Visualize Piecewise Regression with Negative Binomial Type I Error on Real Data using `brms` Custom Family

Michael Gilchrist

Created: 2023-03-29; Compiled: Wed Apr 12 17:33:32 2023

Goal

- Visualize two piece negative binomial type 1 formulation to data

Recap

- Update to `2023-03-09_visualize.2023-02-28-fit...` for use with more recent fits in `2023-03-18_fit...` folder.
- Part of challenge is bimodal nature of some male's `x0` and `y0` fits
  - `T235` and `T236` in particular

Insights

Set up

Install libraries

```
# install packages user might not have by replacing FALSE with TRUE

## load libraries
library(stats)
library(MASS) # provides negative binomial fitting:  glm.nb
library(ggplot2)
library(ggpubr)
library(grid)
library(gridExtra)
library(ggExtra)
library(cowplot)
library(GGally)
library(RColorBrewer) ## needed to have more than 8 colors with `palette="Set2"
library(broom)
library(tidyverse)
library(viridisLite)
library(cmdstanr)
library(rstan)
```

```
options(mc.cores = (parallel::detectCores()-2))
rstan_options(auto_write = TRUE)
library(brms)
library(bayesplot)
library(tidybayes)
library(loo)
library(modelr)

## options(ggplot2.continuous.colour="viridis",
##         ggplot2.discrete.colour="viridis",
##         ggplot2.scale_fill_discrete = scale_fill_viridis_d,
##         ggplot2.scale_fill_continuous = scale_fill_viridis_c)

library(reshape2)
library(latex2exp)

ggplot2::theme_set(theme_default(base_size = 10))
## Restore theme settings
## ggplot2::theme_set(theme_default())

n_cores <- 4
```

Source family

Load `input`

```
infile <- last(dir(file.path(input_dir, "tibbles"), "fit_tbl.*"))
infile_tbl <- file.path(input_dir, "tibbles", infile)
load(infile_tbl, verbose = TRUE)
```

```
## Loading objects:
##   fit_tbl
```

```
## save filtered tibble incase it becomes corrupted locally
fit_tbl_orig <- fit_tbl %>% filter(!(is.na(fit)))
```

Add columns to `fit_tbl` for future use

```
fit_tbl <- fit_tbl_orig

names_col <- c("obs", "epred", "pred", "plots")
for(tmp in names_col) {
  ## note syntax for using value of variable to dynamically assign column name
  if(!(tmp %in% names(fit_tbl))) fit_tbl <- fit_tbl %>%  add_column("{tmp}" := list(list()))
}
```

Define and/or Expose Functions used by STAN

```
source("../../../custom-brms-families/families/nbinom_type1.R")

## Define fmax() in case expose_functions()
fmax <- function(x, x0) max(x, x0)

## Create functions defined in stan's function header
## Only need to do this once since all fits use the same user
## defined functions such as "two_piece"
## Only do this if not already defined or forcing
force_expose_functions <- FALSE
if(!exists("two_piece") | force_expose_functions) expose_functions(fit_tbl[[1, "fit"]], vectorize = TRU
```

```
## Error in UseMethod("expose_functions"): no applicable method for 'expose_functions' applied to an ob
```

Define local functions

```
which_tbl_row <- function(filter_male = FALSE, x0_flag = "individual", y0_flag = "individual", disp_flag
  which( tbl$filter_male %in% filter_male &
           tbl$x0_flag %in% x0_flag &
           tbl$y0_flag %in% y0_flag &
           tbl$disp_flag %in% disp_flag &
           tbl$disp_value %in% disp_value &
           tbl$model %in% model
           # tbl$sampling_dist %in% sampling &
         )
}

clean_var_names <- function(fit) {
  fit %>% setNames(gsub("b_", "", names(.)) %>%
             gsub("(x0|s0|y0|disp)_male(T[0-9]{3})", "\\2_\\1", .) %>%
             gsub("__", "_", .) %>%
             gsub("r_male_(x0|s0|y0|disp)\\[(T[0-9]{3}),Intercept\\]", "\\2_\\1_r", .) %>%
             gsub("\\.", " ", .))
}

## Generate data for plotting expected values
generate_epred_data <- function(fit_brms = NULL, ndraws = 1000, ncores = 4) {

    ## Create grid of x values for epred/predictions

  data_fit_brms <- fit_brms$data
  data_grid <- data_fit_brms %>%
    group_by(male) %>%
    data_grid(x = seq_range(c(20, 46), n = 51))


  ## add expected values
  data_epred <- data_grid %>%
```

```
    add_epred_draws(object = fit_brms, ndraws = ndraws, cores = ncores)

  return(data_epred)
}

## Generate data for plotting range of predicted values
generate_pred_data <- function(fit_brms = NULL, ndraws = 1000, ncores = 4) {

  data_fit_brms <- fit_brms$data

  data_grid <- data_fit_brms %>%
    group_by(male) %>%
    data_grid(x = seq_range(c(20, 45.9), n = 51)) %>%  #, .model = fit_brms) %>%
    ungroup()

  ## add simulated values
  data_pred <- data_grid %>%
    add_predicted_draws(object = fit_brms, ndraws = ndraws, ncores = ncores )

  return(data_pred)
  }


plot_epred<- function(data_epred, male_vec, desc, data_obs = NULL) {
                            ## Plot
  ## Only plot specified males
  data_epred_tmp <- data_epred %>%
    filter(male %in% male_vec)
  plot_tmp <- ggplot(data = data_epred_tmp, aes(x = x, y = .epred)) + #, color = male)) +
    ## Combine Scatter Plots and Model vs Data Plots
    scale_fill_brewer(palette = "Greys") +
    stat_lineribbon(aes(y=.epred)) #, color = "blue"))

    #scale_color_manual(values = colors_male) +
    #scale_color_brewer(palette = "Set2") +

  if(!is.null(data_obs)){

    data_obs_tmp <- data_obs %>% filter(male %in% male_vec)
    plot_tmp <- plot_tmp +
      geom_point(data = data_obs_tmp,
                 aes(x = x, y = y), color = "red")
    title <- "Expected Values & Data vs. Temp"
    y_max <- max(data_obs_tmp$y)*1.1
  } else {

    title <- "Expected Values vs. Temp"
    y_max <- NA
  }

  plot_tmp <- plot_tmp +
    ylim(0, y_max) +
    facet_wrap(vars(male)) +
```

```
    labs(title = title, subtitle = desc)

  return(plot_tmp)
}

plot_pred <- function(data_pred, male_vec, desc, data_obs = NULL) {
                            ## Plot
  ## Only plot specified males
  data_pred_tmp <- data_pred %>%
    filter(male %in% male_vec)
  plot_tmp <- ggplot(data = data_pred_tmp, aes(x = x, y = .prediction)) + #, color = male)) +
    ## Combine Scatter Plots and Model vs Data Plots
    scale_fill_brewer(palette = "Greys", direction =  -1) +
    stat_lineribbon(aes(y=.prediction)) #, fill = "blue"))

    #scale_color_manual(values = colors_male) +
    #scale_color_brewer(palette = "Set2") +

  if(!is.null(data_obs)){

    data_obs_tmp <- data_obs %>% filter(male %in% male_vec)
    plot_tmp <- plot_tmp +
      geom_point(data = data_obs_tmp,
                 aes(x = x, y = y), color = "red")
    title <- "Predicted Values & Data vs. Temp"
    y_max <- max(data_obs_tmp$y)*1.1
  } else {
    ymax <- NA
    title <- "Predicted Values vs. Temp"
  }


  plot_tmp <- plot_tmp +
    ylim(0, y_max) +
    facet_wrap(vars(male)) +
    labs(title = title, subtitle = desc)

  return(plot_tmp)
}
```

