Fitting and evaluating a binomial logit t model

Divan Burger and Sean van der Merwe

2024-09-18

Table of contents

Introduction **2**

Base sample **3**

Generating functions . . . 3
Parameters . . . 3
Example dataset . . . 4
Using real data . . . 4
Data illustration . . . 4

Models to be compared **5**

Binominal Logit t (BLT) Model **5**

Preparation and fitting function . . . 5
Fitting the model via posterior simulation . . . 7

Results **8**

Summary table . . . 8
Trace plot . . . 8

Fit evaluation and prediction **9**

Fixed effects linear predictor . . . 9
Random effects predictor . . . 10
Moving to the measurement scale . . . 11
Standardised residuals . . . 13
Probability mass function and likelihood . . . 16
Information criteria . . . 19
KL Divergence . . . 19

Binomial logit normal (BN) model **20**

Results . . . 22
Summary table . . . 22
Trace plot . . . 23
Fit evaluation and prediction . . . 23
Predictor . . . 23
Random effects predictor . . . 23
Standardised residuals . . . 24

Probability mass function and likelihood . . . 26
Information criteria . . . 27
KL Divergence . . . 27

Beta-Binomial **28**
Results . . . 30
Summary table . . . 30
Trace plot . . . 30
Fit evaluation and prediction . . . 31
Predictor . . . 31
Random effects predictor . . . 31
Standardised residuals . . . 32
Probability mass function and likelihood . . . 34
Information criteria . . . 35
KL Divergence . . . 35

Binomial **36**
Results . . . 38
Summary table . . . 38
Trace plot . . . 38
Fit evaluation and prediction . . . 39
Predictor . . . 39
Random effects predictor . . . 39
Standardised residuals . . . 39
Probability mass function and likelihood . . . 42
Information criteria . . . 43
KL Divergence . . . 43

Final comparison and summary **44**

Introduction

The binomial generalized linear model with logit link function is well known to accurately capture many types of real-world processes. However, it assumes a perfect relationship between the linear predictor and probability of success with no additional variability above that implied by the binomial distribution itself. This formulation often fails to accommodate the observed variability fully. When situations are encountered where the probability of success itself has additional random variation, it is natural to introduce an innovation term in the linear predictor. Further, this innovation term does not need to be restrictive. Where the additional unexplained variation is not homogeneous, additional flexibility can be brought in via a Student-t innovation. This accommodates both normal additional variation and extremely heavy tails. Such a model is far less affected by extreme and outlying values.

```r
options(scipen = 12)
library(tidyverse)
library(knitr)
library(runjags)
theme_set(theme_minimal()) # Set preferred ggplot theme here
```

Base sample

For testing a base sample is generated from the proposed distribution that includes some extreme observations, by defining a linear predictor on $logit\,\pi$ of the $Bin(n, \pi)$ distribution, but with extra innovations on the predictor, via a t density.

Generating functions

Sample generating functions are defined. First a function for the binomial logistic t (BLT) distribution given location and scale, then a function for a dataset with the desired example properties.

```r
rBLT <- function(n, m, mu, sigma_logit, nu) {
  t_values <- rt(n, nu)*sigma_logit + mu
  rbinom(n, m, plogis(t_values))
}

rBLTdataset <- function(params) {
  Sigma <- matrix(c(
      params$sigma_a_0^2, params$rho_a_01*params$sigma_a_0*params$sigma_a_1,
      params$rho_a_01*params$sigma_a_0*params$sigma_a_1, params$sigma_a_1^2
    ), nrow = 2)
  inv_Sigma <- solve(Sigma)
  inv_omega2 <- 1/params$omega^2
  N <- params$N
  random <- MASS::mvrnorm(N, mu = c(0, 0), Sigma = Sigma)
  group <- sample(c(0, 1), N, replace = TRUE)
  cov_cnt <- rnorm(N, 60, 5)
  time <- 0:params$N_time
  n_times <- length(time)
  dataset <- seq_len(N) |> lapply(\(i) {
    y <- rBLT(n_times, params$m,
              mu = params$beta_0 + random[i, 1] + random[i, 2]*time +
                params$beta_time*time + params$beta_xt*group[i]*time +
                params$beta_cnt*cov_cnt[i] + params$beta_grp*group[i],
              sigma_logit = params$omega, nu = params$nu)
    data.frame(Subject = i, Time = time, Group = group[i],
               Y = y, cov_cnt = cov_cnt[i], m = params$m)
  }) |> do.call(rbind, args = _)
}
```

Parameters

The parameters for simulation are specified.

```r
params <- list(
  omega = 0.2,       # \sigma
  sigma_a_0 = 0.4,   # \sigma_{a_0}
  sigma_a_1 = 0.3,   # \sigma_{a_1}
  rho_a_01 = -0.5,   # \rho_{a_0, a_1}
  nu = 3,            # \nu
  beta_0 = 1.5,      # \beta_0
  beta_time = -0.1,# \beta_{\text{time}}
  beta_grp = 0,    # \beta_{group}
  beta_cnt = 0.01,   # \beta_{\text{cnt}}
  beta_xt = 0.2,     # \beta_{\text{tx}}
  N = 120,           # Number of subjects
  N_time = 12,       # Number of timepoints post baseline
  m = 30             # Fixed binomial count parameter
)
```

Example dataset

Then a sample is generated.

```
set.seed(1)
dataset <- rBLTdataset(params)
```

Using real data

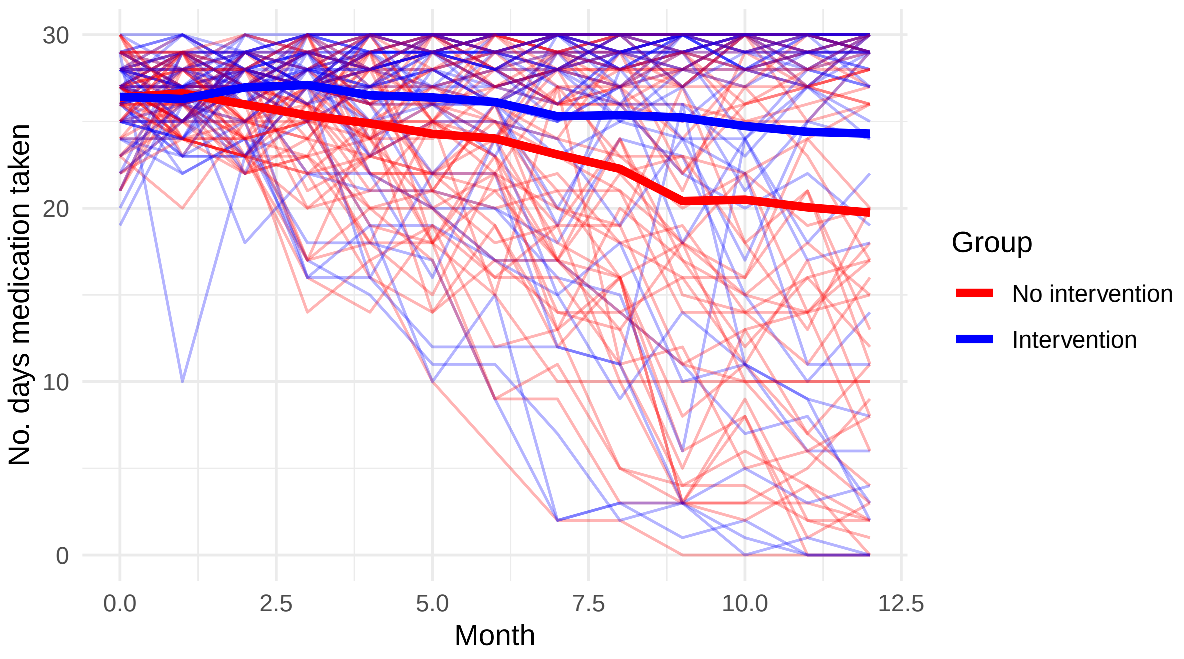
For a real dataset, ensure that columns Time, Y, Group, Subject, and m exist with those names. Subject must be recoded to 1, 1, ..., 1, 2, 2, ... with no gaps. Group should be 0, 1. Time should be 0, 1, ... but may have gaps.

```
# Read in data set from file
Compliance <- "compliance.csv" |> read.csv()
dataset <- data.frame(
  Time = Compliance$StudyMonth,
  Subject = Compliance$ID,
  Group = Compliance$int,
  cov_cnt = Compliance$Age,
  Y = Compliance$nummed,
  m = Compliance$dur
)
```

Data illustration

Illustrating the raw data is critical for checking its integrity.

```
# Profile plot
dataset |> ggplot(aes(x = Time, y = Y, group = Subject, colour = as.factor(Group))) +
    geom_line(alpha = 0.3) +
    stat_summary(fun = mean, geom = "line", linewidth = 1.5, aes(group = as.factor(Group))) +
    labs(x = "Month",
         y = "No. days medication taken",
         color = "Group") +
    scale_color_manual(values = c("red", "blue"),
                       labels = c("No intervention", "Intervention"))
```



Models to be compared

A list of models is first created to store the information that will be used for model comparison at the end.

```
models <- list(
  BLT = list(Name = "Binomial-logit-$t$"),
  BLN = list(Name = "Binomial-logit-normal"),
  BB = list(Name = "Beta-binomial"),
  Bin = list(Name = "Binomial")
)
```

Binominal Logit t (BLT) Model

The model must be defined and then fitted to the example data set.

Preparation and fitting function

First the model is specified in JAGS notation. Then initial values are prepared and the model fitted according to the simulation strategy. These steps are captured in a single function for reproducibility. The same function can be applied to any sample with similar structure.

```r
# number of chains and simulations per chain are set globally because they are reused
# later for model comparison calculations
n_chains <- 4
n_sims_per_chain = 5000
fitBLT <- function(dataset, thin = 5, timeout = 3600) {
# Model specification
  ModelSpec <- "
    model {
      Psi[1, 1] ~ dgamma(0.5, 1/50^2)
      Psi[1, 2] <- 0
      Psi[2, 1] <- 0
      Psi[2, 2] ~ dgamma(0.5, 1/50^2)
      inv_Sigma ~ dwish(2*kappa*Psi, kappa + 1)
      for (i in 1:N) {
        a[i, 1:2] ~ dmnorm(mu[], inv_Sigma[,])
        a_0[i] <- a[i, 1]
        a_1[i] <- a[i, 2]
      }
      for (i in 1:N_total) {
        eta[i] <- beta_0 + a_0[id[i]] + a_1[id[i]]*time[i] + beta_grp*group[i]
        + beta_time*time[i] + beta_xt*group[i]*time[i] + beta_cnt*cov_cnt[i]
        tau[i] ~ dgamma(nu/2, nu/2)
        xi[i] ~ dnorm(eta[i], inv_omega2*tau[i])
        logit(p[i]) <- xi[i]
        y[i] ~ dbin(p[i], m[i])
      }
      beta_0 ~ dnorm(0, 0.001)
      beta_grp ~ dnorm(0, 0.001)
      beta_time ~ dnorm(0, 0.001)
      beta_xt ~ dnorm(0, 0.001)
      beta_cnt ~ dnorm(0, 0.001)
      sigma ~ dt(0, 0.25, 2)T(0, )
      inv_omega2 <- 1/sigma^2
      lambda ~ dexp(0.75)
      nu ~ dgamma(2, lambda)
      log_nu <- log(nu)
    }
  "
# Parameters to be monitored
  Monitor <- c("beta_0", "beta_time", "beta_grp", "beta_cnt", "beta_xt",
               "inv_omega2", "nu", "inv_Sigma[1,1]",
               "inv_Sigma[1,2]", "inv_Sigma[2,1]", "inv_Sigma[2,2]", 'sigma', 'a')
# Initial values
  Inits <- replicate(n_chains, list(
    beta_0 = 0,
    beta_time = 0,
    beta_xt = 0,
    beta_grp = 0,
    sigma = 1,
    lambda = 0.1,
    nu = 10,
    inv_Sigma = diag(2),
    Psi = matrix(c(1, NA, NA, 1), nrow = 2),
    .RNG.name = 'base::Mersenne-Twister',
    .RNG.seed = sample.int(1e5, 1)
  ), simplify = FALSE)
# Data reworked for JAGS
  JAGSData <- list(
    N = max(dataset$Subject),
    N_total = nrow(dataset),
    id = dataset$Subject,
    time = dataset$Time,
    group = dataset$Group,
    cov_cnt = dataset$cov_cnt,
    y = dataset$Y,
    m = dataset$m,
    mu = rep(0, 2),
    kappa = 2
  )
# JAGS is called to do the simulation
  Sample <- tryCatch({R.utils::withTimeout({
    runjags::run.jags(
      model = ModelSpec,
      data = JAGSData,
      inits = Inits,
      monitor = Monitor,
      n.chains = n_chains,
      burnin = n_sims_per_chain,
      thin = thin,
      sample = n_sims_per_chain,
      summarise = FALSE,
      method = 'parallel',
      modules = 'glm',
      factories = 'bugs::MNormal sampler off'
    )
  }, timeout = timeout, onTimeout = "silent")}, error = function(e) {NULL}
  )
# The posterior simulations are returned in a raw runjags format for now
  Sample
}
```

Fitting the model via posterior simulation

```r
post_sims_runjags <- dataset |> fitBLT()
```

```r
# The simulated posterior samples are converted to a data frame for easy manipulation
runjags_to_dataframe <- function(Sample, shuffle_samples = FALSE) {
  samples <- do.call(rbind, Sample$mcmc)
  if (shuffle_samples) {
    samples <- samples[sample(seq_len(nrow(samples)), nrow(samples)),] # Shuffle samples
  }
  samples |> as.data.frame()
}
post_sims <- post_sims_runjags |> runjags_to_dataframe()
```

Results

The posterior simulations are now evaluated in a number of ways.

