Source code 3

Geometric model incorporating prey's turn and predator attack endpoint explains multiple preferred escape trajectories

Yuuki Kawabata Hideyuki Akada Ken-ichiro Shimatani
Gregory N. Nishihara Hibiki Kimura Nozomi Nishiumi Paolo Domenici

2022-03-20

Contents

1 Prepare packages

We use five packages in this supplement.

```
library(tidyverse)    # Data wrangling
library(brms)         # Sampling from the posterior distribution
library(tidybayes)    # Data wrangling of brms output
library(bayesplot)    # Diagnostic plots
library(patchwork)    # Simplified plot layout
```

```r
library(emmeans)      # Simplifies extraction of posterior draws
options(mc.cores = 5) # Five CPU cores were made available
```

2 Prepare data for Figure 3B

The data, analysis, diagnostics, and results for Fig. 3A will be illustrated. The data for this analysis is stored in a CSV file.

```r
data = read_csv("Dataset1.csv") data
= data |>
  mutate(cumd10 = V_s1, ea = abs(ea)) |>
  select(cumd10, ea, fish) |>
  mutate(fish = factor(fish)) |>
  group_by(fish) |>
  filter(n() > 1) |>
  ungroup()
```

```r
xlabel = "'Turn angle'~group('|', alpha, '|' )~(degree)"
ylabel = "'Velocity after s1 turn'~(m~s^{-1})"
ggplot(data) +
  geom_point(aes(x = ea, y = cumd10)) +
  scale_x_continuous(parse(text = xlabel)) +
  scale_y_continuous(parse(text = ylabel)) +
  scale_color_viridis_d()
```

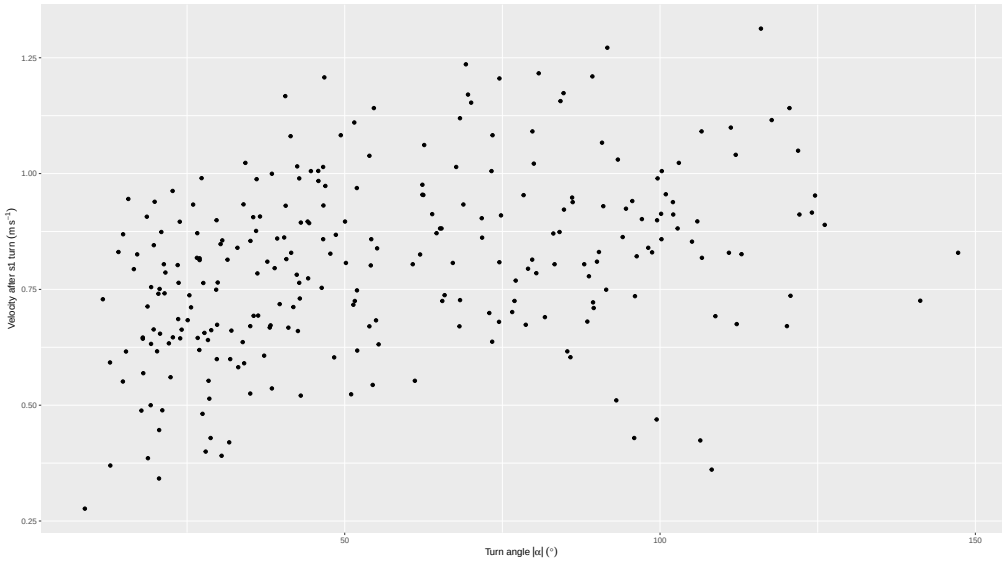

Figure 1: Plot of the observations to be analyzed.

3 BRMS models

- Model 1: Multilevel model
- Model 2: Multilevel constant model
- Model 3: Multilevel linear model

3.1 Model 1

The multilevel model (Eq. (1)) is modified after (Brilleman et al., 2017).

$$
\begin{aligned}
\mu &= \begin{cases} \beta_{0k} + \beta_{1k}x, & \text{if } x < \omega \\ \beta_{0k} + \beta_{1k}\omega + \beta_{2k}(x-\omega), & \text{if } x \geq \omega \end{cases} \\
y_{obs} &\sim N(\mu, \sigma) \\
\beta_{0k} &\sim N(\mu_{\beta_0}, \sigma_{\beta_0}) \\
\beta_{1k} &\sim N(\mu_{\beta_1}, \sigma_{\beta_1}) \\
\beta_{2k} &\sim N(\mu_{\beta_2}, \sigma_{\beta_2}) \\
\mu_{\beta_0} &\sim N(30, 60) \\
\mu_{\beta_1} &\sim N(0.01, 0.3) \\
\mu_{\beta_2} &\sim N(0.15, 0.3) \\
\mu_{\omega} &\sim N(45, 30) \\
\sigma &\sim \textit{half-Cauchy}(0, 2.5) \\
\sigma_{\beta_0} &\sim \textit{half-Cauchy}(0, 2.5) \\
\sigma_{\beta_1} &\sim \textit{half-Cauchy}(0, 2.5) \\
\sigma_{\beta_2} &\sim \textit{half-Cauchy}(0, 2.5) \\
\sigma_{\omega} &\sim \textit{half-Cauchy}(0, 2.5)
\end{aligned}
\tag{1}
$$

y_{obs} are the observations, β_{ik} are the ith coefficient for group k, μ_{β_i} and σ_{β_i} are the population-level coefficients mean and standard deviation, μ_ω and σ_ω are the mean and standard deviation of the change-point parameter, and σ is the error term of the model.

3.2 Model 2

The multilevel constant slope model (Eq. (2)).

$$
\begin{aligned}
\mu &= \beta_{0k} \\
y_{obs} &\sim N(\mu, \sigma) \\
\beta_{0k} &\sim N(\mu_{\beta_0}, \sigma_{\beta_0}) \\
\mu_{\beta_0} &\sim N(30, 60) \\
\sigma &\sim \textit{half-Cauchy}(0, 2.5) \\
\sigma_{\beta_0} &\sim \textit{half-Cauchy}(0, 2.5)
\end{aligned}
\tag{2}
$$

3.3 Model 3

The multilevel single slope model (Eq. (3)).

$$
\begin{aligned}
\mu &= \beta_{0k} + \beta_{1k} x \\
y_{obs} &\sim N(\mu, \sigma) \\
\beta_{0k} &\sim N(\mu_{\beta_0}, \sigma_{\beta_0}) \\
\beta_{1k} &\sim N(\mu_{\beta_1}, \sigma_{\beta_1}) \\
\mu_{\beta_0} &\sim N(30, 60) \\
\mu_{\beta_1} &\sim N(0.01, 0.3) \\
\sigma &\sim \textit{half-Cauchy}(0, 2.5) \\
\sigma_{\beta_0} &\sim \textit{half-Cauchy}(0, 2.5) \\
\sigma_{\beta_1} &\sim \textit{half-Cauchy}(0, 2.5)
\end{aligned}
\tag{3}
$$