Generate Columns of Observed Data, Predicted Data and Expected Values

```
## add rows to fit_tbl if needed
## Can restore fit_tbl to original values using
##    fit_tbl <- fit_tbl_orig

force_generate_epred <- FALSE
force_generate_pred <- force_generate_epred
force_generate_obs <- force_generate_epred

curr_row_max <- nrow(fit_tbl)
##curr_row_max <- 2
```

```
for(curr_row in 1:curr_row_max) {

  fit_brms <- fit_tbl[[curr_row, "fit"]][[1]]

  if(length(fit_tbl[[curr_row, "epred"]][[1]]) ==0 | force_generate_epred) {
    epred <- generate_epred_data(fit_brms)
    fit_tbl[[curr_row, "epred"]] <-list(epred)
  }

  if(length(fit_tbl[[curr_row, "pred"]][[1]]) ==0 | force_generate_pred) {
    pred <- generate_pred_data(fit_brms)
    fit_tbl[[curr_row, "pred"]] <- list(pred)
  }

  if(length(fit_tbl[[curr_row, "obs"]][[1]]) ==0 | force_generate_obs) {
    obs <- fit_brms$data
    fit_tbl[[curr_row, "obs"]] <-list(obs)
  }

}
```

```
## Error in two_piece(x, x0, y0) : could not find function "two_piece"
##  Most likely this is because you used a Stan function in the non-linear model formula that is not de
```

```
## Error in dim(eta) <- dim_eta: dims [product 561000] do not match the length of object [1]
```

Visualize Model Fits

Set up functions, parameters, and results tibble

```
xmax <- 45.5

## Here I create a variable using a string and then access it using a string
for(name in names(fit_tbl)[1:8]) {
  # create
  var <- paste0(name, "_vec")
  assign(var, pull(fit_tbl, name) %>% unique(.))
  # access via get()
  cat(paste0(var, ": ", paste0(get(var), sep = "\n\t")))
}
```

```
## model_vec: two_piece
##  x0_flag_vec: groups_1
##   x0_flag_vec: individual
##   x0_flag_vec: uniform_1
##  y0_flag_vec: individual
##  disp_value_vec: 0.01
##  disp_flag_vec: groups_1
```

```
##   disp_flag_vec: uniform_1
##  desc_vec: nbinom_type1; two_piece; x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: 0
##   desc_vec: nbinom_type1; two_piece; x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: (
##   desc_vec: nbinom_type1; two_piece; x0: groups_1; y0: individual; disp_flag: uniform_1; disp prior:
##   desc_vec: nbinom_type1; two_piece; x0: groups_1; y0: individual; disp_flag: uniform_1; disp prior:
##   desc_vec: nbinom_type1; two_piece; x0: individual; y0: individual; disp_flag: groups_1; disp prior
##   desc_vec: nbinom_type1; two_piece; x0: individual; y0: individual; disp_flag: groups_1; disp prior
##   desc_vec: nbinom_type1; two_piece; x0: individual; y0: individual; disp_flag: uniform_1; disp prio
##   desc_vec: nbinom_type1; two_piece; x0: individual; y0: individual; disp_flag: uniform_1; disp prio
##   desc_vec: nbinom_type1; two_piece; x0: uniform_1; y0: individual; disp_flag: groups_1; disp prior:
##   desc_vec: nbinom_type1; two_piece; x0: uniform_1; y0: individual; disp_flag: groups_1; disp prior:
##   desc_vec: nbinom_type1; two_piece; x0: uniform_1; y0: individual; disp_flag: uniform_1; disp prior
##   desc_vec: nbinom_type1; two_piece; x0: uniform_1; y0: individual; disp_flag: uniform_1; disp prior
##  filter_male_vec: FALSE
##   filter_male_vec: TRUE
##  x0_group_list_vec: NA
##
```

