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

Michael Gilchrist

Created: 2023-02-28; Compiled: Wed Apr 5 10:59:48 2023

Goal

- Fit two piece negative binomial type 1 formulation to data
- From 2023-02-28 version of `nbinom_type1.R`

  Negative Binomial distribution parameterized by mean (mu) and overdispersion parameter (theta). This parameterization is referred to as NEGBIN type I (Cameron and Trivedi, 1998) as cited by https://doi.org/10.1080/03610926.2018.1563164 ## `x ~ nbinom_type1(mu, theta)`, where `E(x) = mu, Var(x) = (theta + 1) mu` This should not be confused with the mu and shape parameterization of nbinom in R or the 'alternative' NB (`neg_binomial_2_...`) in stan Note using disp instead of theta because using theta gives the error > Error: Currently 'dirichlet' is the only valid prior for simplex parameters. See help(set_prior) for more details. when trying to fit the model.

Recap

- Earlier work generated poor estimates of `x0`.
- Visualization of data and model fit indicates there's very little information on x0.
- While I can generate predictions of expected values, I can't generate expected values of the data itself. I expect this is due to fact that we generate parameters which result in y = `NaN`
- TODO
  - Figure out better model definition that avoids generating NaN values. I expect this can be done by imposing a better prior on `x0|male`.
  - Allow disp to vary between males.
- On 3/22/2023 I added the missing $|dg^{-1}(disp)/ddisp| = |-mu/disp^2| = mu/disp^2$ term to llikelihood function ## Insights
- When the `disp` (dispersal or `theta`) gets unrealistically large, we get the emergence of a bimodal distribution at both ends of `x0` values.
  Even though we included this value, it is very unlikely to be 25C values. I interpret this to mean that when things are really noisy (high `theta`), one way to interpret the data is that one set of males has a very long (presumably slow) decline. It would be good to look at the correlations via `pairs()`.
- To me this is consistent with the infomal knowledge that the
- Two males have fitting issues, "T235" and "T236". This appears to be due to a bimodal posterior surface where one region has low 'x0' (< 40C), but low 'y0', ane the other has a high `x0` and low `y0`

Set up

Install libraries

\small

```
## load libraries
library(MASS) # provides negative binomial fitting:  glm.nb
library(stats)
library(tidyverse)
library(brms)
library(loo)
library(ggplot2)
#library(tidybayes)
library(ggpubr)
library(grid)
library(gridExtra)
library(ragg)
library(GGally)
library(cowplot)

ggplot2::theme_set(theme_default(base_size = 10))
#ggplot2::theme_set(theme_default(plot.background = element_rect(color = "black")))

library(broom)
library(viridisLite)
library(cmdstanr)
library(rstan)
options(mc.cores = (parallel::detectCores()-2))
rstan_options(auto_write = TRUE)

## 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(lme4)
library(latex2exp)
```

Source family

\small

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

Load Data

\small

```
sapply(file.path("input", dir("input")),
       load, verbose = TRUE, envir = .GlobalEnv)
```

```
## Loading objects:
##   data_ind
## Loading objects:
##   motif_data
##   motif_data_40C
##   motif_stats
##   motif_stats_40C
##   bird_bill_data
## Loading objects:
##   summary_stats
## Loading objects:
##   stats_ind
```

```
## $`input/data_ind.Rda`
## [1] "data_ind"
##
## $`input/data.processing_2022-12-15.Rda`
## [1] "motif_data"      "motif_data_40C"  "motif_stats"     "motif_stats_40C"
## [5] "bird_bill_data"
##
## $`input/obs_summary_stats.Rda`
## [1] "summary_stats"
##
## $`input/stats_ind.Rda`
## [1] "stats_ind"
```

```
head(stats_ind)
```

```
## # A tibble: 6 x 9
##   male  round n_obs total_round mean_round sd_round cv_round total  mean
##   <fct> <dbl> <int>       <int>      <dbl>    <dbl>    <dbl> <int> <dbl>
## 1 T234      1    13         203       40.6     32.0    0.787   601  46.2
## 2 T235      1    13         882      176.     132.     0.748  2333 179.
## 3 T236      1    13         758      152.      46.0    0.303  2095 161.
## 4 T243      1    13         438       87.6     76.4    0.872  1861 143.
## 5 T244      1    13         270       54       14.7    0.272   993  76.4
## 6 T246      1     5         253       50.6     54.6    1.08    253  50.6
```

```
names(stats_ind)
```

```
## [1] "male"        "round"       "n_obs"       "total_round" "mean_round"
## [6] "sd_round"    "cv_round"    "total"       "mean"
```

```
head(data_ind)
```

```
## # A tibble: 6 x 11
##   male  index motif_count temp_target  temp round trial_round date     counter
##   <chr> <int>       <int>       <dbl> <dbl> <dbl>       <dbl> <chr>    <chr>
## 1 T234      1           0          42  43.0     1           1 02/03/22 RAS
## 2 T234      1          30          44  44.5     1           2 02/05/22 RAS
## 3 T234      1          34          27  27.2     1           3 02/07/22 RAS
```

```
## 4 T234      1          87          40  41.1     1           4 02/09/22 RAS
## 5 T234      1          52          35  36.1     1           5 02/11/22 RAS
## 6 T234      1          32          40  39.5     2           1 04/23/22 KIM
## # i 2 more variables: y0_simple_est <dbl>, phi_ind <dbl>
```

```
names(data_ind)
```

```
##  [1] "male"          "index"         "motif_count"   "temp_target"
##  [5] "temp"          "round"         "trial_round"   "date"
##  [9] "counter"       "y0_simple_est" "phi_ind"
```

