens and spec)
r1 <- inla(formula = formula, data = df,
           family = "binomial",
           control.family = list(
             control.link = list(
               model = "sslogit",
               hyper = list(
                 sens = list(
                   prior = "logitbeta",
                   initial = inla.link.logit(sens),
                   fixed = TRUE
```

```
                ),
                spec = list(
                  prior = "logitbeta",
                  initial = inla.link.logit(spec),
                  fixed = TRUE)
              )
            )
          )
        )
```

```
r0$summary.fixed
```

```
##                  mean        sd 0.025quant 0.5quant 0.975quant     mode kld
## (Intercept) -1.50499 0.0819573  -1.665623 -1.50499  -1.344356 -1.50499   0
```

```
r1$summary.fixed
```

```
##                  mean        sd 0.025quant 0.5quant 0.975quant     mode kld
## (Intercept) -2.24994 0.1225932  -2.490218 -2.24994  -2.009661 -2.24994   0
```

Alternative approach when sensitivity and specificity are not known exactly

For a third approach, we fit the model for a grid of different sensitivities and specificities, as described in the end of Section 4.1.

```
IC.sens = c(0.925, 0.975)
IC.spec = c(0.85, 0.95)

# Latin square grid
n.latin <- 50
sens.values <- seq(from = IC.sens[1],
                   to = IC.sens[2],
                   length.out = n.latin)
spec.values <- seq(from = IC.spec[1], to = IC.spec[2],
                   length.out = n.latin)

# For the Latin-square
reord = sample(1:n.latin)
SSgrid = data.frame(sens = sens.values,
                    spec = spec.values[reord])
# Weights
# Using independent normals
sd.sens = (IC.sens[1] - sens) / qnorm(0.025)
sd.spec = (IC.spec[1] - spec) / qnorm(0.025)
SS.weights = dnorm(SSgrid$sens, sens, sd.sens) * dnorm(SSgrid$spec, spec, sd.spec)
SS.weights = SS.weights / sum(SS.weights)
```

We run INLA for each pair of sensitivity and specificity of the Latin square, and keep the results in a list:

```
inla.SSlogit.grid <- function(SS, data, formula){
  r.temp = inla(data = data,
                formula = formula,
                family = "binomial",
                control.family = list(
```

```
                  control.link = list(
                    model = "sslogit",
                    hyper = list(
                      sens = list(
                        prior = "logitbeta",
                        initial = inla.link.logit(SS$sens),
                        fixed = TRUE),
                      spec = list(
                        prior = "logitbeta",
                        initial = inla.link.logit(SS$spec),
                        fixed = TRUE)
                      )
                    )
                )
  )
  r.temp
}

r.list <- lapply(X = 1:nrow(SSgrid),
                 FUN = function(x){
                   inla.SSlogit.grid(SSgrid[x,],
                                     formula = formula,
                                     data = df)
                   }
                 )

# Merging all INLA outputs
r2 <- inla.merge(loo = r.list, prob = SS.weights)
```

Comparing all approaches

The model that does no adjustment:

```
r0$summary.fixed[,c(1,3,5)]
```

```
##                 mean 0.025quant 0.975quant
## (Intercept) -1.50499  -1.665623  -1.344356
```

The model that adjusts for misclassification using fixed values for sensitivity and specificity:

