Distributional Regression Illustration

Tim Wolock

05/03/2021

Contents

In this document, which accompanies Wolock *et al.* (2021), we will demonstrate how to fit a distributional model with a sinh-arcsinh likelihood in `brms` (Bürkner 2018). First, we will outline how to implement a custom family in `brms`, then we will simulate sinh-arcsinh distributed data with moments that vary with data, and finally, we will fit one conventional regression model and two distributional models.

The likelihood we will be using comes from Jones and Pewsey (2009). The density of a sinh-arcsinh random variable, X, is

$$p(x \mid \mu, \sigma, \epsilon, \delta) = \frac{1}{\sigma\sqrt{2\pi}} \frac{\delta C_{\epsilon,\delta}(x_z)}{\sqrt{1+x_z^2}} \exp\left[-\frac{S_{\epsilon,\delta}(x_z)^2}{2}\right],$$

where $x_z = (x-\mu)/\sigma$, $S_{\epsilon,\delta}(x) = \sinh(\epsilon + \delta \text{asinh}(x))$, and $C_{\epsilon,\delta} = \cosh(\epsilon + \delta \text{asinh}(x))$. As expected, μ and σ control the location and scale of the distribution. Both σ and δ must be greater than zero.

Setup

To step through this analysis yourself, clone or download this repository, open `distreg-illustration.Rproj` in RStudio, and open the `.Rmd` file in the RStudio editor. Be sure to have the following package installed:

1. `rstan`
2. `brms`
3. `data.table`
4. `ggplot2`

`data.table` is not strictly necessary for the analysis, but it will be useful when constructing posterior summaries.

First, we will load these required packages and configure some useful options.

```
library(brms)
library(rstan)
library(data.table)
library(ggplot2)

set.seed(10)
```

```
options(mc.cores = parallel::detectCores())
theme_set(theme_bw())
```

Now, we will load the functions necessary to fit a custom family in `brms`.

```
source('R/stan_funs.R')
stanvars <- stanvar(scode = stan_funs, block = "functions")
```

This `.R` file loads a `brms` family, two functions, and a string containing Stan functions into the global environment. The custom family, `sinhasinh`, will allow us to fit a model in `brms` with a likelihood that is not included in `brms` by default. It is defined as

```
sinhasinh <- custom_family(
  name = "sinhasinh",
  dpars = c("mu", "sigma", "eps", "delta"),
  links = c("identity", "log", "identity", "log"),
  lb = c(NA, NA, NA, NA),
  type = "real"
)
```

In the `dpars` argument, we are defining the names of the parameters of custom family. Note that every custom family in `brms` must have "mu" as its first distributional parameter. With `links`, we are defining the link functions for the linear models for each distributional parameter: both σ and δ must be greater than zero, so we assign them `log` link functions. The `lb` argument sets the lower bounds of each parameter, and the `type` argument defines the support of the distribution.

The R script we ran also adds two functions to the global environment:

1. `log_lik_sinhasinh`: allows `brms` to calculate the log-likelihood of the sinh-arcsinh distribution using Stan functions that we will expose later on.
2. `posterior_predict_sinhasinh`: allows `brms` to sample from the sinh-arcsinh distribution similarly using exposed functions from Stan.

Both of these functions are named in accordance with `brms` convention (`log_lik_FAMILYNAME` and `predict_FAMILYNAME`) so that we will be able to use the posterior prediction and checking tools built-in to the package.

Finally, the script sets a string called `stan_funs` in the global environment. This string contains two Stan functions, `sinhasinh_lpdf` and `sinhasinh_rng`, which allows to calculate the sinh-arcsinh log-density and take sinh-arcsinh samples, respectively. The log-density function with that particular name is required for the custom family to work. We will expose both functions to R after fitting the first model, and the two R functions defined above will wrap them.

