Supplementary information for the paper 'Customizing material into embodied cap by sponge crab'

This document is a supplementary information for the paper 'Customizing material into embodied cap by sponge crab' by Harada and Kagaya.

The basic conceptural framework for data analysis in the paper is a Bayesian framework based on the mathematical theory founded by Dr. Akaike and Dr. Watanabe (Watanabe, 'Mathematical Theory of Bayesian Statistics', 2018). The author of this document wrote an example of model selection by Widely Applicable Information Criterion (WAIC) using the data of 'artificial case' (http://rpubs.com/katzkagaya/460937). It may be of help for clarifying what the analysis of this document is doing, by using numerically generated 'artificial' data and several relatively simple hierarchical models .

In this study, we deal with the behavioral data of 'natural case' (Watanabe, 2018). Therefore, we do not know the true distribution generating the data. Here, we applied 26 statistical models in total for each behavioral aspect of the crab and compare the performances of models in terms of predictatility by WAIC. The models are categorized into four observed variables as random response variables.

1. selected sizes of sponge — 6 models
2. cut (removed) off area of the sponge which is not used for making cap — 8 models
3. hole area size digged for the cap — 6 models
4. used time for making the cap — 6 models

We construct generalized linear models for these random variables as response variables. Additionally, a key feature of the dataset is that in several cases we repeatedly observed the data for each crab, because we are interested in the 'individuality' of each crab. Therefore, we parametrized this point as hierarchical structure into the model. After building up each predictive distributions, we compared the appropriateness of each model using WAIC including the hierarchical model. The best performed models for the four behaviours were described in the Materials and Methods section in the main text. We here walk through all of the models, with its implementation in Stan and plots for the main figures with its R codes.

load libraries

Load packages used in this analysis.

```
library(tidyverse)
library(rstan)
library(forcats)
library(ggrepel)
library(ggforce)
library(lubridate)
```

utilities

We define several utility functions used in this analysis.

```
nth_dens <- function(pdens, n){
  # sample n from the highest density in decreasing order
  # n / 512 points
    dens <- density(pdens)
    nth_x <- c()
    for (i in 1:n){
      nth_x <- c(nth_x,
                 dens$y %>% order(decreasing=T) %>% nth(i) %>% dens$x[.]
 )
    }
    nth_x
  }

waic <- function(log_lik) {
  # calculate WAIC in the 'focus' of this study, cf. http://rpubs.com/kat
zkagaya/460937
  # Shortly, the definition of WAIC is different when the focus of the pr
ediction change
  Tn <- - mean(log(colMeans(exp(log_lik))))
  V_div_N <- mean(colMeans(log_lik^2) - colMeans(log_lik)^2)
  waic <- Tn + V_div_N
  return(waic)
}

check_rhat <- function(fit){
  # We are not using this function in this document, but you can check Rh
at
  all(summary(fit)$summary[,"Rhat"] <= 1.10, na.rm=T)
}



add_chosen_col_choice <- function(d){
  # needed for the preprocessing the data for the analysis of sponge choi
ce (#1)
```

```
  d <- d %>% arrange(id, choice)
  x <- paste(d$id, d$choice, sep="") %>% factor() %>% fct_inorder()
  xx <- table(x)
  chosen_ind <- names(xx)
  chosen_num <- c()
  for (i in xx){
    chosen_num <- c(chosen_num, 1:i)
  }
  d$chosen_num <- chosen_num
  d
}


add_chosen_col_days <- function(d){
  # needed for the preprocessing the data for the analysis of the 'days'
for making cap (#4)
  d <- d %>% arrange(id, days_to_choice)
  x <- paste(d$id, d$days_to_choice, sep="") %>% factor() %>% fct_inorder
()
  xx <- table(x)
  chosen_ind <- names(xx)
  chosen_num <- c()
  for (i in xx){
    chosen_num <- c(chosen_num, 1:i)
  }
  d$chosen_num <- chosen_num
  d
}
```

1. sponge choice

preprocessing

Read data from the gist repository, preprocess the data, and construct the data as 'list' data type given to the Stan code later.

```
d <- read_csv('https://gist.github.com/kagaya/0d309397300b7e8294fe53c44b6
fc525/raw/2cfe44334da66f22a3db59caf0d5e6967434cda1/Dromiidae.csv')

## Parsed with column specification:
## cols(
##   test_num = col_integer(),
##   id = col_integer(),
##   shell_width = col_double(),
##   gender = col_character(),
```

```
##   start_date = col_character(),
##   end_date = col_character(),
##   choice = col_character(),
##   making_process = col_character(),
##   whole_px = col_integer(),
##   hole_px = col_integer(),
##   cm_per_px = col_double(),
##   leg_lack = col_character(),
##   cutting = col_character()
## )

d <- d %>%
  select(c('shell_width', 'choice',
           #'whole_px', 'cm_per_px',
           'id', 'leg_lack'           ))

d <- d[complete.cases(d), ]



# renumber animal id from 1
id <- as.factor(d$id)
levels(id) <- as.character(1:length(levels(id)))
d$id <- as.integer(id)

d$choice <- as.factor(d$choice)
levels(d$choice) <- c(2,1,3) # label M,L,No, as 1,2,3
d$choice <- as.integer(as.character(d$choice))

d$leg_lack <- as.factor(d$leg_lack)
levels(d$leg_lack) <- c(2,3,1) # label no_leg_lack, A, C, as 1, 2, 3
d$leg_lack <- as.integer(as.character(d$leg_lack))

data <- list(N=nrow(d),
             K=max(d$choice),
             L=max(d$id),
             Y=d$choice,
             #cap_ex_size=d$cap_ex_size,
             C_width=d$shell_width,
             Leg_lack=d$leg_lack,
             ID=d$id)
```

choice, model 1_1

Stan code

```
// model 1_1 in Harada and Kagaya, 2018
functions {
```

```
  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
      }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Y, real cwl, real cwno, real ll, real lno, real C_width, re
al Leg_lack,
          real bias_no0,
          real bias_l_K){
    vector[3] mu;
    mu[1] = 0;
    mu[2] = bias_l_K + cwl*C_width + ll*Leg_lack;
    mu[3] = bias_no0 + cwno*C_width + lno*Leg_lack;
    return(categorical_logit_lpmf(Y | mu));
  }


  real log_lik_Simpson_rest(int Y, real cwl, real cwno, real ll, real lno,
 real C_width, real Leg_lack,
                            real bias_no0,
                            real bias_l_lower, real bias_l_upper,
                            int M){
    vector[M+1] lp;
    real h;
    h = (bias_l_upper - bias_l_lower)/M;
    lp[1] = f2(Y, cwl, cwno, ll, lno, C_width, Leg_lack,
                bias_no0,
                bias_l_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Y, cwl, cwno, ll, lno, C_width, Leg_lack,
                            bias_no0,
                            bias_l_lower + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Y, cwl, cwno, ll, lno, C_width, Leg_lack,
```

```
                                bias_no0,
                                bias_l_lower + h*2*m);
    lp[M+1] = f2(Y, cwl, cwno, ll, lno, C_width, Leg_lack,
                  bias_no0,
                  bias_l_upper);
    return(log(h/3) + log_sum_exp(lp));
  }
}

data {
    int N;
    int K;
    int L;
    int<lower=1, upper=K> Y[N];
    real C_width[N];
    real C_width_new[50];
    int Leg_lack[N];
    int<lower=1, upper=L> ID[N];
}

parameters {
    real bias_l[L];
    real bias_l0;
    real ll;
    real cwl;
    real<lower=0> s_bias_l;

    real cwno;
    real bias_no0;
    real lno;
}

transformed parameters {
    matrix[N,K] mu;

    for (n in 1:N){
        mu[n,1] = 0;
        mu[n,2] =  bias_l[ID[n]] + cwl*C_width[n] + ll*Leg_lack[n];
        mu[n,3] =  bias_no0 + cwno*C_width[n] + lno*Leg_lack[n];
    }
}

model {
    for (l in 1:L){
        bias_l[l] ~ normal(bias_l0, s_bias_l);
    }
```

```
    for (n in 1:N){
        Y[n] ~ categorical_logit(mu[n,]');
    }
}

generated quantities {
  vector[N] log_lik;
  real log_lik_l;
  int pred_Y[50];
  matrix[50,3] mu_new;
  real bias_l_new;

  // for making computing fast, separately compute loglik unrelated to Y
[n] and other parameters
  log_lik_l = log_lik_Simpson(bias_l0, s_bias_l,
                              bias_l0 - 5*s_bias_l, bias_l0 + 5*s_bias_l,
 100);
  for (n in 1:N){
    log_lik[n] = log_lik_l +
                 log_lik_Simpson_rest(Y[n],
                                      cwl, cwno, ll, lno, C_width[n], Leg
_lack[n],
                                      bias_no0,
                                      bias_l0 - 5*s_bias_l, bias_l0 + 5*s
_bias_l,
                                      100);
  }

  bias_l_new = normal_rng(bias_l0, s_bias_l);
  for ( k in 1:50) {
     mu_new[k,1] = 0;
     mu_new[k,2] =  bias_l_new + cwl*C_width_new[k] + ll; // Leg_lack is
fixed to 1
     mu_new[k,3] =  bias_no0 + cwno*C_width_new[k] + lno; // Leg_lack is
fixed to 1
     pred_Y[k] = categorical_logit_rng(mu_new[k,]');
  }

}
```

Note that the parameters $bias\_l[l], l = 1, \ldots, L$ are integrated out in the 'generated quantities' block using Simpson's rule to compute the WAIC in the focus of our prediction.

sampling

```
# model <- stan_model("choice_RE_random_intercept3_cw_leg.stan")
# fit <- sampling(model, ###
#             data=data, seed=1234,
#             chains=4, iter=6000, warmup=1000, thin=1)
#
# save(fit, file= "choice_RE_random_intercept3_cw_leg.rdata")
load("choice_RE_random_intercept3_cw_leg.rdata")
```

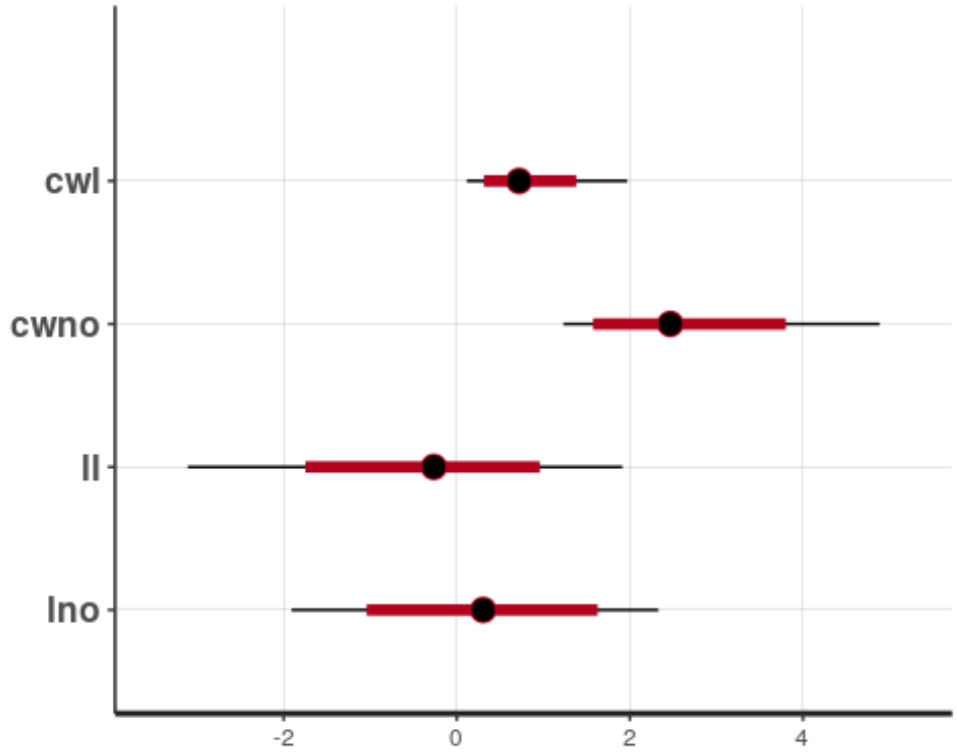
Please save the Stan code above as 'choice_RE_random_intercept3_cw_leg.stan' and uncomment the region to perform sampling in your environment. Here we just load the result saved as 'choice_RE_random_intercept3_cw_leg.rdata' in our local environment (the session information is added the last part of this document). Do the same thing for all of the models below.