Print fits

```
pairs_include_lp = TRUE
n_rows_print_half <- 4 ## 1/2 of rows to print of prior
print_pairs_grid <- FALSE ## if true print grid, if false print row
print_full_prior <- FALSE
off_diag_fun <- "scatter" # "scatter" or "hex" for mcmc_pairs
#use_mcmc_pairs <- FALSE ## indicate if to make scatter plots using mcmc_pairs

print_mcmc_pairs <- TRUE # individual level parameters
print_mcmc_pairs_group <- TRUE # lp and group level parameters
print_mcmc_scatter <- TRUE # individual level parameters, histograms printed on margins
print_stan_hist <- FALSE

print_get_prior <- TRUE ##
print_prior_summary <- TRUE

for(row_index in 1) {#1:nrow(fit_tbl)) {

  cat(paste("Row: ", row_index))

  row_values <- fit_tbl[row_index, ]
  desc <- row_values$desc
  filename_desc <- gsub("_", "-", desc) %>%
    gsub("; ", "_", .) %>%
    gsub(":? ", "-", .)
  fit_brms <- row_values$fit[[1]]
  data <- fit_brms[["data"]]
  fit_stan <- fit_brms$fit
  chains_n <- length(dimnames(fit_stan)$chains)

  desc_filename <- gsub("_", "-", desc) %>%
    gsub("; ", "_", .) %>%
    gsub(":? ", "-", .)
```

```
  desc_short <- desc %>% str_replace("nbinom_type1; two_piece; ","")

  ## two ways of specifying a title
  ## Second is more 'automatic'
    title_row <- ggdraw() + draw_label(desc_short, fontface='bold', size = 12)

  ## Print and plot results, regardless of which fits one uses
  print(desc)
  # print(filename_desc)

  if(print_prior_summary) {
    print("Fit Prior Information")
    if(print_full_prior) {
      print(prior_summary(fit_brms)) # %>% filter(nlpar!="y0"))
    } else {
      ## Condensed version
      tmp_prior <- prior_summary(fit_brms)
      #class(tmp_prior) <- "data.frame"
      #tmp_prior <- tmp_prior %>% select(-source)
      rows_print <- unique(c(1:n_rows_print_half, nrow(tmp_prior) - (n_rows_print_half):0))
      tmp_top <- tmp_prior[1:n_rows_print_half, ]
      tmp_bottom <- tmp_prior[rows_print[-(1:n_rows_print_half)], ]
      tmp_join <- bind_rows(tmp_top, tmp_bottom)
      tmp_join[ n_rows_print_half + 1, ] <- "..."
      ## remove brms class so we can select
      class(tmp_join) <- "data.frame"
      print(tmp_join %>% select(-source))
    }
#    print_prior_summary <- TRUE
  }

  print("Fit Information")
  print(summary(fit_brms) ) # %>% gsub("disp_value", row_values[["disp_value"]]), .) #, pars = "x0*"))


  #clean up variable names
  fit_stan_rename <-
    fit_stan %>%
    clean_var_names()

  ##

  vars_clean <- names(fit_stan_rename) %>% na.omit(.)


  male_vec <- unique(data$male) %>% as.character(.)
  ## get male specific vars (start with "T")
  vars_T <- grep("^T[0-9]{3}", vars_clean, value = TRUE)
  vars_Intercept <- grep("Intercept", vars_clean, value = TRUE)
  vars_non_T <- vars_clean[!(vars_clean %in% c(vars_T, vars_Intercept))]
  #print(vars_clean)

  ## Examine priors and exclude any constants from vars_non_T
```

```r
priors_tmp <- prior_summary(fit_brms)
constants_non_T <- ggdist::parse_dist(priors_tmp) %>% filter(grepl("constant", .dist)) %>% pull(class)
vars_non_T <- vars_non_T[!(vars_non_T %in% constants_non_T)]


## Count occurence of each male in model fit_brms
male_instance <- sapply(male_vec, function(x) {sum(str_detect(x, string=vars_clean))})

## Check to make sure plotting will work and/or is desired
if((all(male_instance > 1) | pairs_include_lp) & print_mcmc_pairs) {
  ## Use mcmc_pairs
  pairs_list <- list()
  ## update panel.boarder
  ## Note brms uses 'variable' while stanfit uses 'pars'
  for(male in male_vec) {
    # print(male)
    vars_male <- grep(male, vars_T, value = TRUE)

    ## Generate pairs() plot using mcmc_pairs()
    ## Don't use pairs.stanfit which does not return an object
    pairs_pars <- vars_male
    if(pairs_include_lp) {
      pairs_pars <- c(pairs_pars, "lp_")
    }

    if(TRUE) {
    ## Change text size
    # bayesplot_theme_update(text = element_text(size = 6)) #, family = "sans"))
    pairs_tmp <-  mcmc_pairs(
      fit_stan_rename,
      pars = pairs_pars,
      condition = pairs_condition(
        # plot all chains on lower diagonal
        # Could also split by draws
        chains <- list(top = 1:(chains_n), bottom = NULL)
      ),
      ## I believe div_color means color of divergent samples
      ##np_style = pairs_style_np(div_color = chain)
      ## set point size and transparency
      # grid_args = list(text = element_text(size = 20)),
      # off_diag_args are passed to mcmc_scatter() or mcmc_hex
      # off_diag_args = list(color = chain)
      ## Try to use hex densities
      off_diag_fun = if_else(off_diag_fun == "hex", "hex", "scatter") ## "hex" won't likely work if ti

    )

    #        bayesplot_theme_set() # same as bayesplot_theme_set(theme_default())

    # add density curves to off diagonal
    plots_tmp <- pairs_tmp$bayesplots

      ## get list of off diagonal positions
```

```
    n_tmp <- sqrt(length(plots_tmp))
    ## plot_index = lower diagonal
    plot_index <- sapply(1:(n_tmp-1), function(x) (x+1):n_tmp+(x-1)*n_tmp) %>% unlist()

    xy_pars <- combn(pairs_pars, 2) %>% t() %>% as_tibble()
    names(xy_pars) <- c("y", "x")

    xlab_vec <-  xy_pars %>% transmute(x = gsub("T[0-9]{3}_","", x)) %>% unlist()
    ylab_vec <- xy_pars %>% transmute(y = gsub("T[0-9]{3}_","", y)) %>% unlist()
    ## These don't work
    #xlab_vec <- lapply(pairs_pars, rep, times = n_tmp) %>% unlist() %>% gsub("T[0-9]{3}_","", .)
    #ylab_vec <- rev(xlab_vec)

  } else { ## use ggplot2 scatterplots, not mcmc_pairs
    plots_tmp <- list()
    ## get df of plots to make based on xy axes
    xy_pars <- combn(pairs_pars, 2) %>% t() %>% as_tibble()
    names(xy_pars) <- c("y", "x")

    xy_labs <-  xy_pars %>% mutate(x = gsub("T[0-9]{3}_","", x), y = gsub("T[0-9]{3}_","", y))
    #xy_labs <-  xy_pars %>% across(x = gsub("T[0-9]{3}_","", x), y = gsub("T[0-9]{3}_","", y))
    ## This 'works', but I can't modify the figure
    plot_tmp <- apply(xy_pars, 1, function(x) {
      mcmc_scatter(fit_stan_rename, pars =  x)}
      )

  }

  plots_row <- list()

  ## add contour lines
  for(j in 1:length(plot_index)) {
    i = plot_index[[j]]
    # pars = c(vars_male): list of 4 ggplot objects diagonal plots are 2 and 3
    plots_tmp[[i]] <- plots_tmp[[i]] +
         stat_density_2d(color = if_else(off_diag_fun == "hex", "black", "red"),
                         size = .75,
                         bins = 5) +
      labs(x = xlab_vec[[j]], y = ylab_vec[[j]])

    plots_row[[j]] <- plots_tmp[[i]]
    #ggExtra::ggMarginal(plots_tmp[[i]] +
    #                                        geom_point(col="transparent") +
    #                                        geom_hex() +
    #                                        theme(legend.position = "none"), type = "histogram")
  }


  #text_axes <- str_remove(pairs_pars, "T[0-9]+_")
  labels_subfig <- str_extract(pairs_pars, "T[0-9]+")[[1]]

#      title_subfig <- ggdraw() + draw_label(text_subfig, fontface='bold')
```

```
    if(print_pairs_grid) {
      print("Begin `print_pairs_grid`")
      pairs_male <-bayesplot_grid(plots = plots_tmp) ## Change font size
    } else {
      ## Print summaries as a row
      pairs_male <- plot_grid(
        plotlist = plots_row,
        labels = labels_subfig,
        label_size = 8,
        #hjust = -16,  # don't understand why this is so large
        label_fontface = "plain",
        nrow = 1)
    }
    pairs_list[[male]] <- pairs_male
  } # end for(male in male_vec)

  ## Combine plots across males
  ifelse(length(pairs_pars) > 3, ncol <- 1, ncol <- 2)

  use_plot_grid <- FALSE

  if(use_plot_grid) {
  p <- plot_grid(plotlist = pairs_list,
                 ncol = ncol)

  plot_pairs_list <- plot_grid(p,
                               labels = desc_short,
                               ncol = 1,
                               label_size = 10,
                               hjust = 0,
                               vjust = - 0.75) +
    # Add some space around the edges
    theme(plot.margin = unit(c(1,0.5,0.5,0.5), "cm"))
  } else {

    plot_pairs_list <- marrangeGrob(pairs_list, ncol = 1, nrow = length(pairs_pars),
                                    top = desc_short,
                                    bottom = quote(paste("page", g, "of", npages))
                                    )
  }

  print(plot_pairs_list)
  filename <- paste0("plot-pairs_", filename_desc, ".pdf")
  ggsave(filename = filename, path = file.path(output_dir, "figures"),
         plot = plot_pairs_list,
         width = 8, height = 11, units = "in",
         scale = 1,
         dpi=300,
         bg = "white")
}  ## end if(all(male... for mcmc_pairs


if(all(male_instance > 1) & print_mcmc_scatter) {
```

