Serological Analysis Examples

Morgan P. Kain, Jonathan H. Epstein, Noam Ross

Overview

Here we present the analysis of three serological datasets (3 of the 500 analyzed in the main text). These three datasets represent three alternatives in regards to percent population seroposivity, differences in mean MFI between seropositive and seronegative groups, and distributional skew. As described in the main text, seropositivity was simulated to be a function of 2 categorical covariates with two levels each (that could capture, for example, sex and species) and 1 continuous covariate (e.g., age in years).

We analyze each dataset here with a single method well-suited for that dataset (the consequences of using a poorly-suited analysis are summarized in the main text). Specifically, we consider both a two step extreme value + logistic regression approach as well as a single step Bayesian latent class regression (LCR) approach. Though we generally advocate against a two step approach when seeking inference about risk factors of seropositivity (what we call here an "epidemiological analysis"), an extreme value approach (e.g., 3sd method) is well suited for serostatus assignment and seroprevalence estimates when true seroprevalence is low (e.g., under 3%) and the positive MFI values have high variance–which together results in a unimodal right-skewed MFI distribution. Though using dichotomous seropositive assignments can result in over-confidence in logistic regression-estimated parameter values (too-narrow confidence intervals), attempting to estimate risk factors with very low seroposivity may nevertheless result in a lack of clear inference on risk factors (i.e., CI overlapping zero) because of low power. We show a two step extreme value approach with one of the three datasets here.

We work through the analysis from raw data visualization through diagnostics prior to comparing estimated values and simulated "truth", keeping the true simulated coefficients hidden until this point.

Visualization

For the analysis of any new serological (MFI: Median/Mean Fluorescence Intensity) dataset, the first step is a visual inspection of the MFI distribution to check for: a) bimodality, and b) if bimodality is present, evidence of right-skew in the right mode.

Given our findings (see main text) that false-negative and false-positive serostatus assignment error rates–and thus bias in seroprevalence estimates–are lower for the 3sd extreme value approach when MFI is analyzed on a linear scale but for clustering methods when MFI is analyzed on a log scale, we advise visualization of data on both scales.

Our three datasets chosen for analysis here are visualized in Figure 1.

A few things stand out from these visualizations. First, the dataset pictured in the left column has a long right tail on the linear scale and no strong bimodality when visualized on either scale. This observation hints that seroprevalence is likely quite low in this dataset (or that seronegative and seropositive individuals have very similar MFI values; however, this is both unlikely in an empirical dataset and would be very difficult to analyze regardless: see Figure S19 in manuscript's supplemental material). Second, both of the datasets in the right two columns have clear bimodality, which is especially prominent on the log scale. Third, the right mode of the right-most dataset shows clear left skew due to the MFI values getting "stacked up" at the machine maximum.

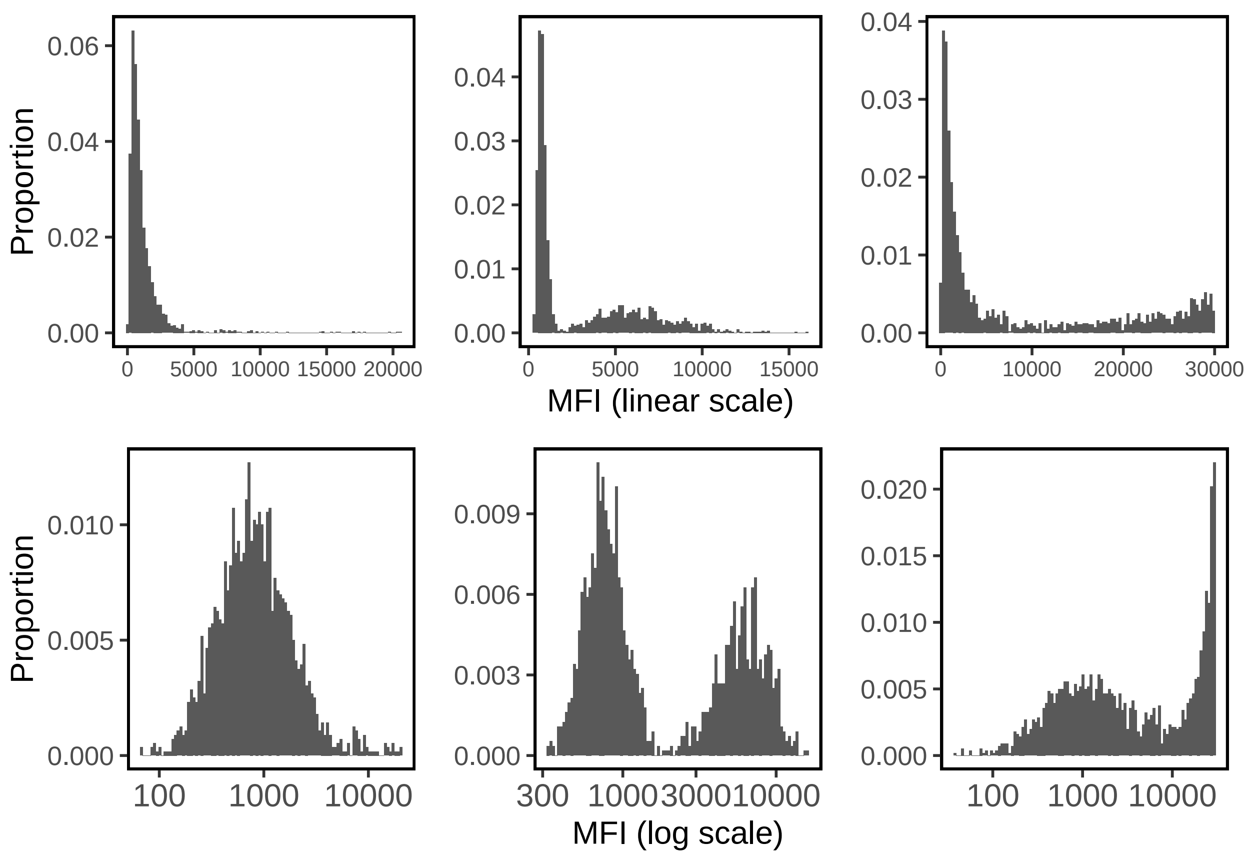

Figure 1: Three example MFI datasets (columns), visualized on the linear scale (top row) and log scale (bottom row).

An analysis plan

This visual inspection leads us to the following analysis plan for the three datasets for assigning serostatus (see the main text for further exposition). Dataset 1 (Figure 1, left column): 3sd with a cutoff derived from a robust mean and sd calculation using the R functions dplR::tbrm and jointseg::estimateSd, respectively; MFI analyzed on a linear scale. Estimation of seroprevalence using a binomial confidence interval. Analysis of risk factors of a positive serostatus using logistic regression with dichotomous serostatus assignments.

Dataset 2 (Figure 1, center column): Bayesian LCR fitting normal distributions to log MFI. Estimation of population seroposivity and risk factors within the Bayesian model.

Dataset 3 (Figure 1, right column): Bayesian LCR fitting a normal and a skewnormal distribution to log MFI. Estimation of population seroposivity and risk factors within the Bayesian model.

Analysis

Dataset 1

A) Assign serostatus based on a 3sd cutoff

```r
dataset1.linear_mfi <- dataset1 %>%
  filter(log_mfi == "mfi") %>%
  mutate(
    cat1f = factor(cat1f)
  , cat2f = factor(cat2f)
  )

mean1  <- dataset1.linear_mfi %>% pull(mfi) %>% dplR::tbrm()
sd1    <- dataset1.linear_mfi %>% pull(mfi) %>% jointseg::estimateSd()
thresh <- mean1 + 3 * sd1

dataset1.linear_mfi %<>% mutate(seropositive = ifelse(mfi > thresh, 1, 0))
```

B) Estimate seroprevalence

```r
dataset1.pop_mod  <- glm(seropositive ~ 1, family = "binomial"
                         , data = dataset1.linear_mfi)

(dataset1.pop_pos <- c(confint(dataset1.pop_mod) %>% plogis()
                       , coef(dataset1.pop_mod) %>% plogis()) %>%
  t() %>%
  as.data.frame() %>%
  mutate(
      dataset  = 1
    , approach = "3sd + glm"
    , .before = 1
    ) %>%
  dplyr::rename(
    mid = `(Intercept)`, lwr = `2.5 %`, upr = `97.5 %`
    ))
```

```
##          lwr        upr        mid dataset  approach .before
## 1 0.01707938 0.03047139 0.02314815       1 3sd + glm       1
```

C) Estimate the impacts of risk factors on individual seropositivity

```r
dataset1.risk_mod   <- glm(seropositive ~ cat1f + cat2f + con1f
                           , family = "binomial", data = dataset1.linear_mfi)
```

```
(dataset1.risk_coef <- confint(dataset1.risk_mod) %>%
    as.data.frame() %>%
    mutate(`50.0 %` = coef(dataset1.risk_mod)) %>%
    mutate(
      dataset  = 1
    , approach = "3sd + glm"
    , coef     = rownames(.)
    , .before  = 1
    ) %>%
    mutate(coef = plyr::mapvalues(
      coef
    , from = c("(Intercept)", "cat1f1", "cat2f1")
    , to = c("baseline", "cat1f", "cat2f")
    )) %>%
  dplyr::rename(
    mid = `50.0 %`, lwr = `2.5 %`, upr = `97.5 %`
    )
)
```

```
##                    lwr        upr        mid dataset  approach     coef .before
## (Intercept) -4.6515414 -3.5395417 -4.0606762       1 3sd + glm baseline       1
## cat1f1      -0.7988781  0.4108698 -0.1876991       1 3sd + glm    cat1f       1
## cat2f1      -0.4629272  0.7426436  0.1363321       1 3sd + glm    cat2f       1
## con1f        0.2636053  0.5867791  0.4224838       1 3sd + glm    con1f       1
```