```
r1$summary.fixed[,c(1,3,5)]
```

```
##                 mean 0.025quant 0.975quant
## (Intercept) -2.24994  -2.490218  -2.009661
```

The model that uses a grid of sensitivity and specificity:

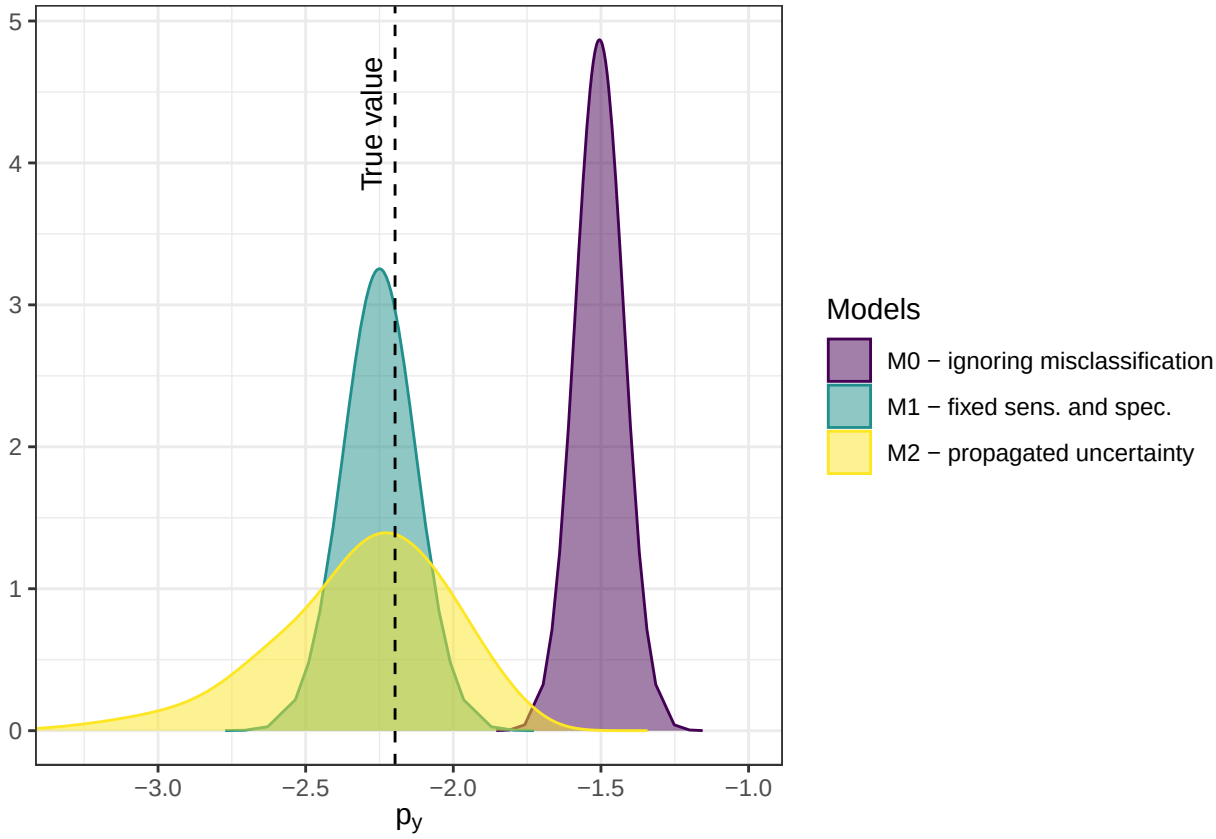
```
# Approximate 95% interval
c(r2$summary.fixed$mean, r2$summary.fixed$mean + c(-2, 2)*r2$summary.fixed$sd)
```

```
## [1] -2.308922 -2.931946 -1.685898
```

```
# Using inla.qmarginal
c(r2$summary.fixed$mean,
  inla.qmarginal(p = c(0.025, 0.975),
                 marginal = r2$marginals.fixed$`(Intercept)`))
```

```
## [1] -2.308922 -3.027396 -1.790307
```

```
marginals.df <- dplyr::bind_rows(
  'M0 - ignoring misclassification' = data.frame(r0$marginals.fixed$`(Intercept)`),
  'M1 - fixed sens. and spec.' = data.frame(r1$marginals.fixed$`(Intercept)`),
  'M2 - propagated uncertainty' = data.frame(r2$marginals.fixed$`(Intercept)`),
  .id = "Models")

library(ggplot2)

ggplot(marginals.df, aes(x = x, y = y, color = Models, fill = Models)) +
  #geom_line() +
  geom_area(alpha = 0.5, position = "identity") +
  geom_vline(xintercept = inla.link.logit(p_y), show.legend = F,
             linetype = "dashed") +
  annotate("text", x = inla.link.logit(p_y) - 0.08, y = 3.8, label="True value",
           hjust = 0, angle = 90) +
  scale_color_viridis_d() + scale_fill_viridis_d() +
  coord_cartesian(xlim = c(-3.3, -1)) +
  theme_bw() +
  xlab(bquote(p[y])) +
  theme(legend.position.inside = c(0.2, 0.8),
        axis.title.y = element_blank())
```




```
ggsave("figures/response_mc.pdf", height = 4, width = 7)
```

Birth weight analysis

```
run_birthweight1 <- FALSE
run_birthweight2 <- FALSE
run_birthweight_mcmc <- FALSE
```

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

Across all cases, we will use the same misclassification matrix and number of iterations, so we define these.

```
MC_matrix <- matrix(c(0.95, 0.05, 0.2, 0.8), nrow = 2, byrow = T)
niter <- 10000
```

```
birthweight_naive <- INLA::inla(bwt ~ lwt + smoke, data = birthweight)
```

Birth weight analysis, case 1

In this case, we assume that the proportion of smokers in the study is 0.4, and that the probability of a woman smoking is independent of other covariates. That means that the exposure model will only have an intercept, and this we set to be $\alpha_0 = \log(\frac{p}{1-p})$, where $p = 0.4$.

```
p <- 0.4

start_time <- Sys.time()
birthweight_model1 <- inla_is(formula_moi = bwt ~ smoke + lwt,
                              formula_imp = smoke ~ 1,
                              alpha = log(p/(1-p)),
                              MC_matrix = MC_matrix,
                              data = birthweight, niter = niter)
end_time <- Sys.time()

birthweight_results1 <- list(runtime = end_time - start_time, model = birthweight_model1,
                             summary = make_results_df(birthweight_model1, niter = niter))

saveRDS(list(model1 = birthweight_results1,
             niter = niter, nburnin = 0, rundate = Sys.time()),
        file = "code/results/birthweight_results1.rds")
```

```
birthweight_results1 <- readRDS("code/results/birthweight_results1.rds")
```

Birth weight analysis, case 2

In this case, we let the prbability that a person is smoking depend on their weight according to a logistic exposure model with $\alpha_0 = -0.3$ and $\alpha_z = 0.02$, meaning that higher body weight leads to a higher probability of smoking.