Summary table

```r
models$BLT$summary <- post_summary_table <- post_sims_runjags |> summary()
```

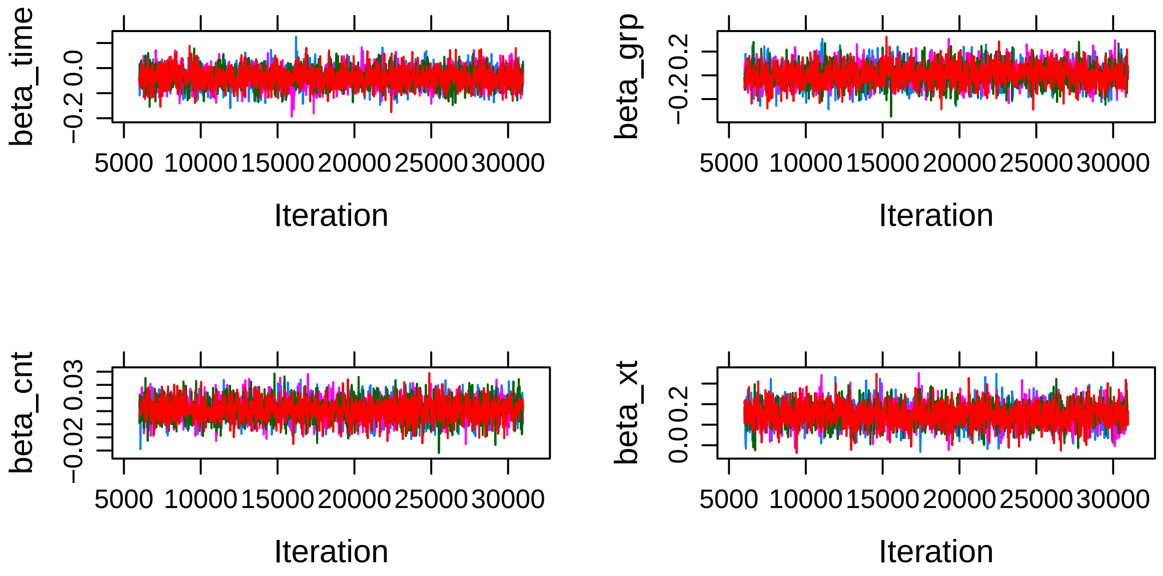
```r
post_summary_table[!startsWith(rownames(post_summary_table), "a"), ] |> kable(digits = 3)
```

	Lower95	Median	Upper95	Mean	SD	Mode	MCerr	MC%ofSD	SSeff	AC.50	psrf
beta_0	0.403	1.399	2.410	1.401	0.510	NA	0.006	1.3	6321	0.029	1.000
beta_time	-0.115	-0.038	0.036	-0.038	0.039	NA	0.000	0.8	14489	0.016	1.000
beta_grp	-0.169	0.011	0.188	0.011	0.091	NA	0.001	1.2	6714	0.022	1.000
beta_cnt	-0.006	0.011	0.027	0.011	0.008	NA	0.000	1.3	6251	0.029	1.000
beta_xt	0.038	0.148	0.256	0.148	0.056	NA	0.001	1.0	10358	0.027	1.001
inv_omega2	10.159	21.349	42.586	23.622	9.638	NA	0.579	6.0	277	0.742	1.003
nu	1.820	3.293	5.708	3.520	1.115	NA	0.064	5.7	302	0.742	1.003
inv_Sigma[1,1]	5.620	11.750	22.101	12.866	5.104	NA	0.117	2.3	1894	0.182	1.002
inv_Sigma[1,2]	1.762	5.932	11.559	6.324	2.707	NA	0.047	1.7	3305	0.076	1.003
inv_Sigma[2,1]	1.762	5.932	11.559	6.324	2.707	NA	0.047	1.7	3305	0.076	1.003
inv_Sigma[2,2]	10.334	14.762	19.824	14.984	2.478	NA	0.039	1.6	4113	0.075	1.000
sigma	0.146	0.216	0.291	0.216	0.037	NA	0.002	5.8	299	0.747	1.003

Trace plot

```r
fixed_effects <- c("beta_time", "beta_grp", "beta_cnt", "beta_xt")
plot(post_sims_runjags, vars = fixed_effects, plot.type = c('trace'), new.window = FALSE)
```

```
Generating summary statistics and plots (these will NOT be saved for
reuse)...
Calculating summary statistics...
Calculating the Gelman-Rubin statistic for 4 variables....
```



Fit evaluation and prediction

In this section we evaluate the posterior distribution of each observation, given the model fit. These can be compared to the observed values in various ways to evaluate the fit and diagnose issues.

The distributions will be built up in stages using functions to minimise code replication.

Fixed effects linear predictor

The linear predictor is first evaluated without the random effects. This is done so then when predictions are to be made we can use the same code again for both the average subject and a random future subject. The first function will thus be used in at least three ways.

It requires as input a data set that is already in the form of a model matrix (use the *model.matrix* function or construct a similar result) such that this matrix can be multiplied by the corresponding initial section of the posterior simulation data frame or matrix. The output of the function is a matrix of size *observations by simulations*.

As example, the function is evaluated for the existing data set.

```
lin_pred_eval <- function(Xmat, post_sims) {
  Xmat %*% t(as.matrix(post_sims[,seq_len(ncol(Xmat))]))
}
lin_pred <- model.matrix(~Time*Group + cov_cnt, data = dataset) |>
  lin_pred_eval(post_sims)
```

The function will now also be applied to a designed data frame that is centered on a specific set of values for the continuous predictors. This enables neat plotting, including beyond the initial time frame. Here we will use the average value for the continuous covariate.

```
n_times_smooth <- 100
extention_factor <- 1.0 # Change to say 1.2 to extend predictions 20% further
time_seq <- seq(0, ceiling(max(dataset$Time)*1.00), length = n_times_smooth)
new_X <- expand.grid(
  Intercept = 1,
  Time = time_seq,
  Group = unique(dataset$Group),
  cov_cnt = mean(dataset$cov_cnt)
) |> mutate(
  xt = Time*Group
) |> as.matrix()
models$BLT$lin_pred_ave <- lin_pred_ave <- new_X |> lin_pred_eval(post_sims)
```

Random effects predictor

Now we consider the task of evaluating the random effects for a given subject, first one that was observed in the sample and then a random future subject yet to be observed.

```
ran_effects_eval <- function(Time, post_sims, subj = 1) {
  ran_effects <- post_sims[[paste0("a[", subj, ",2]")]] %*% t(Time) +
                  post_sims[[paste0("a[", subj, ",1]")]]
  ran_effects |> t()
}
ran_effects_eval_future <- function(post_sims, time_vec) {
  inv_sigmas <- post_sims |> select(
    c("inv_Sigma[1,1]", "inv_Sigma[1,2]", "inv_Sigma[2,1]", "inv_Sigma[2,2]")
  ) |> as.matrix()
  inv_sigmas |> apply(1, \(Sinv) {
    S <- Sinv |> matrix(2) |> solve()
    effects <- MASS::mvrnorm(1, c(0,0), S)
    effects[2]*time_vec + effects[1]
  })
}
```

For each subject we evaluate their random effects. Should the dataset not be in subject order already then the resulting simulations must be reordered to match the dataset.

```
subjects_vec <- unique(dataset$Subject)
ran_effects_subjects <- subjects_vec |> lapply(\(subj) {
  ran_effects_eval(dataset$Time[dataset$Subject %in% subj], post_sims, subj)
}) |> do.call(rbind, args = _)
ran_effects_subjects <- ran_effects_subjects[order(dataset$Subject), ]
```

Now we add the random effects for the existing subjects to their linear predictor to obtain their expected values on the logit scale $logit(\hat{p})$. The same is done for the random future subject.

```
models$BLT$eta <- expected_values_logit <- lin_pred + ran_effects_subjects
```

Moving to the measurement scale

Moving to the measurement scale can be done either purely as a **median predictor**, by taking $m\hat{p}$, or by prediction, that is simulating a new binomial logit t observation centered on the predictor. For true expected values, one must integrate over the t nuisance variable. This is not done here, although the principle is discussed in detail further along in this document.

This will be illustrated by drawing plots for a selected subject, and for random future subjects with average values for continuous predictors.

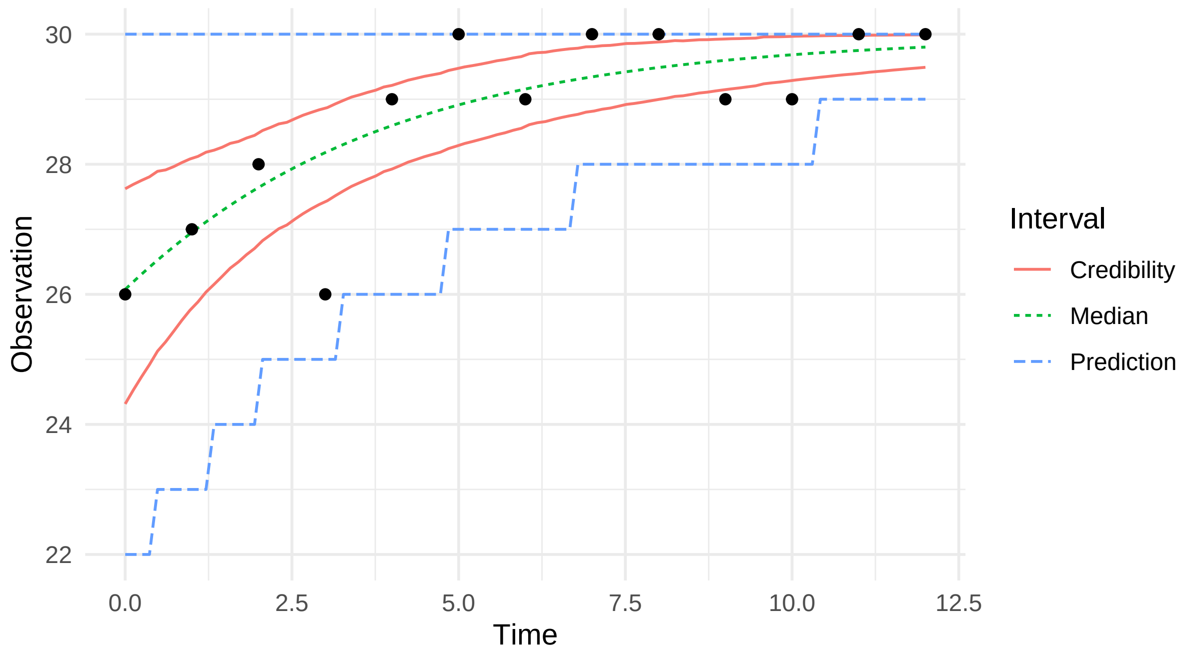
```
n_post_sims <- nrow(post_sims)
n_obs_total <- nrow(dataset)
models$BLT$median <- median_values_bin <- plogis(expected_values_logit)*dataset$m
models$BLT$pred <- predicted_values_bin <- n_post_sims |> seq_len() |> sapply(\(sim) {
  rBLT(n_obs_total, dataset$m, expected_values_logit[,sim], post_sims$sigma[sim],
       post_sims$nu[sim])
})
```

To calculate intervals we first define a function for HPD intervals as these are the shortest (most informative) intervals for a given coverage probability.

```
shortestinterval <- function(postsims, width=0.95) { # Coded by Sean van der Merwe, UFS
  postsims |> sort() -> sorted.postsims
  round(length(postsims)*width) -> gap
  sorted.postsims |> diff(gap) |> which.min() -> pos
  sorted.postsims[c(pos, pos + gap)] }
```

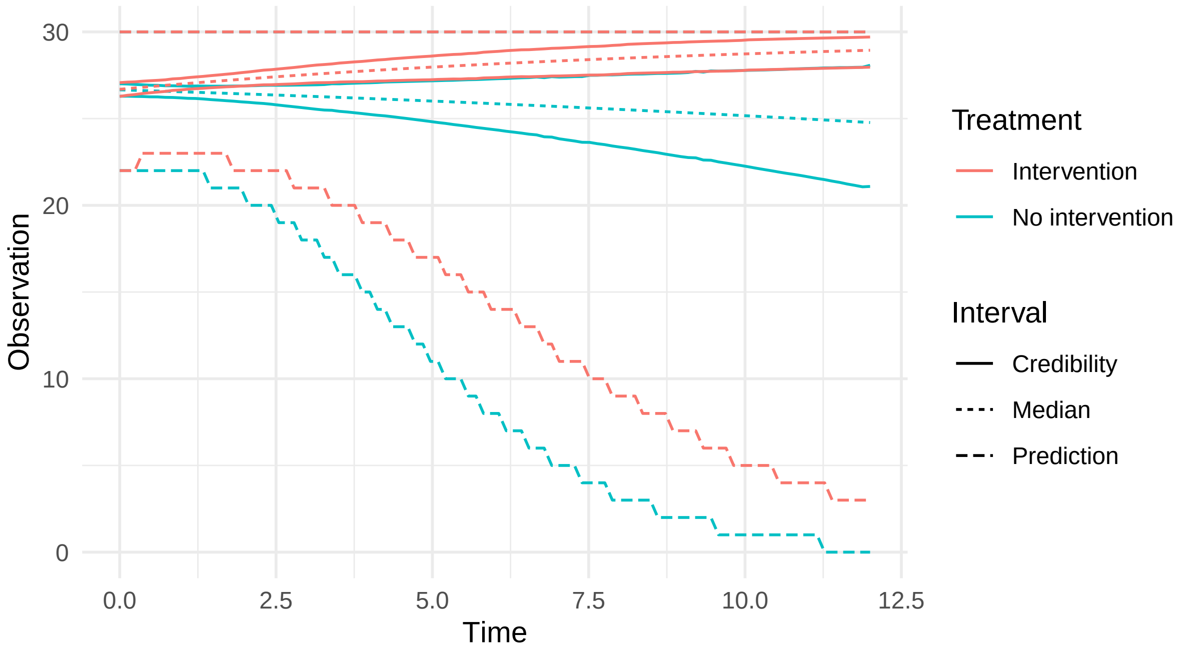
```
plot_subject <- function(subject = 1) {
  subject_rows <- which(dataset$Subject %in% subject)
  smooth_lin_pred_subj <- data.frame(
    Intercept = 1,
    Time = time_seq,
    Group = dataset$Group[subject_rows[1]],
    cov_cnt = dataset$cov_cnt[subject_rows[1]]
  ) |> mutate(
    xt = Time*Group
  ) |> as.matrix() |> lin_pred_eval(post_sims)
  smooth_random_value_subject <- ran_effects_eval(time_seq, post_sims, subject)
  sims <- smooth_lin_pred_subj + smooth_random_value_subject
  sims_trans <- plogis(sims)*dataset$m[subject_rows[1]]
  cred_int <- sims_trans |> apply(1, shortestinterval)
  predicted_values_bin <- n_post_sims |> seq_len() |> sapply(\(sim) {
    rBLT(n_times_smooth, dataset$m[subject_rows[1]], sims[,sim], post_sims$sigma[sim],
         post_sims$nu[sim])
  })
  pred_int <- predicted_values_bin |> apply(1, shortestinterval)
  smooth_subject <- data.frame(
    Time = time_seq,
    Median_value = sims_trans |> rowMeans(),
    Credibility_low = cred_int[1,],
    Credibility_high = cred_int[2,],
    Prediction_low = pred_int[1,],
    Prediction_high = pred_int[2,]
  ) |> pivot_longer(-Time, names_to = "Line", values_to = "Observation") |>
    mutate(Interval = Line |> str_replace("_.*", ""))
  smooth_subject |> ggplot(aes(x = Time)) +
    geom_line(aes(y = Observation, group = Line, colour = Interval, linetype = Interval)) +
    geom_point(aes( y = Y), data = dataset[subject_rows, ])
}
```

```
plot_subject(4)
```



```
Groups <- unique(dataset$Group) |> sort()
group_names <- c("No intervention", "Intervention")
future_plot <- function() {
  smooth_random_value_ave <- ran_effects_eval_future(post_sims, time_seq)
  sims_trans <- plogis(lin_pred_ave)*dataset$m[1]
  cred_int <- sims_trans |> apply(1, shortestinterval)
  predicted_values_bin <- n_post_sims |> seq_len() |> sapply(\(sim) {
    rBLT(nrow(new_X), dataset$m[1], lin_pred_ave[,sim] + smooth_random_value_ave[,sim],
         post_sims$sigma[sim], post_sims$nu[sim])
  })
  pred_int <- predicted_values_bin |> apply(1, shortestinterval)
  smooth_future <- data.frame(
    new_X,
    Median_value = sims_trans |> rowMeans(),
    Credibility_low = cred_int[1,],
    Credibility_high = cred_int[2,],
    Prediction_low = pred_int[1,],
    Prediction_high = pred_int[2,]
  ) |> pivot_longer(-seq_len(ncol(new_X)), names_to = "Line", values_to = "Observation") |>
    mutate(Interval = Line |> str_replace("_.*", ""),
           grouping = factor(paste(Group, Line)),
           Treatment = group_names[Group + 1])
  smooth_future |> ggplot(aes(x = Time)) +
    geom_line(aes(y = Observation, group = grouping, colour = Treatment, linetype = Interval))
}
```

```
future_plot()
```