For the multilevel model (Eq. (1)), we create a function to pass to `brm`.

```
stanfunction = "
real cpmodel (real intercept, real slope1, real slope2, real changepoint, real x) {
  real tmp;
  if(x < changepoint) {
    tmp = intercept + slope1 * x;
  } else {
    tmp = intercept + (slope1 * changepoint) + slope2 * (x - changepoint);
  }
  return(tmp);
}
"
stanvars = stanvar(scode = stanfunction, block = "functions")
```

For the two versions of model 1, the model is fit using the non-linear syntax of `brm`, hence `nl = TRUE`.

```
bmodel1 = bf(cumd10 ~ cpmodel(intercept, slope1, slope2, change, ea),
             change + intercept + slope1 + slope2 ~ (1 | fish),
             nl = TRUE) + gaussian()
```

For the simpler model 2 and 3, which are true linear models, the syntax is simplified.

```
bmodel2 = bf(cumd10 ~ (1|fish)) + gaussian()
bmodel3 = bf(cumd10 ~ ea + (0 + ea||fish)) + gaussian()
```

All models assume a Gaussian (Normal) distribution for the error term. The prior distribution for the `brms` models are declared next.

```
mu_intercept     = 0.5
mu_slope1        = 0.0006
mu_slope2        = -0.0001
mu_change_point  = 15

sigma_intercept     = 1.0
sigma_slope1        = 0.012
sigma_slope2        = 0.012
sigma_change_point  = 30
```

The default priors can be examined using `get_prior()`.

```
# Not run
get_prior(bmodel1, data = data)
```

```
get_prior(bmodel2, data = data)
get_prior(bmodel3, data = data)
```

We compose our own weakly informative priors, based on the values provided above. In `priors1`, we declare the priors for the two versions of model 1. In `priors2` and `priors3`, we declare the priors for model 2 and model 3, respectively.

```
priors1 = c(prior(cauchy(0, 2.5), class = sigma),
            prior_string(str_glue("normal({mu_intercept}, {sigma_intercept})"),
                         class = "b", coef = "Intercept", nlpar = "intercept"),
            prior_string(str_glue("normal({mu_change_point}, {sigma_change_point})"),
                         class = "b", coef = "Intercept", nlpar = "change"),
            prior_string(str_glue("normal({mu_slope1}, {sigma_slope1})"),
                         class = "b", coef = "Intercept", nlpar = "slope1"),
            prior_string(str_glue("normal({mu_slope2}, {sigma_slope2})"),
                         class = "b", coef = "Intercept", nlpar = "slope2"),
            prior(cauchy(0, 2.5), class = sd, nlpar = intercept),
            prior(cauchy(0, 2.5), class = sd, nlpar = change),
            prior(cauchy(0, 2.5), class = sd, nlpar = slope1),
            prior(cauchy(0, 2.5), class = sd, nlpar = slope2))

priors2 = c(prior(cauchy(0, 2.5), class = sigma),
            prior_string(str_glue("normal({mu_intercept}, {sigma_intercept})"),
                         class = "Intercept"),
            prior(cauchy(0, 2.5), class = sd))

priors3 = c(prior(cauchy(0, 2.5), class = sigma),
            prior_string(str_glue("normal({mu_slope1}, {sigma_slope1})"),
                         class = "b", coef = "ea"),
            prior(cauchy(0, 2.5), class = sd))
```

4 Sample from the posterior distribution

Before we compile the models and sample from the prior distribution, we declare a few parameters for `brms` The adapt delta and maximum treedepth of the sampler are set to relatively extreme values, to carefully explore the posterior distribution. Smaller values returned warnings from the sampler. In this example, a total of 200,000 iterations were run and only 2000 samples were retained. A total of 5 chains were explored and a pseudo-random number seed was provided to ensure that the output can be replicated. Note that the manuscript uses an earlier sample of the posterior distribution, where the seed was not provided.

```
ctrl_pars  = list(adapt_delta = 0.9999, max_treedepth = 20)
iterations = 200000
warmup     = iterations - 2000
chains     = 5
seed       = 2022
```

Sample from the posterior distribution using `brm()` from the `brms` package.

```
resample = "on_change"
bout1 = brm(bmodel1, data = data, prior = priors1,
            control = ctrl_pars,
            chains = chains, cores = chains,
            stanvars = stanvars,
            seed = seed,
            save_pars = save_pars(all = TRUE),
```

```
            file = "bmodel1B",
            file_refit = resample,
            iter = iterations, warmup = warmup)
```

```
bout2 = brm(bmodel2, data = data, prior = priors2,
            control = ctrl_pars,
            chains = chains, cores = chains,
            seed = seed,
            save_pars = save_pars(all = TRUE),
            file = "bmodel2B",
            file_refit = resample,
            iter = iterations, warmup = warmup)
```

```
bout3 = brm(bmodel3, data = data, prior = priors3,
            control = ctrl_pars,
            chains = chains, cores = chains,
            seed = seed,
            file = "bmodel3B",
            save_pars = save_pars(all = TRUE),
            file_refit = resample,
            iter = iterations, warmup = warmup)
```

We need to expose the function defined by `stanvar()`, so that the remaining computation can occur.

```
# Expose the function defined in bmodel1 for use in R.
expose_functions(bout1, vectorize = TRUE)
```

The stan code generated by `brm()` can be examined with `stancode()`.

```
# Not run
stancode(bout1)
stancode(bout2)
stancode(bout3)
```

5 Model selection

Conduct the approximate leave-one-out cross-validation from the posterior likelihood fo the models. The LOO is calculated with subsampling. The `loo1` is passed to the `observations` argument to ensure that the same observations are used in the computation.

```
# This procedure is very time-consuming
loo1 = loo_subsample(bout1, observations = 250, cores = 10)
loo2 = loo_subsample(bout2, observations = loo1, cores = 10)
loo3 = loo_subsample(bout3, observations = loo1, cores = 10)
```

Here, we compare all of the models and based on the difference in the expected log-point-wise density (elpd_diff), we select model 1 over model 2 and model 3. Note that decision theory may be better for model selection see **How to use cross-validation for model selection?** [https://mc-stan.org/loo/articles/online-only/faq.html] for some background.

```
loo_compare(loo1, loo2, loo3)
```

```
##       elpd_diff se_diff subsampling_se_diff
## bout1  0.0       0.0     0.0
## bout3  9.3       5.2     0.5
## bout2 22.6       6.8     0.4
```

Calculate the widely applicable information criterion (WAIC) from the posterior likelihood of the models.

```
waic1 = waic(bout1)
waic2 = waic(bout2)
waic3 = waic(bout3)
```

```
loo_compare(waic1, waic2, waic3) |> as_tibble(rownames = "model")
```

```
## # A tibble: 3 x 9
##   model elpd_diff  se_diff elpd_waic se_elpd_waic p_waic se_p_waic waic  se_waic
##   <chr> <compar.l> <compa> <compar.> <compar.l>   <comp> <compar.> <com> <compa>
## 1 bout1   0.000000 0.0000~ 108.98296 11.90517     26.67~ 2.621312  -217~ 23.810~
## 2 bout3  -9.655941 5.2034~  99.32702 12.53697     17.54~ 2.433624  -198~ 25.073~
## 3 bout2 -23.340382 6.8066~  85.64258 11.74250     18.26~ 1.694246  -171~ 23.485~
```