Determine reasonable priors for y0

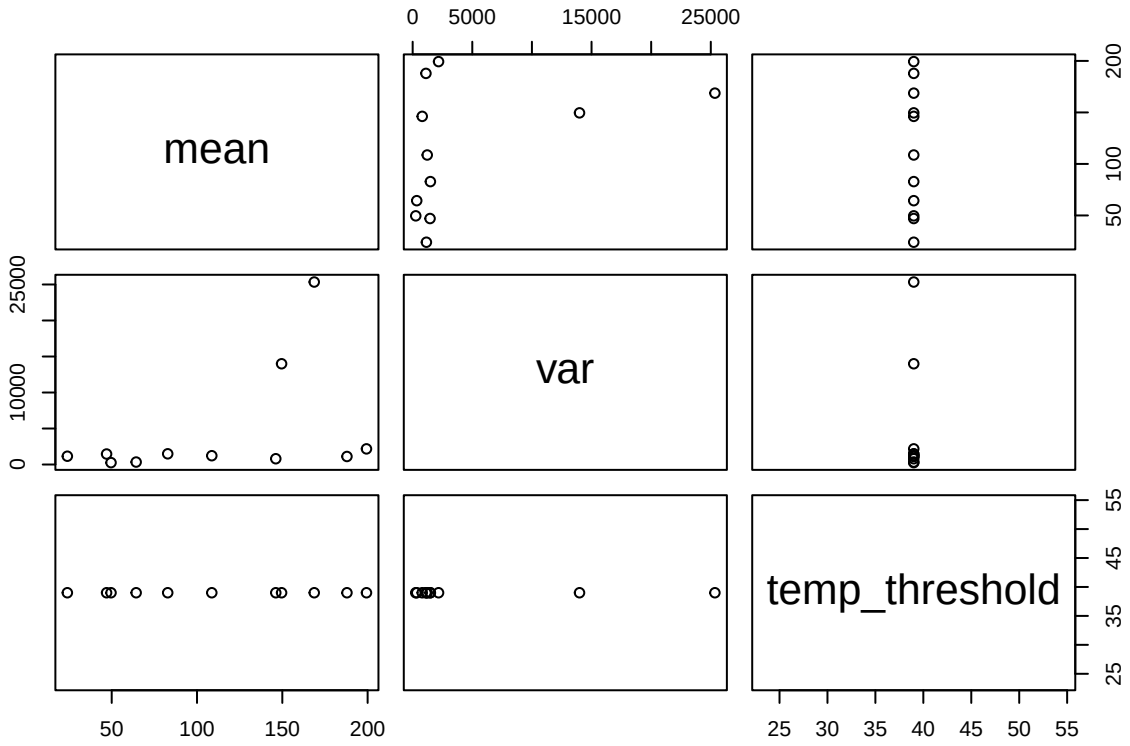
\small

```
cumulative_count_vs_temp <- list()

for(temp_threshold in 26:45) {
  tmp <- data_ind %>% group_by(male) %>% filter(temp < temp_threshold) %>% summarize(mean = mean(motif_
  mean <- mean(tmp$mean)
  sd <- sd(tmp$mean)
  cumulative_count_vs_temp[[temp_threshold]] <- tmp
  print(paste0("Temp: ", temp_threshold, ", mean: ", mean, ", sd: ", sd))
}
```

```
## [1] "Temp: 26, mean: 63, sd: 89.095454429505"
## [1] "Temp: 27, mean: 126.2, sd: 126.689383927778"
## [1] "Temp: 28, mean: 118.142857142857, sd: 110.701443187564"
## [1] "Temp: 29, mean: 116.071428571429, sd: 111.492312355093"
## [1] "Temp: 30, mean: 116.944444444444, sd: 102.314907407366"
## [1] "Temp: 31, mean: 120.055555555556, sd: 114.670572850134"
## [1] "Temp: 32, mean: 120.257575757576, sd: 104.858697610863"
## [1] "Temp: 33, mean: 120.257575757576, sd: 104.858697610863"
## [1] "Temp: 34, mean: 116.606060606061, sd: 89.5174183151982"
## [1] "Temp: 35, mean: 111.068181818182, sd: 69.0805871808609"
## [1] "Temp: 36, mean: 109.833333333333, sd: 69.568651145629"
## [1] "Temp: 37, mean: 108.231818181818, sd: 69.5451985140861"
## [1] "Temp: 38, mean: 110.348484848485, sd: 68.53139688191"
## [1] "Temp: 39, mean: 111.646176046176, sd: 61.7875535115004"
## [1] "Temp: 40, mean: 112.341666666667, sd: 64.4998385010898"
## [1] "Temp: 41, mean: 113.540928768201, sd: 56.0919858489611"
## [1] "Temp: 42, mean: 116.815308933491, sd: 60.694488706503"
## [1] "Temp: 43, mean: 115.212741046832, sd: 60.7911617164827"
## [1] "Temp: 44, mean: 114.864362152999, sd: 61.2514194145016"
## [1] "Temp: 45, mean: 113.168939393939, sd: 62.0293112972156"
```

```
plot_pairs <- pairs(cumulative_count_vs_temp[[39]] %>% select(-c(male, sd, cv)))
```




```
print(plot_pairs)
```

```
## NULL
```

```
hist <- ggplot(cumulative_count_vs_temp[[39]], aes(mean)) +
  geom_histogram(bins = 6)

hist_log <- hist + scale_x_log10()

plot_grid <- plot_grid(plotlist = list(hist, hist_log))
```