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w11
w11

## [1] -2.012721
```

plot posterior distributions of the parameters

```
stan_plot(fit, c("cwl", "cwno", "ll", "lno"))

## ci_level: 0.8 (80% intervals)

## outer_level: 0.95 (95% intervals)
```



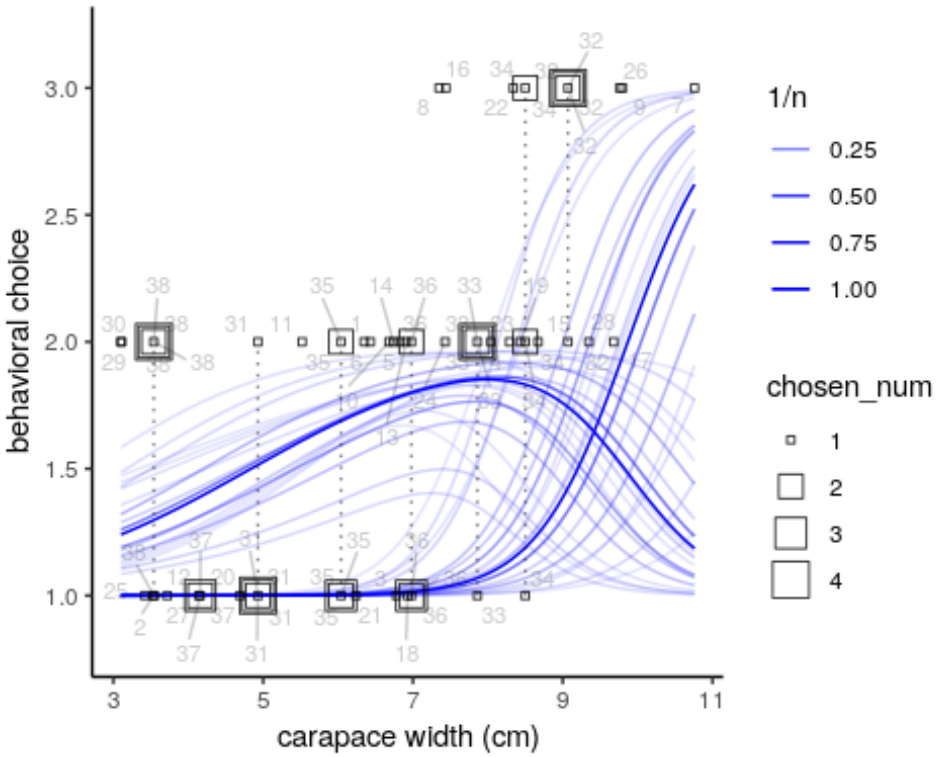
plot figure

This is the code generating the figure 5.

```
plot_choice_random_intercept_lines <- function(d, fit){
  # load("choice_RE_random_intercept3_cw_leg.rdata")

  ms <- rstan::extract(fit)
  new_C_width <- seq(min(d$shell_width), max(d$shell_width), length.out=5
00)

  line_n <- 20
  bias_l0 <- nth_dens(ms$bias_l0, line_n)
  cw_l <- nth_dens(ms$cwl, line_n)
  bias_no0 <- nth_dens(ms$bias_no0, line_n)
  cw_no <- nth_dens(ms$cwno, line_n)

  mu2 <- mu3 <- theta2 <- theta3 <- matrix(nrow=500, ncol=line_n)

  for (j in 1:line_n) {
    for (i in 1:500){
      mu2[i, j] <- bias_l0[j] + cw_l[j]*new_C_width[i]
      mu3[i, j] <- bias_no0[j] + cw_no[j]*new_C_width[i]
```

```
      theta2[i, j] <- 1 + exp(mu2[i, j])/(1 + exp(mu2[i, j]) + exp(mu3[i,
 j]))
      theta3[i, j] <- 1 + 2 * exp(mu3[i, j])/(1 + exp(mu2[i, j]) + exp(mu
3[i, j]))
    }
  }

  pred <- tibble()
  for (j in 1:line_n) {
    pred <- rbind(pred, tibble(n=j,
                               carapace_width=new_C_width,
                               theta2=theta2[,j],
                               theta3=theta3[,j]))
  }

  gid <- levels(d$id)[table(d$id) > 1]

  mul <- factor(rep("s", dim(d)[1]), levels=c("g", "s"))
  for (i in 1:dim(d)[1]){
    if (d$id[i] %in% gid){
      mul[i] <- c("g")
    }
  }

  d$mul <- mul

  dd <- d
  dd$choice <- as.integer(d$choice)
  d_seg <- dd %>%
    group_by(id) %>%
    summarise( choice_min=min(choice),
               choice_max=max(choice),
               shell_width=first(shell_width)) %>%
    drop_na()

  d <- add_chosen_col_choice(d)

  plt <- ggplot(d, aes(shell_width, choice)) +
    geom_line(aes(carapace_width, theta2, group=n, alpha=1/n), colour="bl
ue", data=pred) +
    geom_line(aes(carapace_width, theta3, group=n, alpha=1/n), colour="bl
ue", data=pred) +
    geom_point(aes(size=chosen_num), pch=0, alpha=0.8) +
    geom_segment(
      data = d_seg,
      alpha = 0.5,
```

```
      lty=3,
      aes(x = shell_width,
          y = choice_min,
          xend = shell_width,
          yend = choice_max)) +
    geom_text_repel(aes(shell_width, choice, label=id), alpha=0.2, size=
3) +
    scale_shape_manual(values=c(1,3)) +
    scale_fill_gradient(low="white", high="gray") +
    ylim(c(0.8, 3.2)) +
    xlab("carapace width (cm)") +
    ylab("behavioral choice") +
    theme_classic()
  return(plt)
}
plot_choice_random_intercept_lines(d, fit)
```



choice, model 1_2

Stan code

```
functions {
  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
```

```
    }

real log_lik_Simpson(real mu, real s, real a, real b, int M) {
  vector[M+1] lp;
  real h;
  h = (b-a)/M;
  lp[1] = f(mu, s, a);
  for (m in 1:(M/2))
    lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
  for (m in 1:(M/2-1))
    lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
  lp[M+1] = f(mu, s, b);
  return(log(h/3) + log_sum_exp(lp));
}

real f2(int Y,
        real cw_l,
        real cw_no,
        real C_width,
        real bias_no0,
        real bias_l_K){
  vector[3] mu;
  mu[1] = 0;
  mu[2] = bias_l_K + cw_l*C_width;
  mu[3] = bias_no0 + cw_no*C_width;
  return(categorical_logit_lpmf(Y | mu));
}


real log_lik_Simpson_rest(int Y, real cw_l, real cw_no,
                          real C_width,
                          real bias_no0,
                          real bias_l_lower, real bias_l_upper,
                          int M){
  vector[M+1] lp;
  real h;
  h = (bias_l_upper - bias_l_lower)/M;
  lp[1] = f2(Y, cw_l, cw_no, C_width,
             bias_no0,
             bias_l_lower);
  for (m in 1:(M/2))
    lp[2*m] = log(4) + f2(Y, cw_l, cw_no,  C_width,
                          bias_no0,
                          bias_l_lower + h*(2*m-1));
  for (m in 1:(M/2-1))
    lp[2*m+1] = log(2) + f2(Y, cw_l, cw_no,  C_width,
```

```
                                bias_no0,
                                bias_l_lower + h*2*m);
    lp[M+1] = f2(Y, cw_l, cw_no,  C_width,
                  bias_no0,
                  bias_l_upper);
    return(log(h/3) + log_sum_exp(lp));
  }
}

data {
    int N;
    int K;
    int L;
    int<lower=1, upper=K> Y[N];
    real C_width[N];
    int Leg_lack[N];
    int<lower=1, upper=L> ID[N];
}

parameters {
    real bias_l[L];
    real bias_l0;
    real cw_l;
    real<lower=0> s_bias_l;

    real bias_no0;
    real cw_no;
}

transformed parameters {
    matrix[N,K] mu;

    for (n in 1:N){
        mu[n,1] = 0;
        mu[n,2] =  bias_l[ID[n]] + cw_l*C_width[n];
        mu[n,3] =  bias_no0 + cw_no*C_width[n];
    }
}

model {
    for (l in 1:L){
        bias_l[l] ~ normal(bias_l0, s_bias_l);
    }
    for (n in 1:N){
        Y[n] ~ categorical_logit(mu[n,]');
    }
```

```
}

generated quantities {
  vector[N] log_lik;
  real log_lik_l;
  log_lik_l = log_lik_Simpson(bias_l0, s_bias_l,
                              bias_l0 - 5*s_bias_l, bias_l0 + 5*s_bias_l,
 100);
  for (n in 1:N){
    log_lik[n] = log_lik_l +
                 log_lik_Simpson_rest(Y[n],
                                      cw_l, cw_no,
                                      C_width[n],
                                      bias_no0,
                                      bias_l0 - 5*s_bias_l, bias_l0 + 5*s
_bias_l,
                                      100);
  }
}
```

sampling

```
# fit <- stan(file='choice_RE_random_intercept3.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=6000, warmup=1000)
# save(fit, file="choice_RE_random_intercept3.rdata")

load("choice_RE_random_intercept3.rdata")

ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w12
w12

## [1] -1.867784
```

choice, model 1_3

Stan code

```
functions {
  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
      }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
```

```
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Y,
          real bias_no0,
          real bias_l_K){
    vector[3] mu;
    mu[1] = 0;
    mu[2] = bias_l_K;
    mu[3] = bias_no0;
    return(categorical_logit_lpmf(Y | mu));
  }


  real log_lik_Simpson_rest(int Y,
                            real bias_no0,
                            real bias_l_lower, real bias_l_upper,
                            int M){
    vector[M+1] lp;
    real h;
    h = (bias_l_upper - bias_l_lower)/M;
    lp[1] = f2(Y,
               bias_no0,
               bias_l_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Y,
                            bias_no0,
                            bias_l_lower + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Y,
                              bias_no0,
                              bias_l_lower + h*2*m);
    lp[M+1] = f2(Y,
                 bias_no0,
                 bias_l_upper);
    return(log(h/3) + log_sum_exp(lp));
  }
}

data {
```

```
    int N;
    int K;
    int L;
    int<lower=1, upper=K> Y[N];
    real C_width[N];
    int Leg_lack[N];
    int<lower=1, upper=L> ID[N];
}

parameters {
    real bias_l[L];
    real bias_l0;
    real<lower=0> s_bias_l;

    real bias_no0;
}

transformed parameters {
    matrix[N,K] mu;

    for (n in 1:N){
        mu[n,1] = 0;
        mu[n,2] =  bias_l[ID[n]];
        mu[n,3] =  bias_no0;
    }
}

model {
    for (l in 1:L){
        bias_l[l] ~ normal(bias_l0, s_bias_l);
    }
    for (n in 1:N){
        Y[n] ~ categorical_logit(mu[n,]');
    }
}

generated quantities {
  vector[N] log_lik;
  real log_lik_l;
  log_lik_l = log_lik_Simpson(bias_l0, s_bias_l,
                              bias_l0 - 5*s_bias_l, bias_l0 + 5*s_bias_l,
 100);
  for (n in 1:N){
    log_lik[n] = log_lik_l +
                 log_lik_Simpson_rest(Y[n],
                                      bias_no0,
```

```
                                        bias_l0 - 5*s_bias_l, bias_l0 + 5*s
_bias_l,
                                        100);
  }
}
```

sampling

```
# fit <- stan(file='choice_RE_random_intercept3_only.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=6000, warmup=1000, thin=1)
# save(fit, file="choice_RE_random_intercept3_only.rdata")
load("choice_RE_random_intercept3_only.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w13
w13

## [1] -0.8782529
```

choice, model 1_4

Stan code

```
functions {
  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
      }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Y,
          real ll,
          real lno,
          real Leg_lack,
          real bias_no0,
```

```
          real bias_l_K){
    vector[3] mu;
    mu[1] = 0;
    mu[2] = bias_l_K + ll*Leg_lack;
    mu[3] = bias_no0 + lno*Leg_lack;
    return(categorical_logit_lpmf(Y | mu));
  }


  real log_lik_Simpson_rest(int Y, real ll, real lno,
                            real Leg_lack,
                            real bias_no0,
                            real bias_l_lower, real bias_l_upper,
                            int M){
    vector[M+1] lp;
    real h;
    h = (bias_l_upper - bias_l_lower)/M;
    lp[1] = f2(Y, ll, lno, Leg_lack,
                bias_no0,
                bias_l_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Y, ll, lno,  Leg_lack,
                            bias_no0,
                            bias_l_lower + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Y, ll, lno,  Leg_lack,
                              bias_no0,
                              bias_l_lower + h*2*m);
    lp[M+1] = f2(Y, ll, lno,  Leg_lack,
                  bias_no0,
                  bias_l_upper);
    return(log(h/3) + log_sum_exp(lp));
  }
}

data {
    int N;
    int K;
    int L;
    int<lower=1, upper=K> Y[N];
    real C_width[N];
    int Leg_lack[N];
    int<lower=1, upper=L> ID[N];
}

parameters {
```

```
    real bias_l[L];
    real bias_l0;
    real ll;
    real<lower=0> s_bias_l;

    real bias_no0;
    real lno;
}

transformed parameters {
    matrix[N,K] mu;

    for (n in 1:N){
        mu[n,1] = 0;
        mu[n,2] =  bias_l[ID[n]] + ll*Leg_lack[n];
        mu[n,3] =  bias_no0 + lno*Leg_lack[n];
    }
}

model {
    for (l in 1:L){
        bias_l[l] ~ normal(bias_l0, s_bias_l);
    }
    for (n in 1:N){
        Y[n] ~ categorical_logit(mu[n,]');
    }
}

generated quantities {
  vector[N] log_lik;
  real log_lik_l;
  log_lik_l = log_lik_Simpson(bias_l0, s_bias_l,
                              bias_l0 - 5*s_bias_l, bias_l0 + 5*s_bias_l,
 100);
  for (n in 1:N){
    log_lik[n] = log_lik_l +
                 log_lik_Simpson_rest(Y[n],
                                      ll, lno,
                                      Leg_lack[n],
                                      bias_no0,
                                      bias_l0 - 5*s_bias_l, bias_l0 + 5*s
_bias_l,
                                      100);
  }
}
```

sampling