Dataset 2

A) Assign serostatus, predict seroprevalence, and estimate risk factors for individual seropositivity

```
  ## just log mfi
  dataset2.log <- dataset2 %>% filter(log_mfi == "log_mfi")

  ## See code for function details. Establishes priors (which are sent in as data)
  stan_priors <- build_stan_priors(
    simulated_data = dataset2.log
  , skew_fit       = FALSE
    ## First trying with "naive" priors
  , fit_attempt    = 1
  )

  ## Compile model
  stan_model <- cmdstanr::cmdstan_model(
    "stan_models/publication_model_normal_2.stan"
  , pedantic = FALSE)

  ## Fit model. For model definition see ".stan" in the online supplemental material
  stan_fit2    <- try(R.utils::withTimeout(stan_model$sample(
    data    = list(
        N           = length(dataset2.log$mfi)
      , y           = dataset2.log$mfi
      , cat1f       = dataset2.log$cat1f
      , cat2f       = dataset2.log$cat2f
      , con1f       = dataset2.log$con1f
      , beta_base_prior_m   = stan_priors$priors %>%
        filter(param == "beta_base_prior_m") %>% pull(prior)
```

```r
      , beta_base_prior_v   = stan_priors$priors %>%
        filter(param == "beta_base_prior_v") %>% pull(prior)
      , mu_base_prior_m     = stan_priors$priors %>%
        filter(param == "mu_base_prior_m") %>% pull(prior)
      , mu_diff_prior_m     = stan_priors$priors %>%
        filter(param == "mu_diff_prior_m") %>% pull(prior)
      , sigma_base_prior_m  = stan_priors$priors %>%
        filter(param == "sigma_base_prior_m") %>% pull(prior)
      , sigma_diff_prior_m  = stan_priors$priors %>%
        filter(param == "sigma_diff_prior_m") %>% pull(prior)
      , mu_base_prior_v     = stan_priors$priors %>%
        filter(param == "mu_base_prior_v") %>% pull(prior)
      , mu_pos_prior_v      = stan_priors$priors %>%
        filter(param == "mu_pos_prior_v") %>% pull(prior)
      , sigma_base_prior_v  = stan_priors$priors %>%
        filter(param == "sigma_base_prior_v") %>% pull(prior)
      , sigma_diff_prior_v  = stan_priors$priors %>%
        filter(param == "sigma_diff_prior_v") %>% pull(prior)
    )
    , init            = stan_priors$starting_conditions
    , chains          = 4
    , parallel_chains = 4
    , max_treedepth   = 12
    , iter_warmup     = 2000
    , iter_sampling   = 1000
    , adapt_delta     = 0.98
    , seed            = 48389
    , refresh         = 1000
  ), timeout = 3600
  ), silent  = TRUE)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:    1 / 3000 [  0%]  (Warmup)

## Chain 2 Iteration:    1 / 3000 [  0%]  (Warmup)

## Chain 3 Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 4 Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 1 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 2 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 3 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 4 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 1 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 1 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 2 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 2 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 3 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 3 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 4 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 4 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1 finished in 25.1 seconds.
## Chain 4 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4 finished in 27.4 seconds.
## Chain 2 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3 finished in 27.5 seconds.
## Chain 2 finished in 27.6 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 26.9 seconds.
## Total execution time: 27.7 seconds.
```

```
## Extract predictions
stanfit      <- rstan::read_stan_csv(stan_fit2$output_files())
samps.mat2   <- rstan::extract(stanfit, permuted = FALSE)
samps.simp2  <- rstan::extract(stanfit, permuted = TRUE)
```

B) Run some diagnostics (for more extensive diagnostics and a broader discussion of model convergence see: Bayesian Workflow. Gelman et al. 2020, arXiv and the Stan manual)

```
stan_fit2$diagnostic_summary()
```

```
## $num_divergent
## [1] 0 0 0 0
##
## $num_max_treedepth
## [1] 0 0 0 0
##
## $ebfmi
## [1] 1.0531172 1.0484517 0.9710891 1.0480584
```

```
samps.mat2[ , , "beta_base"] %>%
  as.data.frame() %>%
  dplyr::mutate(Sample = seq(dplyr::n())) %>%
  pivot_longer(-Sample, values_to = "Estimate", names_to = "Chain") %>% {
  ggplot(., aes(Sample, Estimate)) + geom_line(aes(colour = Chain)) +
      scale_color_brewer(palette = "Dark2")
}
```

```
(samps.mat2[ , , "pop_sero"] / nrow(dataset2.log)) %>%
  as.data.frame() %>%
  dplyr::mutate(Sample = seq(dplyr::n())) %>%
  pivot_longer(-Sample, values_to = "Estimate", names_to = "Chain") %>% {
  ggplot(., aes(Sample, Estimate)) + geom_line(aes(colour = Chain)) +
      scale_color_brewer(palette = "Dark2")
}
```

C) Summarize model output for downstream model comparisons

```
quants <- c(0.025, 0.975, 0.500)
## get CI for predictions of interest
dataset2.risk_coef <- data.frame(
  baseline = samps.simp2$beta_base %>% quantile(quants)
, cat1f    = samps.simp2$beta_cat1f_delta %>% quantile(quants)
, cat2f    = samps.simp2$beta_cat2f_delta %>% quantile(quants)
, con1f    = samps.simp2$beta_con1f_delta %>% quantile(quants)
) %>% t() %>% as.data.frame() %>%
  mutate(
      dataset  = 2
    , approach = "Normal-Normal LCR"
```