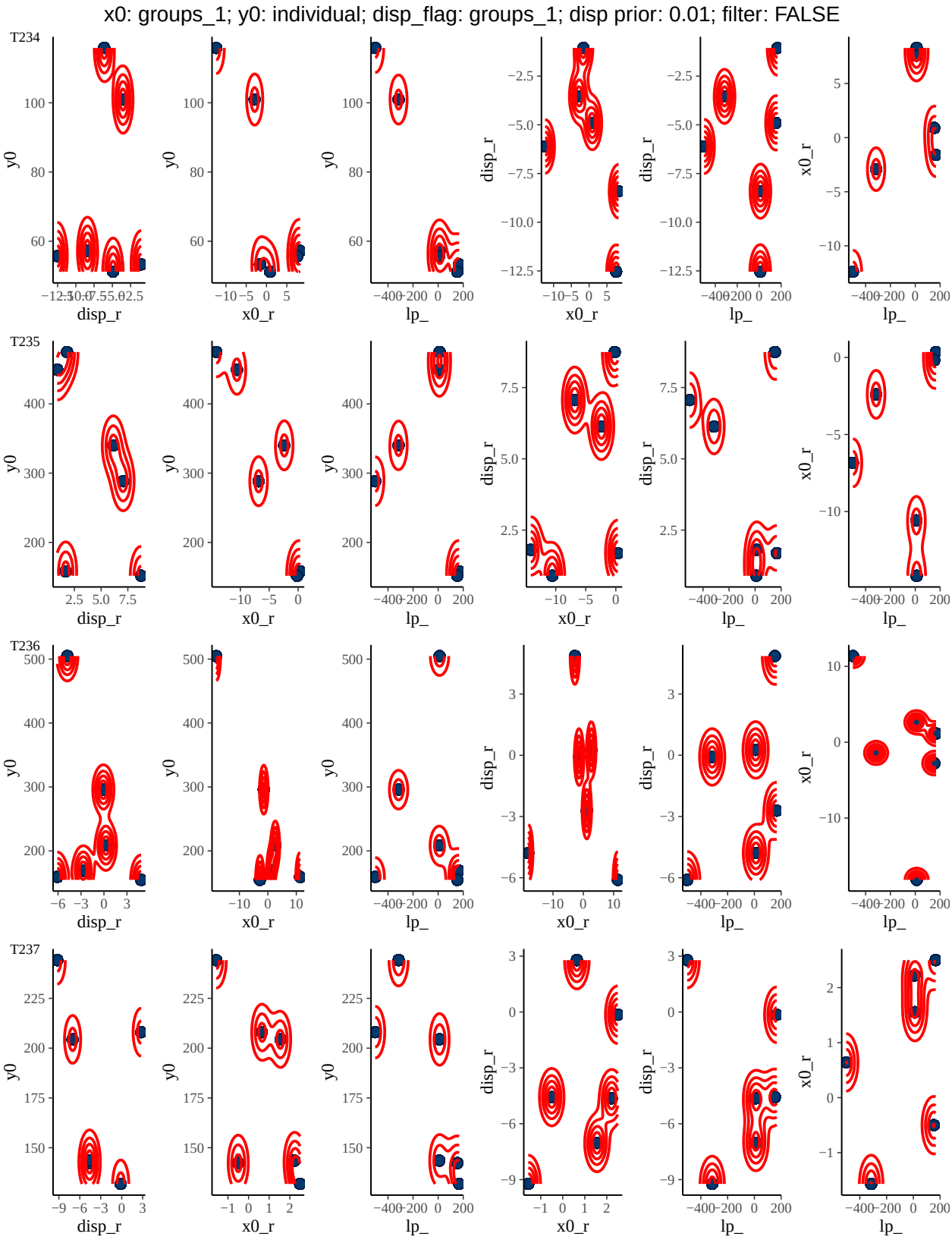
```
  scatter_list <- list()
  for(male in male_vec) {
    # print(male)
    vars_male <- grep(male, vars_T, value = TRUE)
   # if(length(vars_male) > 2) vars_pair <-
    scatter_tmp <-  mcmc_hex( #was mcmc_scatter
      fit_stan_rename,
      ## Can only use two variables
      pars = c(first(vars_male), last(vars_male)) #, vars_Intercept),
    )

    ## ggMarginal doesn't work natively with mcmc_hex, so we need to make the
    ##  points transparent and then add a hex layer
    scatter_list[[male]] <- ggExtra::ggMarginal(scatter_tmp +
                                                  geom_point(col="transparent") +
                                                  geom_hex() +
                                                  theme(legend.position = "none"), type = "histogram"

  }

  p <- plot_grid(plotlist = scatter_list,
                 ncol = 3)

  plot_scatter_list <- plot_grid(title_row, p, ncol = 1, rel_heights=c(0.1, 1))

  print(plot_scatter_list)

  filename <- paste0("plot-scatter_", filename_desc, ".pdf")
  ggsave(filename = filename, path = file.path(output_dir, "figures"),
         width = 8, height = 11, units = "in",
         scale = 1,
         dpi=300,
         bg = "white")


    } # end if(print_mcmc_scatter)



if(print_mcmc_pairs_group | length(vars_non_T > 1)) {
  pairs_list2 <- list()

  if(length(vars_Intercept) > 0) {
    pairs_list2[["Intercept"]] <- mcmc_pairs(
      fit_stan_rename, pars = c(vars_Intercept, "lp_"),
      off_diag_fun = c("hex"))
  }

  pairs_list2[["non-T"]] <- mcmc_pairs(fit_stan_rename,
                                       pars = vars_non_T,
                                       off_diag_fun = c("hex")
                                       )
```