Standardised residuals

Here the DHARMa approach is applied for residual analysis.

```
library(DHARMa)
```

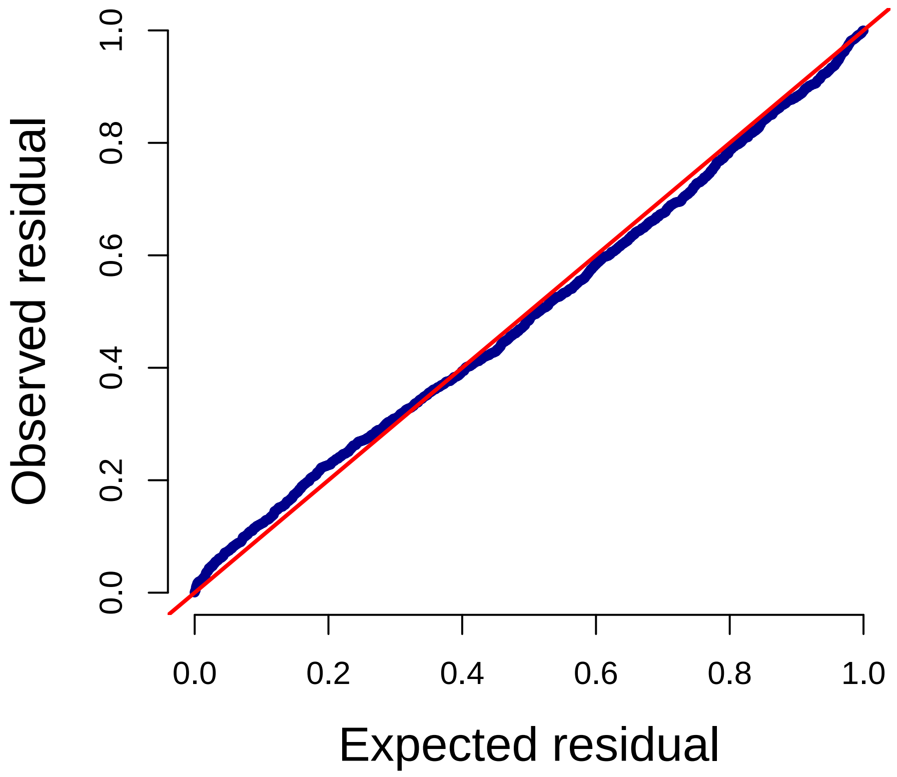
```
fitted <- median_values_bin |> apply(1, median)
DHARMa <- createDHARMa(
  simulatedResponse = predicted_values_bin,
  observedResponse = dataset$Y,
  fittedPredictedResponse = fitted
)
errors <- residuals(DHARMa)
```

```r
# plot(DHARMa)
ResidTest <- testResiduals(DHARMa, plot = FALSE)
```

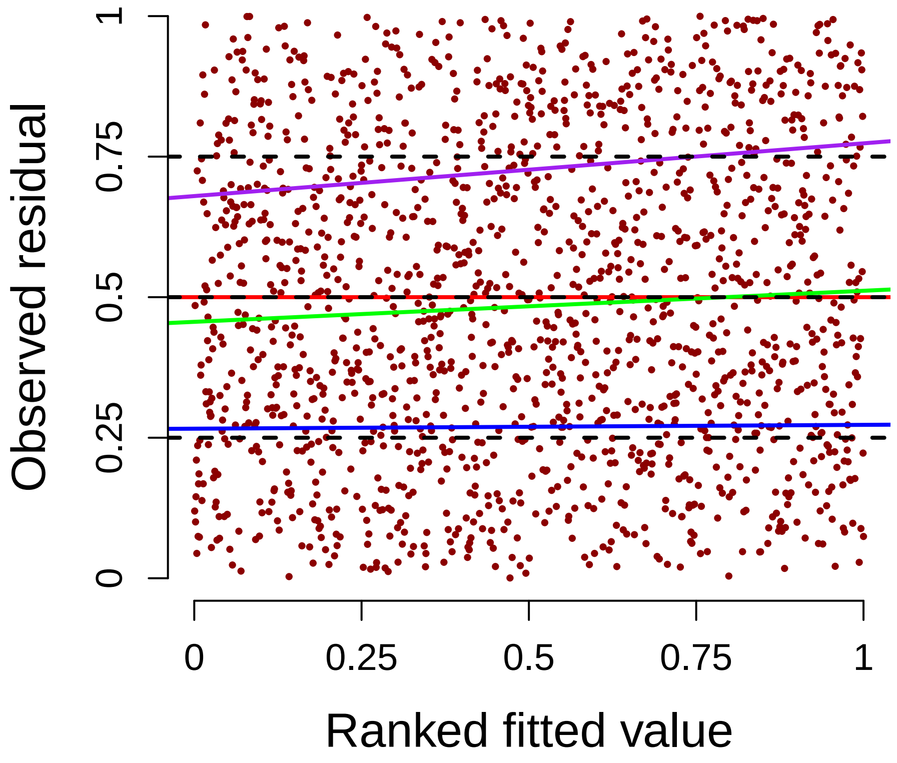
```
$uniformity

	Asymptotic one-sample Kolmogorov-Smirnov test

data:  simulationOutput$scaledResiduals
D = 0.033193, p-value = 0.06429
alternative hypothesis: two-sided


$dispersion

	DHARMa nonparametric dispersion test via sd of residuals fitted vs.
	simulated

data:  simulationOutput
dispersion = 0.81302, p-value = 0.0187
alternative hypothesis: two.sided


$outliers

	DHARMa outlier test based on exact binomial test with approximate
	expectations

data:  simulationOutput
outliers at both margin(s) = 0, observations = 1560, p-value = 1
alternative hypothesis: true probability of success is not equal to 0.000099995
95 percent confidence interval:
 0.000000000 0.002361873
sample estimates:
frequency of outliers (expected: 0.0000999950002499875 )
                                                       0
```

```r
p_values_DHARMa <- c(
  Uniformity = ResidTest$uniformity$p.value,
  Dispersion = ResidTest$dispersion$p.value,
  Outliers   = ResidTest$outliers$p.value
)
models$BLT$p_values <- p_values_DHARMa
```

```r
qqplot(seq(0, 1, length.out = length(errors)), sort(errors),
       main = "", xlab = "", ylab = "", pch = 16, col = "darkblue", cex = 0.7, bty = "n")
title(cex.main = 1.8)
mtext("Expected residual", side = 1, line = 3, cex = 1.5)
mtext("Observed residual", side = 2, line = 3, cex = 1.5)
abline(0, 1, col = "red", lwd = 2)
```



```r
quantiles <- c(0.25, 0.5, 0.75)
colors <- c("blue", "green", "purple")

RankFitted <- rank(fitted)/length(fitted)
plot(RankFitted, errors,
     main = "", xlab = "", ylab = "", pch = 16, col = "darkred", cex = 0.5, bty = "n",
     xaxt = 'n', yaxt = 'n', ylim = c(0, 1))
mtext("Ranked fitted value", side = 1, line = 3, cex = 1.5)
mtext("Observed residual", side = 2, line = 3, cex = 1.5)
abline(h = 0.5, col = "red", lwd = 2)
for (i in seq_along(quantiles)) {
  fit <- quantreg::rq(errors ~ RankFitted, tau = quantiles[i])
  abline(fit, col = colors[i], lwd = 2)
}
axis(1, at = seq(0, 1, by = 0.25), labels = seq(0, 1, by = 0.25), cex.axis = 1.2)
axis(2, at = seq(0, 1, by = 0.25), labels = seq(0, 1, by = 0.25), cex.axis = 1.2)
abline(h = c(0.25, 0.5, 0.75), col = "black", lty = 2, lwd = 2)
```



Probability mass function and likelihood

Many diagnostic tools make use of the (log) probability mass function of the observed values given each set of simulated parameters.

The PMF of the binomial-logit-t model for a given bounded count outcome y_{ij} with m_{ij} trials is given by

$$P\left(y_{ij}\left|\beta, \mathbf{u}_i, \sigma^2, \nu\right.\right) = \int_{-\infty}^{\infty} \binom{m_{ij}}{y_{ij}} \left(\frac{1}{1+e^{-\xi_{ij}}}\right)^{y_{ij}} \left(1 - \frac{1}{1+e^{-\xi_{ij}}}\right)^{m_{ij}-y_{ij}} t\left(\xi_{ij}\left|\eta_{ij}, \sigma^2, \nu\right.\right) \, \mathrm{d}\xi_{ij},$$

where $\eta_{ij} = \mathbf{x}_{ij}'\beta + z_{ij}'\mathbf{u}_i$, and ξ_{ij} are the nuisance parameters that are integrated out.

If accurate values are needed at each simulation iteration then the integration can be performed numerically as follows, demonstrated here for a single observation.

```r
dBLTint <- function(k, eta, sigma, df, m) {
  integrand <- \(x) {
    dbinom(k, m, plogis(x))*dt((x - eta)/sigma, df = df)/sigma
  }
  integrate(integrand, lower = -Inf, upper = Inf, rel.tol = .Machine$double.eps^.25)$value
}
dBLTobs <- function(obs = 1) {
  seq_len(n_post_sims) |> sapply(\(sim) {
    dBLTint(dataset$Y[obs], expected_values_logit[obs, sim],
            post_sims$sigma[sim], post_sims$nu[sim],
            dataset$m[obs])
  })
}
target_obs <- 4
system.time({
obs_pmf <- dBLTobs(target_obs)
})
```

```
   user  system elapsed 
   6.03    0.08    6.47 
```

The numeric integration can be replaced by Monte Carlo integration, where instead of integrating over the nuisance parameters, we replace the nuisance parameters with randomly generated values from their conditional distributions. This is generally less accurate, and possibly slower, in the case of a univariate integration.

```
dBLTmc <- function(k, eta, sigma, df, m, n_mc = 100) {
  x <- rt(n_mc, df)*sigma + eta
  dbinom(k, m, plogis(x)) |> mean()
}
dBLTobs_mc <- function(obs = 1, n_mc = 100) {
  seq_len(n_post_sims) |> sapply(\(sim) {
    dBLTmc(dataset$Y[obs], expected_values_logit[obs, sim],
            post_sims$sigma[sim], post_sims$nu[sim],
            dataset$m[obs], n_mc)
  })
}
system.time({
obs_pmf_mc100 <- dBLTobs_mc(target_obs, n_mc = 100)
})
```

```
   user  system elapsed 
   1.16    0.04    1.22 
```

```
system.time({
obs_pmf_mc10 <- dBLTobs_mc(target_obs, n_mc = 10)
})
```

```
   user  system elapsed 
   0.41    0.00    0.42 
```

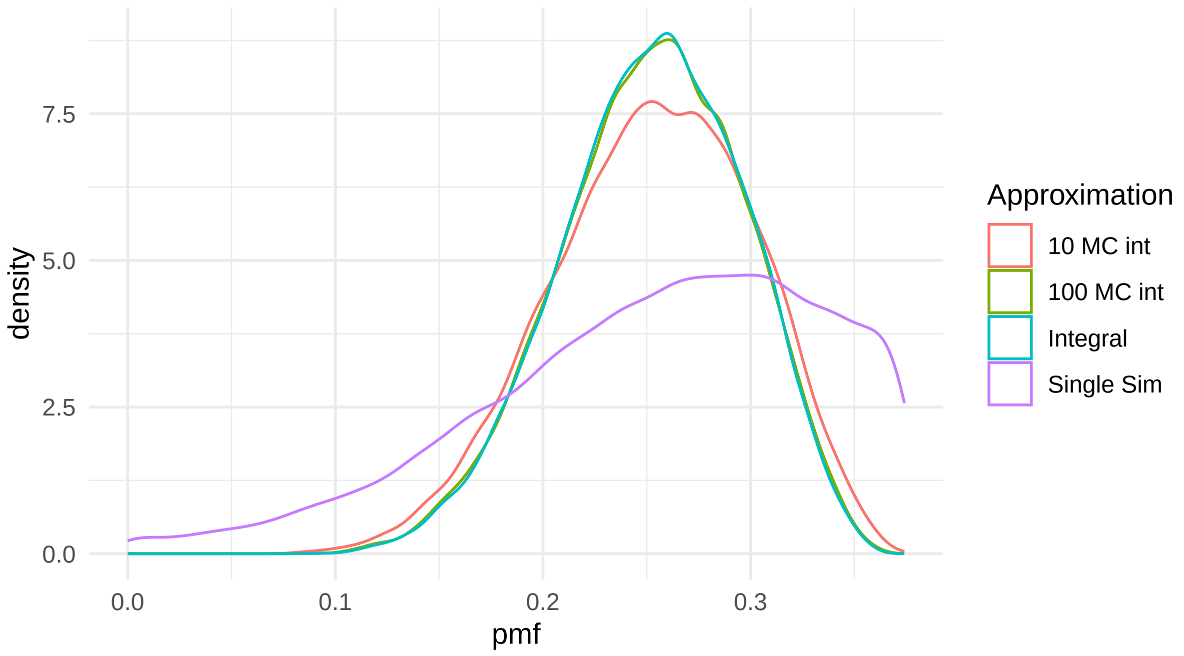
However, if one is only interested in the distribution across simulations, not the value for a specific simulation, as is usually the case, then, given enough posterior simulations, the number of nuisance parameter random values generated can collapse to one each, resulting in a massive speed increase (at the cost of some accuracy).