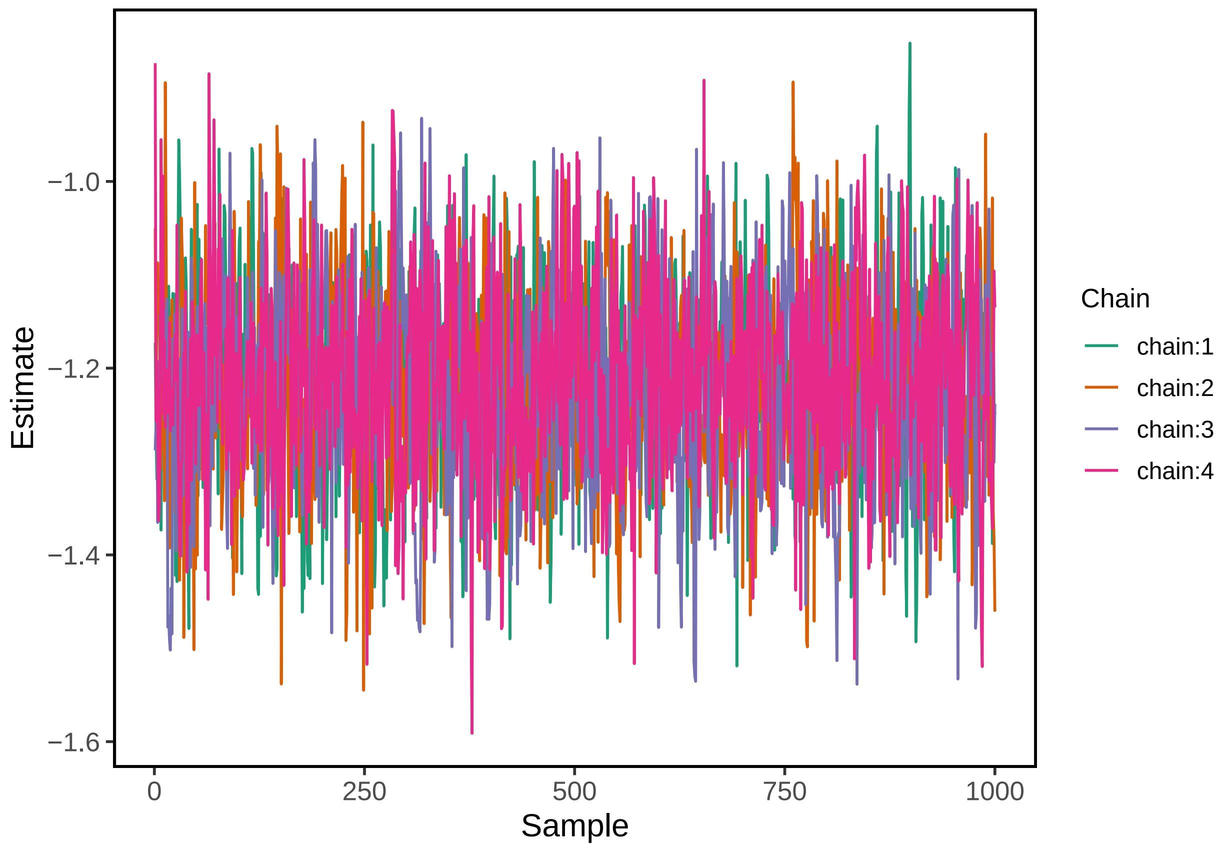

Figure 2: Traceplot for baseline seropositivity (`Intercept`) parameter

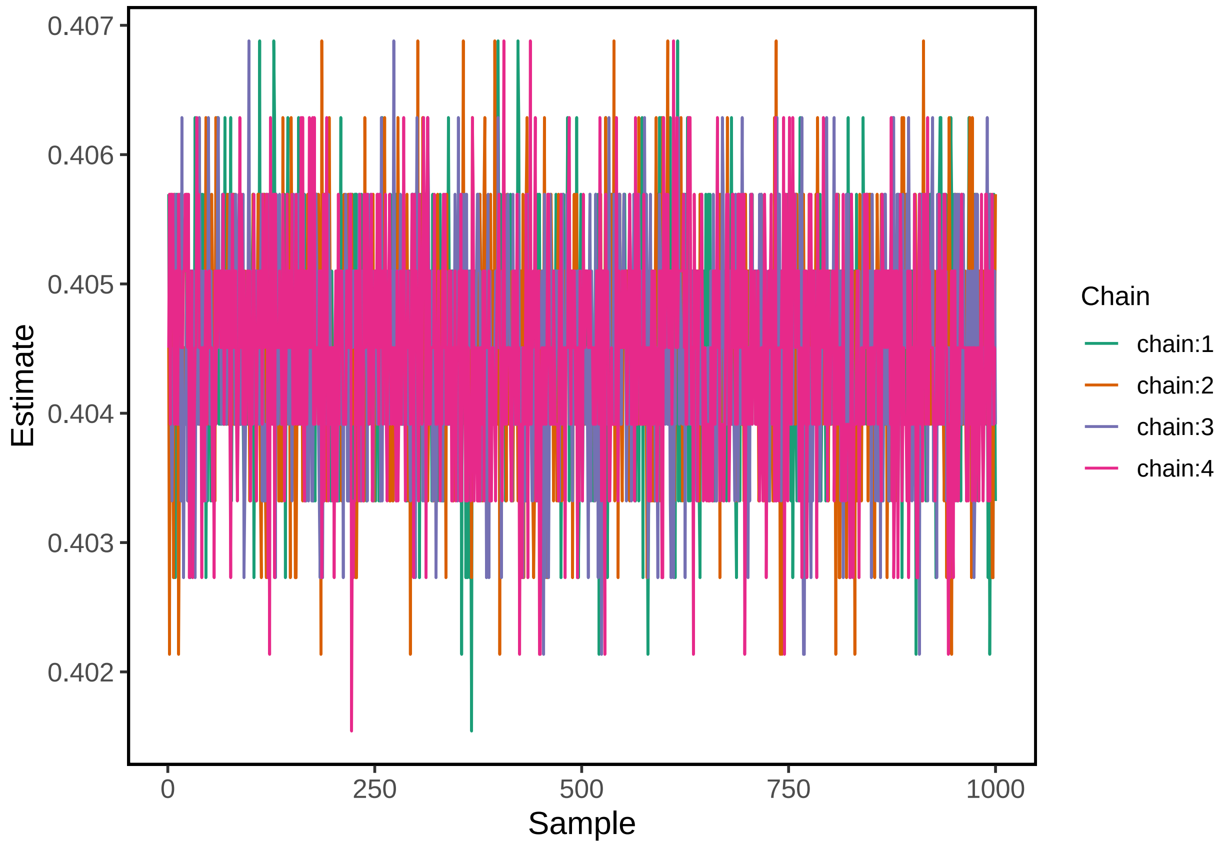

Figure 3: Traceplot for seroprevalence estimates

```
    , coef     = rownames(.)
    , .before  = 1
  ) %>% dplyr::rename(
    mid = `50%`, lwr = `2.5%`, upr = `97.5%`
    )

dataset2.pop_pos <- (samps.simp2$pop_sero/nrow(dataset2.log)) %>%
  quantile(quants) %>% t() %>%
  as.data.frame() %>%
  mutate(
      dataset  = 2
    , approach = "Normal-Normal LCR"
    , .before = 1
    ) %>% dplyr::rename(
    mid = `50%`, lwr = `2.5%`, upr = `97.5%`
    )
```

Dataset 3

A) Assign serostatus, predict seroprevalence, and risk factors for individual seropositivity

```
  ## just log mfi
  dataset3.log <- dataset3 %>% filter(log_mfi == "log_mfi")

  ## See code for function details. Establishes priors (which are sent in as data)
  stan_priors <- build_stan_priors(
    simulated_data = dataset3.log
  , skew_fit       = TRUE
    ## First trying with "naive" priors
  , fit_attempt    = 1
  )

  ## Compile model
  stan_model <- cmdstanr::cmdstan_model(
    "stan_models/publication_model_skew_normal_2.stan"
  , pedantic = FALSE)

  ## Fit model. For model definition see ".stan" in the online supplemental material
  stan_fit3    <- try(R.utils::withTimeout(stan_model$sample(
    data    = list(
        N           = length(dataset3.log$mfi)
      , y           = dataset3.log$mfi
      , cat1f       = dataset3.log$cat1f
      , cat2f       = dataset3.log$cat2f
      , con1f       = dataset3.log$con1f
      , beta_base_prior_m   = stan_priors$priors %>%
        filter(param == "beta_base_prior_m") %>% pull(prior)
      , beta_base_prior_v   = stan_priors$priors %>%
        filter(param == "beta_base_prior_v") %>% pull(prior)
      , mu_diff_prior_m     = stan_priors$priors %>%
        filter(param == "mu_diff_prior_m") %>% pull(prior)
      , mu_diff_prior_v     = stan_priors$priors %>%
        filter(param == "mu_diff_prior_v") %>% pull(prior)
      , mu_pos_prior_m      = stan_priors$priors %>%
        filter(param == "mu_pos_prior_m") %>% pull(prior)
      , mu_pos_prior_v      = stan_priors$priors %>%
        filter(param == "mu_pos_prior_v") %>% pull(prior)
      , sigma_base_prior_m  = stan_priors$priors %>%
        filter(param == "sigma_base_prior_m") %>% pull(prior)
      , sigma_diff_prior_m  = stan_priors$priors %>%
        filter(param == "sigma_diff_prior_m") %>% pull(prior)
      , sigma_base_prior_v  = stan_priors$priors %>%
        filter(param == "sigma_base_prior_v") %>% pull(prior)
      , sigma_diff_prior_v  = stan_priors$priors %>%
        filter(param == "sigma_diff_prior_v") %>% pull(prior)
      , skew_pos_prior_m    = min(dataset3.log %>%
                                   filter(group == 2) %>% pull(mfi) %>%
                                   moments::skewness(), -1)
      , skew_pos_prior_v    = 3
      , skew_neg_prior_m    = 0
      , skew_neg_prior_v    = 0.5
    )
    , init            = stan_priors$starting_conditions
    , chains          = 4
    , parallel_chains = 4
    , max_treedepth   = 12
    , iter_warmup     = 2000
    , iter_sampling   = 1000
    , adapt_delta     = 0.98
    , seed            = 48389
    , refresh         = 1000
  ), timeout = 3600
  ), silent  = TRUE)
```

```
## Running MCMC with 4 parallel chains...

## Chain 1 Iteration:    1 / 3000 [  0%]  (Warmup)

## Chain 2 Iteration:    1 / 3000 [  0%]  (Warmup)

## Chain 3 Iteration:    1 / 3000 [  0%]  (Warmup)

## Chain 4 Iteration:    1 / 3000 [  0%]  (Warmup)

## Chain 1 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 3 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 2 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 4 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 1 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 1 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 3 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 3 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 2 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 2 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 4 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 4 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 3 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3 finished in 108.9 seconds.
## Chain 1 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1 finished in 113.8 seconds.
## Chain 2 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2 finished in 118.6 seconds.
## Chain 4 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4 finished in 123.7 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 116.3 seconds.
## Total execution time: 123.9 seconds.
```

```
  ## Extract predictions
stanfit      <- rstan::read_stan_csv(stan_fit3$output_files())
samps.mat3   <- rstan::extract(stanfit, permuted = FALSE)
samps.simp3  <- rstan::extract(stanfit, permuted = TRUE)
```

B) Some diagnostics for this second fit

```
stan_fit3$diagnostic_summary()
```

```
## $num_divergent
## [1] 0 0 0 0
##
## $num_max_treedepth
## [1] 0 0 0 0
##
## $ebfmi
## [1] 0.9814353 0.9542093 0.9752124 0.9522289
```

```
samps.mat3[ , , "beta_base"] %>%
  as.data.frame() %>%
  dplyr::mutate(Sample = seq(dplyr::n())) %>%
  pivot_longer(-Sample, values_to = "Estimate", names_to = "Chain") %>% {
  ggplot(., aes(Sample, Estimate)) + geom_line(aes(colour = Chain)) +
      scale_color_brewer(palette = "Dark2")
  }
```

```
(samps.mat3[ , , "pop_sero"] / nrow(dataset3.log)) %>%
  as.data.frame() %>%
  dplyr::mutate(Sample = seq(dplyr::n())) %>%
  pivot_longer(-Sample, values_to = "Estimate", names_to = "Chain") %>% {
  ggplot(., aes(Sample, Estimate)) + geom_line(aes(colour = Chain)) +
      scale_color_brewer(palette = "Dark2")
}
```

C) Summarize model output for downstream model comparisons

```
## get CI for predictions of interest
quants <- c(0.025, 0.975, 0.500)

dataset3.risk_coef <- data.frame(
  baseline = samps.simp3$beta_base %>% quantile(quants)
, cat1f    = samps.simp3$beta_cat1f_delta %>% quantile(quants)
, cat2f    = samps.simp3$beta_cat2f_delta %>% quantile(quants)
, con1f    = samps.simp3$beta_con1f_delta %>% quantile(quants)
) %>% t() %>% as.data.frame() %>%
  mutate(
      dataset  = 3
    , approach = "Normal-Skew Normal LCR"
```