Based on these results, we select model 1 for further analysis.

```
usemodel = bout1
```

6 Examine the posterior distributions

Extract the expected predictions of the parameters for the model.

```
changeem = emmeans(usemodel, specs = ~ 1,nlpar = "change", data = usemodel$data)
slope1em = emmeans(usemodel, specs = ~ 1, nlpar = "slope1", data = usemodel$data)
slope2em = emmeans(usemodel, specs = ~ 1, nlpar = "slope2", data = usemodel$data)
interceptem = emmeans(usemodel, specs = ~ 1, nlpar = "intercept", data = usemodel$data)
sigma_post = spread_draws(usemodel, sigma)

changeem_post = changeem |> gather_emmeans_draws()
slope1em_post = slope1em |> gather_emmeans_draws()
slope2em_post = slope2em |> gather_emmeans_draws()
interceptem_post = interceptem |> gather_emmeans_draws()
```

Build the figures for the prior and posterior distribution of the model coefficients.

```
dnorm2 = function(x,mean, sd) {
  # normal distribution, normalized to 1
  z = dnorm(x,mean, sd)
  z / max(z)
}
dhcauchy2 = function(x, sigma) {
# half-cauchy distribution normalized to 1
 z = extraDistr::dhcauchy(x, sigma)
 z / max(z)
}
p1 = ggplot(slope1em_post) +
  geom_freqpoly(aes(x = .value, y = stat(ndensity), color = "Posterior"), binwidth = 0.0005) +
  geom_function(aes(color = "Prior"),
                fun = dnorm2, args = list(mean = mu_slope1, sd = sigma_slope1)) +
  scale_color_viridis_d(end = 0.8) +
  xlim(mu_slope1 - 2*sigma_slope1, mu_slope1 + 2*sigma_slope1) +
  labs(title = "slope1")
p2 = ggplot(slope2em_post) +
  geom_freqpoly(aes(x = .value, y = stat(ndensity), color = "Posterior"), binwidth = 0.0005) +
  geom_function(aes(color = "Prior"),
                fun = dnorm2, args = list(mean = mu_slope2, sd = sigma_slope2)) +
```

```
  scale_color_viridis_d(end = 0.8) +
  xlim(mu_slope2 - 2*sigma_slope2, mu_slope2 + 2*sigma_slope2) +
  labs(title = "slope2")
p3 = ggplot(interceptem_post) +
  geom_freqpoly(aes(x = .value, y = stat(ndensity), color = "Posterior"), binwidth = 0.01) +
  geom_function(aes(color = "Prior"),
                fun = dnorm2, args = list(mean = mu_intercept, sd = sigma_intercept)) +
  scale_color_viridis_d(end = 0.8) +
  xlim(mu_intercept - 2*sigma_intercept, mu_intercept + 2*sigma_intercept) +
  labs(title = "intercept")
p4 = ggplot(changeem_post) +
  geom_freqpoly(aes(x = .value, y = stat(ndensity), color = "Posterior"), binwidth = 1) +
  geom_function(aes(color = "Prior"),
                fun = dnorm2, args = list(mean = mu_change_point, sd = sigma_change_point)) +
  scale_color_viridis_d(end = 0.8) +
  xlim(mu_change_point - 2*sigma_change_point, mu_change_point + 2*sigma_change_point) +
  labs(title = "change")

p5 = ggplot(sigma_post) +
  geom_freqpoly(aes(x = sigma, y = stat(ndensity), color = "Posterior"), binwidth = 0.005) +
  geom_function(aes(color = "Prior"),
                fun = dhcauchy2, args = list(sigma = 2.5)) +
  scale_color_viridis_d(end = 0.8) +
  xlim(0, 5) +
  labs(title = "sigma")
```

```
p1 + p3 + p2 + p4 + p5 +
  plot_layout(nrow = 3, ncol = 2, guides = "collect") &
  theme(legend.title = element_blank(),
        legend.position = "bottom")
```

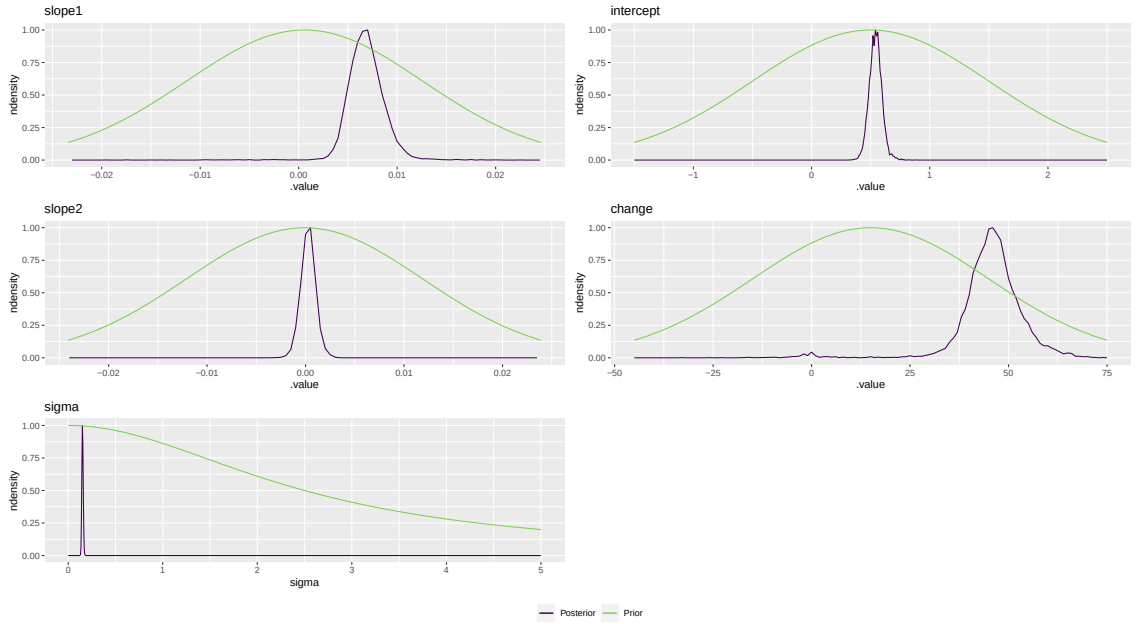

Figure 2: The prior and posterior distributions of the model coefficients and error. The range of the x-axis for the coefficients is the mean ± two standard deviations. The range of the x-axis for the error term is 0 to twice the scale (i.e., 2×2.5).