For convenience, we also define a native R function to produce sinh-arcsinh samples.

```
rsinhasinh <- function(n, mu = 0, sigma = 1, eps = 0, delta = 1) {
  mu + sigma * sinh((asinh(rnorm(n, 0, 1)) - eps) / delta)
}
```

Simulated Data

We will generate sinh-arcsinh distributed data with moments that vary with data to fit to. First, we define the population of interest.

```
N <- 5000
ages <- round(runif(N, 15, 64))
pct_f <- 0.6
sexes <- c('Male', 'Female')[rbinom(N, 1, pct_f) + 1]
```

```r
# Load everything into a dataframe
age_df <- data.frame(age = ages, sex = sexes, id = 1:N)
age_df$scaled_age <- (ages - mean(ages)) / sd(ages)
age_df$sex <- relevel(factor(age_df$sex), ref = 'Female')
age_df$age_bin <- with(age_df, age - age %% 5)
```

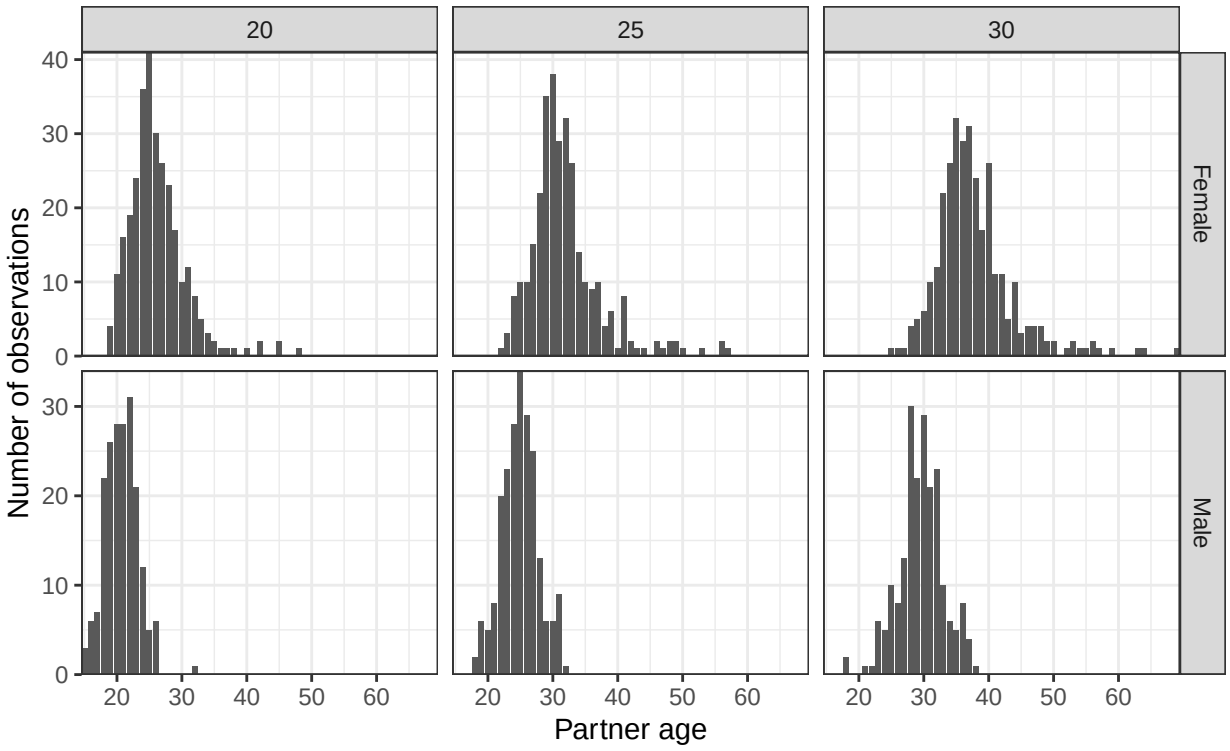
Now we will use distribution regression coefficients from Wolock (2021) to predict distributional parameters for each individual. These coefficients come from a model with linear age-sex interactions for all four parameters, so we need to be sure to define the model matrix in the same way.

```r
# Hard-coded coefficients from paper
B_mu <- c(0.1, -0.02, -0.18, 0.01)
B_sigma <- c(-2.44, -0.11, 0.00, 0.26)
B_eps <- c(-0.2, 0.01, 0.36, 0.06)
B_delta <- c(-0.46, -0.05, 0.01, 0.04)

# Create model matrix for simulating data
X <- model.matrix(id ~ scaled_age * sex, data = age_df)

# Get true distributional parameter values
age_df$mu <- X %*% B_mu
age_df$delta <- exp(X %*% B_delta)
age_df$sigma <- exp(X %*% B_sigma) * age_df$delta
age_df$eps <- X %*% B_eps
```