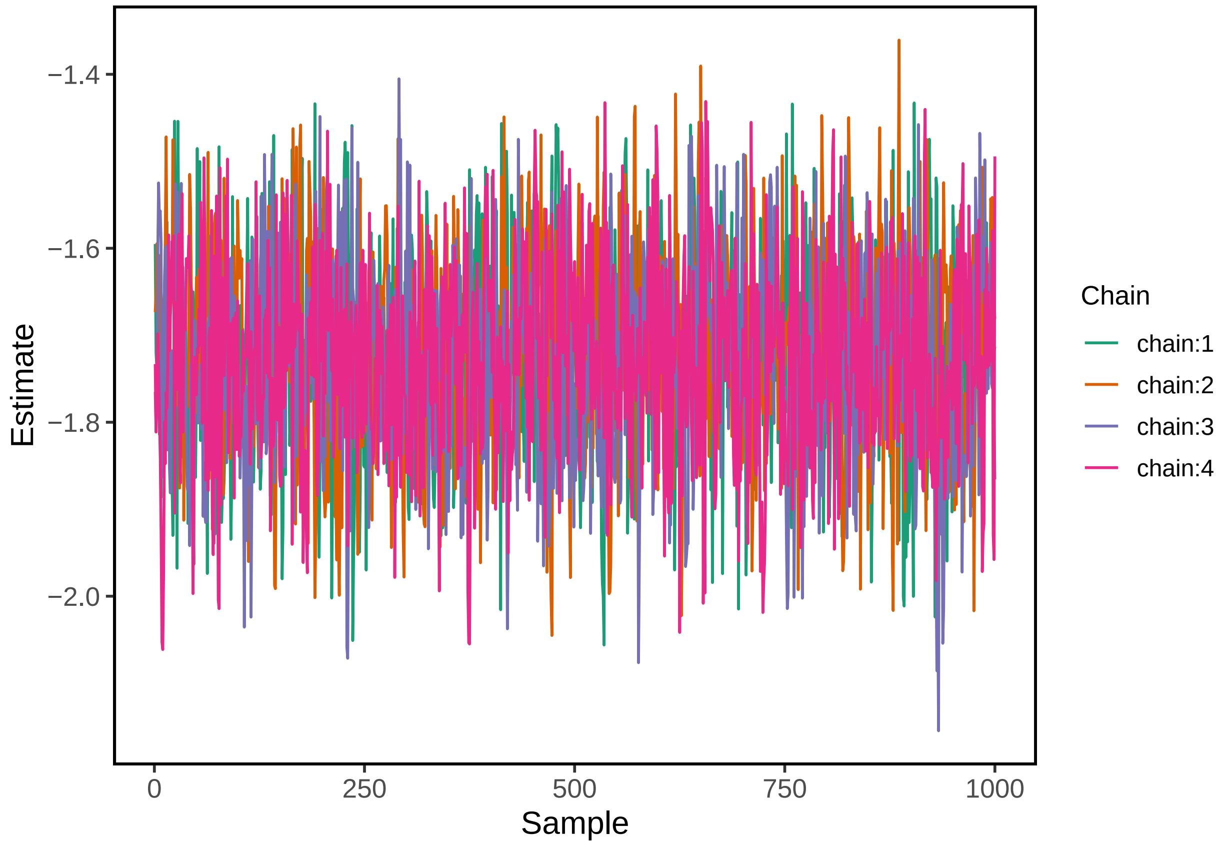

Figure 4: Traceplot for baseline seropositivity (`Intercept`) parameter

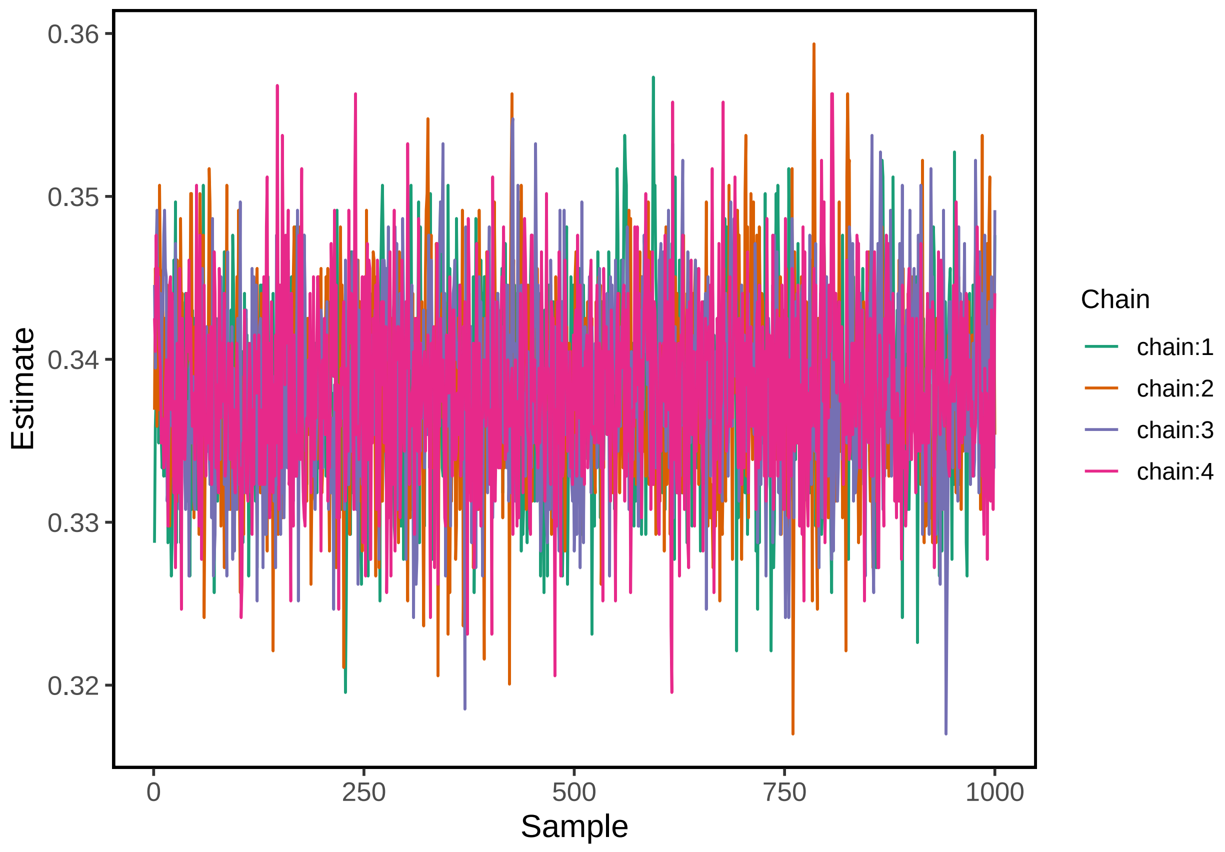

Figure 5: Traceplot for seroprevalence estimates

```
    , coef     = rownames(.)
    , .before  = 1
  ) %>% dplyr::rename(
    mid = `50%`, lwr = `2.5%`, upr = `97.5%`
  )

dataset3.pop_pos <- (samps.simp3$pop_sero/nrow(dataset3.log)) %>%
  quantile(quants) %>% t() %>%
  as.data.frame() %>%
  mutate(
      dataset  = 3
    , approach = "Normal-Skew Normal LCR"
    , .before = 1
    ) %>% dplyr::rename(
    mid = `50%`, lwr = `2.5%`, upr = `97.5%`
    )
```

Comparison to truth

```
pop_pos_pred   <- rbind(dataset1.pop_pos, dataset2.pop_pos, dataset3.pop_pos)
risk_coef_pred <- rbind(dataset1.risk_coef, dataset2.risk_coef, dataset3.risk_coef)
```

Individual seropositivity, dataset 1, analyzed with a 3sd approach: visualized in Figure 6. This analysis resulted in five false-positives (true positive assigned negative; positive-negative), 13 false-negatives (negative-positive), and many correct negative-negative and positive-positive values.

```
dataset1.linear_mfi %>% mutate(group = as.numeric(group) - 1) %>%
  mutate(pred_c = seropositive - group) %>% {
  ggplot(., aes(mfi, pred_c)) +
    geom_jitter(height = 0.1, alpha = 0.3) +
    geom_vline(xintercept = thresh, linetype = "dashed") +
      scale_y_continuous(breaks = c(-1, 0, 1), labels = c(
        "False-Negative", "Correct", "False-Positive")) +
      xlab("MFI") + ylab("Prediction")
}
```

Individual seropositivity, dataset 2, analyzed with a Normal-Normal LCR approach: visualized in Figure 7. With these clearly separated distributions the mixture model is highly accurate; that is, few true seropositive individuals are given a high probability of being seronegative and vice versa.

```
dataset2.log %>% mutate(
  pred_prob = samps.simp2$membership_p[,1,] %>% colMeans()
) %>% mutate(group = as.numeric(group) - 1) %>%
  mutate(mfi = exp(mfi)) %>% {
  ggplot(., aes(mfi, pred_prob)) +
    geom_point(colour = "dodgerblue3") +
    geom_jitter(aes(mfi, group), height = 0.1, alpha = 0.3) +
      scale_x_log10() +
    xlab("MFI") + ylab("Seropositive Probability")
}
```

Individual seropositivity, dataset 3, analyzed with a Normal-Skew Normal LCR approach: visualized in Figure 8. Because of the sizeable overlap in distributions and large left-skew in the seropositive MFI distribution, a large number of false-negatives arise (i.e., small positive probabilities assigned to true seropositivies). We note however that analyzing such a dataset with a 3sd approach using a robust mean and sd calculation