If using a normal prior, go with `mean = 125` and `sd = 125 * 4 = 500`. However, the data doesn't seem to follow any real distribution and its at the motif rather than song scale, as a result each male has its own, unknown scaling factor between motifs and songs (i.e. 1/E(# motifs/song) so a flat prior is justifiable.

Fit Models

- Code derived from `../2023-02-28_fit.real.data.using.nbinom_type1/brms_two.piece_fit.nbionm_type1.Rmd`

Set up functions, parameters, and results tibble

```
data_stan <- data_ind %>% rename(y = motif_count, x = temp) %>%
  mutate(male = factor(male))
males <- unique(data_stan$male)
nmales <- length(males)
```

```r
xmax <- 46 # maximum value for x0
xignore <- 39 # x value above which data is ignored in one_piece model

stan_two_piece_func <- paste0(" real  two_piece(real x, real x0, real y0) {
 real xmax = ", xmax, "; ## paste in value for xmax\n
 real y;

 if(x0 > xmax) {
    y = log(0);
 } else {
    y = y0 * (xmax - fmax(x0, x))/(xmax - x0);
 }
 return(y);
 } ")

stan_one_piece_func <- paste0(" real  one_piece(real y0) {
 return(y0);
 } ")

stan_asymptotic_func <- paste0(" real  asymp(real x, real phi, real y0) {
 real xmax = ", xmax, "; ## paste in value for xmax\n
 return(y0 * (1 - exp( - phi * (xmax - x))) );
 } ")

## Function to drop chains, such as those that get stuck on a suboptimal posterior peak
## Taken from:
remove_chains <- function(brm_fit, chains_to_drop) {
  # brm_fit is the output of brms::brm

  sim <- brm_fit$fit@sim  # Handy shortcut
  sim$samples <- sim$samples[-chains_to_drop]

  # Update the meta-info
  sim$chains <- sim$chains - length(chains_to_drop)
  sim$warmup2 <- sim$warmup2[-chains_to_drop]

  # Add the modified sim back to x
  brm_fit$fit@sim <- sim
  brm_fit
}
## Function to find row(s) in tbl that match criteria

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 &
         )
}
```