With all four distributional parameters in hand for every individual, we can sample sinh-arcsinh distributed outcomes. In this case, the dependent variable of the model those coefficients are associated with the log-ratio of partner's age to respondent's age.

```r
# Sample with function we defined earlier
age_df$log_ratio <- with(age_df, rsinhasinh(N, mu, sigma, eps, delta))
# Calculate partner age
age_df$p_age <- round(exp(age_df$log_ratio) * age_df$age)
```

We can plot the resulting distribution for several five-year age bins by sex.



Fitting Models

Now, we will fit three models in `brms`:

1. **Conventional:** regression with linear age-sex interaction for the location parameter and constant higher-order parameters
2. **Distributional 1:** distributional regression with linear age-sex interaction for location and independent age and sex effects for higher-order parameters
3. **Distributional 2:** distributional regression with linear age-sex interactions for all four parameters

Conventional Regression

First, we fit the conventional model. We use the `bf` function to build a `brms` formula with models for all four distributional parameters in the `sinhasinh` family. Note that we use the first formula (corresponding to `mu`) to set the outcome variable. We include the `stanvars` object defined by `R/stan_funs.R` in the model using the `stanvars` argument of `brm`.

```
bf_1 <- bf(
  log_ratio ~ scaled_age * sex,
  sigma ~ 1,
  eps ~ 1,
  delta ~ 1
)

brm_1 <- brm(formula = bf_1,
             data = age_df,
             family = sinhasinh,
             stanvars = stanvars)
```

```
kbl(posterior_summary(brm_1), digits = 2, booktabs = T) %>%
    kable_styling(c("striped"))
```

	Estimate	Est.Error	Q2.5	Q97.5
b_Intercept	0.11	0.00	0.11	0.12
b_sigma_Intercept	-2.47	0.02	-2.52	-2.43
b_eps_Intercept	-0.06	0.01	-0.08	-0.04
b_delta_Intercept	-0.53	0.02	-0.56	-0.49
b_scaled_age	-0.02	0.00	-0.02	-0.02
b_sexMale	-0.21	0.00	-0.22	-0.21
b_scaled_age:sexMale	0.00	0.00	-0.01	0.01
lp___	8031.77	1.95	8027.15	8034.51

Because of our slightly unconventional specification for the last three parameters, `brm` returns estimates for `b_Intercept` parameters for all four distributional parameters.

The R functions we defined in `R/stan_funs.R` give us what we need to use the posterior checking tools built into `brms`. We just need to expose the Stan functions that the R functions rely on.

```
expose_functions(brm_1, vectorize = T, show_compiler_warnings=F)
```

Distributional Regression 1

To fit a distributional model in `brms`, we just modify the `bf` formula.

```
bf_2 <- bf(
  log_ratio ~ scaled_age * sex,
  sigma ~ scaled_age + sex,
  eps ~ scaled_age + sex,
  delta ~ scaled_age + sex
)

brm_2 <- brm(formula = bf_2,
             data = age_df,
             family = sinhasinh,
             stanvars = stanvars)
```

```
kbl(posterior_summary(brm_2), digits = 2, booktabs = T) %>%
    kable_styling(c("striped"))
```

	Estimate	Est.Error	Q2.5	Q97.5
b_Intercept	0.10	0.00	0.09	0.10
b_sigma_Intercept	-2.46	0.03	-2.51	-2.41
b_eps_Intercept	-0.21	0.02	-0.24	-0.18
b_delta_Intercept	-0.47	0.02	-0.52	-0.42
b_scaled_age	-0.01	0.00	-0.02	-0.01
b_sexMale	-0.18	0.00	-0.19	-0.17
b_scaled_age:sexMale	0.00	0.00	0.00	0.01
b_sigma_scaled_age	-0.04	0.02	-0.08	0.01
b_sigma_sexMale	-0.01	0.04	-0.10	0.07
b_eps_scaled_age	0.05	0.01	0.03	0.08
b_eps_sexMale	0.35	0.03	0.30	0.40
b_delta_scaled_age	-0.06	0.02	-0.09	-0.02
b_delta_sexMale	0.00	0.04	-0.07	0.08
lp___	8144.56	2.55	8138.47	8148.56

We can see that we now have slopes with respect to age and sex for all four distributional parameters.