We now investigate the speed and accuracy difference formally, for a given observation.

```
dBLTfast <- function(k, eta, sigma, df, m) {
  x <- rt(max(length(k), length(eta)), df)*sigma + eta
  dbinom(k, m, plogis(x))
}
dBLTobs_fast <- function(obs = 1) {
  dBLTfast(dataset$Y[obs], expected_values_logit[obs,],
            post_sims$sigma, post_sims$nu, dataset$m)
}
system.time({
obs_pmf_fast <- dBLTobs_fast(target_obs)
})
```

```
   user  system elapsed
   0.02    0.00    0.01
```

```
obs_pmf_compare <- data.frame(
  pmf <- c(obs_pmf, obs_pmf_mc100, obs_pmf_mc10, obs_pmf_fast),
  Approximation <- rep(c("Integral", "100 MC int", "10 MC int", "Single Sim"),
                       each = n_post_sims)
)
obs_pmf_compare |> ggplot(aes(x = pmf, colour = Approximation)) +
  geom_density()
```



In summary: As the number of simulations averaged increases so we approach the pmf suggested by the integral, but we lose out on speed. At 100 MC simulations we achieve a reasonable approximation with a 4 to 5 fold speed increase.

A further speed increase can be obtained by running across observations in parallel.

```r
library(parallel)
cl <- makeCluster(ceiling(parallel::detectCores(logical = FALSE)*0.75))
cl |> clusterExport(
  c('dBLTmc', 'dataset', 'post_sims', 'n_post_sims', 'expected_values_logit'))
dBLTall <- cl |> parSapplyLB(seq_len(n_obs_total), dBLTobs_mc)
cl |> stopCluster()
```

The resulting matrix is in the form *simulations* by *observations* because that is the form preferred by the *loo* package, discussed below.

Information criteria

Information criteria are an excellent way to compare models. While it is feasible to calculate these manually, the use of an established library streamlines the process.

```r
library(loo)
```

```r
rel_eff <- relative_eff(dBLTall, chain_id = rep(1:n_chains, each = n_sims_per_chain))
models$BLT$loo <- loo(log(dBLTall), r_eff = rel_eff)
models$BLT$WAIC <- waic(log(dBLTall))
```

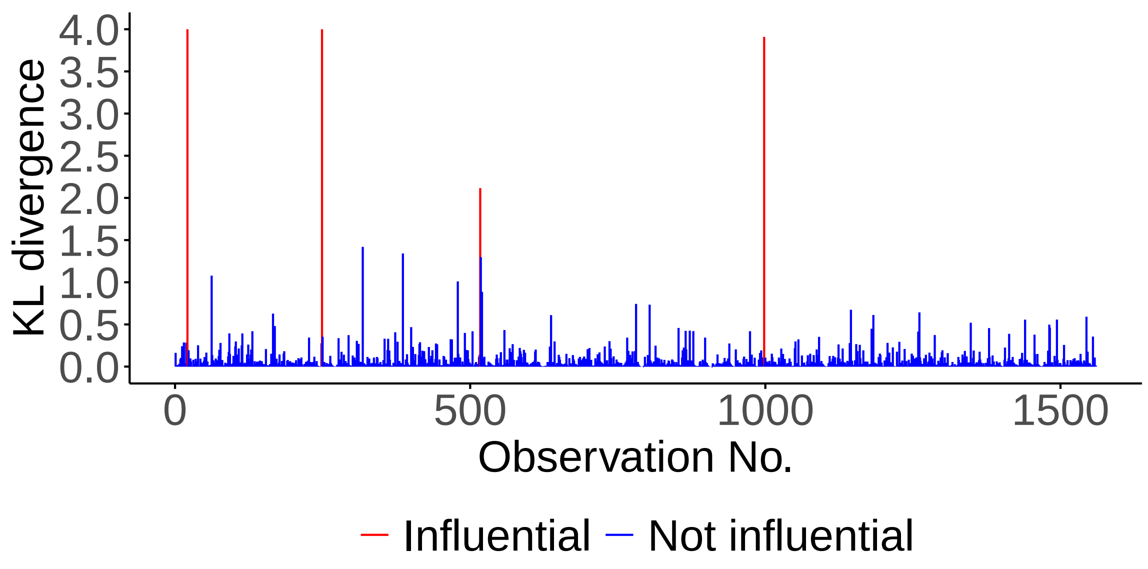
KL Divergence

The divergence is calculated to investigate model fit.

```r
KL_calc <- function(like_matrix) {
  inv_like_mean <- colMeans(1/like_matrix)
  log_like_mean <- colMeans(log(like_matrix))
  log(inv_like_mean) + (log_like_mean)
}
kl_divergences_BLT <- dBLTall |> KL_calc()
```

```r
models$BLT$KLdata <- kl_plot_data <- data.frame(
  Observation = seq_len(n_obs_total),
  KL = kl_divergences_BLT,
  KLCapped = pmin(kl_divergences_BLT, 4),
  Influential = ifelse(0.5*(1 + sqrt(1 - exp(-2*kl_divergences_BLT))) >= 0.99,
                       "Influential", "Not influential")
)
```

```r
kl_plot_data |> ggplot(aes(x = Observation, y = KLCapped, color = Influential)) +
  geom_segment(aes(xend = Observation, yend = 0), linewidth = 0.5) +
  scale_color_manual(values = c("Influential" = "red", "Not influential" = "blue")) +
  labs(x = "Observation No.", y = "KL divergence") +
  scale_x_continuous(breaks = seq(0, max(kl_plot_data$Observation), by = 500)) +
  scale_y_continuous(limits = c(0, 4), breaks = seq(0, 6, by = 0.5)) +
  theme(
    legend.title = element_blank(),
    legend.position = "bottom",
    legend.text = element_text(size = 22),
    axis.text.x = element_text(size = 22),
    axis.text.y = element_text(size = 22),
    axis.title.x = element_text(size = 22),
    axis.title.y = element_text(size = 22),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(color = "black"),
    axis.ticks = element_line(color = "black")
  )
```



Binomial logit normal (BN) model

This is the same model as above, but using a normal distribution instead of a t distributions for the additional innovations. This model does not have a degrees of freedom parameter.

```
fitBLN <- function(dataset, thin = 5, timeout = 3600) {
# Model specification
ModelSpec <- "
  model {
    Psi[1, 1] ~ dgamma(0.5, 1/50^2)
    Psi[1, 2] <- 0
    Psi[2, 1] <- Psi[1, 2]
    Psi[2, 2] ~ dgamma(0.5, 1/50^2)
    inv_Sigma ~ dwish(2*kappa*Psi, kappa + 1)
    for (i in 1:N) {
      a[i, 1:2] ~ dmnorm(mu[], inv_Sigma[,])
      a_0[i] <- a[i, 1]
      a_1[i] <- a[i, 2]
    }
    for (i in 1:N_total) {
      eta[i] <- beta_0 + a_0[id[i]] + a_1[id[i]]*time[i] + beta_grp*group[i] +
                beta_time*time[i] + beta_xt*group[i]*time[i] + beta_cnt*cov_cnt[i]
      xi[i] ~ dnorm(eta[i], 1/sigma^2)
      logit(p[i]) <- xi[i]
      y[i] ~ dbin(p[i], m[i])
    }
    beta_0 ~ dnorm(0, 0.001)
    beta_grp ~ dnorm(0, 0.001)
    beta_time ~ dnorm(0, 0.001)
    beta_xt ~ dnorm(0, 0.001)
    beta_cnt ~ dnorm(0, 0.001)
    sigma ~ dt(0, 0.25, 2)T(0, )
  }
"
# Parameters to be monitored
  Monitor <- c("beta_0", "beta_time", "beta_grp", "beta_cnt", "beta_xt", "inv_Sigma[1,1]",
             "inv_Sigma[1,2]", "inv_Sigma[2,1]", "inv_Sigma[2,2]", 'sigma', 'a')
# Initial values
  Inits <- replicate(n_chains, list(
    beta_0 = 0,
    beta_time = 0,
    beta_xt = 0,
    beta_grp = 0,
    sigma = 1,
    inv_Sigma = diag(2),
    Psi = matrix(c(1, NA, NA, 1), nrow = 2),
    .RNG.name = 'base::Mersenne-Twister',
    .RNG.seed = sample.int(1e5, 1)
  ), simplify = FALSE)
# Data reworked for JAGS
  JAGSData <- list(
    N = max(dataset$Subject),
    N_total = nrow(dataset),
    id = dataset$Subject,
    time = dataset$Time,
    group = dataset$Group,
    cov_cnt = dataset$cov_cnt,
    y = dataset$Y,
    m = dataset$m,
    mu = rep(0, 2),
    kappa = 2
  )
# JAGS is called to do the simulation
  Sample <- tryCatch({R.utils::withTimeout({
    runjags::run.jags(
      model = ModelSpec,
      data = JAGSData,
      inits = Inits,
      monitor = Monitor,
      n.chains = n_chains,
      burnin = n_sims_per_chain,
      thin = thin,
      sample = n_sims_per_chain,
      summarise = FALSE,
      method = 'parallel',
      modules = 'glm',
      factories = 'bugs::MNormal sampler off'
    )
  }, timeout = timeout, onTimeout = "silent")}, error = function(e) {NULL}
  )
# The posterior simulations are returned in a raw runjags format for now
  Sample
}
```

```
rBLN <- function(n, m, mu, sigma_logit) {
  n_values <- rnorm(n)*sigma_logit + mu
  rbinom(n, m, plogis(n_values))
}
```

```
post_sims <- post_sims_runjags |> runjags_to_dataframe()
```

Results

Summary table

```
models$BLN$summary <- post_summary_table <- post_sims_runjags |> summary()
```

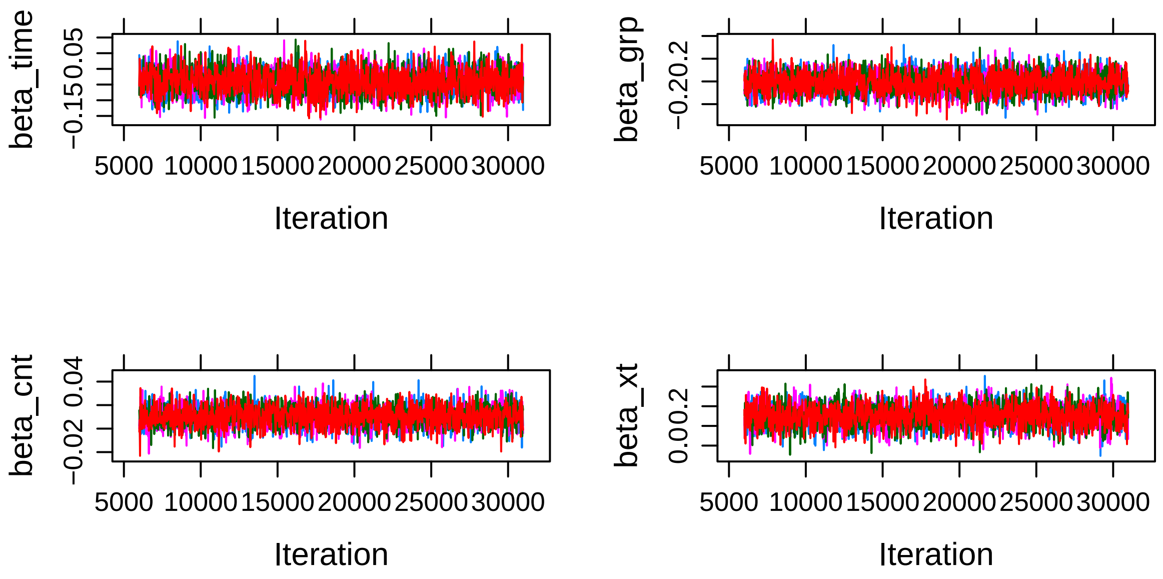
```
post_summary_table[!startsWith(rownames(post_summary_table), "a"), ] |> kable(digits = 3)
```

	Lower95	Median	Upper95	Mean	SD	Mode	MCerr	MC%ofSD	SSeff	AC.50	psrf
beta_0	0.471	1.475	2.529	1.477	0.524	NA	0.005	0.9	12387	0.006	1.000
beta_time	-0.116	-0.040	0.036	-0.040	0.039	NA	0.000	0.8	15347	0.012	1.000
beta_grp	-0.190	-0.012	0.177	-0.011	0.094	NA	0.001	0.9	11942	0.004	1.000
beta_cnt	-0.007	0.010	0.027	0.010	0.009	NA	0.000	0.9	12254	0.008	1.000
beta_xt	0.042	0.152	0.262	0.152	0.056	NA	0.001	1.0	9990	0.021	1.000
inv_Sigma[1,1]	5.380	11.029	20.081	11.905	4.484	NA	0.076	1.7	3447	0.044	1.001
inv_Sigma[1,2]	1.841	5.932	11.157	6.274	2.589	NA	0.035	1.4	5422	0.021	1.002
inv_Sigma[2,1]	1.841	5.932	11.157	6.274	2.589	NA	0.035	1.4	5422	0.021	1.002
inv_Sigma[2,2]	10.727	14.911	20.212	15.132	2.500	NA	0.032	1.3	6139	0.036	1.000
sigma	0.297	0.346	0.395	0.346	0.025	NA	0.001	2.4	1724	0.201	1.000