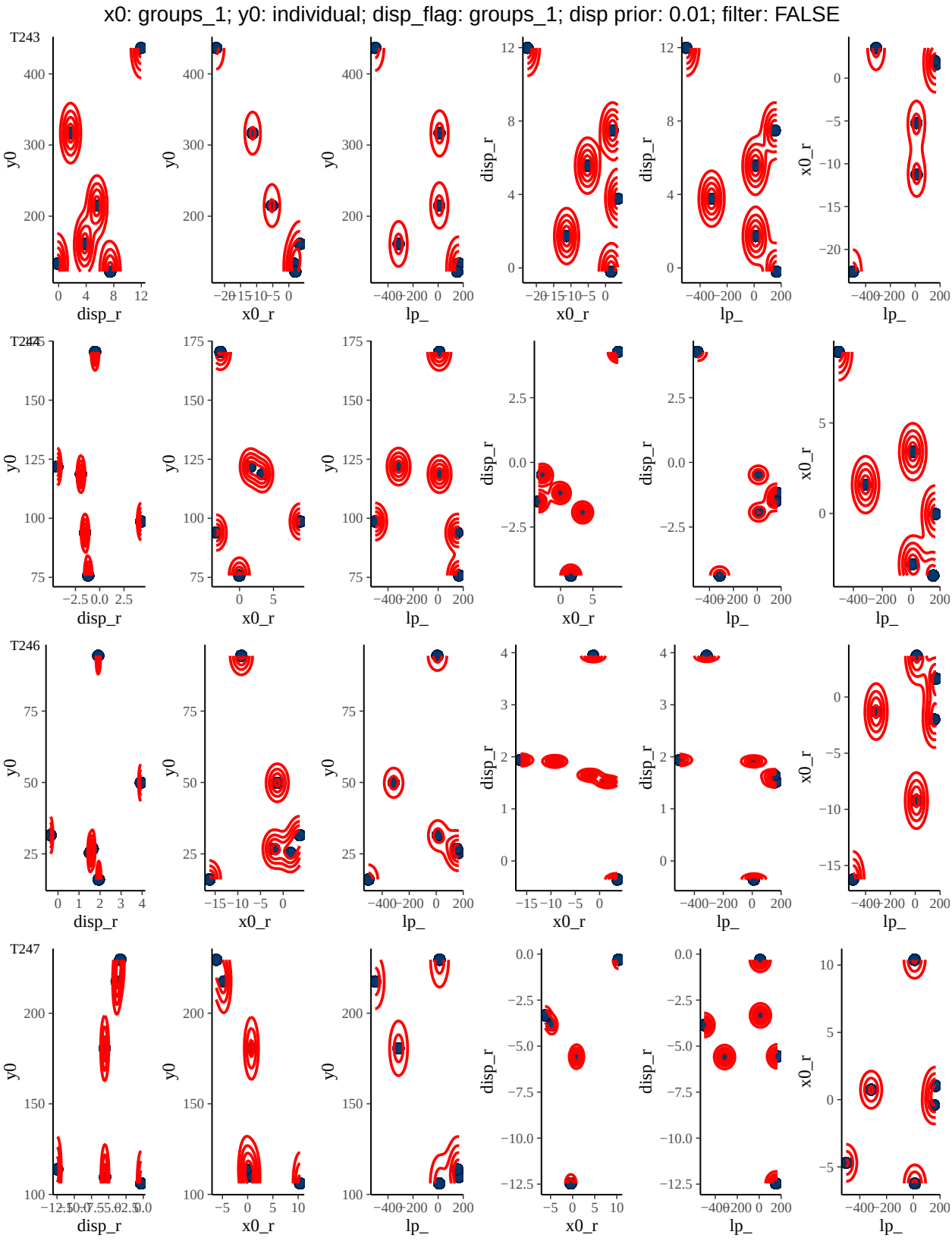
```
    p <- plot_grid(plotlist = pairs_list2,
                   ncol = 1)
    plot_pairs_list2 <- plot_grid(title_row, p,  ncol = 1, rel_heights=c(0.1, 1))

    print(plot_pairs_list2)

        filename <- paste0("plot-pairs2_", filename_desc, ".pdf")
    ggsave(filename = filename, path = file.path(output_dir, "figures"),
           width = 8, height = 11, units = "in",
           scale = 1,
           dpi=300,
           bg = "white")

  }



  vars_tmp <- vars_clean %>% str_subset("y0");

  stan_plot(fit_stan_rename, pars = vars_tmp) +
    ggtitle("Initial Motif Counts", subtitle = desc)

  vars_tmp <- vars_clean %>% str_subset("T[0-9]+_x0")
  if(length(vars_tmp) > 0) {
    fit_stan <- stan_plot(fit_stan_rename, pars = vars_tmp) +
      ggtitle("Thresholds", subtitle = desc_short)

    filename <- paste0("fit-stan_", filename_desc, ".pdf")
    ggsave(filename = filename, path = file.path(output_dir, "figures"), dpi=300)
  }

  if(print_stan_hist) {
    ncol <- 4
    hist <-  stan_hist(fit_stan_rename,
                       pars = vars_fit,
                       bins = 25,
                       ncol = ncol) +
      ggtitle(desc_short)
    filename <- paste0("histogram_", filename_desc, ".pdf")
    ggsave(filename = filename, path = file.path(output_dir, "figures"), dpi=300)
  }
}
```