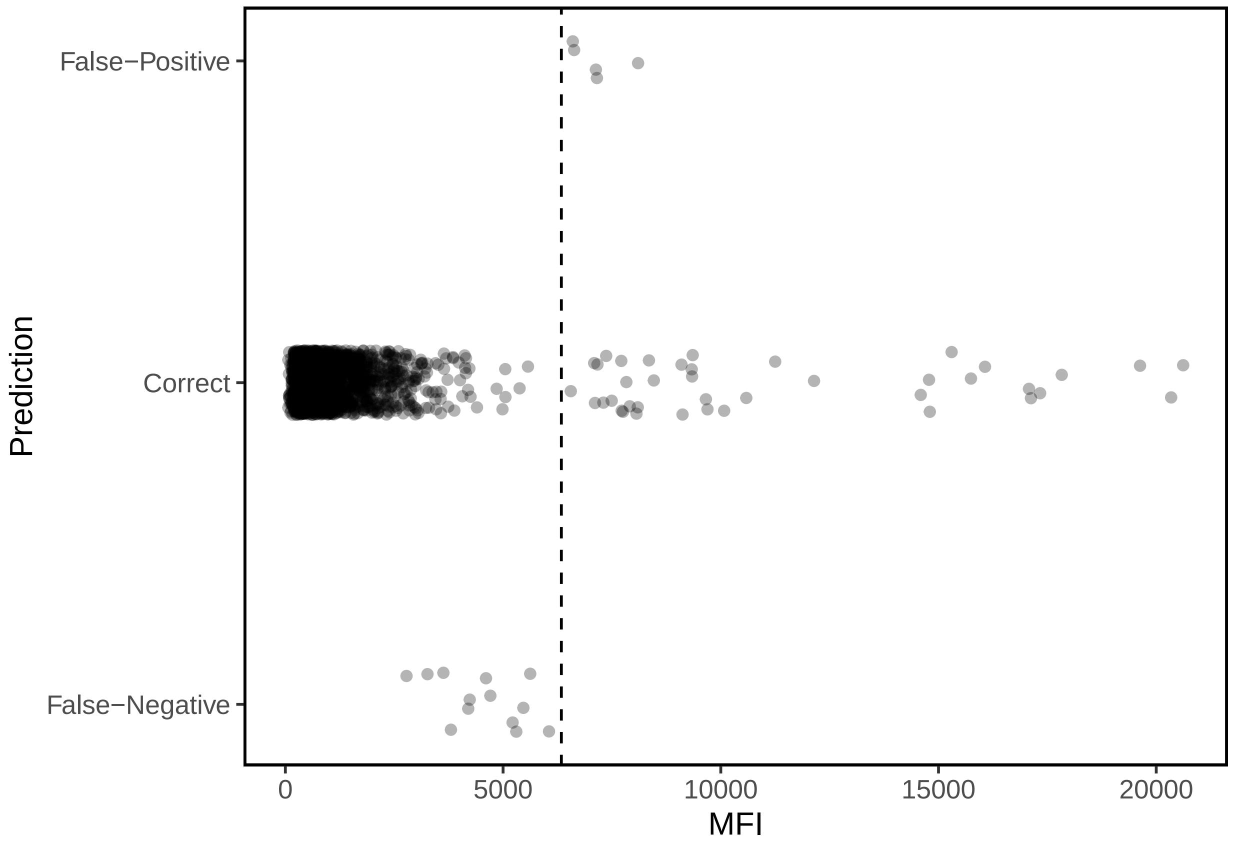

Figure 6: Serostatus assignments. Dashed vertical line gives 3sd cutoff; values to the left are assigned a negative status, values to the right a positive status.

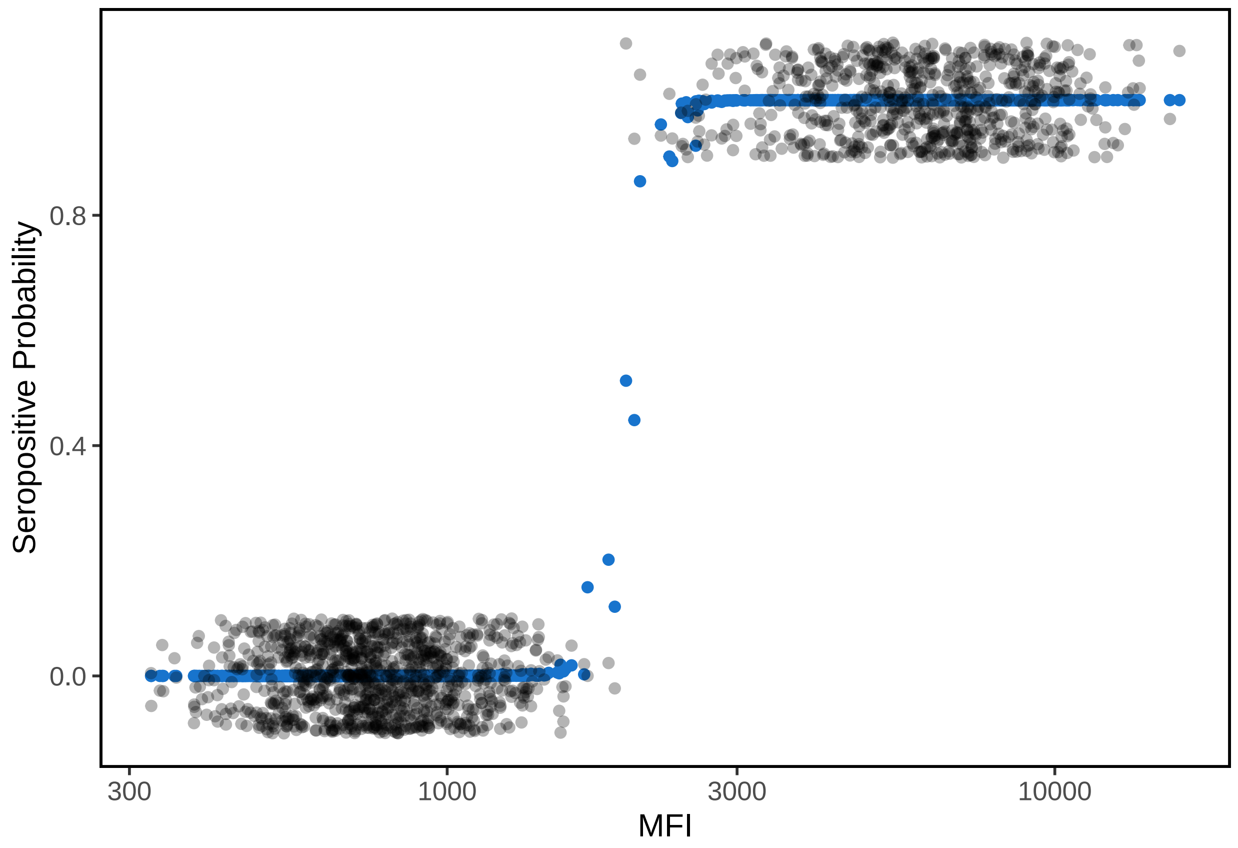

Figure 7: Seropositive probability estimate (median) shown in blue; true serostatus shown as jittered black points (0 = seronegative, 1 = seropositive).

would lead to 100% of seropositives being assigned a seronegative status.

```
dataset3.log %>% mutate(
  pred_prob = samps.simp3$membership_p[,1,] %>% colMeans()
) %>% mutate(group = as.numeric(group) - 1) %>%
  mutate(mfi = exp(mfi)) %>% {
  ggplot(., aes(mfi, pred_prob)) +
    geom_point(colour = "dodgerblue3") +
    geom_jitter(aes(mfi, group), height = 0.1, alpha = 0.3) +
      scale_x_log10() +
    xlab("MFI") + ylab("Seropositive Probability")
}
```

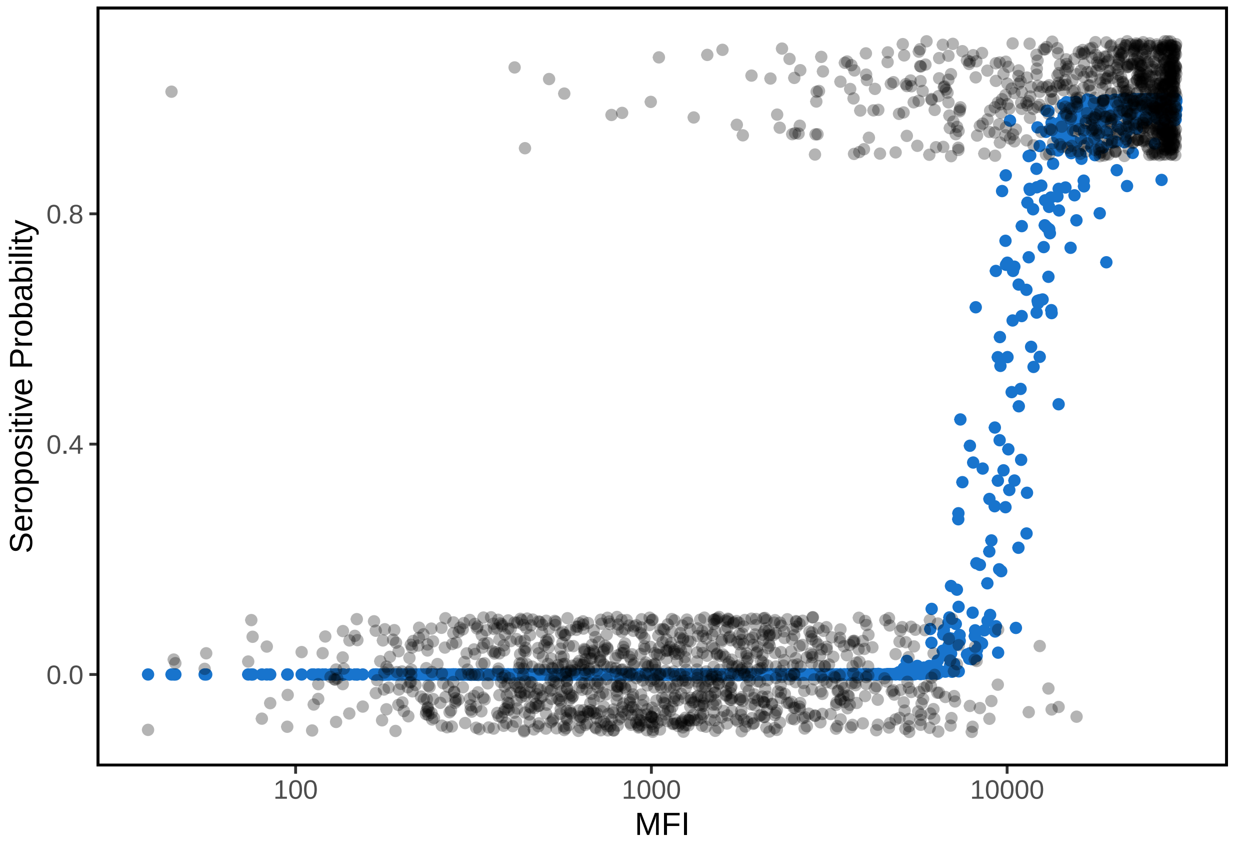

Figure 8: Seropositive probability estimate (median) shown in blue; true serostatus shown as jittered black points (0 = seronegative, 1 = seropositive).