```
# fit <- stan(file='choice_RE_random_intercept3_leg.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=6000, warmup=1000, thin=1)
# save(fit, "choice_RE_random_intercept3_leg.rdata")
load("choice_RE_random_intercept3_leg.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w14
w14

## [1] -0.7828847
```

choice, model 1_5

Stan code

```
data {
    int N;
    int K;
    int L;
    int<lower=1, upper=K> Y[N];
    real C_width[N];
    int Leg_lack[N];
    int<lower=1, upper=L> ID[N];
}

parameters {
    real bias_l0;
    real cw_l;

    real bias_no0;
    real cw_no;
}

transformed parameters {
    matrix[N,K] mu;

    for (n in 1:N){
        mu[n,1] = 0;
        mu[n,2] =  bias_l0 + cw_l*C_width[n];
        mu[n,3] =  bias_no0 + cw_no*C_width[n];
    }
}

model {
```

```
    for (n in 1:N){
        Y[n] ~ categorical_logit(mu[n,]');
    }
}

generated quantities {
  vector[N] log_lik;
  for (n in 1:N){
    log_lik[n] = categorical_logit_lpmf(Y[n] | mu[n,]');
  }
}
```

sampling

```
# fit <- stan(file='choice_no_RE3.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=6000, warmup=1000)
# save(fit, file="choice_no_RE3.rdata")
load("choice_no_RE3.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w15
w15

## [1] 0.8536406
```

choice, model 1_6

Stan code

```
data {
    int N;
    int K;
    int L;
    int<lower=1, upper=3> Y[N];
    real C_width[N];
    int Leg_lack[N];
}

parameters {
    real bias_l0;
    real cw_l;
    real ll_l;

    real bias_no0;
    real cw_no;
    real ll_no;
```

```
}

transformed parameters {
    matrix[N,K] mu;

    for (n in 1:N){
        mu[n,1] = 0;
        mu[n,2] =  bias_l0 + cw_l*C_width[n] + ll_l*Leg_lack[n];
        mu[n,3] =  bias_no0 + cw_no*C_width[n] + ll_no*Leg_lack[n];
    }
}

model {
    for (n in 1:N){
        Y[n] ~ categorical_logit(mu[n,]');
    }
}

generated quantities {
  real log_lik[N];
  for (n in 1:N){
    log_lik[n] = categorical_logit_lpmf(Y[n] | mu[n,]');
  }
}
```

sampling

```
# fit <- stan(file='choice_no_RE.stan',
#             data=data, seed=1234,
#             chains=4, iter=2000, warmup=500, thin=1)
# save(fit, file="choice_no_RE.rdata")
load("choice_no_RE.rdata")
```

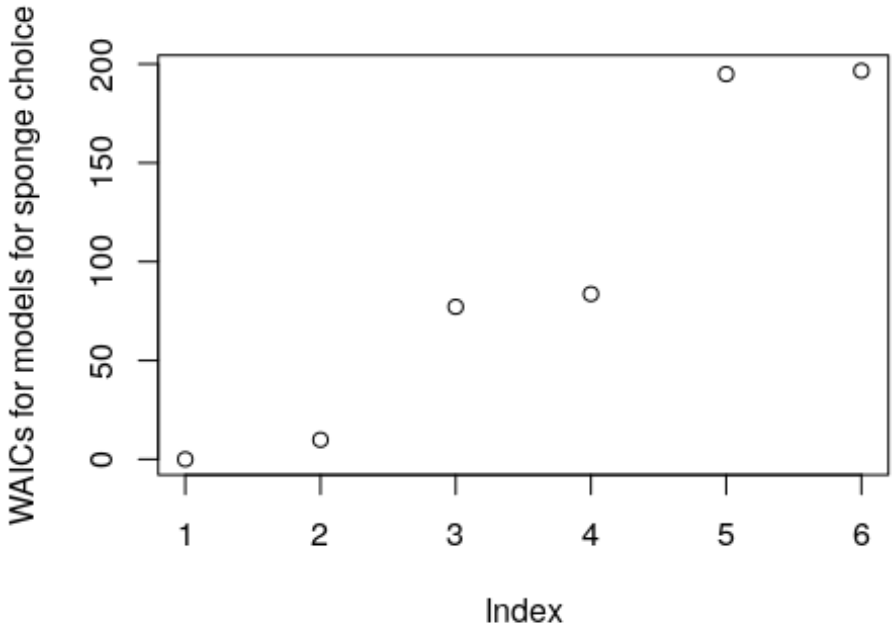
WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w16
w16
```

```
## [1] 0.8780253
```

comparing WAIC values in AIC scale

```
waicv_m1 <- c(w11, w12, w13, w14, w15, w16) * data$N
waicv_m1 <- waicv_m1 - min(waicv_m1)
plot(waicv_m1, ylab=c("WAICs for models for sponge choice"))
```



2. sponge cutting

preprocessing

```
d <- read_csv('https://gist.github.com/kagaya/0d309397300b7e8294fe53c44b6fc525/raw/2cfe44334da66f22a3db59caf0d5e6967434cda1/Dromiidae.csv')
```

```
## Parsed with column specification:
## cols(
##   test_num = col_integer(),
##   id = col_integer(),
##   shell_width = col_double(),
##   gender = col_character(),
##   start_date = col_character(),
##   end_date = col_character(),
##   choice = col_character(),
##   making_process = col_character(),
##   whole_px = col_integer(),
##   hole_px = col_integer(),
##   cm_per_px = col_double(),
##   leg_lack = col_character(),
##   cutting = col_character()
## )
```

```r
d <- d %>%
  dplyr::select(c('shell_width', 'choice',
           'whole_px', 'cm_per_px',
           'id', 'leg_lack', 'cutting' ))

d <- d[complete.cases(d), ]
## calculate exterior cap size
d$cap_ex_size <- d$whole_px * d$cm_per_px^2 / 1.15

# renumber animal id from 1
id <- as.factor(d$id)
levels(id) <- as.character(1:length(levels(id)))
d$id <- as.integer(id)

d$choice <- as.factor(d$choice)
levels(d$choice) <- c(2,1) # label M,L, as 1,2 # 3 is deleted because no
cap_ex_size
d$choice <- as.integer(as.character(d$choice))
d$leg_lack <- as.factor(d$leg_lack)
levels(d$leg_lack) <- c(2,3,1) # label no_leg_lack, A, C, as 1, 2, 3
d$leg_lack <- as.integer(as.character(d$leg_lack))

d$cutting <- as.integer(as.factor(d$cutting))

dd <- d %>% mutate(cap_ex_size=if_else(choice=="1" & cutting == "1", 51,
cap_ex_size))
dd <- dd %>% mutate(cap_ex_size=if_else(choice=="1" & cap_ex_size > 51, 5
1, cap_ex_size))
dd <- dd %>% mutate(cap_ex_size=if_else(choice=="2" & cutting == "1", 210,
 cap_ex_size))
dd <- dd %>% mutate(cap_ex_size=if_else(choice=="2" & cap_ex_size > 210,
210, cap_ex_size))

ddd <- dd %>% mutate(removed_size = if_else(choice=="1", 51 - cap_ex_size,

                                            if_else(choice=="2", 210 - ca
p_ex_size, 1000)))
ddd$removed_size <- round(ddd$removed_size)

data <- list(N=nrow(ddd),
             K=max(ddd$id),
             Choice=ddd$choice,
             C_width=d$shell_width,
             C_width_new=seq(min(d$shell_width), max(d$shell_width), leng
th.out=50),
```

```
            ID=as.integer(ddd$id),
            Removed=ddd$removed_size)
```

cutting, model 2_1

Stan code

```
functions {
  real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }

  int myZIP_rng(real q, real lambda) {
    int choice;
    choice = bernoulli_rng(q);
    if (choice==0) {
      return( 0 );
    } else {
      return( poisson_log_rng(lambda) );
    }
  }

  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
      }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Removed, real a1, real a3, real b1, real b3, real C_w, real
```

```
Choice,
         real ak, real bk){
   real q;
   real lambda;
   q = inv_logit(ak + a1*C_w + a3*Choice);
   lambda = bk + b1*C_w + b3*Choice;

   if (Removed == 0) {
     return log_sum_exp(
       bernoulli_lpmf(0 | q),
       bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
     );
   } else {
     return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
   }
 }

 real log_lik_Simpson_ak(int Removed, real a1, real a3, real b1, real b3,
real C_w, real Choice,
                         real a_lower, real a_upper,
                         real bk,
                         int M){
   vector[M+1] lp;
   real h;

   h = (a_upper - a_lower)/M;
   lp[1] = f2(Removed, a1, a3, b1, b3, C_w, Choice, a_lower, bk);
   for (m in 1:(M/2))
     lp[2*m] = log(4) + f2(Removed, a1, a3, b1, b3, C_w, Choice, a_lower
+ h*(2*m-1), bk);
   for (m in 1:(M/2-1))
     lp[2*m+1] = log(2) + f2(Removed, a1, a3, b1, b3, C_w, Choice, a_low
er + h*2*m, bk);
   lp[M+1] = f2(Removed, a1, a3, b1, b3, C_w, Choice, a_upper, bk);
   return(log(h/3) + log_sum_exp(lp));
   }

 real log_lik_Simpson_rest(int Removed, real a1, real a3, real b1, real
b3, real C_w, real Choice,
                           real a_lower, real a_upper,
                           real b_lower, real b_upper,
                           int M){
   vector[M+1] lp;
   real h;

   h = (b_upper - b_lower)/M;
```

```
    lp[1] = log_lik_Simpson_ak(Removed, a1, a3, b1, b3, C_w, Choice,
                               a_lower, a_upper, b_lower, M);
    for (m in 1:(M/2))
      lp[2*m]   = log(4) + log_lik_Simpson_ak(Removed, a1, a3, b1, b3, C_
w, Choice,
                               a_lower, a_upper, b_lower + h*(2*m-1), M);
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + log_lik_Simpson_ak(Removed, a1, a3, b1, b3, C_
w, Choice,
                               a_lower, a_upper, b_lower + h*2*m, M);
    lp[M+1] = log_lik_Simpson_ak(Removed, a1, a3, b1, b3, C_w, Choice,
                               a_lower, a_upper, b_upper, M);
    return(log(h/3) + log_sum_exp(lp));
  }

}

data {
    int N;
    int K; // number of individuals
    int<lower=0> Removed[N];
    real Choice[N];
    real C_width[N];
    int<lower=1, upper=K> ID[N];
    real C_width_new[50];
}

parameters {
  real a1;
  real a2_0;
  real a2w[K];
  real<lower=0> a2_s;
  real a3;

  real b1;
  real b2_0;
  real b2w[K];
  real<lower=0> b2_s;
  real b3;
}

transformed parameters {
  real q[N];
  real lambda[N];
  real a2[K];
  real b2[K];
```

```
  for (k in 1:K){
    a2[k] = a2_0 + a2w[k]*a2_s;
    b2[k] = b2_0 + b2w[k]*b2_s;
  }

  for (n in 1:N){
    q[n] = inv_logit(a2[ID[n]] + a1*C_width[n] + a3*Choice[n]);
    lambda[n] = b2[ID[n]] + b1*C_width[n] + b3*Choice[n];
  }
}

model {
  a2_s ~ student_t(4, 0, 50);
  b2_s ~ student_t(4, 0, 10);
  for (k in 1:K){
    a2w[k] ~ normal(0, 1);
    b2w[k] ~ normal(0, 1);
  }
  for (n in 1:N){
    Removed[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  real log_lik[N];
  real log_lik_a;
  real log_lik_b;
  int pRemoved[50];
  real pp[50];
  real plambda[50];

  // here we have two local parameters to be marginalized out
  log_lik_a = log_lik_Simpson(a2_0, a2_s, a2_0 - 5*a2_s, a2_0 + 5*a2_s, 1
00);
  log_lik_b = log_lik_Simpson(b2_0, b2_s, b2_0 - 5*b2_s, b2_0 + 5*b2_s, 1
00);


  for (n in 1:N){
    log_lik[n] = log_lik_a + log_lik_b + log_lik_Simpson_rest(Removed[n],
 a1, a3, b1, b3, C_width[n], Choice[n],
                          a2_0 - 5*a2_s, a2_0 + 5*a2_s,
                          b2_0 - 5*b2_s, b2_0 + 5*b2_s,
                          100);
```

```
  }

  for (n in 1:50){
    pp[n] = inv_logit(a2_0 + a1*C_width_new[n] + a3*2); // fixed to choic
e L
    plambda[n] =  b2_0 + b1*C_width_new[n] + b3*2; // fixed to choice L
    pRemoved[n] = myZIP_rng(pp[n], plambda[n]);
  }

}
```

sampling

```
# fit <- stan(file='choice_and_cutting2.stan', ###
#             data=data, seed=2134,
#             chains=4, iter=6000, warmup=2000, thin=1
#             )
# save(fit, file="choice_and_cutting2.rdata")
load(file="choice_and_cutting2.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w21
w21
```