Distributional Regression 2

Finally, we fit a distributional model with age-sex interactions for all four distributional parameters, which we know is the correct model.

```
bf_3 <- bf(
  log_ratio ~ scaled_age * sex,
  sigma ~ scaled_age * sex,
  eps ~ scaled_age * sex,
  delta ~ scaled_age * sex
)

brm_3 <- brm(formula = bf_3,
             data = age_df,
             family = sinhasinh,
             stanvars = stanvars)
```

```
kbl(posterior_summary(brm_3), digits = 2, booktabs = T) %>%
    kable_styling(c("striped"))
```

	Estimate	Est.Error	Q2.5	Q97.5
b_Intercept	0.10	0.00	0.09	0.10
b_sigma_Intercept	-2.47	0.03	-2.52	-2.42
b_eps_Intercept	-0.21	0.02	-0.24	-0.17
b_delta_Intercept	-0.47	0.02	-0.52	-0.42
b_scaled_age	-0.02	0.00	-0.02	-0.01
b_sexMale	-0.18	0.00	-0.19	-0.17
b_scaled_age:sexMale	0.00	0.00	0.00	0.01
b_sigma_scaled_age	-0.13	0.03	-0.18	-0.08
b_sigma_sexMale	0.03	0.04	-0.06	0.12
b_sigma_scaled_age:sexMale	0.25	0.04	0.16	0.33
b_eps_scaled_age	0.04	0.02	0.01	0.07
b_eps_sexMale	0.35	0.02	0.30	0.40
b_eps_scaled_age:sexMale	0.01	0.03	-0.04	0.06
b_delta_scaled_age	-0.07	0.02	-0.11	-0.02
b_delta_sexMale	0.03	0.04	-0.04	0.11
b_delta_scaled_age:sexMale	0.03	0.04	-0.05	0.11
lp___	8174.77	2.91	8168.22	8179.42

Model Comparison

Because we have defined the `log_lik` and `predict` functions `brms` expects, we can use the LOO-CV functions (Vehtari, Gelman, and Gabry 2017) built into the package. Note that this step can be computationally intensive.

```
loo_res <- loo(brm_1, brm_2, brm_3)
```

This function estimates the expected log-posterior densities (ELPDs) for all three model, as well as the comparison of all three models. We can print a single model's results first.

```
print(loo_res$loos$brm_1)
```

```
##
## Computed from 4000 by 5000 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo   8032.4  71.4
## p_loo         7.6   0.2
## looic    -16064.7 142.9
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

We can also print the comparison. A negative value of

```
print(loo_res$diffs)
```

```
##       elpd_diff se_diff
## brm_3    0.0       0.0
## brm_2  -28.8       7.9
## brm_1 -138.9      16.9
```

We can see that, when we compare the two distributional models (`brm_2` and `brm_3`) the absolute value of ratio of the ELPD difference to the standard error of the difference is 3.66, suggesting that the second distributional model is significantly better than the first.

Posterior Prediction

We can create an evenly spaced `data.frame` to predict over.

```
pred_df <- merge(15:64, c('Female', 'Male'))
names(pred_df) <- c('age', 'sex')
pred_df$scaled_age <- (pred_df$age - mean(ages)) / sd(ages)
pred_df$sex <- relevel(factor(pred_df$sex), ref = 'Female')
pred_df$log_ratio <- rep(0, nrow(pred_df))
pred_df$p_age <- round(exp(pred_df$log_ratio) * pred_df$age)
```

We put all three fit objects into a list to make prediction slightly easier. When we apply the `prepare_predictions` function to each fit object, `brms` will generate posterior predictive samples.