Population seropositivity, all datasets. Estimates for the two LCR models show high confidence (narrow CI) because of high confidence in the shapes of the two distributions. This leads to over-confidence (and thus CI that do not cover the true value) for the Normal-Skew Normal model.

```
data.frame(
  dataset  = c(1, 2, 3)
  , true_pos = c(
    length(which(dataset1.linear_mfi$group == 2))/nrow(dataset1.linear_mfi)
    , length(which(dataset2.log$group == 2))/nrow(dataset2.log)
    , length(which(dataset3.log$group == 2))/nrow(dataset3.log)
  )
) %>% left_join(
  ., pop_pos_pred, by = "dataset"
) %>% mutate(
    dataset = paste("Dataset", dataset)
  , app_set = interaction(dataset, approach, sep = ": ")
  , app_set = factor(app_set
                   , levels = rev(unique(app_set)))
) %>% {
  ggplot(., aes(true_pos, app_set)) +
    geom_errorbarh(aes(xmin = lwr, xmax = upr), height = 0.2) +
    geom_point(aes(x = mid)) +
    geom_point(colour = "firebrick3", shape = 10, size = 3)
}
```

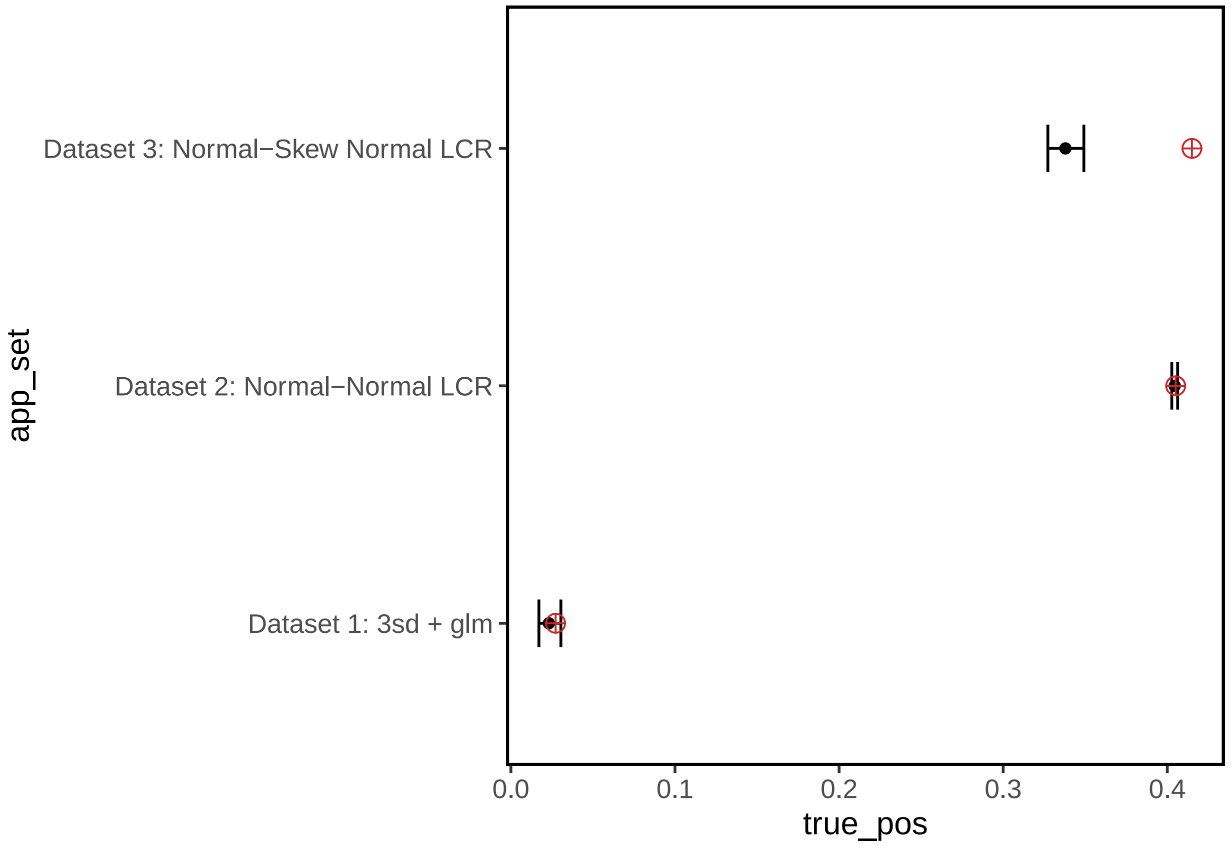

Figure 9: True vs estimated seroprevalence. Intervals are 95% CI.

Risk factors (regression coefficients). Here, because of the low seroprevalence in dataset 1, power is very low to estimate the effects of risk factors on seropositivity and thus CI on most of the coefficients for the 3sd + logistic regression are very wide. This is in contrast to using the two step 3sd + logistic regression approach in general. CI on these regression coefficients tend to be much too narrow (undercover) when seroprevalence is higher (see main text). As was found for a number of other LCR model fits (see main text for more details), for the Normal-Skew Normal example fit here, one level of the categorical covariates (the “Intercept”) was estimated to be too large and another too small.

```
risk_coef_pred %>% left_join(., sim.params.t, by = c("dataset", "coef")) %>% mutate(
  dataset = paste("Dataset", dataset)
  , app_set = interaction(dataset, approach, sep = ": ")
  , app_set = factor(app_set, levels = rev(unique(app_set)))
```

```r
) %>% {
  ggplot(., aes(true, app_set)) +
    geom_errorbarh(aes(xmin = lwr, xmax = upr), height = 0.2) +
    geom_point(aes(x = mid)) +
    geom_point(colour = "firebrick3", shape = 10, size = 3) +
    facet_wrap(~coef, scales = "free_x")
}
```

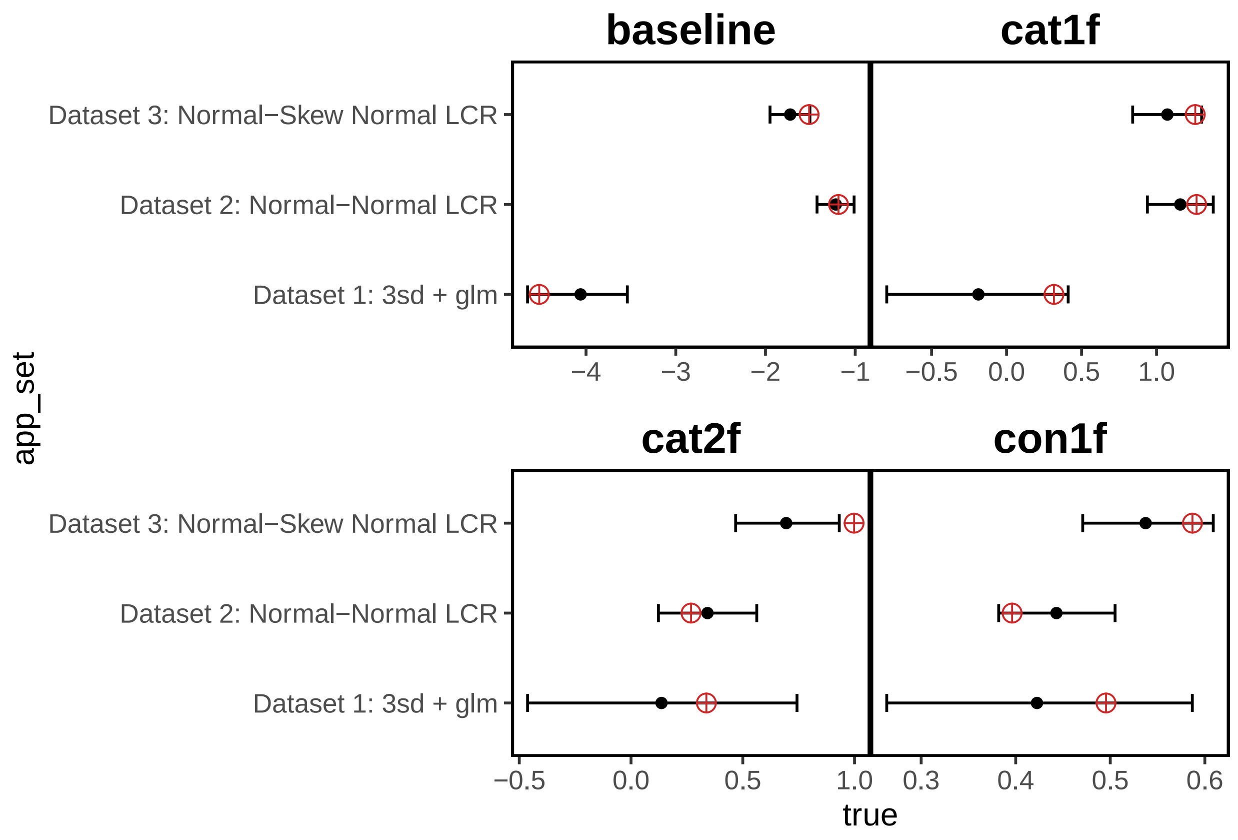

Figure 10: True vs estimated risk factors (regression coefficients). Intervals are 95% CI.