```
start_time <- Sys.time()
birthweight_model2 <- inla_is(formula_moi = bwt ~ smoke + lwt,
```

```
                                formula_imp = smoke ~ lwt,
                                alpha = c(-3, 0.02),
                                MC_matrix = MC_matrix,
                                data = birthweight, niter = niter)
end_time <- Sys.time()

birthweight_results2 <- list(runtime = end_time - start_time, model = birthweight_model2,
                             summary = make_results_df(birthweight_model2, niter = niter))



# Save results ----
saveRDS(list(model2 = birthweight_results2,
             niter = niter, nburnin = 0, rundate = Sys.time()),
        file = "code/results/birthweight_results2.rds")
```

```
birthweight_results2 <- readRDS("code/results/birthweight_results2.rds")
```

```
plot_compare_inlamisclass(list(birthweight_results1$model1$model,
                               birthweight_results2$model2$model),
                          naive_mod = birthweight_naive,
                          niter = niter, num_inlamisclass_models = 2)
```

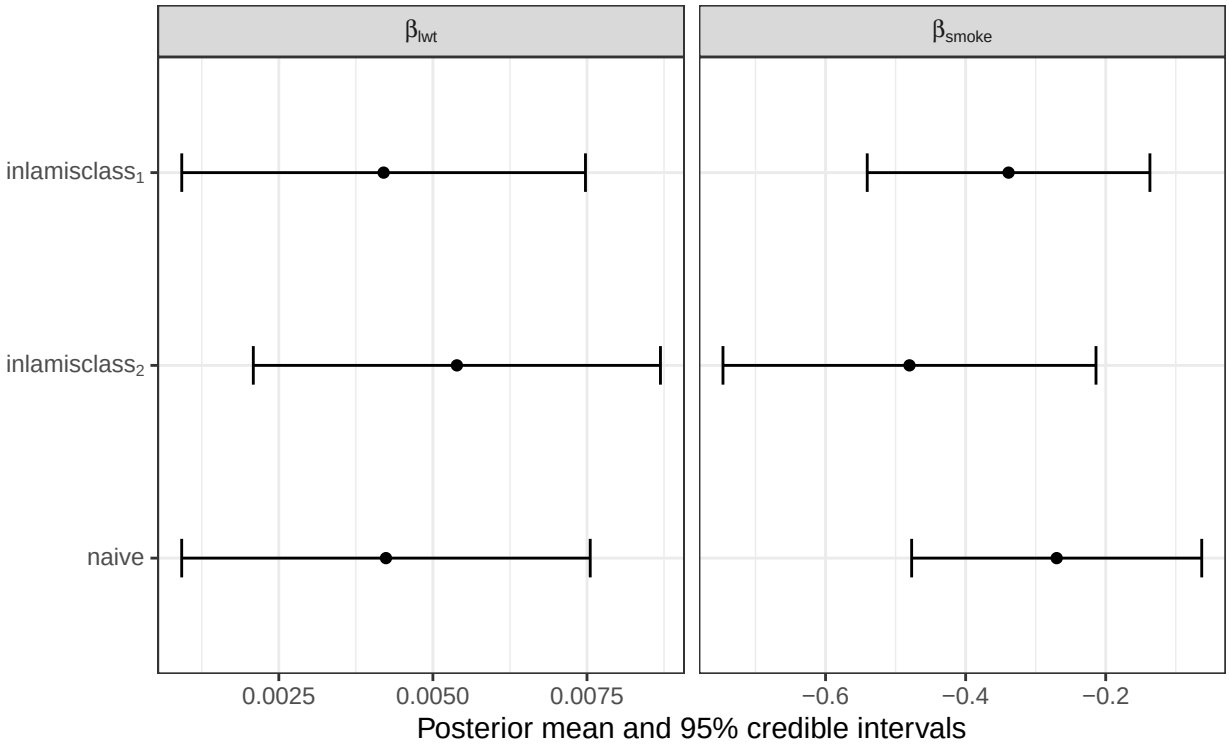
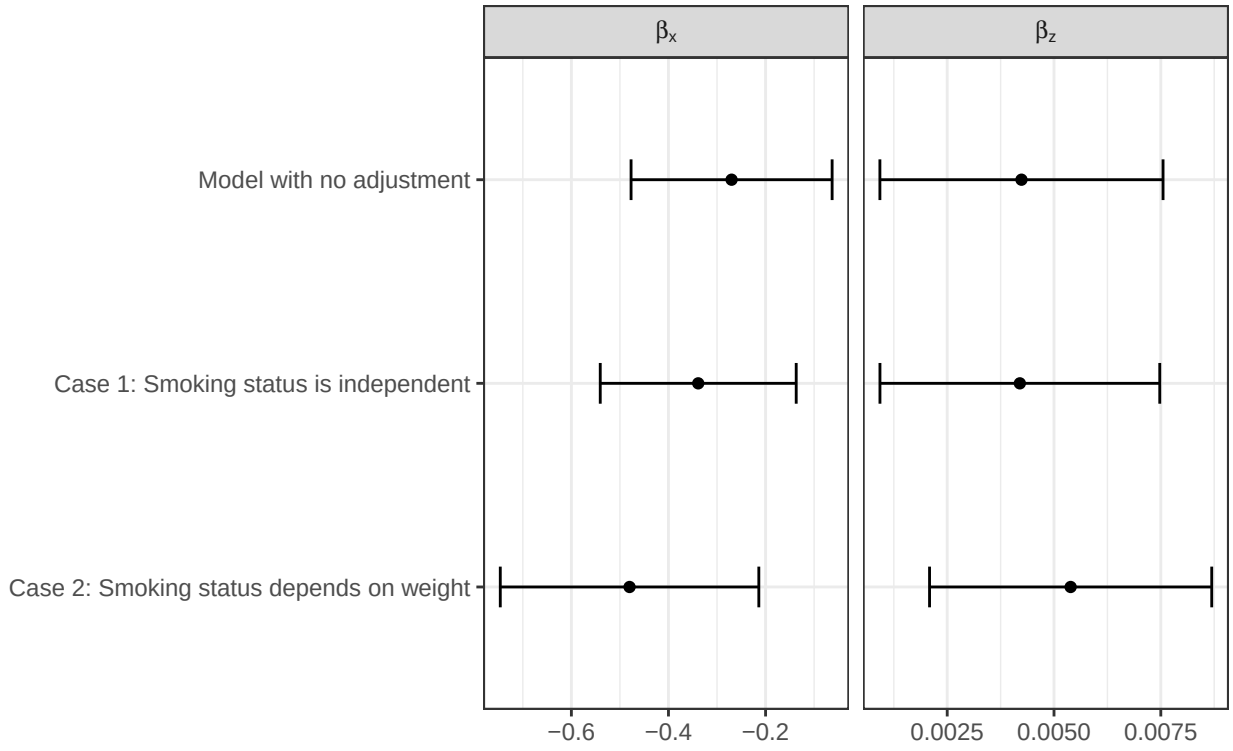


Figure for paper