Trace plot

```r
fixed_effects <- c("beta_time", "beta_grp", "beta_cnt", "beta_xt")
plot(post_sims_runjags, vars = fixed_effects, plot.type = c('trace'), new.window = FALSE)
```

```
Generating summary statistics and plots (these will NOT be saved for
reuse)...
Calculating summary statistics...
Calculating the Gelman-Rubin statistic for 4 variables....
```



Fit evaluation and prediction

Predictor

```r
lin_pred <- model.matrix(~Time*Group + cov_cnt, data = dataset) |>
  lin_pred_eval(post_sims)
models$BLN$lin_pred_ave <- lin_pred_ave <- new_X |> lin_pred_eval(post_sims)
```

Random effects predictor

```r
subjects_vec <- unique(dataset$Subject)
ran_effects_subjects <- subjects_vec |> lapply(\(subj) {
  ran_effects_eval(dataset$Time[dataset$Subject %in% subj], post_sims, subj)
}) |> do.call(rbind, args = _)
```

```r
ran_effects_subjects <- ran_effects_subjects[order(dataset$Subject), ]

models$BLN$eta <- expected_values_logit <- lin_pred + ran_effects_subjects
models$BLN$median <- median_values_bin <- plogis(expected_values_logit)*dataset$m
models$BLN$pred <- predicted_values_bin <- n_post_sims |> seq_len() |> sapply(\(sim) {
  rBLN(n_obs_total, dataset$m, expected_values_logit[,sim], post_sims$sigma[sim])
})
```

Standardised residuals

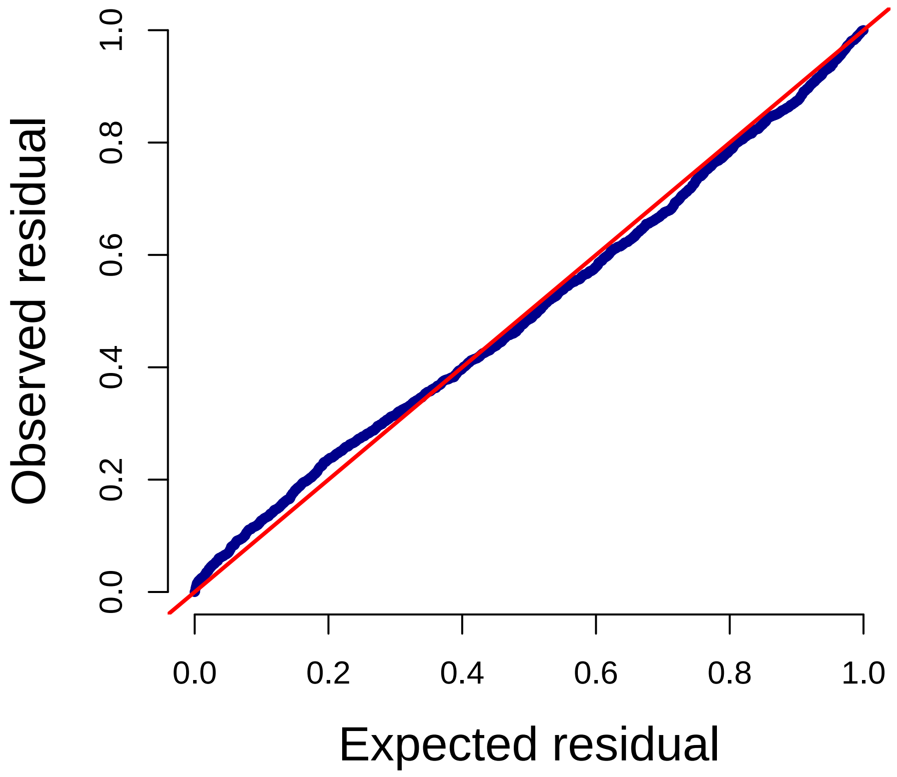
```r
fitted <- median_values_bin |> apply(1, median)
DHARMa <- createDHARMa(
  simulatedResponse = predicted_values_bin,
  observedResponse = dataset$Y,
  fittedPredictedResponse = fitted
)
errors <- residuals(DHARMa)
# plot(DHARMa)
ResidTest <- testResiduals(DHARMa, plot = FALSE)
```

```
$uniformity

    Asymptotic one-sample Kolmogorov-Smirnov test

data:  simulationOutput$scaledResiduals
D = 0.037727, p-value = 0.02358
alternative hypothesis: two-sided


$dispersion

    DHARMa nonparametric dispersion test via sd of residuals fitted vs.
    simulated

data:  simulationOutput
dispersion = 0.80046, p-value = 0.001
alternative hypothesis: two.sided


$outliers

    DHARMa outlier test based on exact binomial test with approximate
    expectations

data:  simulationOutput
outliers at both margin(s) = 2, observations = 1560, p-value = 0.01097
alternative hypothesis: true probability of success is not equal to 0.000099995
```

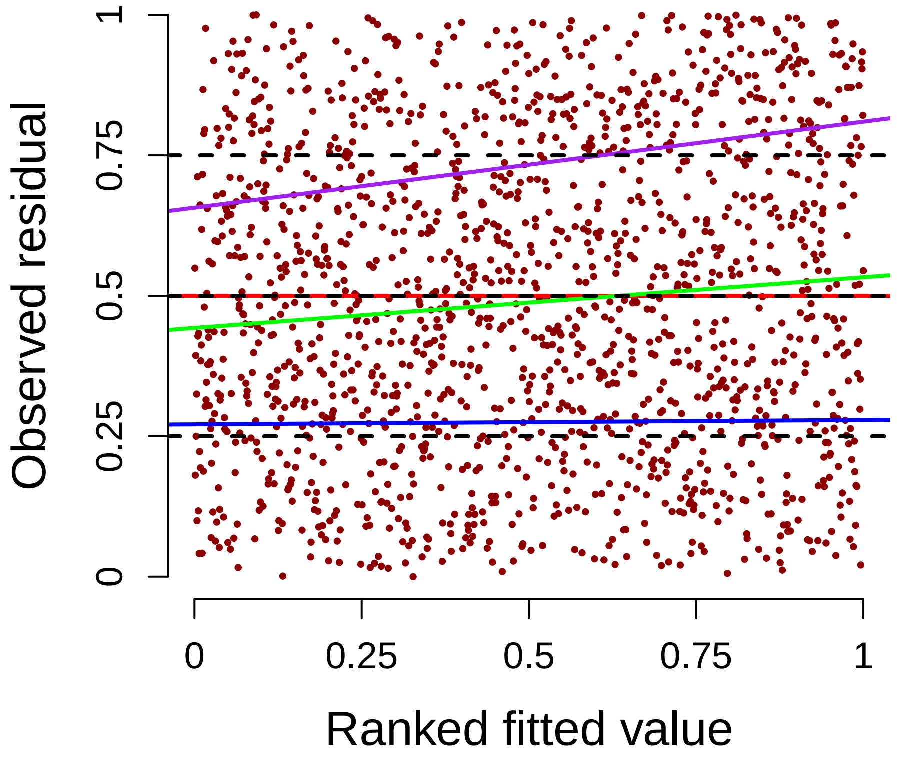
```
95 percent confidence interval:
 0.0001553001 0.0046234612
sample estimates:
frequency of outliers (expected: 0.000999950002499875 )
                                             0.001282051
```

```
p_values_DHARMa <- c(
  Uniformity = ResidTest$uniformity$p.value,
  Dispersion = ResidTest$dispersion$p.value,
  Outliers   = ResidTest$outliers$p.value
)
models$BLN$p_values <- p_values_DHARMa
```

```
qqplot(seq(0, 1, length.out = length(errors)), sort(errors),
       main = "", xlab = "", ylab = "", pch = 16, col = "darkblue", cex = 0.7, bty = "n")
title(cex.main = 1.8)
mtext("Expected residual", side = 1, line = 3, cex = 1.5)
mtext("Observed residual", side = 2, line = 3, cex = 1.5)
abline(0, 1, col = "red", lwd = 2)
```



```
RankFitted <- rank(fitted)/length(fitted)
plot(RankFitted, errors,
     main = "", xlab = "", ylab = "", pch = 16, col = "darkred", cex = 0.5, bty = "n",
     xaxt = 'n', yaxt = 'n', ylim = c(0, 1))
mtext("Ranked fitted value", side = 1, line = 3, cex = 1.5)
mtext("Observed residual", side = 2, line = 3, cex = 1.5)
abline(h = 0.5, col = "red", lwd = 2)
for (i in seq_along(quantiles)) {
  fit <- quantreg::rq(errors ~ RankFitted, tau = quantiles[i])
  abline(fit, col = colors[i], lwd = 2)
}
axis(1, at = seq(0, 1, by = 0.25), labels = seq(0, 1, by = 0.25), cex.axis = 1.2)
axis(2, at = seq(0, 1, by = 0.25), labels = seq(0, 1, by = 0.25), cex.axis = 1.2)
abline(h = c(0.25, 0.5, 0.75), col = "black", lty = 2, lwd = 2)
```



Probability mass function and likelihood

```
dBLNmc <- function(k, eta, sigma, m, n_mc = 100) {
  x <- rnorm(n_mc)*sigma + eta
  dbinom(k, m, plogis(x)) |> mean()
```

```r
}
dBLNobs_mc <- function(obs = 1, n_mc = 100) {
  seq_len(n_post_sims) |> sapply(\(sim) {
    dBLNmc(dataset$Y[obs], expected_values_logit[obs, sim],
            post_sims$sigma[sim], dataset$m[obs], n_mc)
  })
}
```

```r
cl <- makeCluster(ceiling(parallel::detectCores(logical = FALSE)*0.75))
cl |> clusterExport(
  c('dBLNmc', 'dataset', 'post_sims', 'n_post_sims', 'expected_values_logit'))
dBLNall <- cl |> parSapplyLB(seq_len(n_obs_total), dBLNobs_mc)
cl |> stopCluster()
```

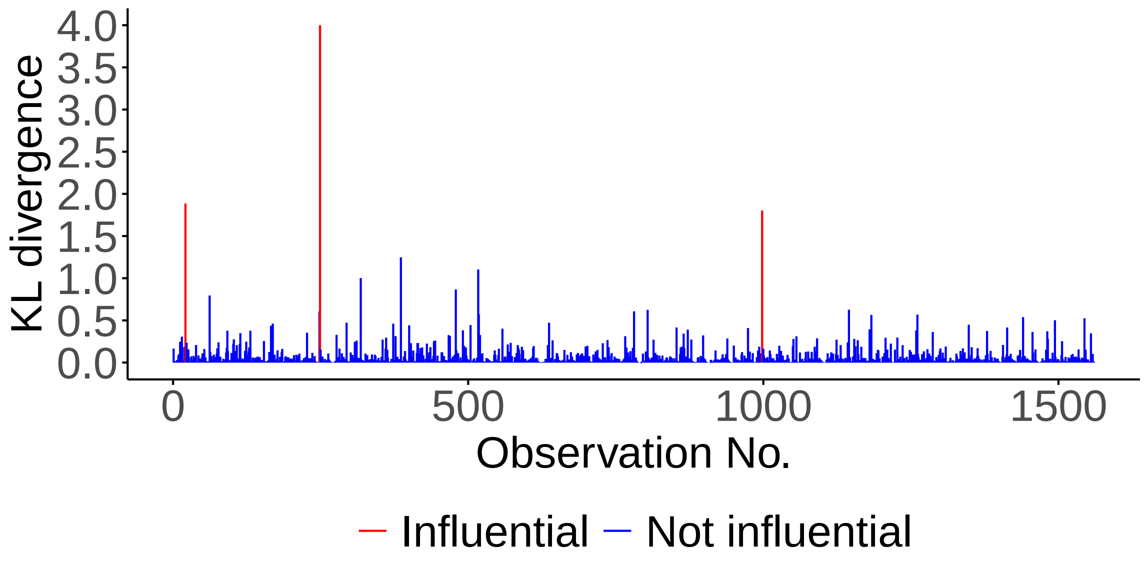
Information criteria

```r
rel_eff <- relative_eff(dBLNall, chain_id = rep(1:n_chains, each = n_sims_per_chain))
models$BLN$loo <- loo(log(dBLNall), r_eff = rel_eff)
models$BLN$WAIC <- waic(log(dBLNall))
```

KL Divergence

```r
kl_divergences_BLN <- dBLNall |> KL_calc()
models$BLN$KLdata <- kl_plot_data <- data.frame(
  Observation = seq_len(n_obs_total),
  KL = kl_divergences_BLN,
  KLCapped = pmin(kl_divergences_BLN, 4),
  Influential = ifelse(0.5*(1 + sqrt(1 - exp(-2*kl_divergences_BLN))) >= 0.99,
                       "Influential", "Not influential")
)
```

```r
kl_plot_data |> ggplot(aes(x = Observation, y = KLCapped, color = Influential)) +
  geom_segment(aes(xend = Observation, yend = 0), linewidth = 0.5) +
  scale_color_manual(values = c("Influential" = "red", "Not influential" = "blue")) +
  labs(x = "Observation No.", y = "KL divergence") +
  scale_x_continuous(breaks = seq(0, max(kl_plot_data$Observation), by = 500)) +
  scale_y_continuous(limits = c(0, 4), breaks = seq(0, 6, by = 0.5)) +
  theme(
    legend.title = element_blank(),
    legend.position = "bottom",
    legend.text = element_text(size = 22),
    axis.text.x = element_text(size = 22),
    axis.text.y = element_text(size = 22),
    axis.title.x = element_text(size = 22),
    axis.title.y = element_text(size = 22),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(color = "black"),
    axis.ticks = element_line(color = "black")
  )
```