Bonus Model: Among Plate Variation

Here we examine a model fit to data with the addition of among-plate/run/batch variation in MFI values. We estimate this among-run variation using a random effect. First, we simulate data with the same structure as used in the above model fits, but with the addition of this source of variation, assuming that data is spread across 10 plates/runs/batches (not pictured in compiled pdf: see Rmd for details). In this simulation we assume that among-plate/run/batch variation manifests as a shift in the average value of both MFI distributions and random mixing seronegative and seropositive samples across plates/runs/batches. We do not allow for/model systematic differences in the proportion of samples seropositive among plates/runs/batches or a difference in seronegative and seropositive samples among plates/runs/batches. We then present parameter estimates for fixed effects (the same as the above models) and these random effects.

This simulated dataset has approximately 10% seropositivity and moderate among-plate variation

A) Fit model

```r
## Compile model
stan_model <- cmdstanr::cmdstan_model(
```

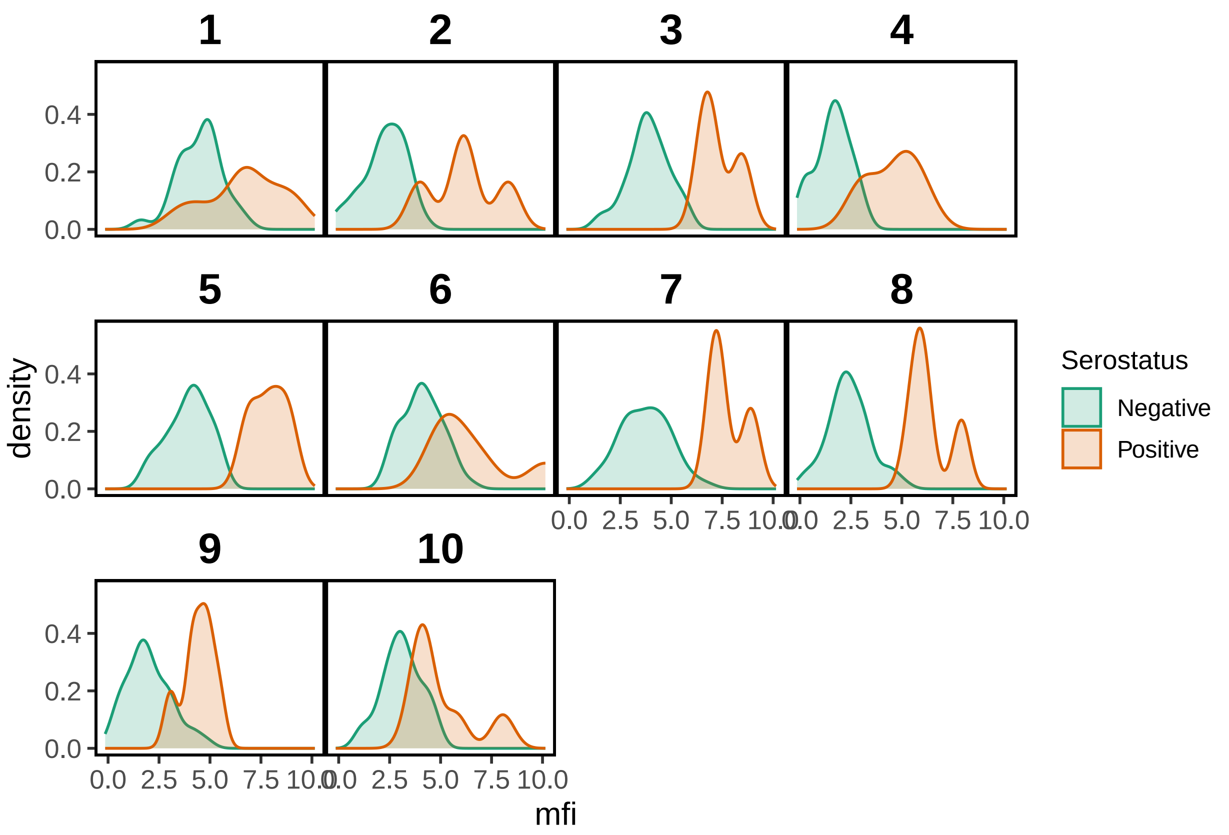

Figure 11: Simulated MFI distributions for seronegative and seropositive individuals among plates (panels)

```r
    "stan_models/cluster_regression_with_rand_plate_var.stan"
  , pedantic = FALSE)

  ## Fit model. For model definition see ".stan" in the online supplemental material
  stan_fit_bonus    <- try(R.utils::withTimeout(stan_model$sample(
    data    = list(
      N           = length(dataset_bonus$mfi)
      , N_cat1r     = n_distinct(dataset_bonus$cat1r)
      , y           = dataset_bonus$mfi
      , cat1f       = dataset_bonus$cat1f
      , cat2f       = dataset_bonus$cat2f
      , cat1r       = dataset_bonus$cat1r
      , con1f       = dataset_bonus$con1f

      , beta_base_prior_m   = stan_priors$priors %>%
        filter(param == "beta_base_prior_m") %>% pull(prior)
      , beta_base_prior_v   = stan_priors$priors %>%
        filter(param == "beta_base_prior_v") %>% pull(prior) / 4

      , mu_base_prior_m     = stan_priors$priors %>%
        filter(param == "mu_base_prior_m") %>% pull(prior) / 2
      , mu_base_prior_v     = stan_priors$priors %>%
        filter(param == "mu_base_prior_v") %>% pull(prior)

      , mu_diff_prior_m     = stan_priors$priors %>%
        filter(param == "mu_diff_prior_m") %>% pull(prior) * 2
      , mu_diff_prior_v     = 2

      , sigma_base_prior_m  = stan_priors$priors %>%
        filter(param == "sigma_base_prior_m") %>% pull(prior)
      , sigma_diff_prior_m  = stan_priors$priors %>%
        filter(param == "sigma_diff_prior_m") %>% pull(prior)

      , sigma_base_prior_v  = stan_priors$priors %>%
        filter(param == "sigma_base_prior_v") %>% pull(prior)
      , sigma_diff_prior_v  = stan_priors$priors %>%
        filter(param == "sigma_diff_prior_v") %>% pull(prior)
    )
    , init            = stan_priors$starting_conditions
    , chains          = 4
    , parallel_chains = 4
    , max_treedepth   = 12
    , iter_warmup     = 2000
    , iter_sampling   = 1000
    , adapt_delta     = 0.98
    , seed            = 1000006
    , refresh         = 1000
  ), timeout = 3600
  ), silent  = TRUE)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 3 Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 4 Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 2 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 3 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 4 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 1 Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 3 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 3 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 2 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 2 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 1 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 1 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 4 Iteration: 2000 / 3000 [ 66%]  (Warmup)
## Chain 4 Iteration: 2001 / 3000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1 finished in 43.4 seconds.
## Chain 4 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4 finished in 45.2 seconds.
## Chain 3 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3 finished in 49.5 seconds.
## Chain 2 Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2 finished in 51.8 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 47.5 seconds.
## Total execution time: 51.9 seconds.
```

```
## Extract predictions
stanfit      <- rstan::read_stan_csv(stan_fit_bonus$output_files())
samps.mat_bonus   <- rstan::extract(stanfit, permuted = FALSE)
samps.simp_bonus  <- rstan::extract(stanfit, permuted = TRUE)
```

B) Run some diagnostics and compare fitted values to truth

```
stan_fit_bonus$diagnostic_summary()
```

```
## $num_divergent
## [1] 0 0 0 0
##
## $num_max_treedepth
## [1] 0 0 0 0
##
## $ebfmi
## [1] 0.9263440 0.7465980 0.6774149 0.8521517
```

```
samps.mat_bonus[ , , "beta_base"] %>%
  as.data.frame() %>%
  dplyr::mutate(Sample = seq(dplyr::n())) %>%
  pivot_longer(-Sample, values_to = "Estimate", names_to = "Chain") %>% {
  ggplot(., aes(Sample, Estimate)) + geom_line(aes(colour = Chain)) +
      scale_color_brewer(palette = "Dark2")
}
```