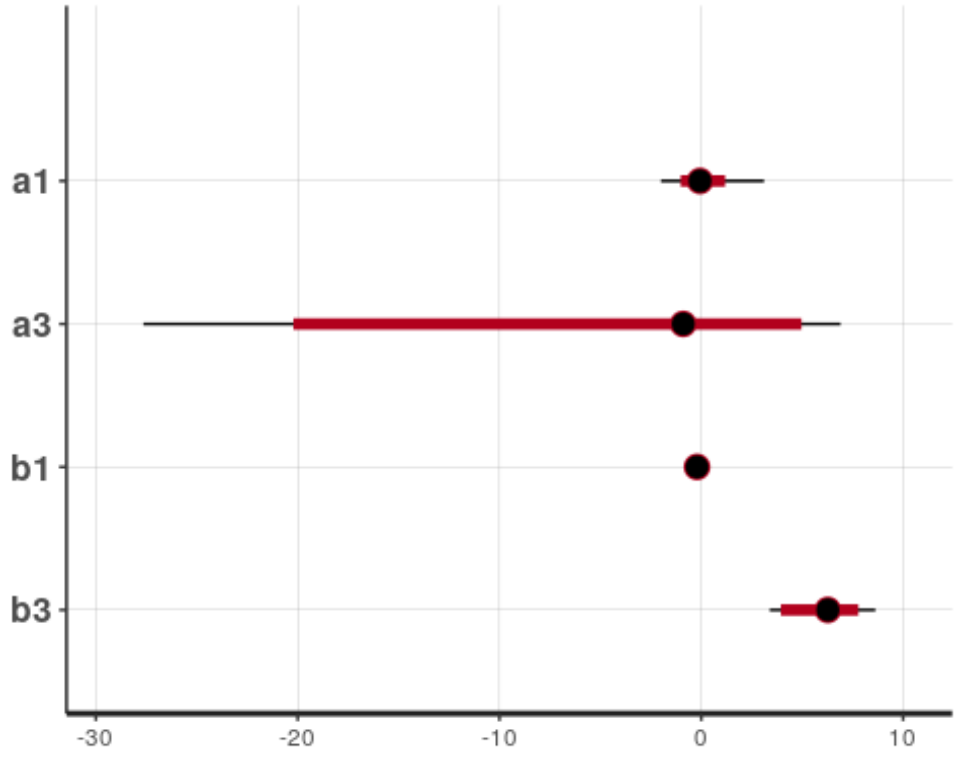
```
## [1] -1.913706
```

plot posterior distribution of the parameters

```
stan_plot(fit, c("a1", "a3", "b1", "b3")) # b_{cut}, c_{cut}, e_{cut}, f_
{cut}
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```

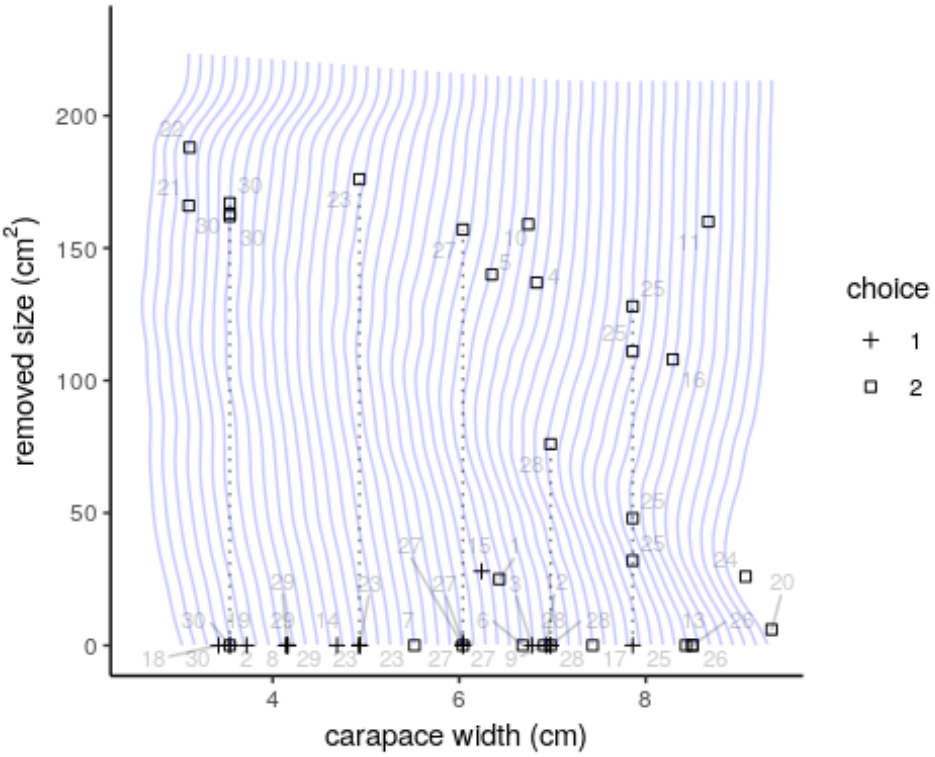


plot figure

This is the code for the figure 6.

```
plot_C_width_vs_removed <- function(d=ddd, fit){
  ### cutting the best model ###
  # choice_and_cutting2, intercept|ID x 2
  d$choice <- as.factor(d$choice)
  shell_width_new <- seq(min(d$shell_width), max(d$shell_width), length.o
ut=50)
  ms <- rstan::extract(fit)

  pred_df <- tibble()
  for (i in 1:50) {
    x <- ms$plambda[,i] %>% exp() %>% .[.<200] %>% density()
    pred_df <- rbind(pred_df, tibble(carapace_width=shell_width_new[i],
                                     removed_size=x$x,
                                     posterior_plambda=x$y/max(x$y)))
  }

  pred_df2 <- tibble()
  for (i in 1:50) {
    x <- ms$pRemoved[,i] %>% .[.<200] %>% density()
```

```
    pred_df2 <- rbind(pred_df2, tibble(carapace_width=shell_width_new[i],
                                       removed_size=x$x,
                                       posterior_pRemoved=x$y/max(x$y)))
  }

  d_seg <- d %>%
    group_by(id) %>%
    summarise( removed_size_min=min(removed_size),
               removed_size_max=max(removed_size),
               shell_width=first(shell_width)) %>%
    drop_na()


  plt <- ggplot(d) +
    geom_point(aes(shell_width, removed_size,
                   shape=choice)) +
    geom_text_repel(aes(shell_width, removed_size, label=id),
                    alpha=0.2, size=3) +
    geom_segment(
      data = d_seg,
      alpha = 0.5,
      lty=3,
      aes(x = shell_width,
          y = removed_size_min,
          xend = shell_width,
          yend = removed_size_max)) +
    geom_path(aes(x=carapace_width-posterior_plambda/2,
                  y=removed_size,
                  group=carapace_width), col="blue", alpha=0.2, data=pred
_df) +
    # geom_path(aes(x=carapace_width-posterior_pRemoved/2,
    #               y=removed_size,
    #               group=carapace_width), alpha=0.2, col="blue", data=pr
ed_df2) +
    scale_shape_manual(values=c(3, 0)) +
    # scale_fill_gradient(low="white", high="blue") +
    ylim(c(0, 230)) +
    xlab("carapace width (cm)") +
    ylab(expression(paste("removed size (", cm ^ 2, ")"))) +
    theme_classic()
  return(plt)
}

plot_C_width_vs_removed(ddd, fit)

## Warning: Removed 1683 rows containing missing values (geom_path).
```



cutting, model 2_2

Stan code

```
functions {
  real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }

  int myZIP_rng(real q, real lambda) {
    int choice;
    choice = bernoulli_rng(q);
    if (choice==0) {
      return( 0 );
    } else {
      return( poisson_log_rng(lambda) );
    }
```

```
  }

  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
  }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Removed, real a1, real a2, real a3, real b2, real b3,
          real C_width, int Choice,
          real bk){
    real q;
    real lambda;
    q = inv_logit(a1 + a2*C_width + a3*Choice);
    lambda = bk + b2*C_width + b3*Choice;
    return(ZIP_lpmf(Removed | q, lambda));
  }

  real log_lik_Simpson_rest(int Removed, real a1, real a2, real a3, real
b2, real b3,
                            real C_width, int Choice,
                            real b1_lower, real b1_upper, int M){
    vector[M+1] lp;
    real h;
    h = (b1_upper - b1_lower)/M;
    lp[1] = f2(Removed, a1, a2, a3, b2, b3, C_width, Choice, b1_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Removed, a1, a2, a3, b2, b3, C_width, Choice,
 b1_lower + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Removed, a1, a2, a3, b2, b3, C_width, Choic
e,  b1_lower + h*2*m);
    lp[M+1] = f2(Removed, a1, a2, a3, b2, b3, C_width, Choice, b1_upper);
    return(log(h/3) + log_sum_exp(lp));
   }
```

```
}

data {
    int N;
    int K; // number of individuals
    int<lower=0> Removed[N];
    int Choice[N];
    real C_width[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
  real a1;
  real a2;
  real a3;

  real b2;
  real b3;
  real b1_0;
  real b1w[K];
  real<lower=0> b1_s;
}

transformed parameters {
  real q[N];
  real lambda[N];
  real b1[K];

  for (k in 1:K){
    b1[k] = b1_0 + b1w[k]*b1_s;
  }

  for (n in 1:N){
    q[n] = inv_logit(a1 + a2*C_width[n] + a3*Choice[n]);
    lambda[n] = b1[ID[n]] + b2*C_width[n] + b3*Choice[n];
  }
}

model {
  for (k in 1:K){
    // b1k[k] ~ normal(0, b1_s);
    b1w[k] ~ normal(0, 1);
  }
  for (n in 1:N){
    Removed[n] ~ ZIP(q[n], lambda[n]);
```

```
  }
}

generated quantities {
  real log_lik_fix;
  vector[N] log_lik;
  int pRemoved[N];

  real b1p;
  real pp[N];
  real plambda[N];

  log_lik_fix = log_lik_Simpson(b1_0, b1_s, b1_0-5*b1_s, b1_0+5*b1_s, 10
0);

  b1p = normal_rng(b1_0, b1_s);

  for (n in 1:N) {
    log_lik[n] = log_lik_fix +
                 log_lik_Simpson_rest(Removed[n], a1, a2, a3, b2, b3,
                                      C_width[n], Choice[n],
                                      b1_0-5*b1_s, b1_0+5*b1_s, 100);

    pp[n] = inv_logit(a1 + a2*C_width[n] + a3*Choice[n]);
    plambda[n] =  b1p + b2*C_width[n] + b3*Choice[n];
    pRemoved[n] = myZIP_rng(pp[n], plambda[n]);
  }
}
```

sampling

```
# fit <- stan(file='choice_and_cutting_waic.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=3000, warmup=1000, thin=1
#             )
# save(fit, file="choice_and_cutting_waic.rdata")
load("choice_and_cutting_waic.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w22
w22

## [1] 0.8766614
```

cutting, model 2_3

Stan code

```
functions {
  real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }

  int myZIP_rng(real q, real lambda) {
    int choice;
    choice = bernoulli_rng(q);
    if (choice==0) {
      return( 0 );
    } else {
      return( poisson_log_rng(lambda) );
    }
  }

  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
  }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Removed, real a1, real a3, real b3,
          int Choice,
          real bk){
    real q;
```

```
    real lambda;
    q = inv_logit(a1 +  a3*Choice);
    lambda = bk + b3*Choice;
    return(ZIP_lpmf(Removed | q, lambda));
  }

  real log_lik_Simpson_rest(int Removed, real a1, real a3, real b3,
                            int Choice,
                            real b1_lower, real b1_upper, int M){
    vector[M+1] lp;
    real h;
    h = (b1_upper - b1_lower)/M;
    lp[1] = f2(Removed, a1,  a3, b3, Choice, b1_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Removed, a1,  a3, b3, Choice, b1_lower + h*(2
*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Removed, a1, a3, b3, Choice,  b1_lower + h*
2*m);
    lp[M+1] = f2(Removed, a1, a3, b3, Choice, b1_upper);
    return(log(h/3) + log_sum_exp(lp));
   }