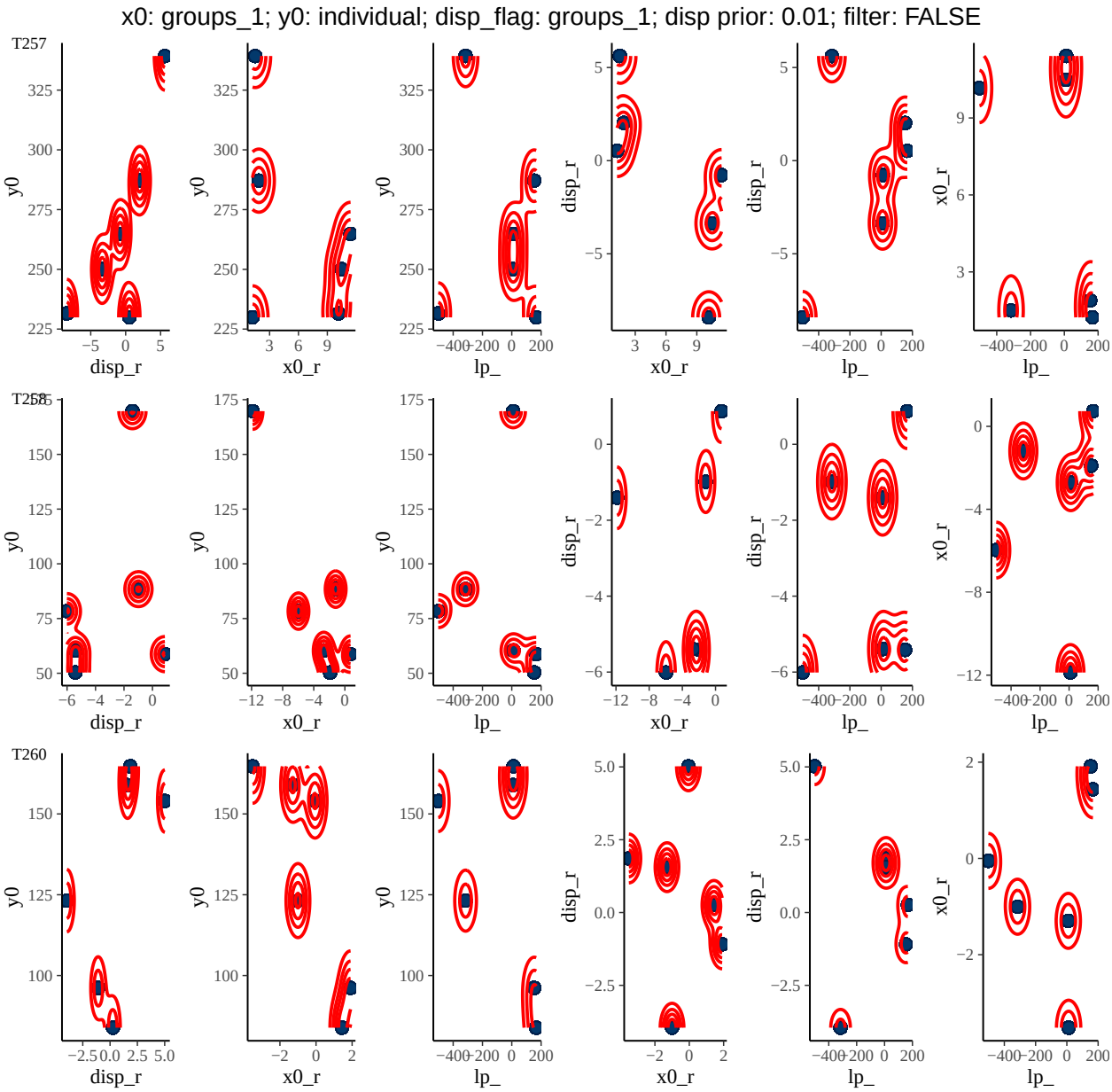
```
## Row:  1[1] "nbinom_type1; two_piece; x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: G
## [1] "Fit Prior Information"
##               prior class      coef group resp dpar nlpar  lb   ub
## 1 uniform(32, 44.5)     b                              x0  32 44.5
## 2                       b Intercept                    x0
## 3 uniform(10, 1000)     b                              y0  10 1000
## 4                       b  maleT234                    y0
## 5               ...   ...       ...   ...  ...  ...   ... ...  ...
## 6                     sd            male              x0
```

```
## 7                     sd Intercept  male             x0
## 8                     sd            male      disp
## 9                     sd Intercept  male      disp
## [1] "Fit Information"

## Warning: Parts of the model have not converged (some Rhats are > 1.05). Be
## careful when analysing the results! We recommend running more iterations and/or
## setting stronger priors.

##  Family: nbinom_type1
##   Links: mu = identity; disp = identity
## Formula: y ~ two_piece(x, x0, y0)
##          x0 ~ 0 + Intercept + (1 | male)
##          disp ~ 0 + Intercept + (1 | male)
##          y0 ~ 0 + male
##    Data: data (Number of observations: 107)
##   Draws: 6 chains, each with iter = 5000; warmup = 3000; thin = 4;
##          total post-warmup draws = 3000
##
## Group-Level Effects:
## ~male (Number of levels: 11)
##                     Estimate Est.Error l-95% CI u-95% CI               Rhat
## sd(disp_Intercept)     3.45      0.86     1.68     4.18 556362293699272.38
## sd(x0_Intercept)       4.59      2.71     1.46     7.92 556362293699272.38
##                     Bulk_ESS Tail_ESS
## sd(disp_Intercept)         6       NA
## sd(x0_Intercept)           6       NA
##
## Population-Level Effects:
##                Estimate Est.Error l-95% CI u-95% CI               Rhat Bulk_ESS
## x0_Intercept      36.47      4.58    32.19    43.46 556362293699272.38        6
## y0_maleT234       72.37     25.91    51.18   115.85 556362293699272.38        6
## y0_maleT235      310.44    126.54   151.81   474.91 556362293699272.38        6
## y0_maleT236      248.66    124.23   154.68   504.74 556362293699272.38        6
## y0_maleT237      179.04     41.98   131.82   244.18 556362293699272.38        6
## y0_maleT243      230.76    112.67   122.09   436.51 556362293699272.38        6
## y0_maleT244      113.22     29.91    75.76   170.43 556362293699272.38        6
## y0_maleT246       40.64     26.11    16.01    94.36 556362293699272.38        6
## y0_maleT247      159.54     51.87   106.06   229.49 556362293699272.38        6
## y0_maleT257      267.11     37.77   229.89   339.27 556362293699272.38        6
## y0_maleT258       84.35     40.28    50.33   169.75 556362293699272.38        6
## y0_maleT260      130.15     31.47    83.86   164.74 556362293699272.38        6
## disp_Intercept     9.44      2.45     5.56    12.54 556362293699272.38        6
##                Tail_ESS
## x0_Intercept         NA
## y0_maleT234          NA
## y0_maleT235          NA
## y0_maleT236          NA
## y0_maleT237          NA
## y0_maleT243          NA
## y0_maleT244          NA
## y0_maleT246          NA
## y0_maleT247          NA
## y0_maleT257          NA
```