```
results1 <- make_results_df(birthweight_results1$model1$model)$moi
results2 <- make_results_df(birthweight_results2$model2$model)$moi
results_naive <- birthweight_naive$summary.fixed
results_naive$variable <- rownames(results_naive)
```

```r
all_res <- dplyr::bind_rows(Ex1 = results1, Ex2 = results2, Naive = results_naive, .id = "Model")
all_res$labels <- paste0("beta", "[", c(0, "z", "x"), "]")
all_res$Model <- factor(all_res$Model, levels = c("Naive", "Ex1", "Ex2"))
all_res$Model <- plyr::revalue(all_res$Model,
                               c("Naive" = "Model with no adjustment",
                                 "Ex1" = "Case 1: Smoking status is independent",
                                 "Ex2" = "Case 2: Smoking status depends on weight"))

ggplot(dplyr::filter(all_res, variable!= "(Intercept)"), aes(y = Model)) +
  geom_point(aes(x = mean)) +
  geom_errorbarh(aes(xmin = .data$"0.025quant", xmax = .data$"0.975quant"), height = .2) +
  scale_y_discrete(limits = rev) +
  facet_wrap(vars(labels), scales = "free_x", labeller = label_parsed) +
  theme_bw() +
  theme(axis.title = element_blank())
```




```r
ggsave("figures/birthweight.pdf", height = 2.3, width = 7)
```

Cervical cancer and herpes

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

The data `case_control_data` consists of $n = 2044$ rows and three columns:

- $y_1, \cdots, y_n$: The cervical cancer status of a patient.
- $x_1, \cdots, x_n$: Exposure to HSV-2, measured with an accurate test, only 115 measurements available.
- $w_1, \cdots, w_n$: Exposure to HSV-2, measured with an inaccurate test.

The response $\boldsymbol{y}$ and the less accurate test result $\boldsymbol{w}$ are available for all patients, while the more accurate test result $\boldsymbol{x}$ is only available for 115 of the patients. That means that we have a validation sample of 115 samples which can be used to estimate the misclassification probabilities.