}

data {
    int N;
    int K; // number of individuals
    int<lower=0> Removed[N];
    int Choice[N];
    real C_width[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
  real a1;
  real a3;

  real b3;
  real b1_0;
  real b1w[K];
  real<lower=0> b1_s;
}

transformed parameters {
  real q[N];
```

```
  real<lower=0> lambda[N];
  real b1[K];

  for (k in 1:K){
    b1[k] = b1_0 + b1w[k]*b1_s;
  }

  for (n in 1:N){
    q[n] = inv_logit(a1 + a3*Choice[n]);
    lambda[n] = b1[ID[n]] + b3*Choice[n];
  }
}

model {
  for (k in 1:K){
    b1w[k] ~ normal(0, 1);
  }
  for (n in 1:N){
    Removed[n] ~ ZIP(q[n], lambda[n]);
  }
  //b1_s ~ student_t(4, 0, 20);
}

generated quantities {
  real log_lik_fix;
  vector[N] log_lik;

  int pRemoved[N];
  real b1p;
  real pp[N];
  real plambda[N];

  b1p = normal_rng(b1_0, b1_s);

  log_lik_fix = log_lik_Simpson(b1_0, b1_s, b1_0-5*b1_s, b1_0+5*b1_s, 10
0);
  for (n in 1:N) {
    log_lik[n] = log_lik_fix +
                 log_lik_Simpson_rest(Removed[n], a1, a3, b3,
                                      Choice[n],
                                      b1_0-5*b1_s, b1_0+5*b1_s, 100);

    pp[n] = inv_logit(a1 + a3*Choice[n]);
    plambda[n] =  b1p + b3*Choice[n];
    pRemoved[n] = myZIP_rng(pp[n], plambda[n]);
```

```
  }
}
```

sampling

```
# fit <- stan(file='choice_and_cutting_waic_no_Cw.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1
#             )
# save(fit, file="choice_and_cutting_waic_no_Cw.rdata")
load("choice_and_cutting_waic_no_Cw.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w23
w23

## [1] 0.9852562
```

cutting, model 2_4

Stan code

```
functions {
  real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }

  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
  }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
```

```
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Removed, real a1,
          real bk){
    real q;
    real lambda;
    q = inv_logit(a1);
    lambda = bk;
    return(ZIP_lpmf(Removed | q, lambda));
  }

  real log_lik_Simpson_rest(int Removed, real a1,
                            real b1_lower, real b1_upper, int M){
    vector[M+1] lp;
    real h;
    h = (b1_upper - b1_lower)/M;
    lp[1] = f2(Removed, a1,  b1_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Removed, a1,  b1_lower + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Removed, a1, b1_lower + h*2*m);
    lp[M+1] = f2(Removed, a1,  b1_upper);
    return(log(h/3) + log_sum_exp(lp));
   }

}

data {
    int N;
    int K; // number of individuals
    int<lower=0> Removed[N];
    int Choice[N];
    real C_width[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
  real a1;

  real b1_0;
  real b1w[K];
  real<lower=0> b1_s;
}
```

```
transformed parameters {
  real q[N];
  real lambda[N];
  real b1[K];

  for (k in 1:K){
    b1[k] = b1_0 + b1w[k]*b1_s;
  }

  for (n in 1:N){
    q[n] = inv_logit(a1);
    lambda[n] = b1[ID[n]];
  }
}

model {
  for (k in 1:K){
    b1w[k] ~ normal(0, 1);
  }
  for (n in 1:N){
    Removed[n] ~ ZIP(q[n], lambda[n]);
  }
  //b1_s ~ student_t(4, 0, 20);
}

generated quantities {
  real log_lik_fix;
  vector[N] log_lik;
  log_lik_fix = log_lik_Simpson(b1_0, b1_s, b1_0-5*b1_s, b1_0+5*b1_s, 10
0);
  for (n in 1:N) {
    log_lik[n] = log_lik_fix +
                 log_lik_Simpson_rest(Removed[n], a1,
                                      b1_0-5*b1_s, b1_0+5*b1_s, 100);
  }
}
```

sampling

```
# fit <- stan(file='choice_and_cutting_waic_no_Cw_no_Choice.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1
#             )
# save(fit, file="choice_and_cutting_waic_no_Cw_no_Choice.rdata")
load("choice_and_cutting_waic_no_Cw_no_Choice.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w24
w24
```

```
## [1] 1.241855
```

cutting, mode 2_5

Stan code

```
functions {
  real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }

  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
  }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Removed, real a1, real a2,  real b2,
          real C_width,
          real bk){
    real q;
    real lambda;
    q = inv_logit(a1 + a2*C_width);
    lambda = bk + b2*C_width;
```

```
    return(ZIP_lpmf(Removed | q, lambda));
  }

  real log_lik_Simpson_rest(int Removed, real a1, real a2, real b2,
                            real C_width,
                            real b1_lower, real b1_upper, int M){
    vector[M+1] lp;
    real h;
    h = (b1_upper - b1_lower)/M;
    lp[1] = f2(Removed, a1, a2, b2,  C_width,  b1_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Removed, a1, a2, b2,  C_width, b1_lower + h*
(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Removed, a1, a2,  b2,  C_width,   b1_lower
+ h*2*m);
    lp[M+1] = f2(Removed, a1, a2, b2,  C_width, b1_upper);
    return(log(h/3) + log_sum_exp(lp));
   }

}

data {
    int N;
    int K; // number of individuals
    int<lower=0> Removed[N];
    int Choice[N];
    real C_width[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
  real a1;
  real a2;

  real b2;
  real b1_0;
  real b1w[K];

  real<lower=0> b1_s;
}

transformed parameters {
  real q[N];
  real<lower=0> lambda[N];
  real b1[K];
```

```
  for (k in 1:K){
    b1[k] = b1_0 + b1w[k] * b1_s;
  }
  for (n in 1:N){
    q[n] = inv_logit(a1 + a2*C_width[n]);
    lambda[n] = b1[ID[n]] + b2*C_width[n];
  }
}

model {
  for (k in 1:K){
    b1w[k] ~ normal(0, 1);
  }
  for (n in 1:N){
    Removed[n] ~ ZIP(q[n], lambda[n]);
  }
  //b1_s ~ student_t(4, 0, 50);
}

generated quantities {
  real log_lik_fix;
  vector[N] log_lik;
  log_lik_fix = log_lik_Simpson(b1_0, b1_s, b1_0-5*b1_s, b1_0+5*b1_s, 10
0);
  for (n in 1:N) {
    log_lik[n] = log_lik_fix +
                 log_lik_Simpson_rest(Removed[n], a1, a2, b2,
                                      C_width[n],
                                      b1_0-5*b1_s, b1_0+5*b1_s, 100);
  }
}
```

sampling

```
# fit <- stan(file='choice_and_cutting_waic_no_Choice.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1
#             )
# save(fit, file="choice_and_cutting_waic_no_Choice.rdata")
load("choice_and_cutting_waic_no_Choice.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w25
w25
```

```
## [1] 1.371657
```

cutting, model 2_6

Stan code

```
functions {
  real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }
}

data {
    int N;
    int K; // number of individuals
    int<lower=0> Removed[N];
    real Choice[N];
    real C_width[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
  real a1;
  real a2;
  real a3;

  real b1;
  real b2;
  real b3;
}

transformed parameters {
  real q[N];
  real lambda[N];

  for (n in 1:N){
    q[n] = inv_logit(a1 + a2*C_width[n] + a3*Choice[n]);
    lambda[n] = b1 + b2*C_width[n] + b3*Choice[n];
  }
}
```

```
model {
  for (n in 1:N){
    Removed[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  vector[N] log_lik;
  for (n in 1:N) {
    log_lik[n] = ZIP_lpmf(Removed[n] | q[n], lambda[n]);
  }
}
```

sampling

```
# fit <- stan(file='choice_and_cutting_no_RE.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1
#             #control = list(max_treedepth = 16,
#             #               adapt_delta=0.99)
#             )
# save(fit, file="choice_and_cutting_no_RE.rdata")
load("choice_and_cutting_no_RE.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w26
w26

## [1] 7.400718
```

cutting, model 2_7

Stan code

```
functions {
  real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }
}
```

```
data {
    int N;
    int K; // number of individuals
    int<lower=0> Removed[N];
    real Choice[N];
    real C_width[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
  real a1;
  real a2;

  real b1;
  real b2;
}

transformed parameters {
  real q[N];
  real lambda[N];

  for (n in 1:N){
    q[n] = inv_logit(a1 + a2*C_width[n]);
    lambda[n] = b1 + b2*C_width[n];
  }
}

model {
  for (n in 1:N){
    Removed[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  vector[N] log_lik;
  for (n in 1:N) {
    log_lik[n] = ZIP_lpmf(Removed[n] | q[n], lambda[n]);
  }
}
```

sampling

```
# fit <- stan(file='choice_and_cutting_no_RE_C_width.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=3000, warmup=1000, thin=1
#             )
```

```r
# save(fit, file="choice_and_cutting_no_RE_C_width.rdata")
load("choice_and_cutting_no_RE_C_width.rdata")
```

WAIC

```r
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w27
w27
```

```
## [1] 10.04331
```

cutting, model 2_8

Stan code

```
functions {
  real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }
}

data {
    int N;
    int K; // number of individuals
    int<lower=0> Removed[N];
    real Choice[N];
    real C_width[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
  real a1;
  real b1;
}

transformed parameters {
  real q[N];
  real lambda[N];

  for (n in 1:N){
    q[n] = inv_logit(a1);
```

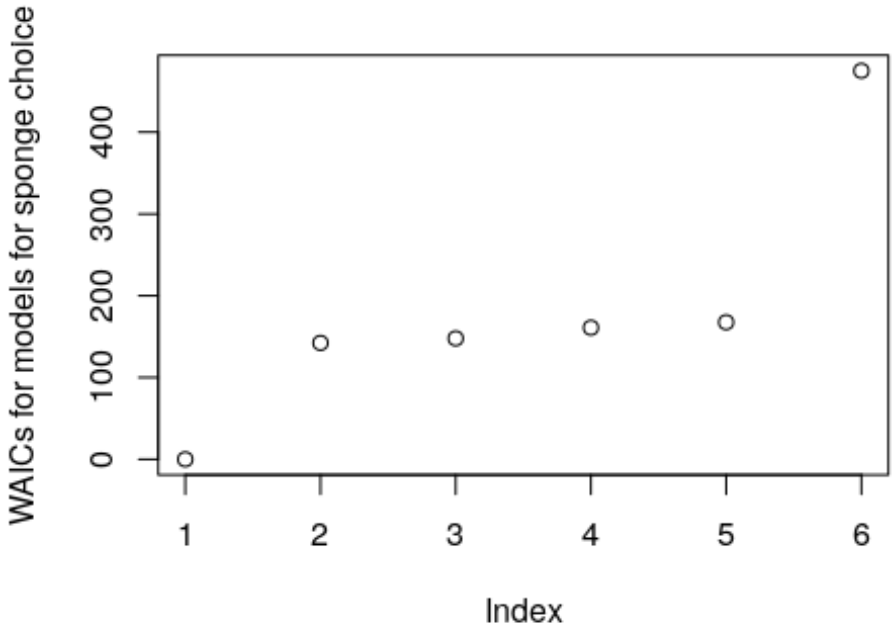
```
    lambda[n] = b1;
  }
}

model {
  for (n in 1:N){
    Removed[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  vector[N] log_lik;
  for (n in 1:N) {
    log_lik[n] = ZIP_lpmf(Removed[n] | q[n], lambda[n]);
  }
}
```

sampling

```
# fit <- stan(file='choice_and_cutting_no_RE_constant.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=3000, warmup=1000, thin=1
#             )
# save(fit, file="choice_and_cutting_no_RE_constant.rdata")
load("choice_and_cutting_no_RE_constant.rdata")

ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w28
w28

## [1] 12.53425
```

comparing WAIC values in AIC scale

```
waicv_m2 <- c(w21, w22, w23, w24, w25, w26) * data$N
waicv_m2<- waicv_m2 - min(waicv_m2)
plot(waicv_m2, ylab=c("WAICs for models for sponge choice"))
```



3. hole making

preprocessing

```
d <- read_csv('https://gist.github.com/kagaya/0d309397300b7e8294fe53c44b6fc525/raw/2cfe44334da66f22a3db59caf0d5e6967434cda1/Dromiidae.csv')
```

```
## Parsed with column specification:
## cols(
##   test_num = col_integer(),
##   id = col_integer(),
##   shell_width = col_double(),
##   gender = col_character(),
##   start_date = col_character(),
##   end_date = col_character(),
##   choice = col_character(),
##   making_process = col_character(),
##   whole_px = col_integer(),
##   hole_px = col_integer(),
##   cm_per_px = col_double(),
##   leg_lack = col_character(),
##   cutting = col_character()
## )
```

```r
d <- d %>%
  select(c('shell_width', 'hole_px', 'id', 'cm_per_px'))

d <- d[complete.cases(d), ]
# calculate hole size
d$hole_size <- d$hole_px * d$cm_per_px^2 / 1.15

# renumber animal id from 1
id <- as.factor(d$id)
levels(id) <- as.character(1:length(levels(id)))
d$id <- as.integer(id)

data <- list(N=nrow(d),
             K=max(d$id),
             Hole_size=d$hole_size,
             C_width=d$shell_width)
```

model 3_1

Stan code