Set up Dataframe for fit results

\small

```
models <- c("one_piece", "two_piece", "asymptotic")[2:3] #, "one_piece") #, "asymptotic")
sampling_dists <- c("nbinom_type1") ##, "com_poisson") ## lognormal doesn't work since the counts can b
flags_x0 <- c("uniform_1",
              "groups_1", ## this doesn't work with x0_Intercept prior, suggests error in priors
#              "groups_2a",
#              "groups_2b",
              "individual")

flags_y0 <- c("uniform_1", "groups_1", "individual")[3]

values_disp <- switch(1,
                      c(0.01), # 0.125 is a good value
                      c(0.01, 0.1), #, 0.25), # used in exploring model behavior
                      list(0.1, "flat"), #, 0.1, 1) # doesn't work yet
                      c("flat"))

flags_disp <- c("uniform_1", "groups_1", "individual")


## whether to filter males with large disp values estimated using one piece model
filter_male <- c(TRUE, FALSE)

N <- length(data)

fit_tbl <- crossing(model = models,
                    #sampling_dist = sampling_dists,
                    x0_flag = flags_x0,
                    y0_flag = flags_y0,
                    disp_value = values_disp,
                    disp_flag = flags_disp,
                    desc = "NA_character",
                    filter_male = filter_male,
                    #y0_group_list = list(NA), #tbl_tmp, #list(NA),
                    x0_group_list = list(NA),
                    fit = list(NA),
                    llik = list(NA),
                    r_eff = list(NA),
                    loo = list(NA)
                    )
```