```
validation <- filter(case_control_data, !is.na(x))
incomplete_data <- filter(case_control_data, is.na(x))
```

By looking at the data in aggregated form, it can be seen that the misclassification probabilities for the cases ($y_i = 1$) and controls ($y_i = 0$) are quite different. Therefore, we will consider two cases: Case 1, where we only consider the overall misclassification matrix, irrespective of the value of y_i, and Case 2, where we switch between the two matrices depending on the value of y_i in the modelling.

Case 1: Misclassification is considered independent of y_i

We estimate the overall misclassification matrix from the validation data:

```
# w = 1 given x = 0
pi10 <- sum((validation$w-validation$x) == 1)/sum(validation$x==0)

# w = 0, given x = 1
pi01 <- sum(validation$w-validation$x == -1)/sum(validation$x==1)

M <- matrix(c(1-pi10, pi10, pi01, 1-pi01), byrow = TRUE, nrow = 2)
```

For the exposure model describing $\boldsymbol{x}$, we estimate the probability of exposure to HSV-2, $P(x_i = 1)$ from the validation data:

```
p <- sum(validation$x == 1)/nrow(validation)
alpha0 <- log(p/(1-p))
```

This means that for the modelling, we will use the misclassification matrix

```
M
```

```
##           [,1]      [,2]
## [1,] 0.7666667 0.2333333
## [2,] 0.3818182 0.6181818
```

and the exposure model $\text{logit}[E(x)] = \alpha_0 \mathbf{1}$, where α_0 is assumed to be -0.0870114.

For running the model, we will run importance sampling for the given number of iterations, and the results will be saved along with the running time:

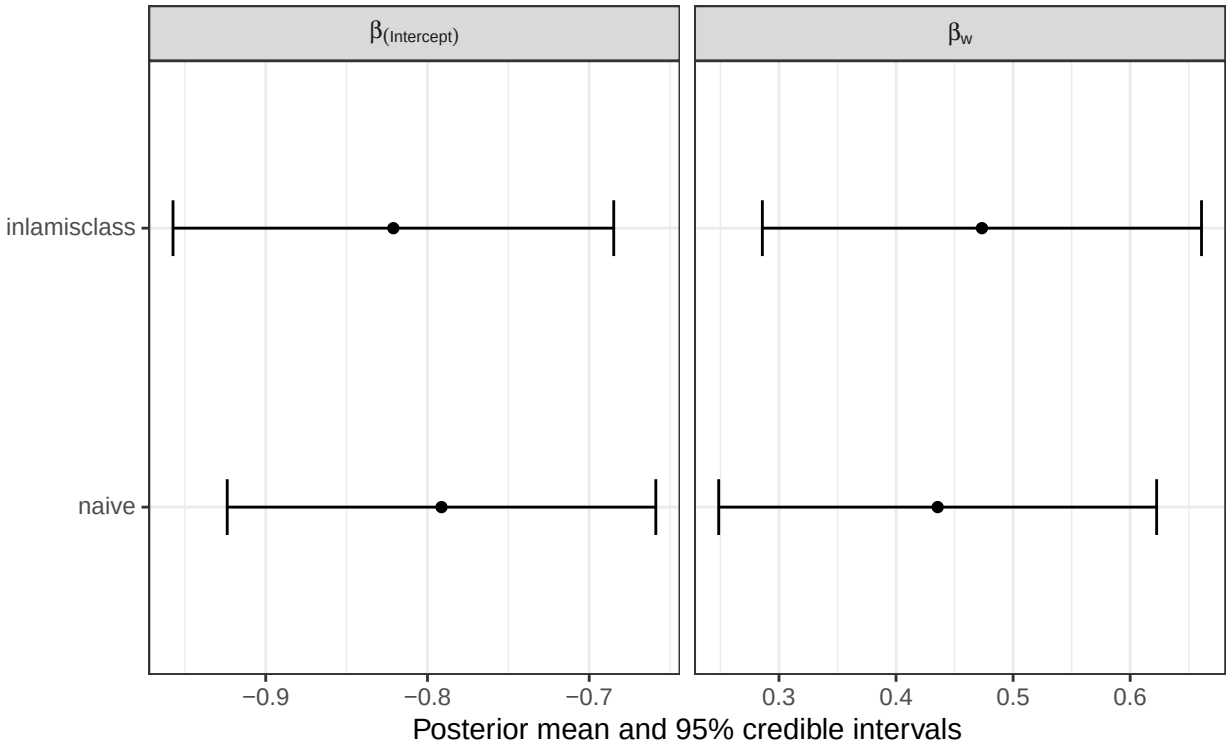
```
niter <- 10000

start_time <- Sys.time()
case_control_model1 <- inla_is(formula_moi = y ~ w,
                               formula_imp = w ~ 1,
                               alpha = alpha0,
                               MC_matrix = M,
                               data = incomplete_data,
                               niter = niter,
                               family = "binomial", Ntrials = 1)
end_time <- Sys.time()

case_control_results1 <- list(runtime = end_time - start_time,
                              model = case_control_model1,
                              summary = make_results_df(case_control_model1,
                                                        niter = niter))

saveRDS(list(case_control_model = case_control_results1,
             niter = niter, rundate = Sys.time()),
        file = "code/results/case_control_results1.rds")
```

```
case_control_results1 <- readRDS("code/results/case_control_results1.rds")
```

We fit a naive model that ignores the misclassification in $\boldsymbol{w}$ for a comparison.