```
functions {
  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
      }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(real Hole_size, real shape, real b,
             real C_w, real ak){
    return(gamma_lpdf(Hole_size | shape, shape/exp(ak + b*C_w)));
  }

  real log_lik_Simpson_rest(real Hole_size,
                            real shape,
                            real b,
```

```
                            real C_w,
                            real ak_lower, real ak_upper,
                            int M){
    vector[M+1] lp;
    real h;

    h = (ak_upper - ak_lower)/M;
    lp[1] = f2(Hole_size, shape, b, C_w, ak_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Hole_size, shape, b, C_w, ak_lower + h*(2*m-
1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Hole_size, shape, b, C_w, ak_lower + h*2*
m);
    lp[M+1] = f2(Hole_size, shape, b, C_w, ak_upper);
    return(log(h/3) + log_sum_exp(lp));
    }
}

data {
    int N;
    int K;
    real C_width[N];
    real C_width_new[50];
    real Hole_size[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
    real<lower=0> shape;
    real a0;
    real b;
    real a_id[K];
    real<lower=0> s_a;
}

transformed parameters {
    real a[K];

    for (k in 1:K){
        a[k] = a0 + a_id[k];
    }
}

model {
    for (k in 1:K){
```

```
        a_id[k] ~ normal(0, s_a);
    }
    for (n in 1:N){
        Hole_size[n] ~ gamma(shape, shape / exp(a[ID[n]] + b*C_width
[n]));
    }
}

generated quantities {
  real log_lik[N];
  real log_lik_a;
  real pred_Y[50];
  real paid;

  log_lik_a = log_lik_Simpson(a0, s_a, a0-5*s_a, a0+5*s_a, 100);
  paid = normal_rng(a0, s_a);
  for (n in 1:N) {
    log_lik[n] = log_lik_a +
    log_lik_Simpson_rest(Hole_size[n],
    shape,
    b,
    C_width[n],
    a0-5*s_a, a0+5*s_a,
    100);
  }

  for (n in 1:50) {
    pred_Y[n] = gamma_rng(shape, shape / exp(paid + b*C_width_new[n]));
  }
}
```

sampling

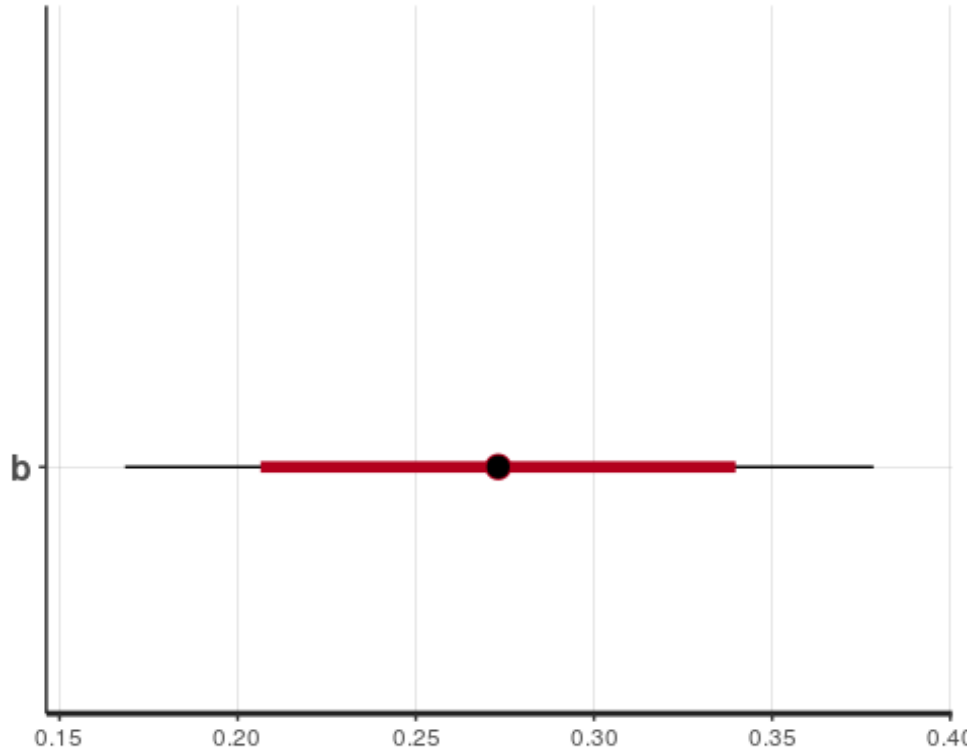
```
# fit <- stan(file='shell_hole2.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=2)
# save(fit, file="shell_hole2.rdata")
load("shell_hole2.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w31
w31

## [1] 4.342192
```

plot posterior

```
stan_plot(fit, c("b"))

## ci_level: 0.8 (80% intervals)

## outer_level: 0.95 (95% intervals)
```



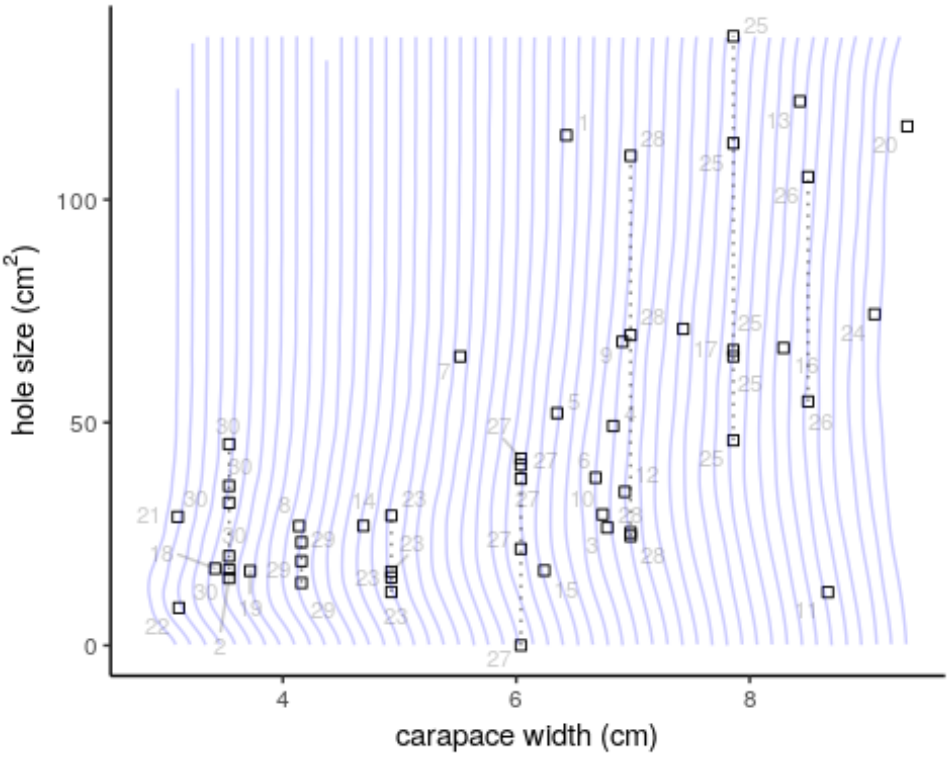
plot prediction

This is the code for the figure 7.

```
plot_C_width_vs_hole <- function(d, fit) {
  ms <- rstan::extract(fit)
  shell_width_new <- seq(min(d$shell_width), max(d$shell_width), length.o
ut=50) ###


  pred_df <- tibble()
  for (i in 1:50) {
    x <- ms$pred_Y[,i] %>% .[.<max(d$hole_size)] %>% density()
    pred_df <- rbind(pred_df, tibble(carapace_width=shell_width_new[i],
                                     hole_size=x$x,
                                     posterior_Y=x$y/max(x$y)))
  }
```

```
  d_seg <- d %>%
    group_by(id) %>%
    summarise( hole_size_min=min(hole_size),
               hole_size_max=max(hole_size),
               shell_width=first(shell_width)) %>%
    drop_na()

  plt <- ggplot(d) +
    geom_point(aes(shell_width, hole_size), pch=0) +
    geom_path(aes(x=carapace_width-posterior_Y/4,
                  y=hole_size,
                  group=carapace_width), alpha=0.2, col="blue", data=pred
_df) +

    geom_segment(
      data = d_seg,
      alpha = 0.5,
      lty=3,
      aes(x = shell_width,
          y = hole_size_min,
          xend = shell_width,
          yend = hole_size_max)) +
    geom_text_repel(aes(shell_width, hole_size, label=id), alpha=0.2, siz
e=3) +
    scale_fill_gradient(low="white", high="blue") +
    xlab('carapace width (cm)') +
    ylab(expression(paste("hole size (", cm ^ 2, ")"))) +
    ylim(c(0, max(d$hole_size))) +
    theme_classic()
  return(plt)
}

plot_C_width_vs_hole(d, fit)
```

```
## Warning: Removed 3245 rows containing missing values (geom_path).
```



model 3_2

Stan code

```
data {
    int N;
    real C_width[N];
    real Hole_size[N];
}

parameters {
    real<lower=0> shape;
    real a0;
    real b0;
}

model {
    for (n in 1:N){
        Hole_size[n] ~ gamma(shape,
        shape / exp(a0 + b0*C_width[n]));
    }
}
```

```
generated quantities {
  real log_lik[N];
  for (n in 1:N){
    log_lik[n] = gamma_lpdf(Hole_size[n] | shape, shape/exp(a0+b0*C_width
[n]));
  }
}
```

sampling

```
# fit <- stan(file='shell_hole_gamma_no_RF.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=2)
# save(fit, file="shell_hole_gamma_no_RF.rdata")
load("shell_hole_gamma_no_RF.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w32
w32

## [1] 4.535921
```

model 3_3

Stan code

```
data {
    int N;
    real shell_width[N];
    real hole_size[N];
    real gender[N];
}

parameters {
    real<lower=0> shape;
    real a0;
    real b0;
    real c0;
}

model {
    for (n in 1:N){
        hole_size[n] ~ gamma(shape,
        shape / exp(a0 + b0*shell_width[n] + c0*gender[n]));
    }
}
```

```
generated quantities {
    real<lower=0> y_base_new[N];
    real<lower=0> y_new[N];
    vector[N] log_lik;
    for (n in 1:N){
        y_base_new[n] = exp(a0 + b0 * shell_width[n] + c0*gender[n]);
        y_new[n] = gamma_rng(shape, shape / y_base_new[n]);
    }
    for (n in 1:N){
        log_lik[n] = gamma_lpdf(hole_size[n]|shape,
        shape / exp(a0 + b0 * shell_width[n] + c0*gender[n]));
    }
}
```

sampling

```
# fit <- stan(file='shell_hole_no_RF.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1)
# save(fit, file='shell_hole_no_RF.rdata') ###
load('shell_hole_no_RF.rdata') ###
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w33
w33

## [1] 4.690959
```

model 3_4

Stan code

```
functions {
  real f(real Y, real[] theta, real x){
    return(normal_lpdf(x | theta[1], theta[2]) +
    gamma_lpdf(Y | theta[3], theta[3]/exp(x)));
  }

  real log_lik_Simpson(real Y, real[] theta, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(Y, theta, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(Y, theta, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(Y, theta, a + h*2*m);
```

```
    lp[M+1] = f(Y, theta, b);
    return(log(h/3) + log_sum_exp(lp));
  }
}

data {
    int N;
    int K;
    real hole_size[N];
    int<lower=1, upper=K> id[N];
}

parameters {
    real<lower=0> shape;
    real a0;
    real a_id[K];
    real<lower=0> s_a;
}

transformed parameters {
    real a[K];
    real theta[3];

    for (k in 1:K){
        a[k] = a0 + a_id[k];
    }
    theta[1] = a0;
    theta[2] = s_a;
    theta[3] = shape;
}

model {
    for (k in 1:K){
        a_id[k] ~ normal(0, s_a);
    }
    for (n in 1:N){
        hole_size[n] ~ gamma(shape, shape / exp(a[id[n]]));
    }
}

generated quantities {
  real log_lik[N];
  for (n in 1:N){
    log_lik[n] = log_lik_Simpson(hole_size[n], theta, a0-5*s_a, a0+5*s_a,
 100);
```

```
  }
}
```

sampling

```
# fit <- stan(file='shell_hole_constant.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=2000, warmup=500, thin=1)
# save(fit, file='shell_hole_constant.rdata')
load('shell_hole_constant.rdata')
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w34
w34

## [1] 4.714697
```

model 3_5

Stan code

```
data {
    int N;
    real C_width[N];
    real Hole_size[N];
}

parameters {
    real a;
    real b;
    real<lower=0> sigma;
}

model {
    for (n in 1:N){
        Hole_size[n] ~ normal(a + b * C_width[n], sigma);
    }
}

generated quantities {
  real log_lik[N];
  for (n in 1:N){
    log_lik[n] = normal_lpdf(Hole_size[n] | a + b*C_width[n], sigma);
  }
}
```

sampling

```
# fit <- stan(file='shell_hole_linear.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1)
# save(fit, file='shell_hole_linear.rdata')
load('shell_hole_linear.rdata')
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w35
w35