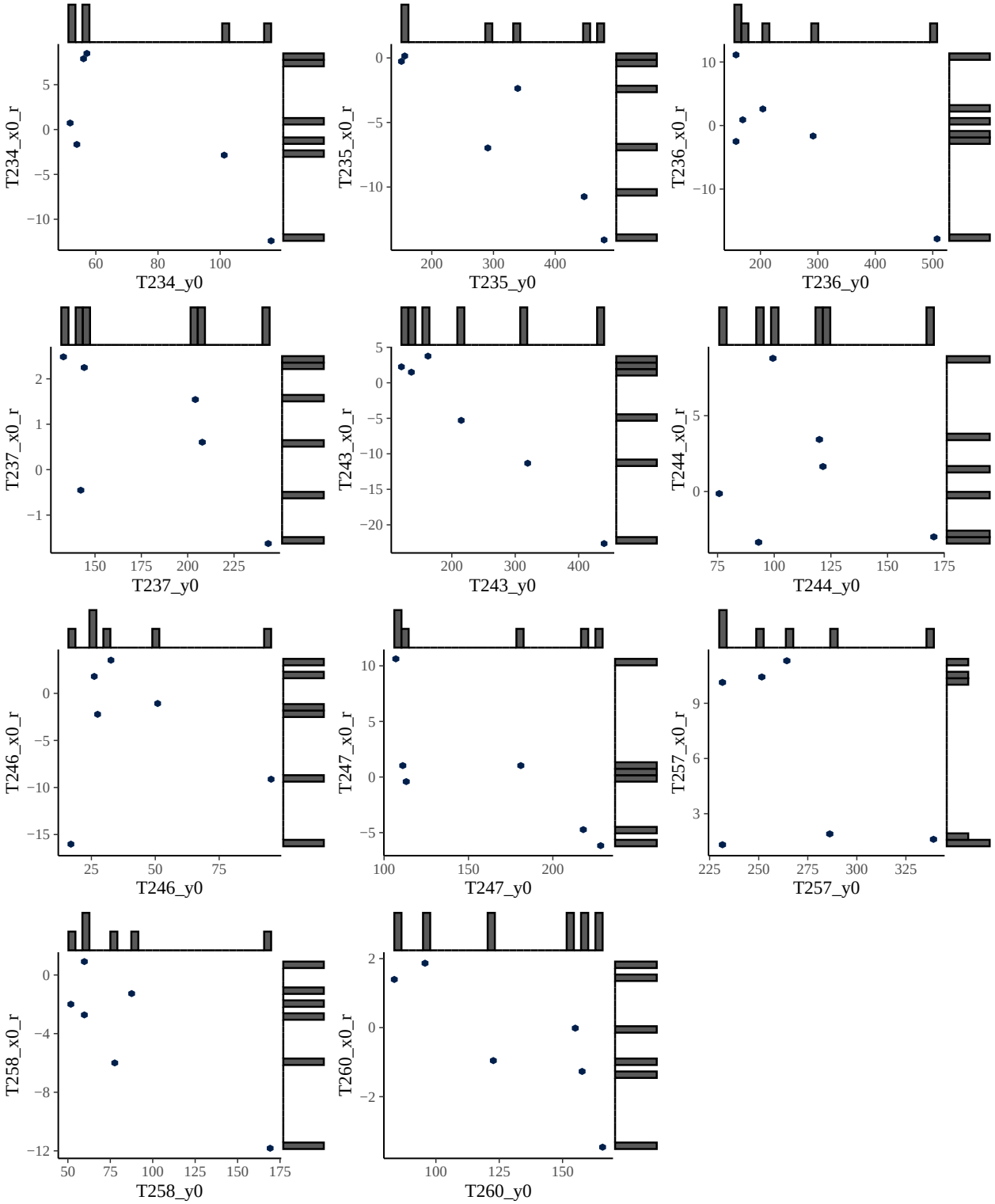
```
## y0_maleT258          NA
## y0_maleT260          NA
## disp_Intercept       NA
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```


x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: 0.01; filter: FALSE
T234
T235
T236
T237
y0
disp_r
x0_r
lp_

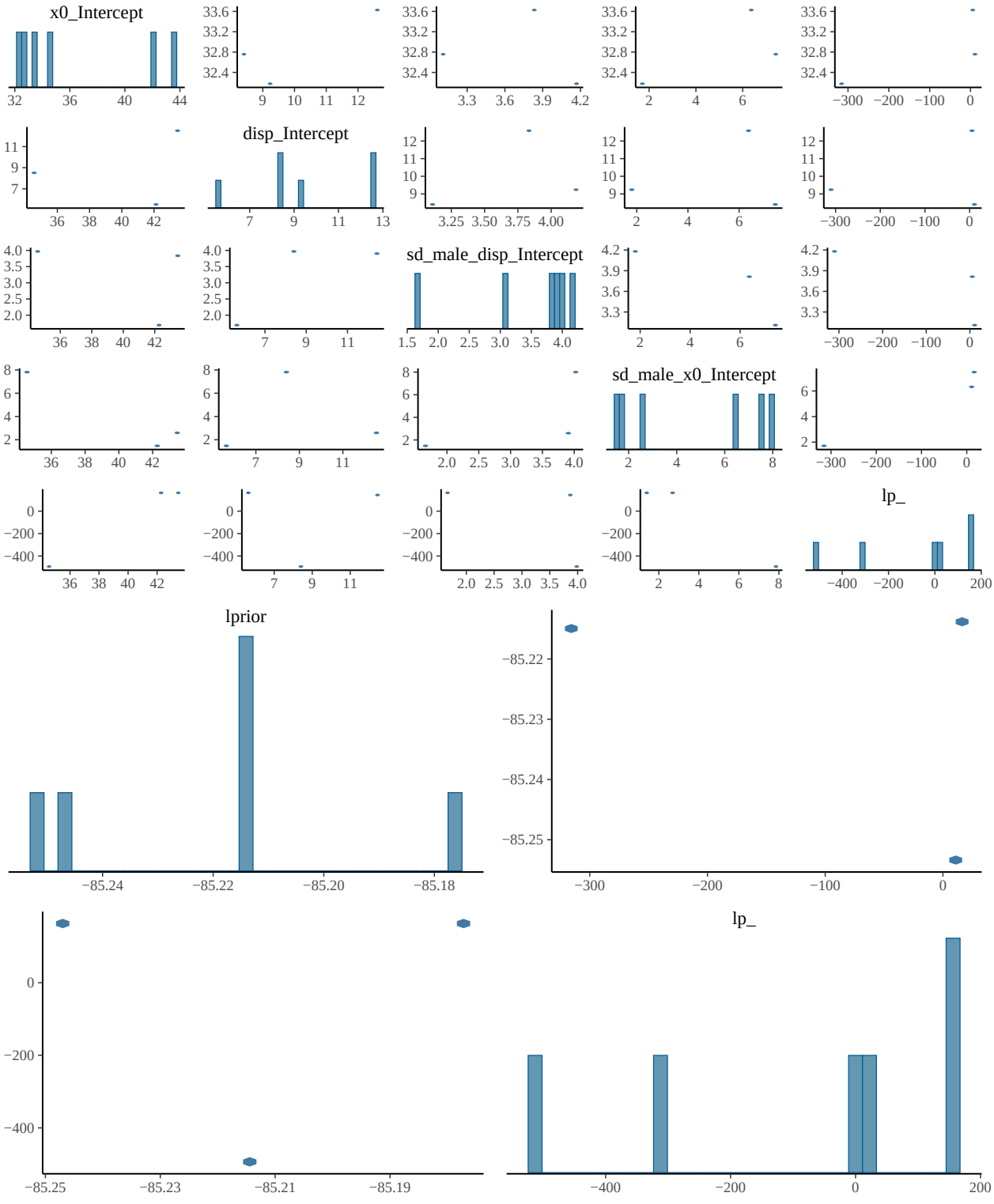

x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: 0.01; filter: FALSE
T243
T244
T246
T247
y0
disp_r
x0_r
lp_


x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: 0.01; filter: FALSE
T257
T258
T260
y0
disp_r
x0_r
lp_

x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: 0.01; filter: FALSE



x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: 0.01; filter: FALSE



Plot Data and Predictions

Set up colors – not currently needed