Run fits

\small

```
run_fits <- FALSE
force_load <- FALSE  #reload fit_tbl even if it already exits
save_fits <- FALSE
```

```r
if(run_fits) {
  infile_tbl <- NULL
  cur_time <- gsub(" ", "_", Sys.time()) %>% gsub(":", ".", .)
  outfile_tbl <- file.path(output_dir, "tibbles", paste0("fit_tbl_", cur_time, ".Rda"))
} else {
  infile <- last(dir(file.path(output_dir, "tibbles"), "fit_tbl.*"))
  infile_tbl <- file.path(output_dir, "tibbles", infile)
  outfile_tbl <- NULL
}



sampling = "nbinom_type1"
prior_shape_y0 = "flat"

flags_x0_used <- c("individual", "groups_1", "uniform_1") %>% rev()#
flags_y0_used <- c("individual")
values_disp_used <- values_disp
flags_disp_used <- c("individual", "groups_1", "uniform_1")[3] #  |> rev() |> first()
models_used <- c("one_piece", "two_piece", "asymptotic")[2] #c("one_piece", "two_piece") #, "two_piece"
shape_y0_prior <- "flat" # flat or normal

## These males produce bimodial posteriors and interfere with model fitting
## Ideally, we'd do a preliminary analysis without them and then include them later.

male_exclude = c("T235", "T236")

fit_index <- 0

for(male_filter in c(FALSE, TRUE)) {
    for(model in models_used) {

        print(model)
        switch(model,
               two_piece = {
                   ## Note issues in non-convergence are related to bimodality of posterior surface.
                   stan_func <- stan_two_piece_func
                   warmup <- 3000 # floor(3/4 * iter)
                   iter <- warmup + 2000
                   adapt_delta <- 0.99
                   thin <- 4
               },
               one_piece = {
                   stan_func <- stan_one_piece_func
                   warmup <- 2000
                   iter <- warmup + 2000
                   thin <- 4
                   adapt_delta <- 0.7
               },
               asymptotic = {
                   stan_func <- stan_asymptotic_func
                   warmup <- 2000
                   iter <- warmup + 2000
```

```
                thin <- 4
                adapt_delta <- 0.9
            }

            )
    for(disp_flag in flags_disp_used) {

        print(stan_func)

        for(x0_flag in flags_x0_used) {

            for(y0_flag in flags_y0_used) {

                ## define variable for labeling figures
                x0_label <- ifelse(model == "one_piece", "NA", x0_flag)


                print_get_prior <- TRUE ## reset value
                print_prior_summary <- TRUE

                for(disp_value in values_disp_used) {

                    ## used when loading fits
                    fit_index <- fit_index + 1
                    ## Set up variables for saving model and fit

                    desc_short <- paste0("x0: " , x0_label, "; y0: ", y0_flag, "; disp_flag: ", disp
                    desc <- paste0(sampling, "; ", model, "; ", desc_short)

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

                                    #stan_model_name <- sub(filename_desc, "_disp-prior-[0-9.]+_fil
                    stan_model_name <- filename_desc #sub("_disp-prior-[0-9.]+_", "_", filename_des

                    curr_row <- which_tbl_row(male_filter,
                                              x0_flag,
                                              y0_flag,
                                              disp_flag,
                                              disp_value,
                                              model,
                                              fit_tbl)

                    fit_tbl[ curr_row, ]$desc <- desc

                    print(desc)

                    if(run_fits) {

                        print("Fitting Models")
                        switch(sampling,
                               "nbinom_type1"= {
```

```
                family <- nbinom_type1(link = "identity", link_disp = "identity"
                adapt_delta <- adapt_delta #0.95 ## will decreasing value increa
                iter <- iter
                warmup <- warmup
                thin <- thin
                n_cores <- 6 ## set to 1 if getting errors from stan in order to
                n_chains <- n_cores
                stanvar_func <-
                    stanvar(scode = paste(
                                stan_func,
                                stan_nbinom_type1, sep = "\n"),
                            block = "functions")
        }
        )

    ## Refresh data in case x0_group or y0_group are all set to 1
    data <- data_stan
    if(male_filter) data <- data %>% filter(!(male %in% male_exclude))

    males_used <- unique(data$male)

    if(model == "one_piece") data <- data %>% filter(x < xignore)

    print("Set flags based on fitted model structure")
    if(x0_flag %in% c("uniform_1", "groups_1")) data <- mutate(data, x0_group =
    if(y0_flag %in% c("uniform_1", "groups_1")) data <-mutate(data, y0_group = 
    if(x0_flag %in% c("individual")) data <- mutate(data, x0_group = male)
    if(y0_flag %in% c("individual")) data <- mutate(data, y0_group = male)

    ## Note we need to put a tibble into a list because row updates, even if do
    ## just one cell, require a list format.
    ## Haven't defined "y0_group" or "x0_group" variables
    if(FALSE) {
        fit_tbl[[curr_row, "x0_group_list"]] <- list(unique(data[, c("male", "x
        fit_tbl[[curr_row, "y0_group_list"]] <- list(unique(data[, c("male", "y
    }

    print("Define parameter formulas")
    x0_form <- switch(x0_flag,
                      uniform_1 = formula(x0 ~ 0 + Intercept),
                      uniform_2 = formula(x0 ~ 0 + x0_group),
                      ## `0 + Intercept` avoids prior being defined on centered
                      groups_1 = formula(x0 ~ 0 + Intercept + (1|male)),
                      groups_2 = formula(x0 ~ 0 + Intercept + (1|male) + x0_grou
                      individual = formula(x0 ~ 0 + male) ## Do not use 1 + mal
                      )

    phi_form <- formula(deparse(x0_form) %>% gsub("x0", "phi", .))

    y0_form <- switch(y0_flag,
                      uniform_1 = formula(y0 ~ 0 + Intercept),
                      ## `0 + Intercept` avoids prior being defined on centered
                      uniform_2 = formula(y0 ~ 0 + y0_group),
```

```
                        groups_1 = formula(y0 ~ 0 + Intercept + (1| male)),
                        groups_2 = formula(y0 ~ 0 + (1| male) + y0_group),