## [1] 4.751062
```

model 3_6

Stan code

```
functions {
  real f(real mu, real s, real x){
      return(normal_lpdf(x | mu, s));
      }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(real Hole_size, real shape,
             real C_w, real ak, real bk){
    return(gamma_lpdf(Hole_size | shape, shape/exp(ak + bk*C_w)));
  }

  real log_lik_Simpson_ak(real Hole_size, real shape, real C_w,
                          real ak_lower, real ak_upper,
                          real bk,
                          int M){

    vector[M+1] lp;
```

```
    real h;

    h = (ak_upper - ak_lower)/M;
    lp[1] = f2(Hole_size, shape, C_w, ak_lower, bk);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Hole_size, shape, C_w, ak_lower + h*(2*m-1),
bk);
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Hole_size, shape, C_w,  ak_lower + h*2*m, b
k);
    lp[M+1] = f2(Hole_size, shape, C_w, ak_upper, bk);
    return(log(h/3) + log_sum_exp(lp));
  }

  real log_lik_Simpson_rest(real Hole_size,
                            real shape,
                            real C_w,
                            real ak_lower, real ak_upper,
                            real bk_lower, real bk_upper, int M){
    vector[M+1] lp;
    real h;

    h = (bk_upper - bk_lower)/M;
    lp[1] = log_lik_Simpson_ak(Hole_size, shape, C_w, ak_lower, ak_upper,
  bk_lower, M);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + log_lik_Simpson_ak(Hole_size, shape, C_w, ak_low
er, ak_upper, bk_lower + h*(2*m-1), M);
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + log_lik_Simpson_ak(Hole_size, shape, C_w, ak_l
ower, ak_upper, bk_lower + h*2*m, M);
    lp[M+1] = log_lik_Simpson_ak(Hole_size, shape, C_w, ak_lower, ak_uppe
r, bk_upper, M);
    return(log(h/3) + log_sum_exp(lp));
    }
}

data {
    int N;
    int K;
    real C_width[N];
    real Hole_size[N];
    int<lower=1, upper=K> ID[N];
}

parameters {
```

```
    real<lower=0> shape;
    real a0;
    real b0;
    real a_id[K];
    real b_id[K];
    real<lower=0> s_a;
    real<lower=0> s_b;
}

transformed parameters {
    real a[K];
    real b[K];

    for (k in 1:K){
        a[k] = a0 + a_id[k];
        b[k] = b0 + b_id[k];
    }
}

model {
    for (k in 1:K){
        a_id[k] ~ normal(0, s_a);
        b_id[k] ~ normal(0, s_b);
    }
    for (n in 1:N){
        Hole_size[n] ~ gamma(shape, shape / exp(a[ID[n]] + b[ID[n]]*C_wid
th[n]));
    }
}

generated quantities {
  real log_lik[N];
  real log_lik_a;
  real log_lik_b;
  log_lik_a = log_lik_Simpson(a0, s_a, a0-5*s_a, a0+5*s_a, 100);
  log_lik_b = log_lik_Simpson(b0, s_b, b0-5*s_b, b0+5*s_b, 100);
  for (n in 1:N) {
    //log_lik[n] = log_lik_a + log_lik_b + gamma_lpdf(Hole_size[n] | shap
e, shape/exp(a0* + b0*C_width[n]));
    log_lik[n] = log_lik_a + log_lik_b +
    log_lik_Simpson_rest(Hole_size[n],
    shape,
    C_width[n],
    a0-5*s_a, a0+5*s_a,
    b0-5*s_b, b0+5*s_b,
    100);
```

```
  }
}
```

sampling

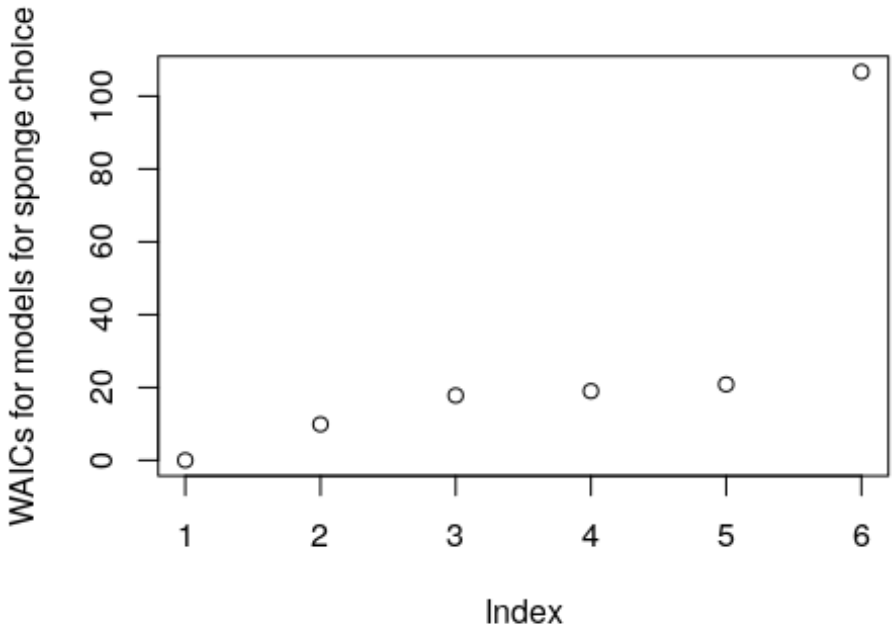
```
# fit <- stan(file='shell_hole.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=2)
# save(fit, file="shell_hole.rdata")
load("shell_hole.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w36
w36

## [1] 6.43384
```

comparing WAIC values in AIC scale

```
waicv_m3 <- c(w31, w32, w33, w34, w35, w36) * data$N
waicv_m3<- waicv_m3 - min(waicv_m3)
plot(waicv_m3, ylab=c("WAICs for models for sponge choice"))
```



4. days to carrying

preprocessing

```
d <- read_csv('https://gist.github.com/kagaya/0d309397300b7e8294fe53c44b6fc525/raw/2cfe44334da66f22a3db59caf0d5e6967434cda1/Dromiidae.csv')

## Parsed with column specification:
## cols(
##   test_num = col_integer(),
##   id = col_integer(),
##   shell_width = col_double(),
##   gender = col_character(),
##   start_date = col_character(),
##   end_date = col_character(),
##   choice = col_character(),
##   making_process = col_character(),
##   whole_px = col_integer(),
##   hole_px = col_integer(),
##   cm_per_px = col_double(),
##   leg_lack = col_character(),
##   cutting = col_character()
## )

d$start_date <- as_date(d$start_date)
d$end_date <- as_date(d$end_date)
d <- d %>% mutate(days_to_choice=end_date-start_date) %>%
  select("id", "gender", "days_to_choice", "shell_width", "choice") %>%
  filter(choice != "no_choice")

d$choice <- as.factor(d$choice)
levels(d$choice) <- c(2,1,3) # label M,L,No, as 1,2,3
d$choice <- as.integer(as.character(d$choice))

d$gender <- as.integer(as.factor(d$gender))

# renumber animal id from 1
id <- as.factor(d$id)
levels(id) <- as.character(1:length(levels(id)))
d$id <- as.integer(id)

d$days_to_choice <- as.integer(d$days_to_choice) - 1

#browser()

data <- list(N=nrow(d),
             K=length(unique(d$id)),
```

```
            Days=d$days_to_choice,
            C_width=d$shell_width,
            C_width_new=seq(min(d$shell_width), max(d$shell_width), leng
th.out=50),
            Choice=d$choice,
            ID=d$id)
```

mode 4_1

Stan code

```
functions {
    real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }

  int myZIP_rng(real q, real lambda) {
    int choice;
    choice = bernoulli_rng(q);
    if (choice==0) {
      return( 0 );
    } else {
      return( poisson_log_rng(lambda) );
    }
  }

  real f(real mu, real s, real x){
    return(normal_lpdf(x | mu, s));
  }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
```

```
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Days, real a, real b2, real C_width, real bk){
    real q;
    real lambda;
    q = inv_logit(a);
    lambda = bk + b2*C_width;

    if (Days == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Days | lambda);
    }
  }

  real log_lik_Simpson_rest(int Days, real a, real b2, real C_width, real
 bk_lower, real bk_upper, int M) {
    vector[M+1] lp;
    real h;
    h = (bk_upper - bk_lower)/M;
    lp[1] = f2(Days, a, b2, C_width, bk_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Days, a, b2, C_width, bk_lower + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Days, a, b2, C_width, bk_lower + h*2*m);
    lp[M+1] = f2(Days, a, b2, C_width, bk_upper);
    return(log(h/3) + log_sum_exp(lp));
  }
}

data {
  int N;
  int K;
  real C_width[N];
  int Days[N];
  int Choice[N];
  int ID[N];
  real C_width_new[50];
}

parameters {
  real a;
```

```
  real b1k[K];
  real b0;
  real<lower=0> s_b;
  real b2;
}

transformed parameters {
  real q[N];
  real lambda[N];
  real b1[K];

  for (k in 1:K){
    b1[k] = b0 + b1k[k];
  }

  for (n in 1:N) {
    q[n] = inv_logit(a);
    lambda[n] = b1[ID[n]] + b2*C_width[n];
  }
}

model {
  for (k in 1:K) {
    b1k[k] ~ normal(0, s_b);
  }
  for (n in 1:N) {
    Days[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  real log_lik[N];
  real log_lik_b;
  real pp[50];
  real pb;
  real plambda[50];
  real pDays[50];

  log_lik_b = log_lik_Simpson(b0, s_b, b0-5*s_b, b0+5*s_b, 100);
  for (n in 1:N){
    log_lik[n] = log_lik_b + log_lik_Simpson_rest(Days[n], a, b2, C_width
[n], b0-5*s_b, b0+5*s_b, 100);
  }
  pb = normal_rng(b0, s_b);
  for (n in 1:50){
    pp[n] = inv_logit(a);
```

```
    plambda[n] = pb + b2*C_width_new[n];
    pDays[n] = myZIP_rng(pp[n], plambda[n]);
  }
}
```

sampling

```
# fit <- stan(file='days_C_width2.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1)
# save(fit, file="days_C_width2.rdata")
load("days_C_width2.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w41
w41

## [1] 1.104198
```

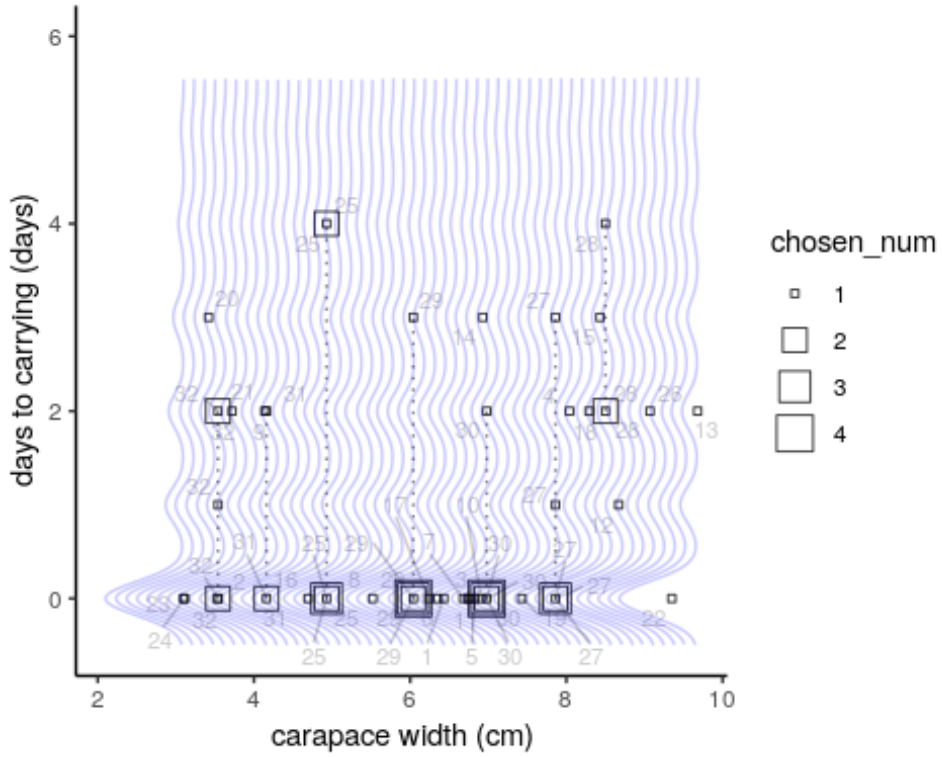
prediction plot

This is the code for the figure 8.

```
plot_days_C_width <- function(d, fit) {
  d <- add_chosen_col_days(d)
  ms <- rstan::extract(fit)
  shell_width_new <- seq(min(d$shell_width), max(d$shell_width), length.o
ut=50)
  pred_df <- tibble()
  for (i in 1:50) {
    x <- ms$pDays[,i] %>% .[.<6] %>% density()
    pred_df <- rbind(pred_df, tibble(carapace_width=shell_width_new[i],
                                     days_to_choice=x$x,
                                     posterior_days=x$y/max(x$y)))
  }

  d_seg <- d %>%
    group_by(id) %>%
    summarise( dates_min=min(days_to_choice),
               dates_max=max(days_to_choice),
               shell_width=first(shell_width)) %>%
    drop_na()

  plt <- ggplot(d) +
    # stat_density_2d(geom="raster", aes(x=x, y=y, fill=..density..), dat
a=p50, contour=F) +
    geom_point(aes(shell_width, days_to_choice,
```