7 Model results

Prepare the expected values and predicted values from the posterior distribution.

```
tmp = changeem |> as_tibble()

edata = usemodel$data |>
  expand(fish =50,ea = c(seq(min(ea), tmp$lower.HPD, length = 6),
                seq(tmp$lower.HPD, tmp$upper.HPD, length = 51),
                seq(tmp$lower.HPD, max(ea), length = 6))) |> distinct()
pdata = edata |> add_predicted_draws(usemodel, allow_new_levels = TRUE)
edata = edata |> add_epred_draws(usemodel, allow_new_levels = TRUE)

pdata = pdata |> group_by(ea) |>
  summarise(estimate = mean(.prediction),
            lower = quantile(.prediction, 0.025),
            upper = quantile(.prediction, 0.975))
edata = edata |> group_by(ea) |>
  summarise(estimate =  mean(.epred),
            lower = quantile(.epred, 0.025),
            upper = quantile(.epred, 0.975))
```

```
coeffs = usemodel |>
  gather_draws(b_change_Intercept, b_slope1_Intercept, b_slope2_Intercept, b_intercept_Intercept) |>
  separate(.variable, c("b", "coef", "x"))

coeffs_sd =
  usemodel |>
  gather_draws(`sd_.*`, `sigma`, regex = T) |>
```

```
  mutate(coef = str_extract(.variable, "sigma|slope1|slope2|change|intercept"))
```

Model coefficients of the population parameter means.

```
coeffs |>
  group_by(coef) |>
  summarise(across(.value,
                   list(mean = mean, sd = sd, median = median, mad = mad,
                        lower = ~quantile(.x, 0.025),
                        upper = ~quantile(.x, 0.975)),
                   .names = "{.fn}"))
```

```
## # A tibble: 4 x 7
##   coef           mean       sd    median      mad    lower    upper
##   <chr>         <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 change    45.5      8.98     46.0      5.62     29.4     60.5
## 2 intercept  0.541    0.0540    0.541    0.0505    0.438    0.649
## 3 slope1     0.00683  0.00209   0.00677  0.00160   0.00381  0.0105
## 4 slope2     0.000265 0.000753  0.000274 0.000710 -0.00122  0.00177
```

Model coefficients of the population parameter standard deviations.

```
coeffs_sd |>
  group_by(coef) |>
  summarise(across(.value,
                   list(mean = mean, sd = sd, median = median, mad = mad,
                        lower = ~quantile(.x, 0.025),
                        upper = ~quantile(.x, 0.975)),
                   .names = "{.fn}"))
```

```
## # A tibble: 5 x 7
##   coef         mean       sd  median      mad     lower    upper
##   <chr>       <dbl>    <dbl>   <dbl>    <dbl>     <dbl>    <dbl>
## 1 change    3.02    3.31     2.00    2.01     0.0743    11.9
## 2 intercept 0.0681  0.0296   0.0692  0.0284   0.00757    0.126
## 3 sigma     0.152   0.00725  0.151   0.00714  0.138      0.166
## 4 slope1    0.00192 0.00699  0.00146 0.000878 0.0000874  0.00344
## 5 slope2    0.00112 0.000729 0.00104 0.000748 0.0000594  0.00278
```

```
# Just a few custom scale_* functions to make the plot pretty.
scale_fill_kawabata = function(...) {
  ggplot2:::manual_scale("fill",
               values = setNames(viridis::viridis(3, end = 0.8),
                                 c("Observations", "Predictions", "Expectations")),
               ...)
}
scale_color_kawabata = function(...) {
  ggplot2:::manual_scale("color",
               values = setNames(viridis::viridis(3, end = 0.8),
                                 c("Observations", "Predictions", "Expectations")),
               ...)
}
```

```
xlabel = "'Turn angle'~group('|', alpha, '|' )~(degree)"
ylabel = "'Velocity after s1 turn'~(m~s^{-1})"

alabel = sprintf("Change point posterior distribution (%0.1f, %0.1f - %0.1f; mean, 95%% HDI)",
```

```
                  tmp$emmean, tmp$lower.HPD, tmp$upper.HPD)
ggplot() +
  geom_freqpoly(aes(x = .value, y = stat(ndensity)/3),
                binwidth = 1, color = "grey50",
                data = changeem_post) +
  geom_vline(xintercept = tmp$emmean, color = "grey50") +
  annotate("text", x = tmp$upper.HPD, y = 0.1, label = alabel,
           vjust = 0, hjust = 0, color = "grey50") +
  geom_ribbon(aes(x = ea, ymin = lower, ymax = upper, fill = "Predictions"),
              data = pdata, alpha = 0.5) +
  geom_ribbon(aes(x = ea, ymin = lower, ymax = upper, fill = "Expectations"),
              data = edata, alpha = 0.5) +
  geom_point(aes(x = ea, y = cumd10, color = "Observations"), data = data,
             show.legend = F) +
  geom_line(aes(x = ea, y = estimate, color = "Expectations"), data = edata, size = 2,
            show.legend = F) +
  scale_x_continuous(parse(text = xlabel), limits = c(0, 150)) +
  scale_y_continuous(parse(text = ylabel), limits = c(0, 1.5)) +
  scale_fill_kawabata() +
  scale_color_kawabata() +
  theme(legend.position = c(0,1),
        legend.justification = c(0,1),
        legend.title = element_blank(),
        legend.background = element_blank())
```

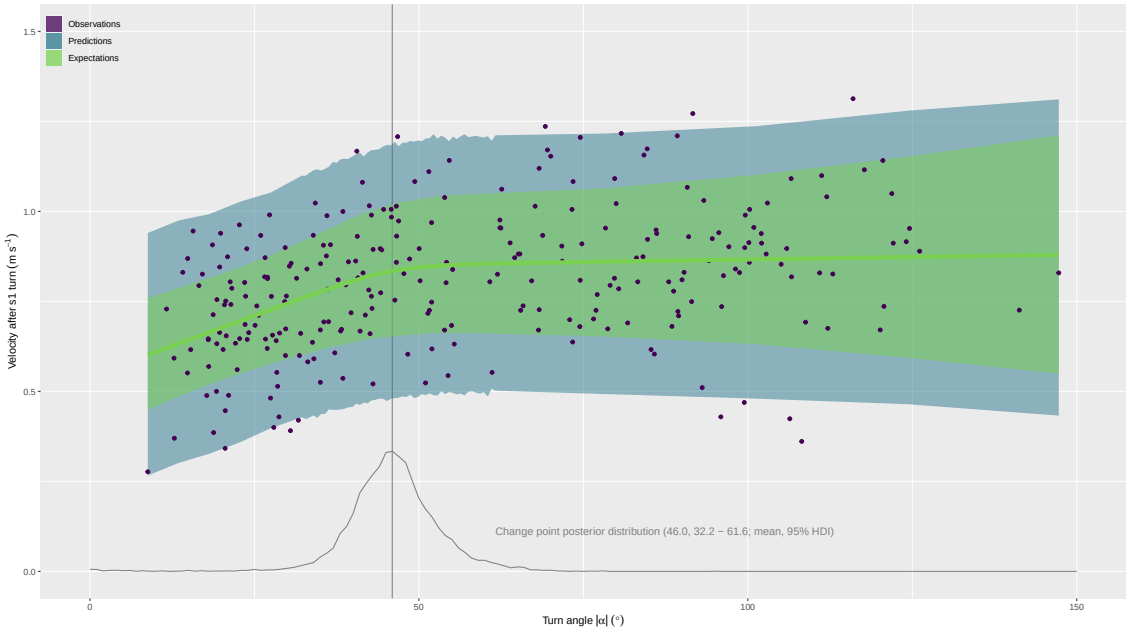

Figure 3: The observations, expectations, and predictions with the respective 95% quantiles.