Beta-Binomial

```
fitBB <- function(dataset, thin = 5, timeout = 3600) {
# Model specification
ModelSpec <- "
  model {
    Psi[1, 1] ~ dgamma(0.5, 1/50^2)
    Psi[1, 2] <- 0
    Psi[2, 1] <- Psi[1, 2]
    Psi[2, 2] ~ dgamma(0.5, 1/50^2)
    inv_Sigma ~ dwish(2*kappa*Psi, kappa + 1)
    for (i in 1:N) {
      a[i, 1:2] ~ dmnorm(mu[], inv_Sigma[,])
      a_0[i] <- a[i, 1]
      a_1[i] <- a[i, 2]
    }
    for (i in 1:N_total) {
      eta[i] <- beta_0 + a_0[id[i]] + a_1[id[i]]*time[i] + beta_grp*group[i] +
                beta_time*time[i] + beta_xt*group[i]*time[i] + beta_cnt*cov_cnt[i]
      logit(p[i]) <- eta[i]
      alpha_bb[i] <- p[i]*delta
      beta_bb[i] <- (1 - p[i])*delta
      y[i] ~ dbetabin(alpha_bb[i], beta_bb[i], m[i])
    }
    delta ~ dgamma(0.001, 0.001)
    beta_0 ~ dnorm(0, 0.001)
    beta_grp ~ dnorm(0, 0.001)
    beta_time ~ dnorm(0, 0.001)
    beta_xt ~ dnorm(0, 0.001)
    beta_cnt ~ dnorm(0, 0.001)
  }
"
# Parameters to be monitored
  Monitor <- c("beta_0", "beta_time", "beta_grp", "beta_cnt", "beta_xt",
               "delta", "inv_Sigma", 'a')
# Initial values
  Inits <- replicate(n_chains, list(
    beta_0 = 2,
    beta_grp = 0,
    beta_time = 0,
    beta_xt = 0,
    delta = 1,
    inv_Sigma = matrix(c(1, 0, 0, 45), nrow = 2),
    Psi = matrix(c(2, NA, NA, 0.01), nrow = 2),
    a = matrix(0, max(dataset$Subject), 2),
    .RNG.name = 'base::Mersenne-Twister',
    .RNG.seed = sample.int(1e5, 1)
  ), simplify = FALSE)
# Data reworked for JAGS
  JAGSData <- list(
    N = max(dataset$Subject),
    N_total = nrow(dataset),
    id = dataset$Subject,
    time = dataset$Time,
    group = dataset$Group,
    cov_cnt = dataset$cov_cnt,
    y = dataset$Y,
    m = dataset$m,
    mu = rep(0, 2),
    kappa = 2
  )
# JAGS is called to do the simulation
  Sample <- tryCatch({R.utils::withTimeout({
    runjags::run.jags(
      model = ModelSpec,
      data = JAGSData,
      inits = Inits,
      monitor = Monitor,
      n.chains = n_chains,
      burnin = n_sims_per_chain,
      thin = thin,
      sample = n_sims_per_chain,
      summarise = FALSE,
      method = 'parallel',
      modules = c('glm', "mix"),
      factories = 'bugs::MNormal sampler off'
    )
  }, timeout = timeout, onTimeout = "silent")}, error = function(e) {NULL}
  )
# The posterior simulations are returned in a raw runjags format for now
  Sample
}
```

```
rBB <- function(n, m, mu, delta) {
  VGAM::rbetabinom(n, m, plogis(mu), 1/(1 + delta))
}
```

```
post_sims <- post_sims_runjags |> runjags_to_dataframe()
```

Results

Summary table

```
models$BB$summary <- post_summary_table <- post_sims_runjags |> summary()
```

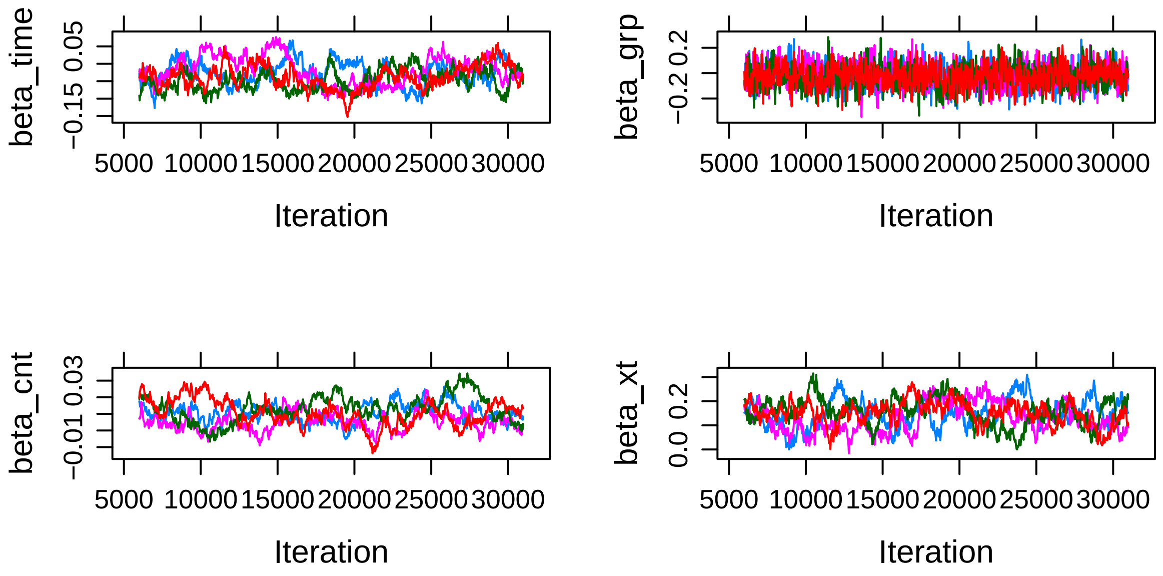
```
post_summary_table[!startsWith(rownames(post_summary_table), "a"), ] |> kable(digits = 3)
```

	Lower95	Median	Upper95	Mean	SD	Mode	MCerr	MC%ofSD	SSeff	AC.50	psrf
beta_0	0.473	1.487	2.255	1.458	0.446	NA	0.056	12.5	64	0.938	1.152
beta_time	-0.098	-0.034	0.038	-0.032	0.036	NA	0.005	12.6	63	0.924	1.103
beta_grp	-0.197	-0.023	0.159	-0.023	0.091	NA	0.002	2.4	1808	0.194	1.005
beta_cnt	-0.004	0.009	0.025	0.010	0.007	NA	0.001	12.4	65	0.936	1.163
beta_xt	0.030	0.149	0.247	0.147	0.056	NA	0.007	11.9	71	0.919	1.040
delta	54.712	75.178	101.250	76.702	12.336	NA	0.128	1.0	9343	0.025	1.002
inv_Sigma[1,1]	5.628	11.596	22.263	12.786	5.293	NA	0.192	3.6	761	0.466	1.005
inv_Sigma[2,1]	1.970	6.091	12.253	6.549	2.901	NA	0.081	2.8	1283	0.282	1.003
inv_Sigma[1,2]	1.970	6.091	12.253	6.549	2.901	NA	0.081	2.8	1283	0.282	1.003
inv_Sigma[2,2]	11.049	15.691	21.247	15.930	2.670	NA	0.046	1.7	3441	0.088	1.002

Trace plot

```
fixed_effects <- c("beta_time", "beta_grp", "beta_cnt", "beta_xt")
plot(post_sims_runjags, vars = fixed_effects, plot.type = c('trace'), new.window = FALSE)
```

```
Generating summary statistics and plots (these will NOT be saved for
reuse)...
Calculating summary statistics...
Calculating the Gelman-Rubin statistic for 4 variables....
```



Fit evaluation and prediction

Predictor

```r
lin_pred <- model.matrix(~Time*Group + cov_cnt, data = dataset) |>
  lin_pred_eval(post_sims)
models$BB$lin_pred_ave <- lin_pred_ave <- new_X |> lin_pred_eval(post_sims)
```

Random effects predictor

```r
subjects_vec <- unique(dataset$Subject)
ran_effects_subjects <- subjects_vec |> lapply(\(subj) {
  ran_effects_eval(dataset$Time[dataset$Subject %in% subj], post_sims, subj)
}) |> do.call(rbind, args = _)
ran_effects_subjects <- ran_effects_subjects[order(dataset$Subject), ]

models$BB$eta <- expected_values_logit <- lin_pred + ran_effects_subjects
models$BB$median <- median_values_bin <- plogis(expected_values_logit)*dataset$m
models$BB$pred <- predicted_values_bin <- n_post_sims |> seq_len() |> sapply(\(sim) {
  rBB(n_obs_total, dataset$m, expected_values_logit[,sim], post_sims$delta[sim])
})
```

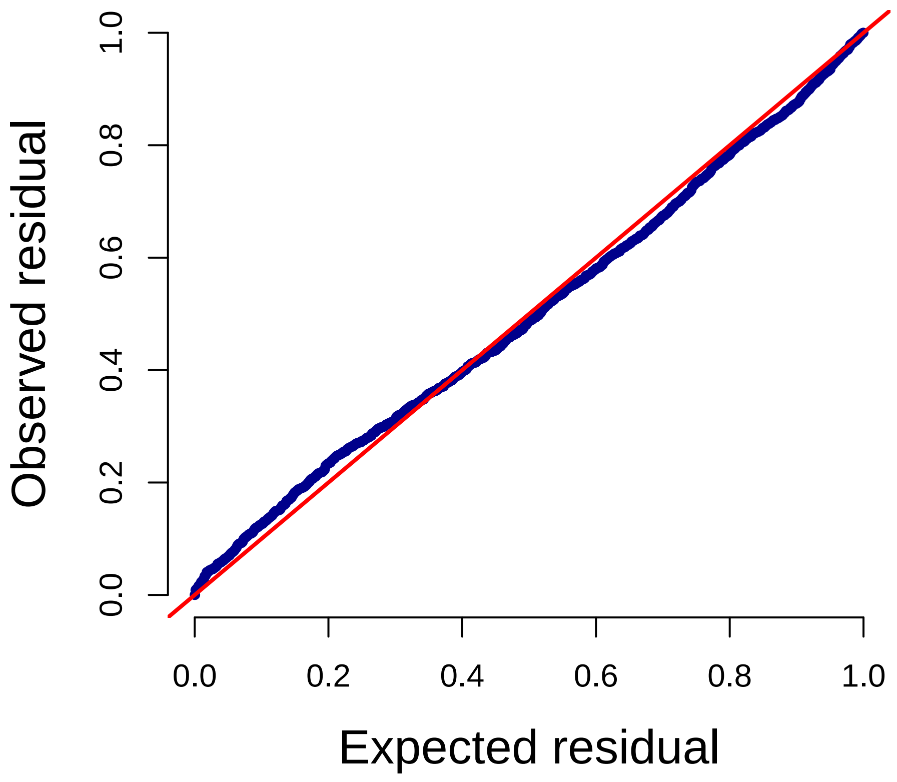
Standardised residuals

```r
fitted <- median_values_bin |> apply(1, median)
DHARMa <- createDHARMa(
  simulatedResponse = predicted_values_bin,
  observedResponse = dataset$Y,
  fittedPredictedResponse = fitted
)
errors <- residuals(DHARMa)
# plot(DHARMa)
ResidTest <- testResiduals(DHARMa, plot = FALSE)
```

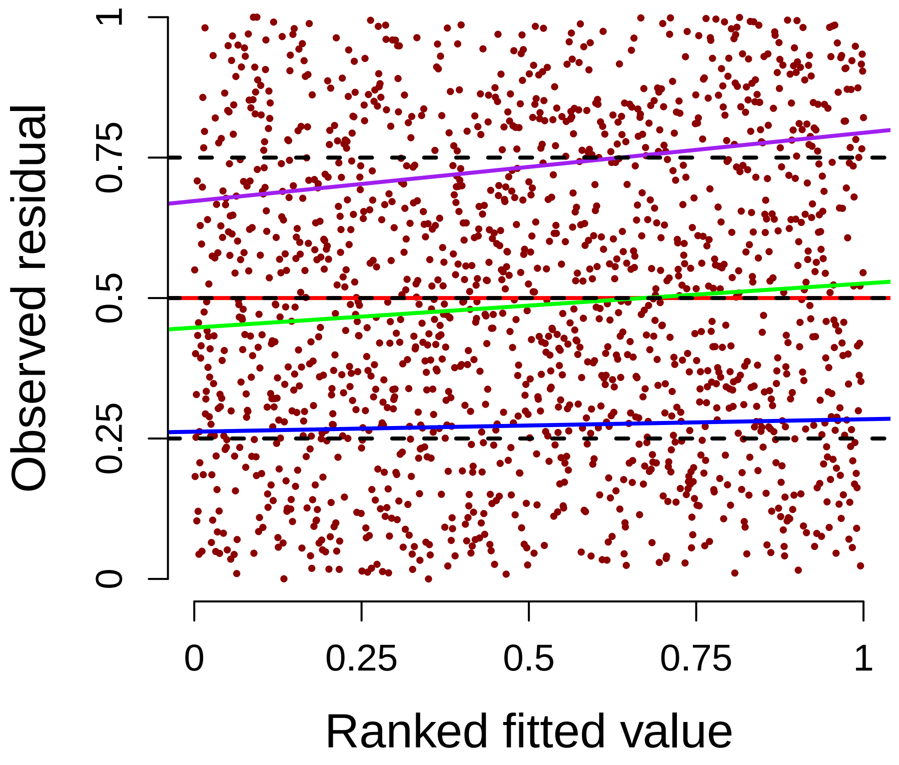
```
$uniformity

	Asymptotic one-sample Kolmogorov-Smirnov test

data:  simulationOutput$scaledResiduals
D = 0.035197, p-value = 0.04192
alternative hypothesis: two-sided


$dispersion

	DHARMa nonparametric dispersion test via sd of residuals fitted vs.
	simulated

data:  simulationOutput
dispersion = 0.87864, p-value = 0.0493
alternative hypothesis: two.sided


$outliers

	DHARMa outlier test based on exact binomial test with approximate
	expectations

data:  simulationOutput
outliers at both margin(s) = 2, observations = 1560, p-value = 0.01097
alternative hypothesis: true probability of success is not equal to 0.000099995
95 percent confidence interval:
 0.0001553001 0.0046234612
sample estimates:
frequency of outliers (expected: 0.0000999950002499875 )
                                             0.001282051
```

```r
p_values_DHARMa <- c(
  Uniformity = ResidTest$uniformity$p.value,
  Dispersion = ResidTest$dispersion$p.value,
  Outliers    = ResidTest$outliers$p.value
)
models$BB$p_values <- p_values_DHARMa
```

```
qqplot(seq(0, 1, length.out = length(errors)), sort(errors),
       main = "", xlab = "", ylab = "", pch = 16, col = "darkblue", cex = 0.7, bty = "n")
title(cex.main = 1.8)
mtext("Expected residual", side = 1, line = 3, cex = 1.5)
mtext("Observed residual", side = 2, line = 3, cex = 1.5)
abline(0, 1, col = "red", lwd = 2)
```



```
RankFitted <- rank(fitted)/length(fitted)
plot(RankFitted, errors,
     main = "", xlab = "", ylab = "", pch = 16, col = "darkred", cex = 0.5, bty = "n",
     xaxt = 'n', yaxt = 'n', ylim = c(0, 1))
mtext("Ranked fitted value", side = 1, line = 3, cex = 1.5)
mtext("Observed residual", side = 2, line = 3, cex = 1.5)
abline(h = 0.5, col = "red", lwd = 2)
for (i in seq_along(quantiles)) {
  fit <- quantreg::rq(errors ~ RankFitted, tau = quantiles[i])
  abline(fit, col = colors[i], lwd = 2)
}
axis(1, at = seq(0, 1, by = 0.25), labels = seq(0, 1, by = 0.25), cex.axis = 1.2)
axis(2, at = seq(0, 1, by = 0.25), labels = seq(0, 1, by = 0.25), cex.axis = 1.2)
abline(h = c(0.25, 0.5, 0.75), col = "black", lty = 2, lwd = 2)
```