```
                    size=chosen_num
                    #shape=mul
    ), alpha=0.8, pch=0) +
    geom_segment(data=d_seg, lty=3, aes(x=shell_width, y=dates_min,
                                        xend=shell_width, yend=dates_max),
 alpha=0.5) +
    geom_text_repel(aes(shell_width, days_to_choice, label=id), alpha=0.2,
 size=3) +
    geom_path(aes(x=carapace_width-posterior_days,
                  y=days_to_choice,
                  group=carapace_width), alpha=0.2, col="blue", data=pred
_df) +
    # scale_y_continuous(breaks=pretty_breaks()) +
    # scale_fill_gradient(low="white", high="gray") +
    ylim(c(-0.5, 6)) +
    #xlim(c(2.8, 10)) +
    xlab("carapace width (cm)") +
    ylab("days to carrying (days)") +
    theme_classic()
  return(plt)
}

plot_days_C_width(d, fit)
```

```
## Warning: Removed 218 rows containing missing values (geom_path).
```



model 4_2

Stan code

```
functions {
    real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }

  int myZIP_rng(real q, real lambda) {
    int choice;
    choice = bernoulli_rng(q);
    if (choice==0) {
      return( 0 );
    } else {
      return( poisson_log_rng(lambda) );
    }
```

```
  }

  real f(real mu, real s, real x){
    return(normal_lpdf(x | mu, s));
  }

  real log_lik_Simpson(real mu, real s, real a, real b, int M) {
    vector[M+1] lp;
    real h;
    h = (b-a)/M;
    lp[1] = f(mu, s, a);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f(mu, s, a + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f(mu, s,  a + h*2*m);
    lp[M+1] = f(mu, s, b);
    return(log(h/3) + log_sum_exp(lp));
  }

  real f2(int Days, real a, real bk){
    real q;
    real lambda;
    q = inv_logit(a);
    lambda = bk;

    if (Days == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Days | lambda);
    }
  }

  real log_lik_Simpson_rest(int Days, real a, real bk_lower, real bk_uppe
r, int M) {
    vector[M+1] lp;
    real h;
    h = (bk_upper - bk_lower)/M;
    lp[1] = f2(Days, a, bk_lower);
    for (m in 1:(M/2))
      lp[2*m] = log(4) + f2(Days, a, bk_lower + h*(2*m-1));
    for (m in 1:(M/2-1))
      lp[2*m+1] = log(2) + f2(Days, a, bk_lower + h*2*m);
    lp[M+1] = f2(Days, a, bk_upper);
```

```
    return(log(h/3) + log_sum_exp(lp));
  }
}

data {
  int N;
  int K;
  real C_width[N];
  int Days[N];
  int Choice[N];
  int ID[N];
}

parameters {
  real a;
  real b0;
  real rb[K];
  real<lower=0> s_b;
}

transformed parameters {
  real q[N];
  real lambda[N];
  real b[K];

  for (k in 1:K){
    b[k] = b0 + rb[k];
  }
  for (n in 1:N) {
    q[n] = inv_logit(a);
    lambda[n] = b[ID[n]];
  }
}

model {
  for (k in 1:K) {
    rb[k] ~ normal(0, s_b);
  }
  for (n in 1:N) {
    Days[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  real log_lik[N];
  real log_lik_b;
```

```
  real pp[50];
  real pb;
  real plambda[50];
  real pDays[50];

  log_lik_b = log_lik_Simpson(b0, s_b, b0-5*s_b, b0+5*s_b, 100);
  for (n in 1:N){
    log_lik[n] = log_lik_b + log_lik_Simpson_rest(Days[n], a, b0-5*s_b, b
0+5*s_b, 100);
  }


  pb = normal_rng(b0, s_b);
  for (n in 1:50){
    pp[n] = inv_logit(a);
    plambda[n] = pb;
    pDays[n] = myZIP_rng(pp[n], plambda[n]);
  }

}
```

sampling

```
# fit <- stan(file='days_constant2.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=6000, warmup=2000, thin=1)

# save(fit, file="days_constant2.rdata")
load("days_constant2.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w42
w42

## [1] 1.215824
```

model 4_3

Stan code

```
functions {
    real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
```

```
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }
}

data {
  int N;
  int K;
  real C_width[N];
  int Days[N];
  int Choice[N];
}

parameters {
  real a;
  real b;
}

transformed parameters {
  real q[N];
  real lambda[N];
  for (n in 1:N) {
    q[n] = inv_logit(a);
    lambda[n] = b;
  }
}

model {
  for (n in 1:N) {
    Days[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  real log_lik[N];
  for (n in 1:N){
    log_lik[n] = ZIP_lpmf(Days[n] | q[n], lambda[n]);
  }
}
```

sampling

```
# fit <- stan(file='days_no_RE_constant2.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1)
```

```r
# save(fit, file="days_no_RE_constant2.rdata")
load(file="days_no_RE_constant2.rdata")
```

WAIC

```r
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w43
w43

## [1] 1.281563
```

model 4_4

Stan code

```
functions {
    real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }
}

data {
  int N;
  int K;
  real C_width[N];
  int Days[N];
  int Choice[N];
}

parameters {
  real a;
  real b1;
  real b2;
}

transformed parameters {
  real q[N];
  real lambda[N];
  for (n in 1:N) {
    q[n] = inv_logit(a);
    lambda[n] = b1 + b2*Choice[n];
```

```
  }
}

model {
  for (n in 1:N) {
    Days[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  real log_lik[N];
  for (n in 1:N){
    log_lik[n] = ZIP_lpmf(Days[n] | q[n], lambda[n]);
  }
}
```

sampling

```
# fit <- stan(file='days_no_RE_no_C_width2.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1)
# save(fit, file="days_no_RE_no_C_width2.rdata")
load("days_no_RE_no_C_width2.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w44
w44

## [1] 1.284781
```

model 4_5

Stan code

```
functions {
    real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }
}
```

```
data {
  int N;
  int K;
  real C_width[N];
  int Days[N];
  int Choice[N];
}

parameters {
  real a;
  real b1;
  real b2;
}

transformed parameters {
  real q[N];
  real lambda[N];
  for (n in 1:N) {
    q[n] = inv_logit(a);
    lambda[n] = b1 + b2*C_width[n];
  }
}

model {
  for (n in 1:N) {
    Days[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  real log_lik[N];
  for (n in 1:N){
    log_lik[n] = ZIP_lpmf(Days[n] | q[n], lambda[n]);
  }
}
```

sampling

```
# fit <- stan(file='days_no_RE_C_width2.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1)
# save(fit, file="days_no_RE_C_width2.rdata")
load("days_no_RE_C_width2.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w45
w45

## [1] 1.295895
```

model 4_6

Stan code

```
functions {
    real ZIP_lpmf(int Removed, real q, real lambda) {
    if (Removed == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q),
        bernoulli_lpmf(1 | q) + poisson_log_lpmf(0 | lambda)
      );
    } else {
      return bernoulli_lpmf(1 | q) + poisson_log_lpmf(Removed | lambda);
    }
  }
}

data {
  int N;
  int K;
  real C_width[N];
  int Days[N];
  int Choice[N];
}

parameters {
  real a1;
  real a2;
  real a3;
  real b1;
  real b2;
  real b3;
}

transformed parameters {
  real q[N];
  real lambda[N];
  for (n in 1:N) {
    q[n] = inv_logit(a1 + a2*C_width[n] + a3*Choice[n]);
    lambda[n] = b1 + b2*C_width[n] + b3*Choice[n];
```

```
  }
}

model {
  for (n in 1:N) {
    Days[n] ~ ZIP(q[n], lambda[n]);
  }
}

generated quantities {
  real log_lik[N];
  for (n in 1:N){
    log_lik[n] = ZIP_lpmf(Days[n] | q[n], lambda[n]);
  }
}
```

sampling

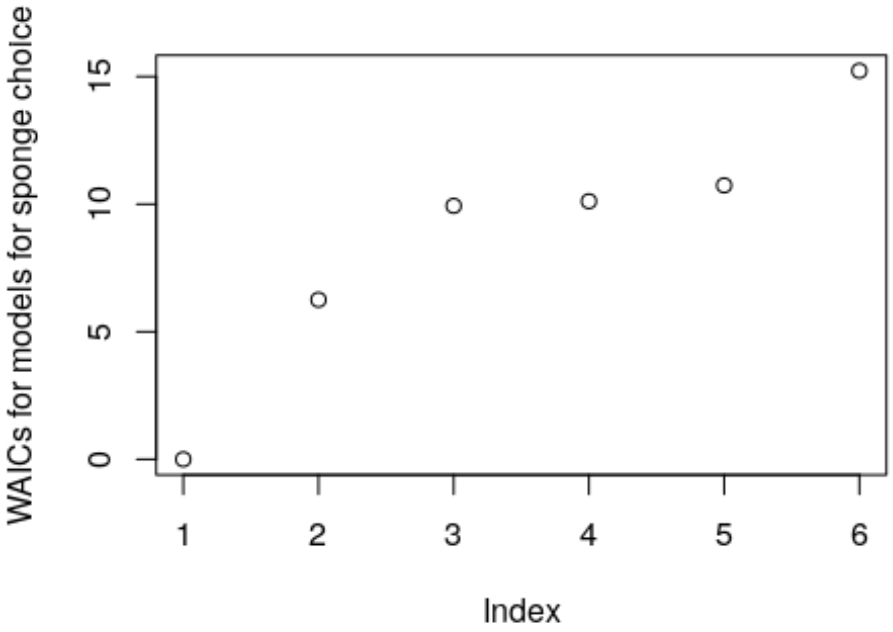
```
# fit <- stan(file='days_no_RE2.stan', ###
#             data=data, seed=1234,
#             chains=4, iter=4000, warmup=1000, thin=1)
# save(fit, file="days_no_RE2.rdata")
load("days_no_RE2.rdata")
```

WAIC

```
ms <- rstan::extract(fit)
ms$log_lik %>% waic() -> w46
w46

## [1] 1.376063
```

comparing WAIC values in AIC scale

```
waicv_m4 <- c(w41, w42, w43, w44, w45, w46) * data$N
waicv_m4<- waicv_m4 - min(waicv_m4)
plot(waicv_m4, ylab=c("WAICs for models for sponge choice"))
```



session info

```
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.1 LTS
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=ja_JP.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=ja_JP.UTF-8        LC_COLLATE=ja_JP.UTF-8
##  [5] LC_MONETARY=ja_JP.UTF-8    LC_MESSAGES=ja_JP.UTF-8
##  [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] bindrcpp_0.2.2     lubridate_1.7.4    ggforce_0.1.3
##  [4] ggrepel_0.8.0      rstan_2.18.1       StanHeaders_2.18.0
##  [7] forcats_0.3.0      stringr_1.3.1      dplyr_0.7.7
## [10] purrr_0.2.5        readr_1.1.1        tidyr_0.8.1
## [13] tibble_1.4.2       ggplot2_3.0.0      tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.19       lattice_0.20-38    prettyunits_1.0.2
##  [4] ps_1.2.0           assertthat_0.2.0   rprojroot_1.3-2
##  [7] digest_0.6.18      R6_2.3.0           cellranger_1.1.0
## [10] plyr_1.8.4         backports_1.1.2    stats4_3.5.2
## [13] evaluate_0.12      httr_1.3.1         pillar_1.3.0
## [16] rlang_0.2.2        curl_3.2           lazyeval_0.2.1
## [19] readxl_1.1.0       data.table_1.12.0  rstudioapi_0.8
## [22] callr_3.0.0        rmarkdown_1.10     labeling_0.3
## [25] loo_2.0.0          munsell_0.5.0      ggdistribute_1.0.3
## [28] broom_0.5.0        compiler_3.5.2     modelr_0.1.2
## [31] pkgconfig_2.0.2    base64enc_0.1-3    pkgbuild_1.0.2
## [34] htmltools_0.3.6    tidyselect_0.2.5   gridExtra_2.3
## [37] matrixStats_0.54.0 crayon_1.3.4       withr_2.1.2
## [40] MASS_7.3-51        grid_3.5.2         nlme_3.1-137
## [43] jsonlite_1.5       gtable_0.2.0       magrittr_1.5
## [46] units_0.6-2        scales_1.0.0       cli_1.0.1
## [49] stringi_1.2.4      farver_1.1.0       xml2_1.2.0
## [52] tools_3.5.2        glue_1.3.0         tweenr_1.0.1
## [55] hms_0.4.2          processx_3.2.0     parallel_3.5.2
## [58] yaml_2.2.0         inline_0.3.15      colorspace_1.3-2
## [61] rvest_0.3.2        knitr_1.20         bindr_0.1.1
## [64] haven_1.1.2
```