8 Model validation of `bout1` with posterior predictive checks

```
parameters = usemodel |> get_variables()
```

```
mcmc_rank_hist(usemodel, regex_pars = "b_")
```

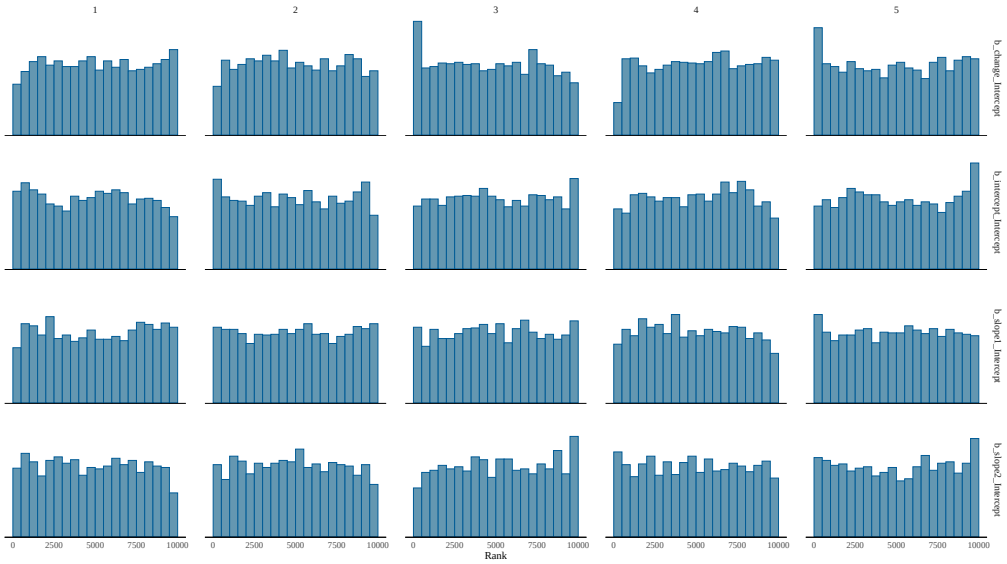

Figure 4: Rank plot of the MCMC draws for population coefficients. Each column represents a single chain. The histograms should appear uniform when the chains are well mixed.

```
mcmc_rank_hist(usemodel, regex_pars = "sd|sigma")
```

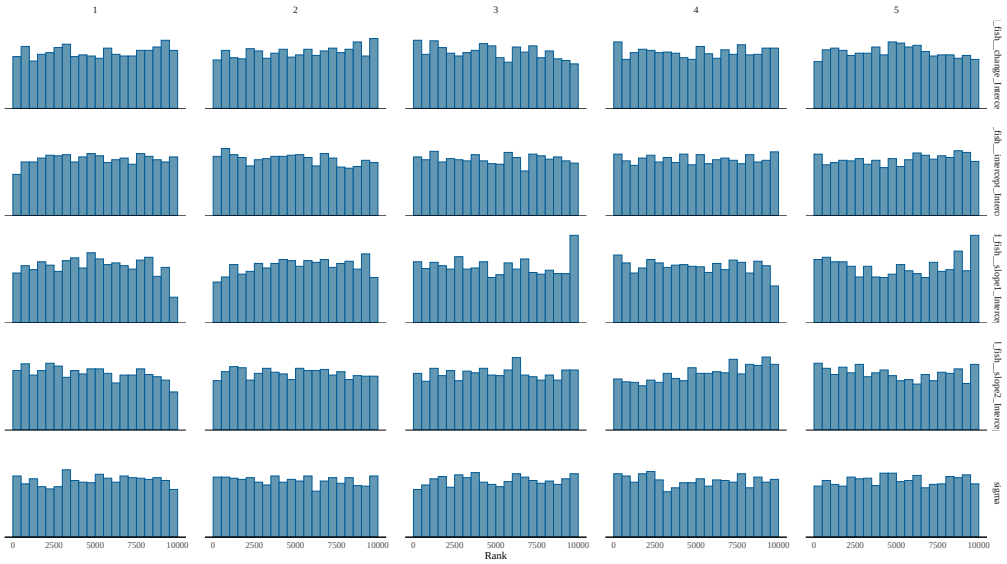

Figure 5: Rank plot of the MCMC draws for population standard deviations and the model σ. Each column represents a single chain. The histograms should appear uniform when the chains are well mixed.

```
pp_check(usemodel, type = "dens_overlay", ndraws = 50)
```

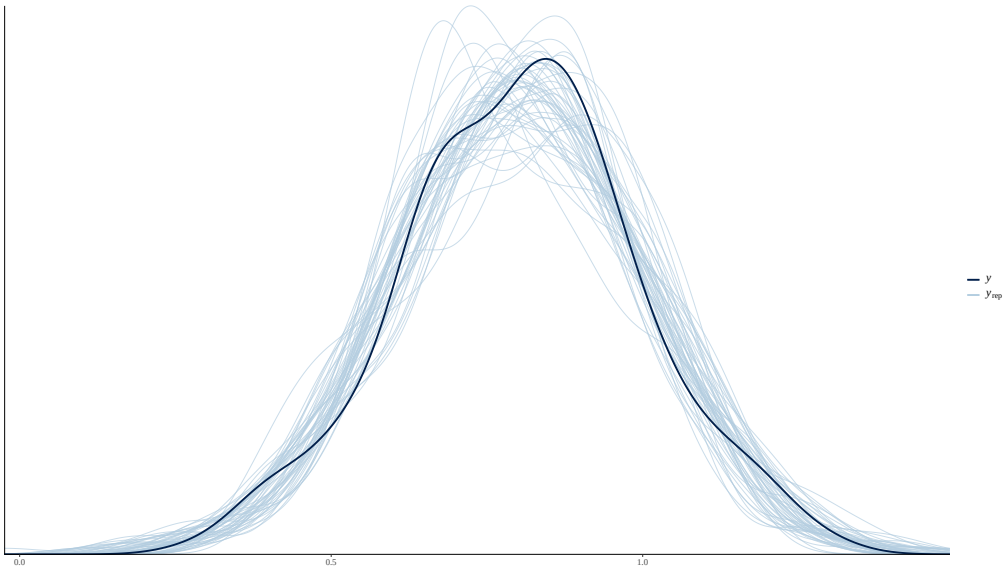

Figure 6: The kernel density estimate of the observations y and the results (y_{rep}) from the posterior predictive distribution of the model. If the model is correct, then we should be able to generate predictions that describe the observations. Therefore, the kernel density estimate and the posterior predictive distribution should appear similar.

```
pp_check(usemodel, type = "stat_2d", stat = c("mean", "sd"))
```