                        individual = formula(y0 ~ 0 + male)
                        ## `-1 + ` gives me the error:
                        ## Warning in parallel::mclapply(1:chains, FUN = callFun,
                        ##  4 function calls resulted in an error
                        ## Error in FUN(X[[i]], ...) :
                        ##  trying to get slot "mode" from an object (class "try-
                        ## should I use `0 +` or `-1 +` ?
                        )

threshold_form <- switch(model,
                         two_piece = x0_form,
                         one_piece = NULL,
                         asymptotic = phi_form
                         )

disp_form <- switch(disp_flag,
                    uniform_1 = NULL, #formula(disp ~ 1),
                    groups_1 = formula(disp ~ 0 + Intercept + (1|male)),
                    individual = formula(disp ~ 0 + male)
                    )

nlform <- switch(model,
                 two_piece = bf(y ~ two_piece(x, x0, y0), nl = TRUE),
                 one_piece = bf(y ~ one_piece(y0), nl = TRUE),
                 asymptotic = bf(y ~ asymp(x, phi, y0), nl = TRUE)
                 ) +
    threshold_form +
    disp_form +
    y0_form


print("Define priors")

            # pass disp_value via stanvar argument
stanvar_prior <- stanvar(disp_value, name = "disp_value")

prior_string <- if(disp_value == "flat") {
                    "uniform(0, 20)"
                } else {
            # encode non-flat prior here, which force recompling when disp_
            #paste0("exponential(", disp_value, ")")
            # pass disp_value via stanvar argument
            # Allows disp_value to be changed w/o recompiling
                    "exponential(disp_value)"
                }

## x0 only used in two_piece model
x0_prior <- switch(x0_flag,
                   uniform_1 = NULL,
                   uniform_2 = NULL,
                   groups_1 = prior(student_t(3, 0, 66.7), lb = 0, ub = 10,
```

```
                        groups_2 = NULL,
                        individual = NULL
                        )

x0_priors <- prior(uniform(32, 44.5), lb = 32, ub = 44.5, nlpar = "x0") + x0

phi_priors <- prior(uniform(0.1, 100), lb = 0.01, ub = 17, nlpar = "phi")

y0_priors <- switch(prior_shape_y0,
                        ## Values based on calculations at top of file using `t0
                        normal = prior(normal(125, 500), nlpar = "y0", lb = 10,
                # flat prior
                # - consistent with fact we're working with motifs, not songs
                # - avoids bimodal posterior sampling issues with T235 and 236
                        flat =  prior(uniform(10, 1000), nlpar = "y0", lb = 10,

                        )

threshold_priors <- switch(model,
                               two_piece = x0_priors,
                               one_piece = NULL,
                               asymptotic = phi_priors
                               )

disp_priors <- switch(disp_flag,
                          uniform_1 = set_prior(prior_string, class = "disp", lb
                          uniform_2 = NULL,
                          groups_1 = set_prior(prior_string,
                                               class = "b", dpar = "disp", lb =
                              set_prior("uniform(0.1, 5)", class = "sd", dpar =
                          groups_2a = NULL,
                          groups_2b = NULL,
                          individual = set_prior(prior_string,
                                                 dpar = "disp", lb = 0, ub = 20

                          )

prior <- switch(model,
                    one_piece = {
                        y0_priors + disp_priors
                    },
                    threshold_priors + y0_priors + disp_priors
                    )

if(print_get_prior) {
    tmp <- get_prior(nlform,
                     data = data,
                     family = family
                     )
    print(tmp,
          max.print = 500)
    print_get_prior <- FALSE # will get reset
}
```

```
        stan_code <- file.path(output_dir,
                               "stan", "code", paste0(stan_model_name, ".stan"))
                    #make_stancode( .... save_model = stan_code)

        fit <- brm(nlform,
                   data = data,
                   ## `link` refers to the mapping of the expectation of the distri
                   ## link_shape corresponds to `phi` of `stan`'s
                   ## Negbinomial_2
                   ## Defining `phi = mu/theta` creates a quasipoisson
                   ## distribution with overdispersion parameter (1 +theta)
                   family = family, #negbinomial(link = "identity", link_shape = "i
                   prior = prior,
                   stanvar = stanvar_func + stanvar_prior, ## pass prior values her
                   iter = iter,
                   warmup = warmup,
                   thin = thin,
                   silent = ifelse(interactive(), 1, 2), # 0, 1, or 2. 1 is default
                   control = list(adapt_delta = adapt_delta,
                                  max_treedepth = 12
                                  ##model_name = desc ## Incorrect way to set this.
                                  ),
                   ## Ideally save model to avoid need to recompile
                   stan_model_args = list(
                       model_name = file.path(output_dir, "stan", "binary", stan_mo
                   ),
                    #sample_prior =  "no", ## note improper priors not sampled
                   ## Only print out sampling progress if in interactive mode
                   refresh = ifelse(interactive(),max(iter/5, 1), 0),
                   chains = n_chains,
                   cores = n_cores,
                   save_model = stan_code
                   )

        print("Prior Summary")
        print(prior_summary(fit))
        print("Fit Information")
        print(desc)
        print(fit)

                    #fit_exp <- expose_functions(fit) , vectorize = TRUE)
                    #fit_cr <- add_criterion(fit_exp, c("loo", "waic"))
        fit_tbl[[curr_row, "fit"]] <- list(fit)
        ## Print current warnings
        warnings(summary)
        ## Clear warnings()

        ## End if(run_fit)
    } else {
        print("Working with Pre-existing Fits")


        ## Try to assign from local memory.
```