```
## "Set2" only has 8 colors by default.
## This code expands that ability, based on: https://www.datanovia.com/en/blog/easy-way-to-expand-color
n_colors <- 11 # length(male_vec)
colors_male <- colorRampPalette(brewer.pal(8, "Set2"))(n_colors)
```

Data suggests that disp should vary between males?

Plot Predicted and Expected Values vs. Temp

```
curr_row_max <- nrow(fit_tbl)
curr_row_max <- 1
save_plots = TRUE
force_update_plots = TRUE

for(curr_row in 1:curr_row_max) {

  fit_row <- fit_tbl[curr_row, ]

  curr_desc <- fit_row$desc

  desc <- curr_desc %>% str_replace("nbinom_type1; two_piece; ","")
  filename_desc <- gsub("_", "-", desc) %>%
    gsub("; ", "_", .) %>%
    gsub(":? ", "-", .)

  print(paste0("Row: ", curr_row, ", ", desc))

  obs <- fit_row[["obs"]][[1]]
  males <- levels(obs$male)

  print("Plotting Expectations")

  epred <- fit_row[["epred"]][[1]]
  plot_curr <- fit_tbl[[curr_row, "plots"]][[1]][["epred"]]
  if(length(plot_curr) ==0 | force_update_plots) {
    plot_tmp <- plot_epred(data_epred = epred, male_vec = males, desc = desc, data_obs = obs)
    print(plot_tmp)
    ## Why do I need the [[1]]?
    fit_tbl[[curr_row, "plots"]][[1]][["epred"]] <- plot_tmp

    if(save_plots){
      filename <- paste0("data.and.epred-vs-x_", filename_desc, ".pdf")
      ggsave(filename = filename, plot = plot_tmp, path = file.path(output_dir, "figures"),
             width = 8, height = 11, units = "in",
             scale = 1,
             dpi=300,
```

```
            bg = "white")
    }
  }

## Plot Data and Predicted Values vs. Temperature

## Reserve term "simulated" for when uisng the best fit MODEL
  ## Warning: Removed XXX rows containing missing values (`stat_slabinterval()`).
  ## This is from ggplot2 and indicates there's data outside the y range
  print("Plotting predictions")
  pred <- fit_row[["pred"]][[1]]

  plot_curr <- fit_tbl[[curr_row, "plots"]][[1]][["pred"]]
  if(length(plot_curr) == 0 | force_update_plots) {
    plot_tmp <- plot_pred(data_pred = pred, male_vec = males, desc = desc, data_obs = obs)
    print(plot_tmp)

    fit_tbl[[curr_row, "plots"]][[1]][["pred"]] <- plot_tmp
    if(save_plots) {
      filename <- paste0("data.and.pred-vs-x_", filename_desc, ".pdf")
      ggsave(filename = filename, plot = plot_tmp, path = file.path(output_dir, "figures"),
             width = 8, height = 11, units = "in",
             scale = 1,
             dpi=300,
             bg = "white")
    }
  }
}
```

```
## [1] "Row: 1, x0: groups_1; y0: individual; disp_flag: groups_1; disp prior: 0.01; filter: FALSE"
## [1] "Plotting Expectations"

## Error in UseMethod("filter"): no applicable method for 'filter' applied to an object of class "list"
```

OLD Plot Data and Simulated Values vs. Temperature

```
for(curr_row in 1:curr_row_max) {

  curr_desc <- fit_tbl[[curr_row, "desc"]]
  curr_desc_short <- curr_desc %>% str_replace("nbinom_type1; two_piece; ","")
  fit_brms <- fit_tbl[[curr_row, "fit"]][[1]]

  if(FALSE) {
    object.size(fit_brms)
    ## Unclear where function information is stored
    ## Fit object does not seem to change in size
    expose_functions(fit_brms, vectorize = TRUE, show_compiler_warnings=FALSE)
    object.size(fit_brms)
  }

  data_obs <- fit_brms$data
```

```
  ## Create grid of x values for epred/predictions

  #dataframe_tmp <- crossing(x = seq_range(c(20, 30), n = 51), draw = 1:3, male = male_vec)

  data_grid <- data_obs %>%
    group_by(male) %>%
    data_grid(x = seq_range(c(20, 45.9), n = 51)) %>%  #, .model = fit_brms) %>%
    ungroup()




  ## add simulated values
  ## Only getting 1 draw/(male temp)
  data_pred <- data_grid %>%
    add_predicted_draws(object = fit_brms)


  y_max <- max(data_pred$.prediction, na_rm = TRUE)*1.1

  plot_tmp <- ggplot(data = data_pred, aes(x = x, y = .prediction, color = male)) +
    ## Combine Scatter Plots and Model vs Data Plots
    stat_lineribbon(aes(y=.prediction), .width = c(.95), color = "#08519C") +
    scale_fill_brewer(palette = "Greys") +
    scale_color_manual(values = colors_male) +
    #scale_color_brewer(palette = "Set2") +
    geom_point(data = data_obs,
               aes(x = x, y = y), color = "red") +
    ylim(0, y_max)

  plot_data_vs_pred <- plot_tmp + facet_wrap(vars(male)) +
    labs(title = "Data vs. Simulated Values", subtitle = curr_desc_short)
  last_plot()

  filename <- paste0("data-simulated-vs-x_", filename_desc, ".pdf")
  ggsave(filename = filename, path = file.path(output_dir, "figures"),
         width = 8, height = 11, units = "in",
         scale = 1,
         dpi=300,
         bg = "white")
}
```

```
## Error in two_piece(x, x0, y0) : could not find function "two_piece"
##  Most likely this is because you used a Stan function in the non-linear model formula that is not de
```

```
## Error in dim(eta) <- dim_eta: dims [product 1683000] do not match the length of object [1]
```

Exit rendering

```
knitr::knit_exit()
```