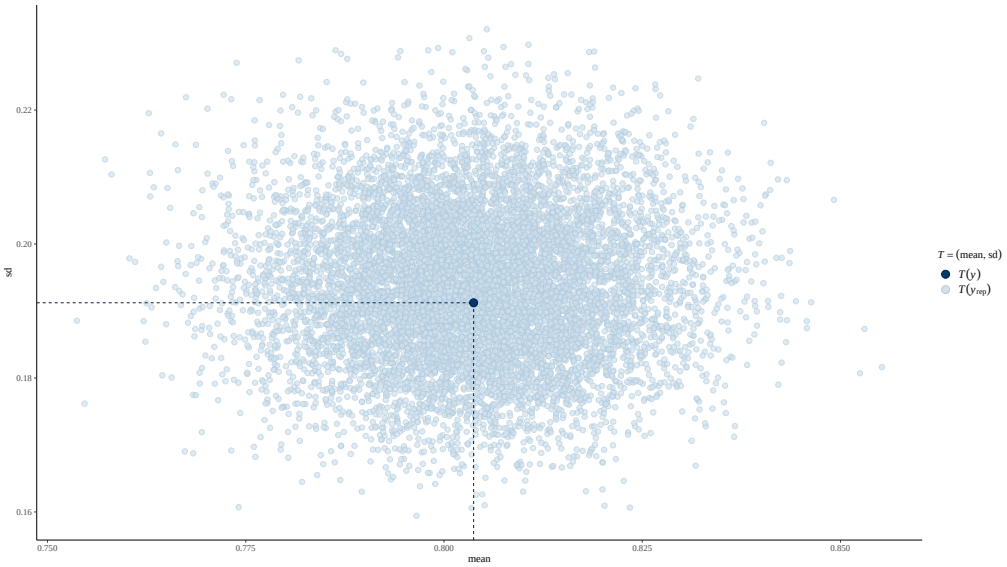

Figure 7: The the mean and standard deviation (sd) of the observations y and the joint distribution of (y_{rep}) from the posterior predictive distribution of the model.

```
pp_check(usemodel, type = "stat_2d", stat = c("median", "mad"))
```

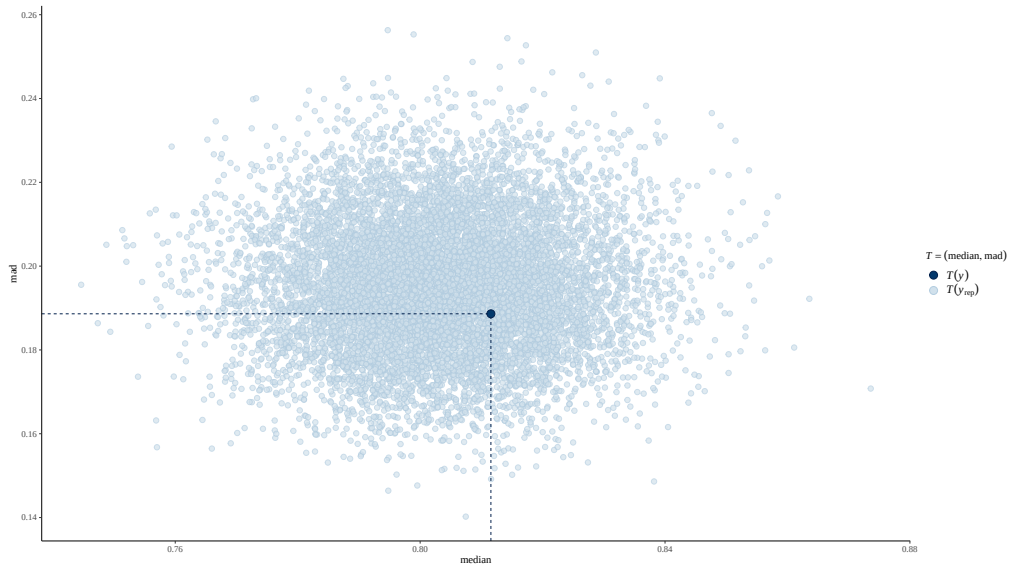

Figure 8: The the median and mean absolute deviation (mad) of the observations y and the joint distribution of (y_{rep}) from the posterior predictive distribution of the model.

9 Stan code for rstan

The original Stan models used in the manuscript.