Probability mass function and likelihood

```
dBB <- function(k, m, eta, delta) {
  VGAM::dbetabinom(k, m, plogis(eta), 1/(1 + delta))
}
dBBobs <- function(obs = 1) {
  seq_len(n_post_sims) |> sapply(\(sim) {
    dBB(dataset$Y[obs], dataset$m[obs], expected_values_logit[obs, sim],
            post_sims$delta[sim])
  })
}
```

```
cl <- makeCluster(ceiling(parallel::detectCores(logical = FALSE)*0.75))
cl |> clusterExport(
```

```r
  c('dBB', 'dataset', 'post_sims', 'n_post_sims', 'expected_values_logit'))
dBBall <- cl |> parSapplyLB(seq_len(n_obs_total), dBBobs)
cl |> stopCluster()
```

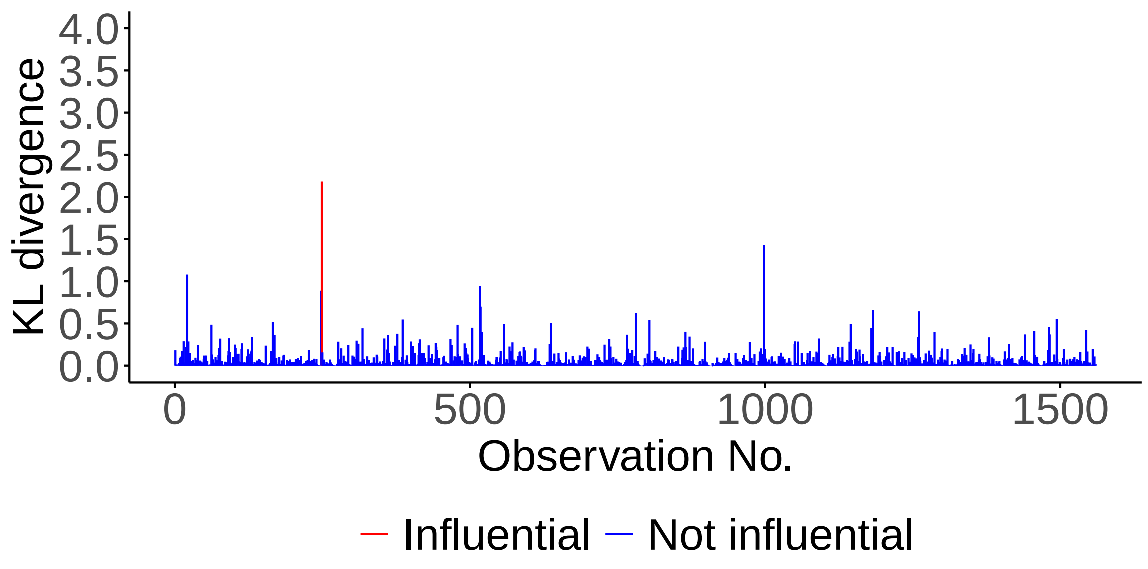
Information criteria

```r
rel_eff <- relative_eff(dBBall, chain_id = rep(1:n_chains, each = n_sims_per_chain))
models$BB$loo <- loo(log(dBBall), r_eff = rel_eff)
models$BB$WAIC <- waic(log(dBBall))
```

KL Divergence

```r
kl_divergences_BB <- dBBall |> KL_calc()
models$BB$KLdata <- kl_plot_data <- data.frame(
  Observation = seq_len(n_obs_total),
  KL = kl_divergences_BB,
  KLCapped = pmin(kl_divergences_BB, 4),
  Influential = ifelse(0.5*(1 + sqrt(1 - exp(-2*kl_divergences_BB))) >= 0.99,
                       "Influential", "Not influential")
)
```

```r
kl_plot_data |> ggplot(aes(x = Observation, y = KLCapped, color = Influential)) +
  geom_segment(aes(xend = Observation, yend = 0), linewidth = 0.5) +
  scale_color_manual(values = c("Influential" = "red", "Not influential" = "blue")) +
  labs(x = "Observation No.", y = "KL divergence") +
  scale_x_continuous(breaks = seq(0, max(kl_plot_data$Observation), by = 500)) +
  scale_y_continuous(limits = c(0, 4), breaks = seq(0, 6, by = 0.5)) +
  theme(
    legend.title = element_blank(),
    legend.position = "bottom",
    legend.text = element_text(size = 22),
    axis.text.x = element_text(size = 22),
    axis.text.y = element_text(size = 22),
    axis.title.x = element_text(size = 22),
    axis.title.y = element_text(size = 22),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(color = "black"),
    axis.ticks = element_line(color = "black")
  )
```



Binomial

Here we do an ordinary Binomial mixed model.

```
fitBin <- function(dataset, thin = 5, timeout = 3600) {
# Model specification
ModelSpec <- "
  model {
    Psi[1, 1] ~ dgamma(0.5, 1/50^2)
    Psi[1, 2] <- 0
    Psi[2, 1] <- Psi[1, 2]
    Psi[2, 2] ~ dgamma(0.5, 1/50^2)
    inv_Sigma ~ dwish(2*kappa*Psi, kappa + 1)
    for (i in 1:N) {
      a[i, 1:2] ~ dmnorm(mu[], inv_Sigma[,])
      a_0[i] <- a[i, 1]
      a_1[i] <- a[i, 2]
    }
    for (i in 1:N_total) {
      eta[i] <- beta_0 + a_0[id[i]] + a_1[id[i]]*time[i] + beta_grp*group[i] +
                beta_time*time[i] + beta_xt*group[i]*time[i] + beta_cnt*cov_cnt[i]
      logit(p[i]) <- eta[i]
      y[i] ~ dbin(p[i], m[i])
    }
    beta_0 ~ dnorm(0, 0.001)
    beta_grp ~ dnorm(0, 0.001)
    beta_time ~ dnorm(0, 0.001)
    beta_xt ~ dnorm(0, 0.001)
    beta_cnt ~ dnorm(0, 0.001)
  }
"
# Parameters to be monitored
  Monitor <- c("beta_0", "beta_time", "beta_grp", "beta_cnt", "beta_xt", "inv_Sigma", "a")
# Initial values
  Inits <- replicate(n_chains, list(
    beta_0 = 0,
    beta_time = 0,
    beta_xt = 0,
    beta_grp = 0,
    inv_Sigma = diag(2),
    Psi = matrix(c(1, NA, NA, 1), nrow = 2),
    .RNG.name = 'base::Mersenne-Twister',
    .RNG.seed = sample.int(1e5, 1)
  ), simplify = FALSE)
# Data reworked for JAGS
  JAGSData <- list(
    N = max(dataset$Subject),
    N_total = nrow(dataset),
    id = dataset$Subject,
    time = dataset$Time,
    group = dataset$Group,
    cov_cnt = dataset$cov_cnt,
    y = dataset$Y,
    m = dataset$m,
    mu = rep(0, 2),
    kappa = 2
  )
# JAGS is called to do the simulation
  Sample <- tryCatch({R.utils::withTimeout({
    runjags::run.jags(
      model = ModelSpec,
      data = JAGSData,
      inits = Inits,
      monitor = Monitor,
      n.chains = n_chains,
      burnin = n_sims_per_chain,
      thin = thin,
      sample = n_sims_per_chain,
      summarise = FALSE,
      method = 'parallel',
      modules = c('glm', "mix"),
      factories = 'bugs::MNormal sampler off'
    )
  }, timeout = timeout, onTimeout = "silent")}, error = function(e) {NULL}
  )
# The posterior simulations are returned in a raw runjags format for now
  Sample
}
```

```
rBin <- function(n, m, mu) {
  rbinom(n, m, plogis(mu))
}
```

```
post_sims <- post_sims_runjags |> runjags_to_dataframe()
```

Results

Summary table

```
models$Bin$summary <- post_summary_table <- post_sims_runjags |> summary()
```

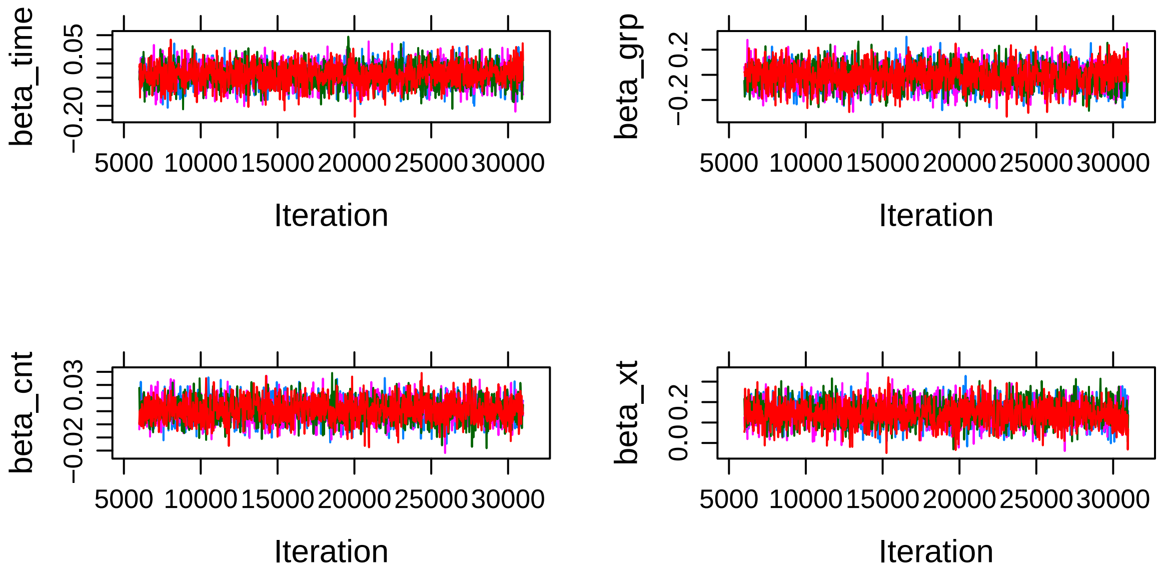
```
post_summary_table[!startsWith(rownames(post_summary_table), "a"), ] |> kable(digits = 3)
```

	Lower95	Median	Upper95	Mean	SD	Mode	MCerr	MC%ofSD	SSeff	AC.50	psrf
beta_0	0.400	1.405	2.409	1.404	0.516	NA	0.007	1.3	6254	0.024	1.000
beta_time	-0.116	-0.042	0.030	-0.042	0.037	NA	0.000	0.8	14642	0.010	1.000
beta_grp	-0.193	-0.016	0.169	-0.015	0.093	NA	0.001	1.3	5504	0.027	1.000
beta_cnt	-0.006	0.011	0.027	0.011	0.008	NA	0.000	1.3	6163	0.024	1.000
beta_xt	0.041	0.147	0.256	0.148	0.055	NA	0.001	1.2	6768	0.031	1.001
inv_Sigma[1,1]	4.959	8.311	12.750	8.594	2.090	NA	0.044	2.1	2227	0.140	1.000
inv_Sigma[2,1]	2.085	4.953	8.426	5.093	1.642	NA	0.027	1.6	3705	0.060	1.000
inv_Sigma[1,2]	2.085	4.953	8.426	5.093	1.642	NA	0.027	1.6	3705	0.060	1.000
inv_Sigma[2,2]	11.198	15.314	20.019	15.440	2.280	NA	0.045	2.0	2569	0.080	1.001

Trace plot

```
fixed_effects <- c("beta_time", "beta_grp", "beta_cnt", "beta_xt")
plot(post_sims_runjags, vars = fixed_effects, plot.type = c('trace'), new.window = FALSE)
```

```
Generating summary statistics and plots (these will NOT be saved for
reuse)...
Calculating summary statistics...
Calculating the Gelman-Rubin statistic for 4 variables....
```



Fit evaluation and prediction

Predictor

```
lin_pred <- model.matrix(~Time*Group + cov_cnt, data = dataset) |>
  lin_pred_eval(post_sims)
models$Bin$lin_pred_ave <- lin_pred_ave <- new_X |> lin_pred_eval(post_sims)
```

Random effects predictor

```
subjects_vec <- unique(dataset$Subject)
ran_effects_subjects <- subjects_vec |> lapply(\(subj) {
  ran_effects_eval(dataset$Time[dataset$Subject %in% subj], post_sims, subj)
}) |> do.call(rbind, args = _)
ran_effects_subjects <- ran_effects_subjects[order(dataset$Subject), ]

models$Bin$eta <- expected_values_logit <- lin_pred + ran_effects_subjects
models$Bin$median <- median_values_bin <- plogis(expected_values_logit)*dataset$m
models$Bin$pred <- predicted_values_bin <- n_post_sims |> seq_len() |> sapply(\(sim) {
  rBin(n_obs_total, dataset$m, expected_values_logit[,sim])
})
```