```
naive_cc <- INLA::inla(y ~ w, data = incomplete_data, family = "binomial",
                       Ntrials = 1)
```

```
plot_compare_inlamisclass(case_control_results1$case_control_model$model,
                          naive_mod = naive_cc, plot_intercept = TRUE)
```



Case 2: Misclassification and exposure depend on y_i

We estimate the conditional misclassification matrices based on the value of $\boldsymbol{y}$:

```
validation1 <- filter(validation, y == 1)
validation0 <- filter(validation, y == 0)

# w = 1 given x = 0, y = 1
pi10_y1 <- sum((validation1$w-validation1$x) == 1)/sum(validation1$x==0)
# w = 0, given x = 1, y = 1
pi01_y1 <- sum(validation1$w-validation1$x == -1)/sum(validation1$x==1)

# w = 1, x = 0, given y = 0
pi10_y0 <- sum((validation0$w-validation0$x) == 1)/sum(validation0$x==0)
# w = 0, x = 1, given y = 0
pi01_y0 <- sum(validation0$w-validation0$x == -1)/sum(validation0$x==1)
```

So the MC matrix for $y_i = 1$ would be

```
M1 <- matrix(c(1-pi10_y1, pi10_y1, pi01_y1, 1-pi01_y1), byrow = TRUE, nrow = 2)
M1
```

```
##           [,1]      [,2]
## [1,] 0.8125000 0.1875000
## [2,] 0.2173913 0.7826087
```

and for $y_i = 0$:

```
M0 <- matrix(c(1-pi10_y0, pi10_y0, pi01_y0, 1-pi01_y0), byrow = TRUE, nrow = 2)
M0
```

```
##      [,1] [,2]
```

```
## [1,] 0.75 0.25
## [2,] 0.50 0.50
```

Similarly, we estimate the coefficients of the exposure model conditional on y_i:

```
p1 <- sum(validation1$x == 1)/nrow(validation1)
alpha0_1 <- log(p1/(1-p1))
p0 <- sum(validation0$x == 1)/nrow(validation0)
alpha0_0 <- log(p0/(1-p0))
```

```
c(alpha0_0 = alpha0_0, alpha0_1 = alpha0_1)
```

```
##   alpha0_0   alpha0_1
## -0.3184537  0.3629055
```

But this is the same as regression model:

```
exp_glm <- glm(x~y, family = "binomial", data = validation)$coef
alphas <- exp_glm["(Intercept)"] + c(0, exp_glm["y"])
alphas
```

```
##                   y
## -0.3184537  0.3629055
```

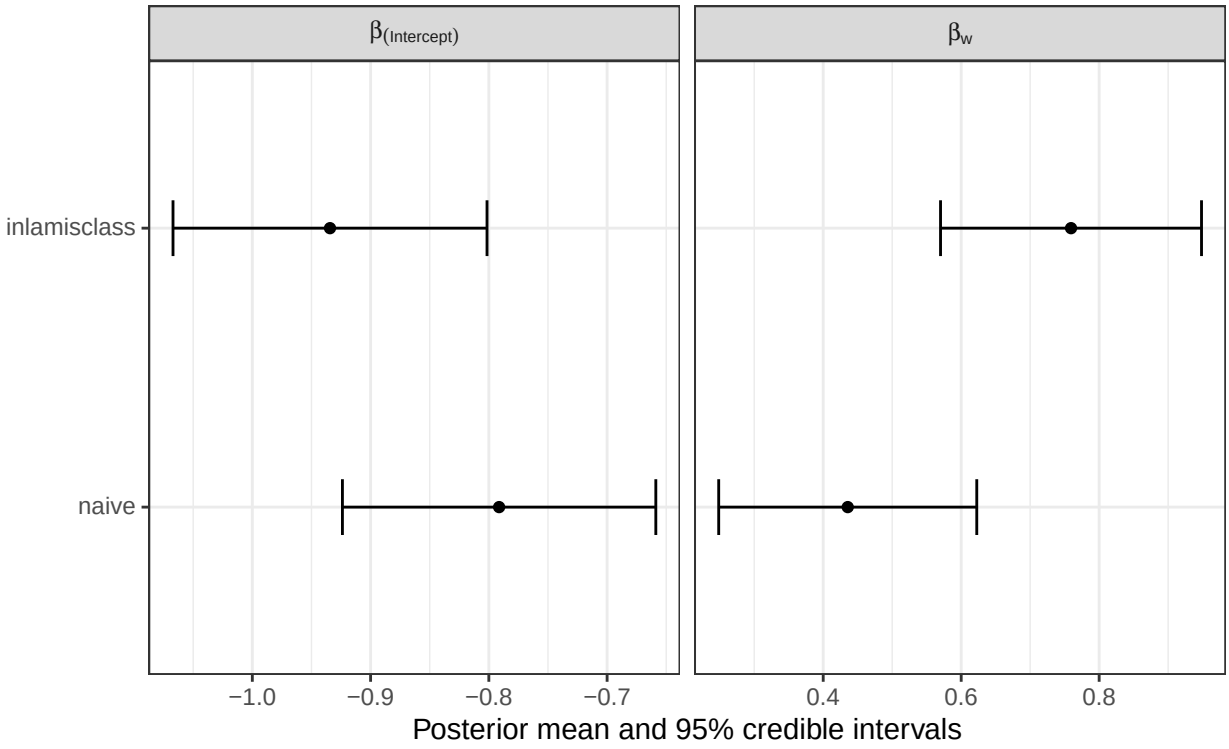
So we use the exposure/imputation model $\text{logit}[E(\boldsymbol{x} \mid \boldsymbol{y})] = \alpha_0 \mathbf{1} + \alpha_y \boldsymbol{y}$, with fixed values for the coefficients, $\alpha_0 = -0.3185$ and $\alpha_y = 0.3629$.