9.1 Multilevel change-point model (model 1)

```
data {
  int<lower=0> N; // Total number of data
  int<lower=0> M; // Total number of fish
  int fish[N]; // Index for each fish

  int<lower=0> Npred; //Total number for prediction
  vector[Npred] model_ea;

  vector[N] cumd10;  // speed after s1 turn
  vector[N] ea;   // EA (angle)

  real mu_intercept;
  real mu_slope1;
  real mu_slope2;

  real sigma_intercept;
  real sigma_slope1;
  real sigma_slope2;

  real mu_change_point;
  real sigma_change_point;

}

// The parameters accepted by the model. Our model
// accepts two parameters 'mu' and 'sigma'.
parameters {
  real  global_intercept; // Intercept
```

```
  real  global_slope1; // Slope before changepoint
  real  global_slope2; // Slope after changepoint
  real<lower=0, upper=180>  global_change_point;

  real  global_tau_intercept; // Intercept
  real  global_tau_slope1; // Slope before changepoint
  real  global_tau_slope2; // Slope after changepoint
  real  global_tau_change_point;

  real<lower=0> global_sigma; // Error

  vector[M] unit_intercept; // Intercept
  vector[M] unit_slope1; // Slope before changepoint
  vector[M] unit_slope2; // Slope after changepoint
  vector[M] unit_change_point;
}
transformed parameters {
  vector[M] intercept; // Intercept
  vector[M] slope1; // Slope before changepoint
  vector[M] slope2; // Slope after changepoint
  vector<lower=0, upper=180>[M] change_point;

  intercept = global_intercept + unit_intercept * global_tau_intercept;
  slope1 = global_slope1 + unit_slope1 * global_tau_slope1;
  slope2 = global_slope2 + unit_slope2 * global_tau_slope2;
  change_point = global_change_point + unit_change_point * global_tau_change_point;

}
model {
  // Prior Distribution
  global_intercept ~ normal(mu_intercept, sigma_intercept);
  global_slope1 ~    normal(mu_slope1, sigma_slope1);
  global_slope2 ~    normal(mu_slope2, sigma_slope2);
  global_change_point ~ normal(mu_change_point, sigma_change_point);

  unit_intercept ~ normal(0.0, 1.0);
  unit_slope1 ~       normal(0.0, 1.0);
  unit_slope2 ~       normal(0.0, 1.0);
  unit_change_point ~ normal(0.0, 1.0);

  global_tau_intercept ~ cauchy(0, 2.5);
  global_tau_slope1 ~ cauchy(0, 2.5);
  global_tau_slope2 ~ cauchy(0, 2.5);
  global_tau_change_point ~ cauchy(0, 2.5);

  global_sigma ~ cauchy(0, 2.5); // Error prior

  // Calculate the likelihood
  for(n in 1:N) {
    if(ea[n] < change_point[fish[n]]) {
      // cumd10[n] ~ normal(intercept + slope1 * ea[n], sigma); // Does not save the true likelihood
      target += normal_lpdf(cumd10[n] | intercept[fish[n]] + slope1[fish[n]] * ea[n], global_sigma) ;
    } else {
      // cumd10[n] ~ normal(intercept + (slope1 * change_point) + slope2 * (ea[n] - change_point), sigma);
      target += normal_lpdf(cumd10[n] | intercept[fish[n]] +
                                        (slope1[fish[n]] * change_point[fish[n]]) +
                                         slope2[fish[n]] * (ea[n] - change_point[fish[n]]), global_sigma);
    }
  }
```

```
}

generated quantities {
  vector[N] log_lik;
  vector[Npred] yfit;
  vector[Npred] ypred;


   for(n in 1:N) {
    if(ea[n] < change_point[fish[n]]) {
      log_lik[n] = normal_lpdf(cumd10[n] | intercept[fish[n]] + slope1[fish[n]] * ea[n], global_sigma) ;
    } else {
      log_lik[n] = normal_lpdf(cumd10[n] | intercept[fish[n]] +
                                           (slope1[fish[n]] * change_point[fish[n]]) +
                                            slope2[fish[n]] * (ea[n] - change_point[fish[n]]), global_sigma);
    }
  }

  for(n in 1:Npred) {
    if(model_ea[n] < global_change_point) {
      yfit[n]  = global_intercept + global_slope1 * model_ea[n];
      ypred[n] = normal_rng(yfit[n], global_sigma);
    } else {
      yfit[n]  = global_intercept + (global_slope1 * global_change_point) +
      global_slope2 * (model_ea[n] - global_change_point);
      ypred[n] = normal_rng(yfit[n], global_sigma);
    }
  }


}

// x +=1 : x = x + 1
```

9.2 Multilevel constant slope model (model 2)

```
data {
  int<lower=0> N; // Total number of data
  int<lower=0> M; // Total number of fish
  int fish[N]; // Index for each fish

  int<lower=0> Npred; //Total number for prediction
  vector[Npred] model_ea;

  vector[N] cumd10;  // speed after s1 turn
  vector[N] ea;   // EA (angle)

  real mu_intercept;

  real sigma_intercept;


}

// The parameters accepted by the model. Our model
// accepts two parameters 'mu' and 'sigma'.
parameters {
  real  global_intercept; // Intercept
```

```
  real  global_tau_intercept; // Intercept

  real<lower=0> global_sigma; // Error

  vector[M] unit_intercept; // Intercept
}

transformed parameters {
  vector[M] intercept; // Intercept

  intercept = global_intercept + unit_intercept * global_tau_intercept;

}
model {
  // Prior Distribution
  global_intercept ~ normal(mu_intercept, sigma_intercept);

  unit_intercept ~ normal(0.0, 1.0);

  global_tau_intercept ~ cauchy(0, 2.5);

  global_sigma ~ cauchy(0, 2.5); // Error prior

  // Calculate the likelihood
  for(n in 1:N) {
      // cumd10[n] ~ normal(intercept + slope1 * ea[n], sigma); // Does not save the true likelihood
      target += normal_lpdf(cumd10[n] | intercept[fish[n]], global_sigma) ;
  }
}

generated quantities {
  vector[N] log_lik;
  vector[Npred] yfit;
  vector[Npred] ypred;


   for(n in 1:N) {
      log_lik[n] = normal_lpdf(cumd10[n] | intercept[fish[n]], global_sigma) ;
  }

  for(n in 1:Npred) {
      yfit[n]  = global_intercept;
      ypred[n] = normal_rng(yfit[n], global_sigma);
  }


}

// x +=1 : x = x + 1
```

9.3 Multilevel single slope model (model 3)

```
data {
  int<lower=0> N; // Total number of data
  int<lower=0> M; // Total number of fish
  int fish[N]; // Index for each fish

  int<lower=0> Npred; //Total number for prediction
  vector[Npred] model_ea;
```

```
  vector[N] cumd10;  // speed after s1 turn
  vector[N] ea;   // EA (angle)

  real mu_intercept;
  real mu_slope1;

  real sigma_intercept;
  real sigma_slope1;


}

// The parameters accepted by the model. Our model
// accepts two parameters 'mu' and 'sigma'.
parameters {
  real  global_intercept; // Intercept
  real  global_slope1; // Slope before changepoint

  real  global_tau_intercept; // Intercept
  real  global_tau_slope1; // Slope before changepoint

  real<lower=0> global_sigma; // Error

  vector[M] unit_intercept; // Intercept
  vector[M] unit_slope1; // Slope before changepoint
}
transformed parameters {
  vector[M] intercept; // Intercept
  vector[M] slope1; // Slope before changepoint

  intercept = global_intercept + unit_intercept * global_tau_intercept;
  slope1 = global_slope1 + unit_slope1 * global_tau_slope1;

}
model {
  // Prior Distribution
  global_intercept ~ normal(mu_intercept, sigma_intercept);
  global_slope1 ~    normal(mu_slope1, sigma_slope1);


  unit_intercept ~ normal(0.0, 1.0);
  unit_slope1 ~       normal(0.0, 1.0);

  global_tau_intercept ~ cauchy(0, 2.5);
  global_tau_slope1 ~ cauchy(0, 2.5);

  global_sigma ~ cauchy(0, 2.5); // Error prior

  // Calculate the likelihood
  for(n in 1:N) {
      // cumd10[n] ~ normal(intercept + slope1 * ea[n], sigma); // Does not save the true likelihood
      target += normal_lpdf(cumd10[n] | intercept[fish[n]] + slope1[fish[n]] * ea[n], global_sigma) ;
  }
}

generated quantities {
  vector[N] log_lik;
  vector[Npred] yfit;
```

```
  vector[Npred] ypred;


   for(n in 1:N) {
      log_lik[n] = normal_lpdf(cumd10[n] | intercept[fish[n]] + slope1[fish[n]] * ea[n], global_sigma) ;
  }

  for(n in 1:Npred) {
      yfit[n]  = global_intercept + global_slope1 * model_ea[n];
      ypred[n] = normal_rng(yfit[n], global_sigma);
  }


}

// x +=1 : x = x + 1
```