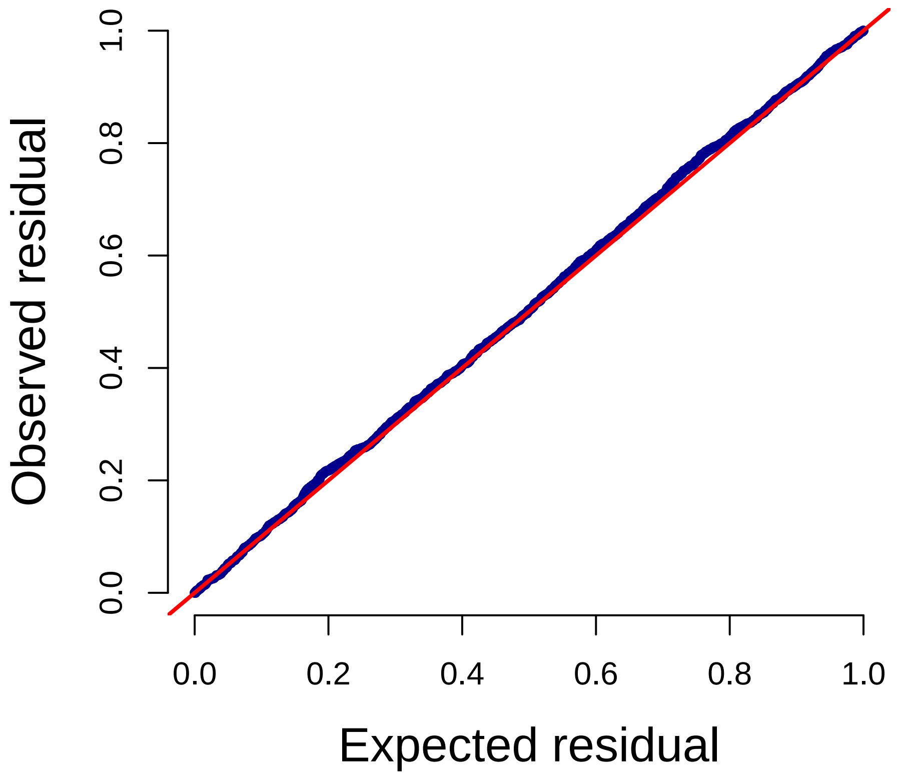
Standardised residuals

```r
fitted <- median_values_bin |> apply(1, median)
DHARMa <- createDHARMa(
  simulatedResponse = predicted_values_bin,
  observedResponse = dataset$Y,
  fittedPredictedResponse = fitted
)
errors <- residuals(DHARMa)
# plot(DHARMa)
ResidTest <- testResiduals(DHARMa, plot = FALSE)
```

```
$uniformity

	Asymptotic one-sample Kolmogorov-Smirnov test

data:  simulationOutput$scaledResiduals
D = 0.0221, p-value = 0.4312
alternative hypothesis: two-sided


$dispersion

	DHARMa nonparametric dispersion test via sd of residuals fitted vs.
	simulated

data:  simulationOutput
dispersion = 1.1653, p-value = 0.001
alternative hypothesis: two.sided


$outliers

	DHARMa outlier test based on exact binomial test with approximate
	expectations

data:  simulationOutput
outliers at both margin(s) = 4, observations = 1560, p-value =
0.00002171
alternative hypothesis: true probability of success is not equal to 0.000099995
95 percent confidence interval:
 0.0006990598 0.0065519973
sample estimates:
frequency of outliers (expected: 0.0000999950002499875 )
                                             0.002564103
```

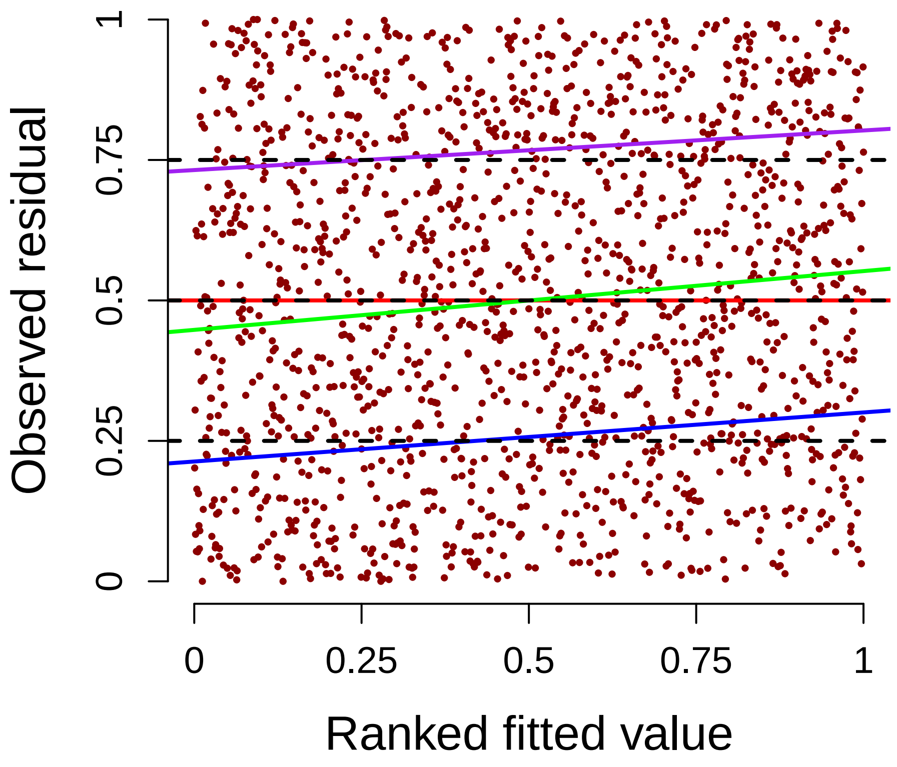
```r
p_values_DHARMa <- c(
  Uniformity = ResidTest$uniformity$p.value,
  Dispersion = ResidTest$dispersion$p.value,
  Outliers   = ResidTest$outliers$p.value
)
models$Bin$p_values <- p_values_DHARMa
```

```
qqplot(seq(0, 1, length.out = length(errors)), sort(errors),
       main = "", xlab = "", ylab = "", pch = 16, col = "darkblue", cex = 0.7, bty = "n")
title(cex.main = 1.8)
mtext("Expected residual", side = 1, line = 3, cex = 1.5)
mtext("Observed residual", side = 2, line = 3, cex = 1.5)
abline(0, 1, col = "red", lwd = 2)
```



```
RankFitted <- rank(fitted)/length(fitted)
plot(RankFitted, errors,
     main = "", xlab = "", ylab = "", pch = 16, col = "darkred", cex = 0.5, bty = "n",
     xaxt = 'n', yaxt = 'n', ylim = c(0, 1))
mtext("Ranked fitted value", side = 1, line = 3, cex = 1.5)
mtext("Observed residual", side = 2, line = 3, cex = 1.5)
abline(h = 0.5, col = "red", lwd = 2)
for (i in seq_along(quantiles)) {
  fit <- quantreg::rq(errors ~ RankFitted, tau = quantiles[i])
  abline(fit, col = colors[i], lwd = 2)
}
```

```r
axis(1, at = seq(0, 1, by = 0.25), labels = seq(0, 1, by = 0.25), cex.axis = 1.2)
axis(2, at = seq(0, 1, by = 0.25), labels = seq(0, 1, by = 0.25), cex.axis = 1.2)
abline(h = c(0.25, 0.5, 0.75), col = "black", lty = 2, lwd = 2)
```



Probability mass function and likelihood

```r
dBin <- function(k, eta, m) {
  dbinom(k, m, plogis(eta))
}
dBinobs <- function(obs = 1) {
  seq_len(n_post_sims) |> sapply(\(sim) {
    dBin(dataset$Y[obs], expected_values_logit[obs, sim],
            dataset$m[obs])
  })
}
```

```r
cl <- makeCluster(ceiling(parallel::detectCores(logical = FALSE)*0.75))
cl |> clusterExport(
  c('dBin', 'dataset', 'post_sims', 'n_post_sims', 'expected_values_logit'))
```

```r
dBinall <- cl |> parSapplyLB(seq_len(n_obs_total), dBinobs)
cl |> stopCluster()
```

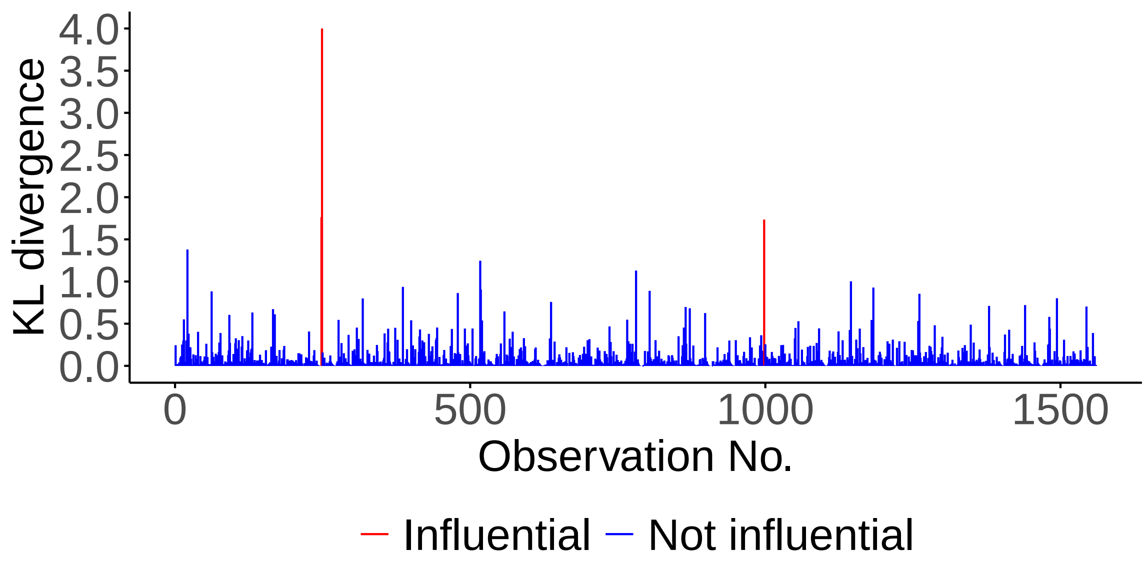
Information criteria

```r
rel_eff <- relative_eff(dBinall, chain_id = rep(1:n_chains, each = n_sims_per_chain))
models$Bin$loo <- loo(log(dBinall), r_eff = rel_eff)
models$Bin$WAIC <- waic(log(dBinall))
```

KL Divergence

```r
kl_divergences_Bin <- dBinall |> KL_calc()
models$Bin$KLdata <- kl_plot_data <- data.frame(
  Observation = seq_len(n_obs_total),
  KL = kl_divergences_Bin,
  KLCapped = pmin(kl_divergences_Bin, 4),
  Influential = ifelse(0.5*(1 + sqrt(1 - exp(-2*kl_divergences_Bin))) >= 0.99,
                       "Influential", "Not influential")
)
```

```r
kl_plot_data |> ggplot(aes(x = Observation, y = KLCapped, color = Influential)) +
  geom_segment(aes(xend = Observation, yend = 0), linewidth = 0.5) +
  scale_color_manual(values = c("Influential" = "red", "Not influential" = "blue")) +
  labs(x = "Observation No.", y = "KL divergence") +
  scale_x_continuous(breaks = seq(0, max(kl_plot_data$Observation), by = 500)) +
  scale_y_continuous(limits = c(0, 4), breaks = seq(0, 6, by = 0.5)) +
  theme(
    legend.title = element_blank(),
    legend.position = "bottom",
    legend.text = element_text(size = 22),
    axis.text.x = element_text(size = 22),
    axis.text.y = element_text(size = 22),
    axis.title.x = element_text(size = 22),
    axis.title.y = element_text(size = 22),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(color = "black"),
    axis.ticks = element_line(color = "black")
  )
```



Final comparison and summary

The models will now have their fit statistics tabulated.

First the recommended LOO statistics table.

```r
loo_table <- loo_compare(list(
  BLT = models$BLT$loo,
  BLN = models$BLN$loo,
  BB = models$BB$loo,
  Bin = models$Bin$loo
))
loo_table |> print(simplify = FALSE) |> kable(digits = 3)
```

```
    elpd_diff se_diff elpd_loo se_elpd_loo p_loo   se_p_loo looic   se_looic
BB      0.0       0.0 -2938.9     44.4       164.5     7.9   5877.8    88.8
BLN    -1.8       5.0 -2940.7     44.6       173.3    11.3   5881.4    89.3
BLT   -24.7      13.9 -2963.6     52.3       209.4    24.1   5927.2   104.5
Bin   -42.0      12.8 -2980.9     53.4       234.3    13.3   5961.8   106.8
```

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
BB	0.000	0.000	-2938.890	44.397	164.467	7.907	5877.781	88.795
BLN	-1.791	4.996	-2940.682	44.649	173.271	11.262	5881.363	89.298
BLT	-24.698	13.878	-2963.588	52.267	209.375	24.107	5927.177	104.534
Bin	-42.018	12.759	-2980.908	53.380	234.337	13.302	5961.816	106.760

Then the WAIC.

```
models |> sapply(\(m) {
  m$WAIC
})
```

```
             BLT          BLN          BB           Bin
estimates    numeric,6    numeric,6    numeric,6    numeric,6
pointwise    numeric,4680 numeric,4680 numeric,4680 numeric,4680
elpd_waic    -2968.419    -2935.781    -2935.66     -2974.624
p_waic       214.2061     168.3705     161.2368     228.0528
waic         5936.839     5871.562     5871.321     5949.247
se_elpd_waic 58.70819     44.45401     44.28376     53.06322
se_p_waic    33.81892     11.09144     7.739054     12.87122
se_waic      117.4164     88.90801     88.56751     106.1264
```

And also the residual test p-values.

```
models |> sapply(\(m) {
  m$p_values
}) |> kable(digits = 3)
```

	BLT	BLN	BB	Bin
Uniformity	0.064	0.024	0.042	0.431
Dispersion	0.019	0.001	0.049	0.001
Outliers	1.000	0.011	0.011	0.000