```
start_time <- Sys.time()
case_control_model2 <- inla_is(formula_moi = y ~ w,
                               formula_imp = w ~ y,
                               alpha = alphas,
                               MC_matrix = list(MC_1 = M1, MC_0 = M0),
                               data = incomplete_data,
                               niter = niter,
                               conditional = "y",
                               family = "binomial", Ntrials = 1)
end_time <- Sys.time()

case_control_results2 <- list(
  runtime = end_time - start_time,
  model = case_control_model2,
  summary = make_results_df(case_control_model2, niter = niter))

saveRDS(list(case_control_model = case_control_results2,
             niter = niter, nburnin = 0, rundate = Sys.time()),
        file = "code/results/case_control_results2.rds")
```

```
case_control_results2 <- readRDS("code/results/case_control_results2.rds")
```

Again, we plot the results from this model together with the estimates from the model that does not account for misclassification.

```
plot_compare_inlamisclass(case_control_results2$case_control_model$model,
                          naive_mod = naive_cc, plot_intercept = TRUE)
```



Comparing the cases

We plot both cases together, along with the naive model, to compare.

```
plot_compare_inlamisclass(
  list(case_control_results1$case_control_model$model,
       case_control_results2$case_control_model$model),
  naive_mod = naive_cc, plot_intercept = TRUE,
  num_inlamisclass_models = 2, niter = case_control_results2$niter)
```

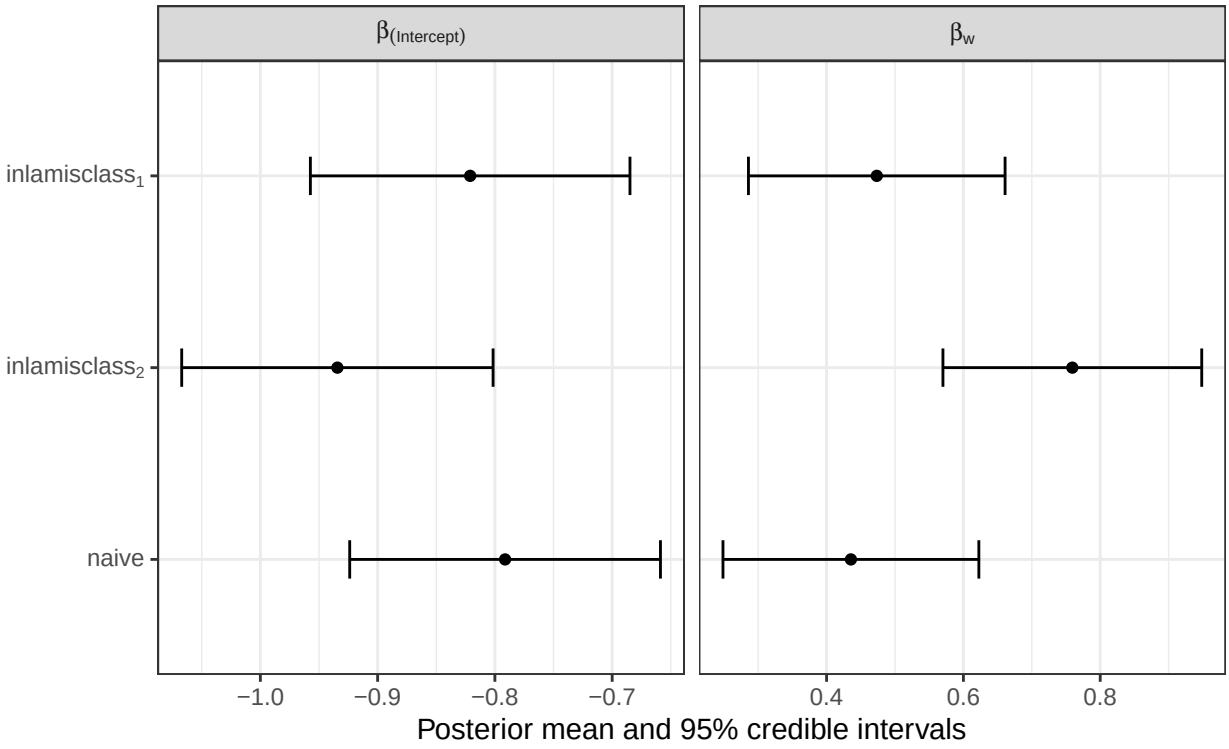
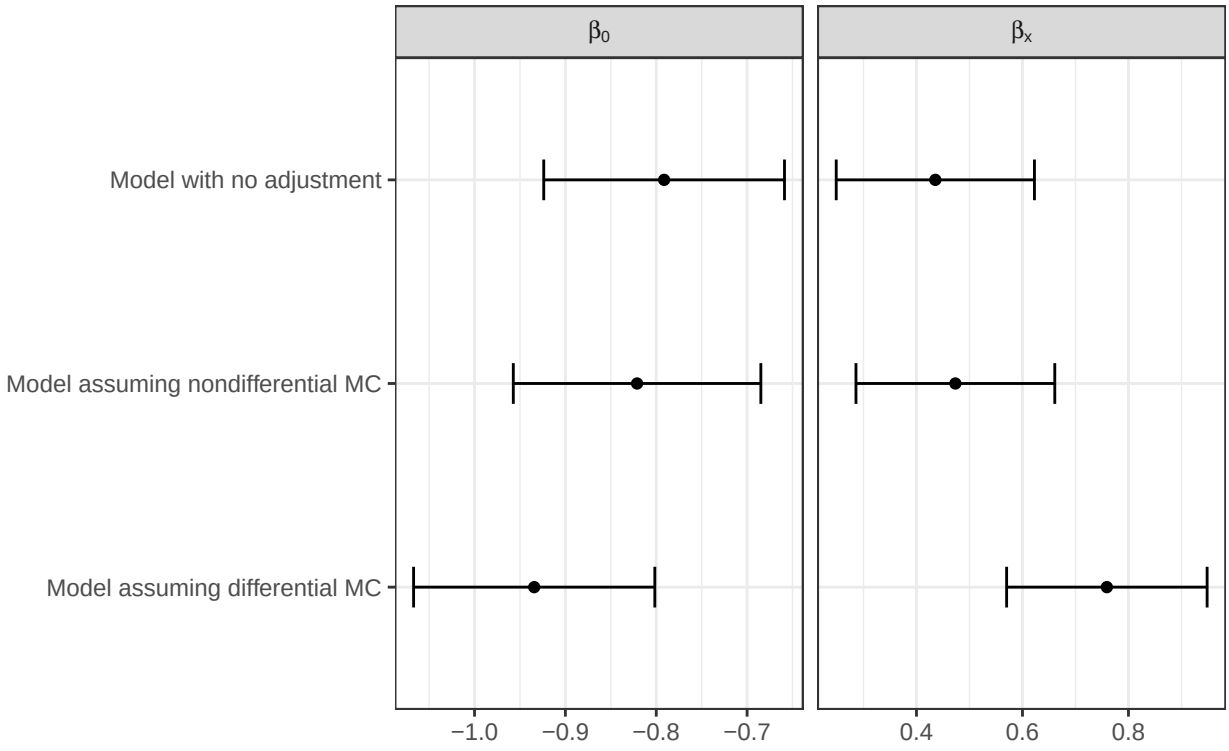


Figure for article

```
results_nondiff <- make_results_df(case_control_results1$case_control_model$model)$moi
results_diff <- make_results_df(case_control_results2$case_control_model$model)$moi

results_naive <- naive_cc$summary.fixed
results_naive$variable <- rownames(results_naive)

all_res <- dplyr::bind_rows(nondiff = results_nondiff, diff = results_diff, Naive = results_naive, .id =
all_res$labels <- paste0("beta", "[", c(0, "x"), "]")
all_res$Model <- factor(all_res$Model, levels = c("Naive", "nondiff", "diff"))
all_res$Model <- plyr::revalue(all_res$Model,
                               c("Naive" = "Model with no adjustment",
                                 "nondiff" = "Model assuming nondifferential MC",
                                 "diff" = "Model assuming differential MC"))

ggplot(all_res, aes(y = Model)) +
  geom_point(aes(x = mean)) +
  geom_errorbarh(aes(xmin = .data$"0.025quant", xmax = .data$"0.975quant"), height = .2) +
  scale_y_discrete(limits = rev) +
  facet_wrap(vars(labels), scales = "free_x", labeller = label_parsed) +
  theme_bw() +
  theme(axis.title = element_blank())
```




```
ggsave("figures/case_control.pdf", height = 2.3, width = 7)
```