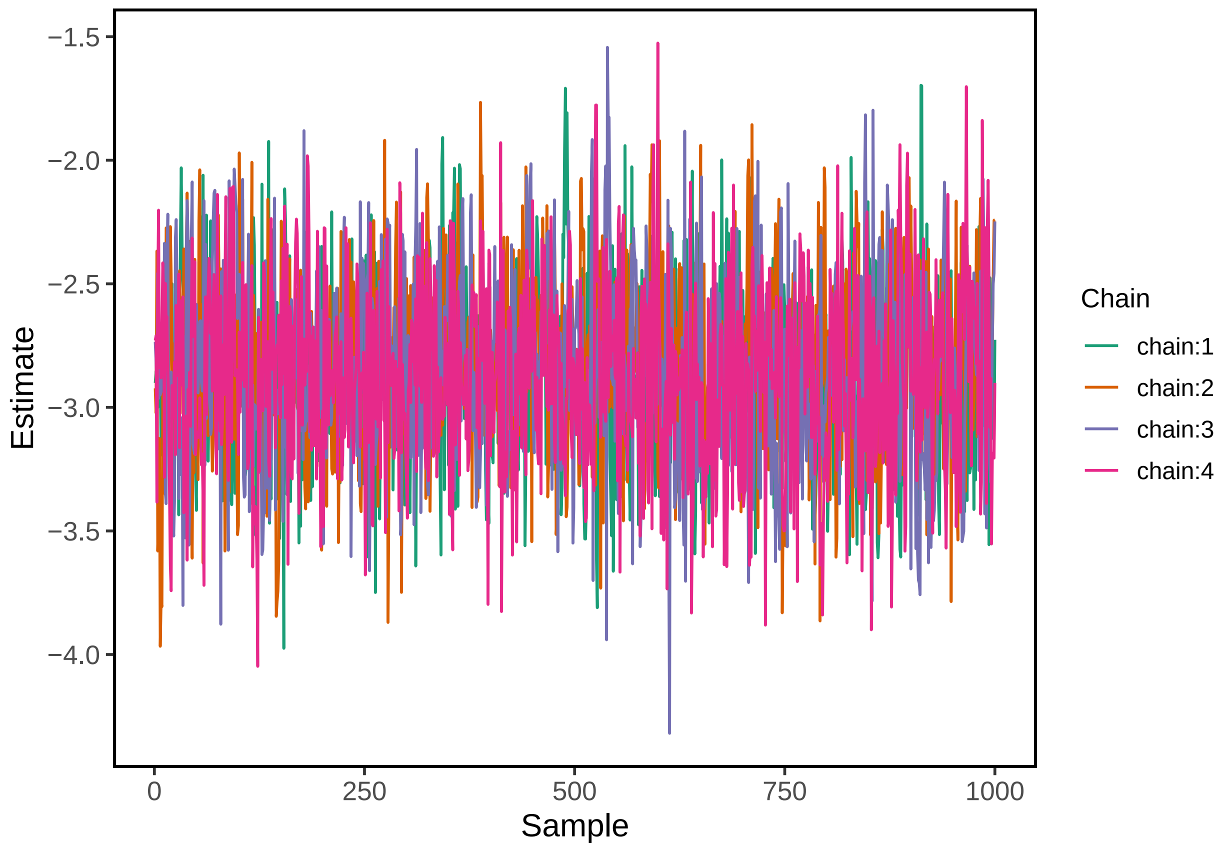

Figure 12: Traceplot for baseline seropositivity (`Intercept`) parameter

```
(samps.mat_bonus[ , , "pop_sero"] / nrow(dataset_bonus)) %>%
  as.data.frame() %>%
  dplyr::mutate(Sample = seq(dplyr::n())) %>%
  pivot_longer(-Sample, values_to = "Estimate", names_to = "Chain") %>% {
  ggplot(., aes(Sample, Estimate)) + geom_line(aes(colour = Chain)) +
      scale_color_brewer(palette = "Dark2")
}
```

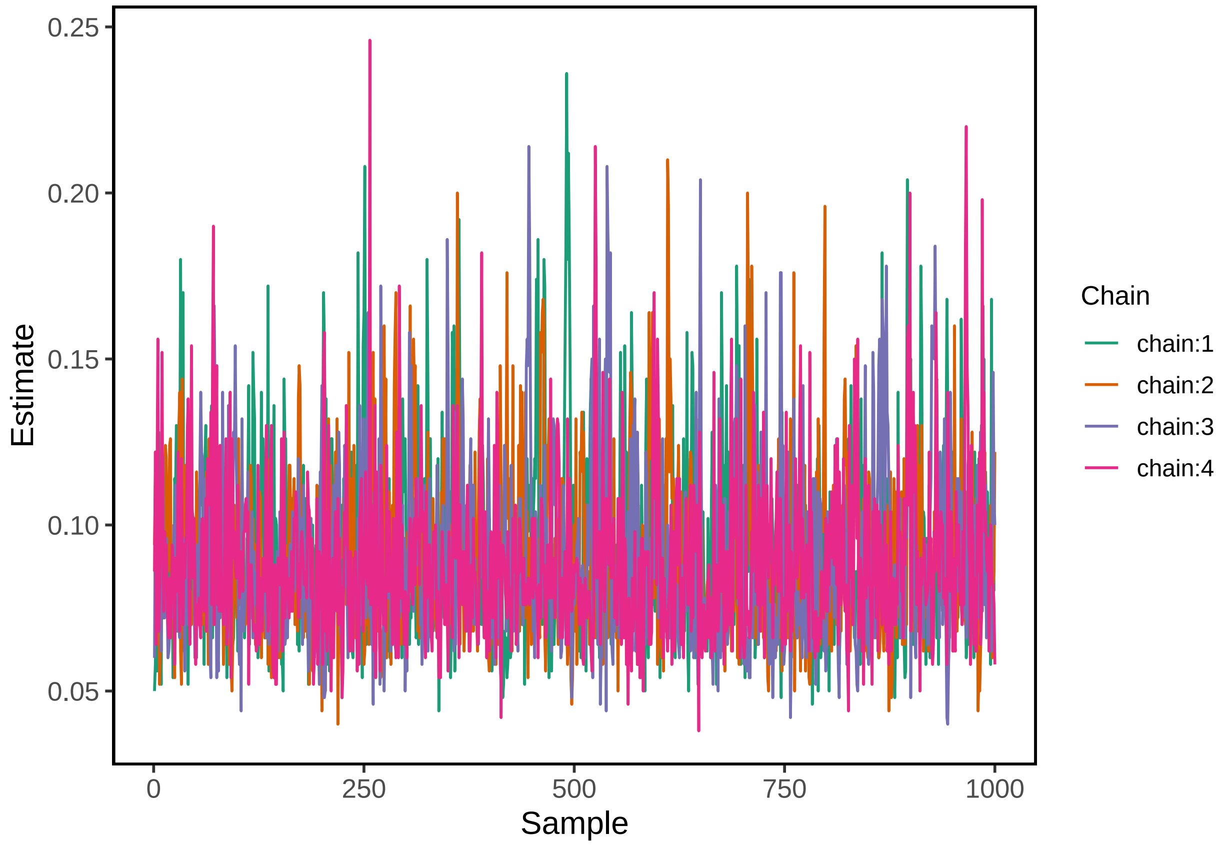

Figure 13: Traceplot for seroprevalence estimates

Estimated levels of the random effect match the real data well

```
data.frame(
  true = dataset_bonus %>% group_by(cat1r) %>% dplyr::summarize(mmfi = mean(mfi)) %>% pull(mmfi)
, pred = samps.mat_bonus[, , 10:19] %>% apply(., 3, c) %>% colMeans()
) %>% {
  ggplot(., aes(true, pred)) + geom_point() +
    xlab("Plate mean MFI") + ylab("Predicted random effect deviate
(conditional mode of the random effect)")
}
```

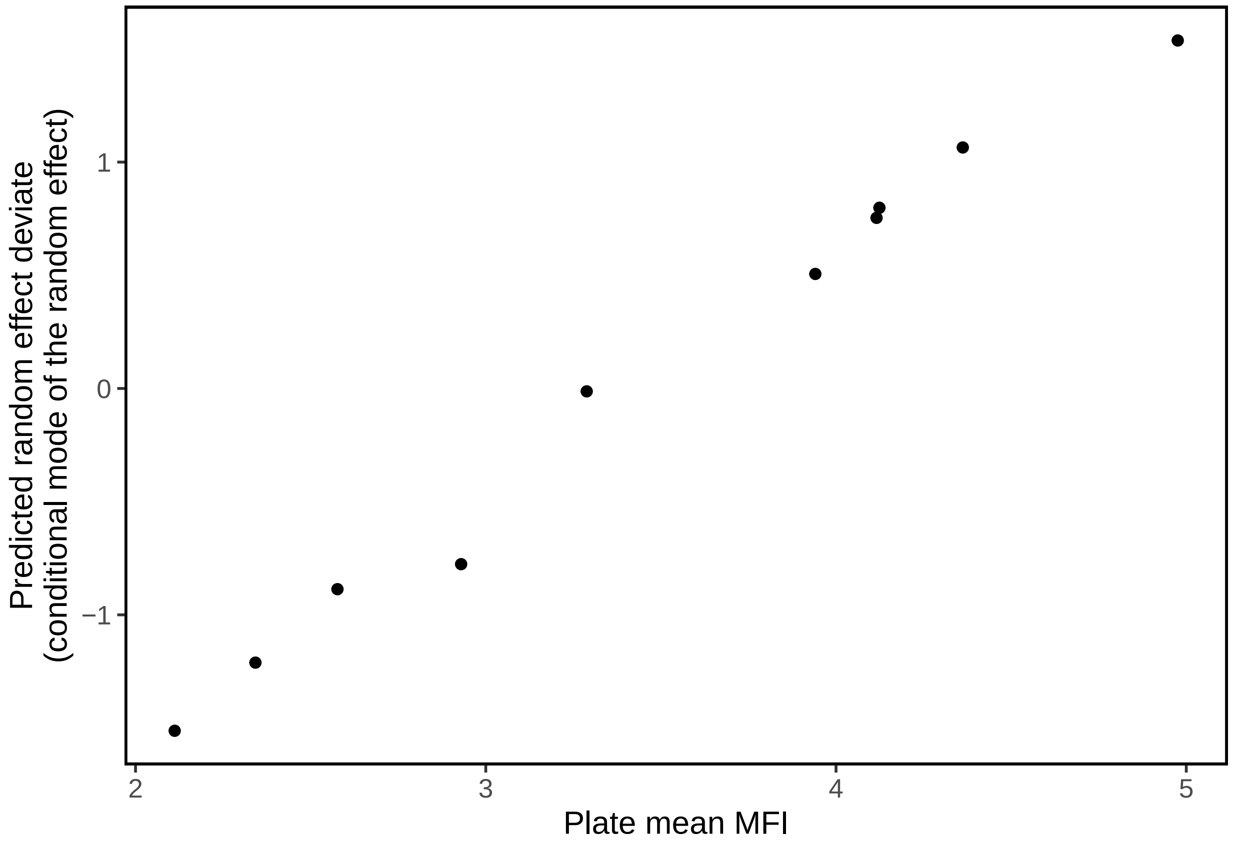

Figure 14: Estimated conditional modes of the random effect vs simulated plate-level mean MFI