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

        ## Load fit if undefined or desired
        if(is.na(list(fit)) | force_load) {
            print("Loading Models")
                    #if(fit_index ==1)
            load(file = infile_tbl)
            fit <- fit_tbl[[curr_row, "fit"]][[1]]
        }

        data <- fit$data
        males_used <- unique(data$male)

    }

    if(is.na(list(fit))) {
        warning(paste0("model fit ", desc, "does not exist.\n Skipping."))
    }else {
        ## Print and plot results, regardless of which fits one uses
        print(desc)
        if(print_prior_summary) {
            print("Fit Prior Information")
            print(prior_summary(fit)) # %>% filter(nlpar!="y0"))
            print_prior_summary <- TRUE
        }
        print("Fit Information")
        print(summary(fit)) #, pars = "x0*"))          %>% filter(nlpar!="y0"))
        fit_stan <- fit$fit
                    #clean up variable names
        fit_stan_rename <-
            fit_stan %>%
            setNames(gsub("b_", "", names(.)) %>%
                         gsub("(x0|phi|y0)_male(T[0-9]{3})", "\\2_\\1", .) %>%
                         gsub("__", "_", .) %>%
                         gsub("r_male_(x0|phi|y0)\\[(T[0-9]{3}),Intercept\\]", "\\2_\\1
                         gsub("\\.", " ", .))

        ##
        vars_fit <- names(fit_stan_rename) %>% na.omit(.) %>% sort(., decreasing =
        ncol <- 4
        hist <-  stan_hist(fit_stan_rename,
                           pars = vars_fit,
                           bins = 30,
                           ncol = ncol) +
            ggtitle(desc_short)
        print(hist)
        filename <- paste0("histogram_", filename_desc, ".pdf")
        ggsave(filename = filename, path = file.path(output_dir, "figures"),
               width = 8, height = 11, units = "in",
               scale = 0.4,
               dpi=300)

        list_plot <- list()
```

```
                            if(model == "one_piece" & disp_flag != "individual") {
                                ## each male only appears once in the one piece models
                                splitby <- 3
                                nsplits <- max(nmales %/% splitby, 1)
                                tmp_stop <- (1:nsplits) * splitby
                                tmp_start <- tmp_stop - (splitby - 1)
                                for(i in 1:nsplits) {
                                    tmp_range <- tmp_start[[i]]:tmp_stop[[i]]
                                    list_plot[[i]] <- pairs(fit,
                                                            variable = c(as.character(males[tmp_range])
                                                                         "disp"),
                                                            regex = TRUE)
                                }
                                list_plot[[nsplits+1]] <- pairs(fit,
                                                                variable = c("disp", "lprior", "lp__")
                                                                )
                            } else {
                                list_plot <- list()
                                for(male in males_used) {
                                    ##print(male);
                                    list_plot[[male]] <-
                                        pairs(fit,
                                              variable = c(male, "lp__"),
                                              regex = TRUE)

                                }
                                grid_plot <- cowplot::plot_grid(
                                                          plotlist = list_plot,
                                                          ncol = 2)
                                       #ggtitle(desc_short)

                                print(grid_plot)
                                filename <- paste0("grid.plot_", filename_desc, ".png")
                                ggsave(filename = filename, path = file.path(output_dir, "figures"),
                                       device = agg_png,
                                       width = 8, height = 11, units = "in",
                                       scale = 0.4,
                                       dpi=300)

                                       #           }
                                       #end else for fitting

                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
## [1] "two_piece"
```

```
## Error in eval(expr, envir, enclos): object 'stan_two_piece_func' not found
```

```
print(outfile_tbl)
```

```
## NULL
```

```
if(save_fits) save(fit_tbl, file = outfile_tbl)
```

Exit rendering

\small

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