10 Session information

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Debian GNU/Linux 11 (bullseye)
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
## LAPACK: /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3
##
## locale:
##  [1] LC_CTYPE=en_US.utf8       LC_NUMERIC=C
##  [3] LC_TIME=ja_JP.UTF-8       LC_COLLATE=en_US.utf8
##  [5] LC_MONETARY=ja_JP.UTF-8   LC_MESSAGES=en_US.utf8
##  [7] LC_PAPER=ja_JP.UTF-8      LC_NAME=C
##  [9] LC_ADDRESS=C              LC_TELEPHONE=C
## [11] LC_MEASUREMENT=ja_JP.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] emmeans_1.7.2   patchwork_1.1.1 bayesplot_1.8.1 tidybayes_3.0.2
##  [5] brms_2.16.3     Rcpp_1.0.8      rmarkdown_2.12  forcats_0.5.1
##  [9] stringr_1.4.0   dplyr_1.0.8     purrr_0.3.4     readr_2.1.2
## [13] tidyr_1.2.0     tibble_3.1.6    ggplot2_3.3.5   tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##   [1] readxl_1.3.1         backports_1.4.1      RcppEigen_0.3.3.9.1
##   [4] plyr_1.8.6           igraph_1.2.11        svUnit_1.0.6
##   [7] splines_4.1.2        crosstalk_1.2.0      TH.data_1.1-0
##  [10] rstantools_2.1.1     inline_0.3.19        digest_0.6.29
##  [13] htmltools_0.5.2      viridis_0.6.2        fansi_1.0.2
##  [16] BH_1.78.0-0          magrittr_2.0.2       checkmate_2.0.0
##  [19] tzdb_0.2.0           modelr_0.1.8         RcppParallel_5.1.5
##  [22] matrixStats_0.61.0   vroom_1.5.7          xts_0.12.1
##  [25] sandwich_3.0-1       prettyunits_1.1.1    colorspace_2.0-3
##  [28] rvest_1.0.2          ggdist_3.1.1         haven_2.4.3
##  [31] xfun_0.30            callr_3.7.0          crayon_1.5.0
##  [34] jsonlite_1.8.0       lme4_1.1-28          survival_3.3-1
##  [37] zoo_1.8-9            glue_1.6.2           gtable_0.3.0
##  [40] distributional_0.3.0 pkgbuild_1.3.1       rstan_2.21.3
##  [43] abind_1.4-5          scales_1.1.1         mvtnorm_1.1-3
##  [46] DBI_1.1.2            miniUI_0.1.1.1       viridisLite_0.4.0
##  [49] xtable_1.8-4         diffobj_0.3.5        bit_4.0.4
##  [52] stats4_4.1.2         StanHeaders_2.21.0-7 DT_0.21
##  [55] htmlwidgets_1.5.4    httr_1.4.2           threejs_0.3.3
##  [58] arrayhelpers_1.1-0   posterior_1.2.1      ellipsis_0.3.2
##  [61] pkgconfig_2.0.3      loo_2.4.1            farver_2.1.0
##  [64] dbplyr_2.1.1         utf8_1.2.2           labeling_0.4.2
##  [67] tidyselect_1.1.2     rlang_1.0.2          reshape2_1.4.4
##  [70] later_1.3.0          munsell_0.5.0        cellranger_1.1.0
##  [73] tools_4.1.2          cli_3.2.0            generics_0.1.2
##  [76] broom_0.7.12         ggridges_0.5.3       evaluate_0.15
##  [79] fastmap_1.1.0        yaml_2.3.5           bit64_4.0.5
##  [82] processx_3.5.2       knitr_1.37           fs_1.5.2
##  [85] nlme_3.1-155         mime_0.12            projpred_2.0.2
##  [88] xml2_1.3.3           compiler_4.1.2       shinythemes_1.2.0
##  [91] rstudioapi_0.13      gamm4_0.2-6          reprex_2.0.1
##  [94] stringi_1.7.6        ps_1.6.0             Brobdingnag_1.2-7
##  [97] lattice_0.20-45      Matrix_1.4-0         nloptr_2.0.0
## [100] markdown_1.1         shinyjs_2.1.0        tensorA_0.36.2
## [103] vctrs_0.3.8          pillar_1.7.0         lifecycle_1.0.1
## [106] bridgesampling_1.1-2 estimability_1.3     httpuv_1.6.5
## [109] extraDistr_1.9.1     R6_2.5.1             bookdown_0.24
## [112] promises_1.2.0.1     gridExtra_2.3        codetools_0.2-18
## [115] boot_1.3-28          colourpicker_1.1.1   MASS_7.3-55
## [118] gtools_3.9.2         assertthat_0.2.1     withr_2.5.0
## [121] shinystan_2.6.0      multcomp_1.4-18      mgcv_1.8-39
## [124] parallel_4.1.2       hms_1.1.1            grid_4.1.2
## [127] coda_0.19-4          minqa_1.2.4          shiny_1.7.1
```

```
## [130] lubridate_1.8.0      base64enc_0.1-3     tinytex_0.37
## [133] dygraphs_1.1.1.6
```

References

Brilleman SL, Howe LD, Wolfe R, Tilling K. 2017. Bayesian piecewise linear mixed models with a random change point: An application to BMI rebound in childhood. *Epidemiology* **28**:827–833. doi:10.1097/EDE.0000000000000723

Bürkner P-C. 2021a. Brms: Bayesian regression models using stan.

Bürkner P-C. 2021b. Bayesian item response modeling in R with brms and Stan. *Journal of Statistical Software* **100**:1–54. doi:10.18637/jss.v100.i05

Bürkner P-C. 2018. Advanced Bayesian multilevel modeling with the R package brms. *The R Journal* **10**:395–411. doi:10.32614/RJ-2018-017

Bürkner P-C. 2017. brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software* **80**:1–28. doi:10.18637/jss.v080.i01

Gabry J, Češnovar R. 2021. Cmdstanr: R interface to cmdstan.

Gabry J, Mahr T. 2021. Bayesplot: Plotting for bayesian models.

Gabry J, Simpson D, Vehtari A, Betancourt M, Gelman A. 2019. Visualization in bayesian workflow. *J R Stat Soc A* **182**:389–402. doi:10.1111/rssa.12378

Guo J, Gabry J, Goodrich B, Weber S. 2021. Rstan: R interface to stan.

Kay M. 2022. Tidybayes: Tidy data and geoms for bayesian models.

Lenth RV. 2022. Emmeans: Estimated marginal means, aka least- squares means.

Pedersen TL. 2020. Patchwork: The composer of plots.

Wickham H. 2021. Tidyverse: Easily install and load the tidyverse.

Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H. 2019. Welcome to the tidyverse. *Journal of Open Source Software* **4**:1686. doi:10.21105/joss.01686