```
fit_l <- list('Conv' = brm_1,
              'Dist 1' = brm_2,
              'Dist 2' = brm_3)
```

Posterior Predictive Distributions

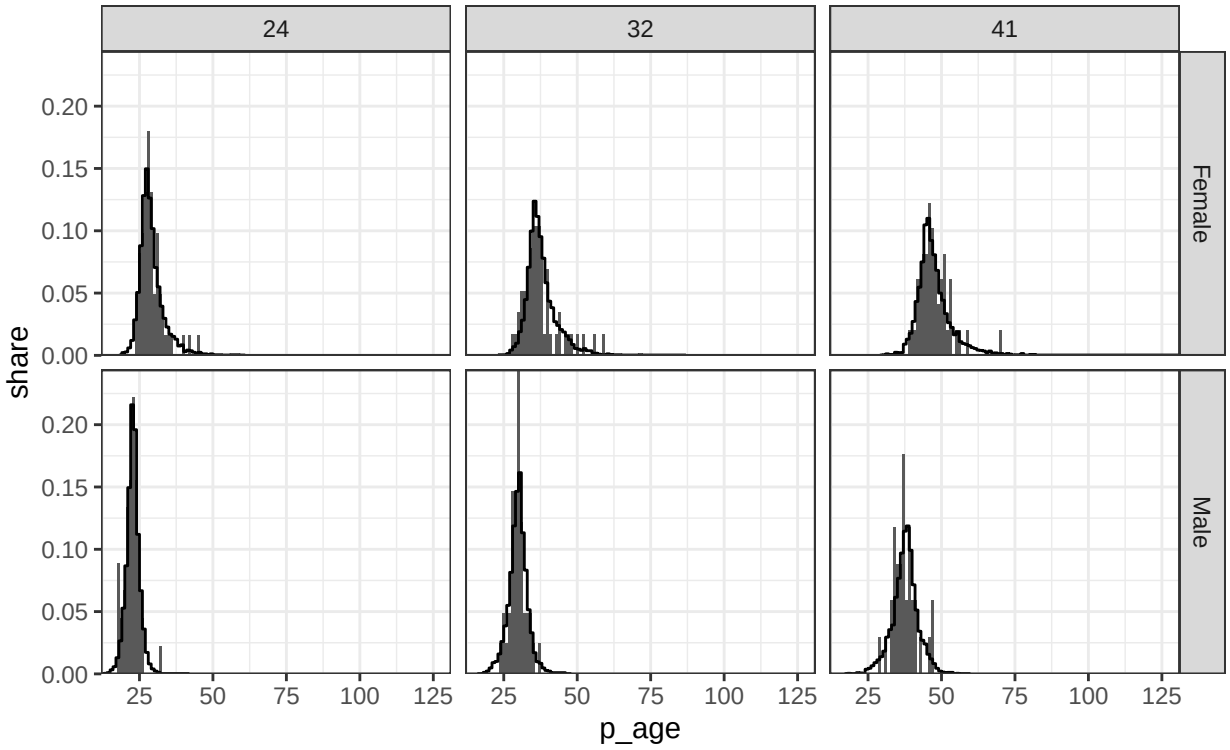
We can get posterior predictive samples and construct a `data.frame` for histograms.

```
posterior_l <- lapply(fit_l, posterior_predict, newdata=pred_df)

# Using data.table for convenience
post_dt <- rbindlist(lapply(posterior_l, function(x) {
  melt(cbind(data.table(pred_df), t(x)),
       id.vars=names(pred_df))}),
  idcol = 'Model')
post_dt[, p_age_post := round(exp(value) * age)]

post_hist_dt <- post_dt[, .N, by = .(Model, sex, age, p_age = p_age_post)]
```

```
post_hist_dt[, total_N := sum(N), by=.(Model, sex, age)]
post_hist_dt[, share := N / total_N]

post_hist_df <- data.frame(post_hist_dt)
```

We can compare our predictive distributions to the observed distributions.

```
hist_dt <- data.table(age_df)[, .N, by=.(age, sex, p_age)]
hist_dt[, total_N := sum(N), by=.(age, sex)]
hist_dt[, share := N / total_N]
hist_df <- data.frame(hist_dt)

ggplot() +
  geom_bar(data = hist_df[hist_df$age %in% c(24, 32, 41),],
           aes(x = p_age,
               y = share),
           stat = 'identity') +
  geom_step(data = post_hist_df[post_hist_df$age %in% c(24, 32, 41) &
                                post_hist_df$Model == 'Dist 2',],
            aes(x = p_age,
                y = share),
            direction = 'hv') +
  facet_grid(sex ~ age) +
  coord_cartesian(expand = 0)
```



Posterior Distributional Parameter Summaries

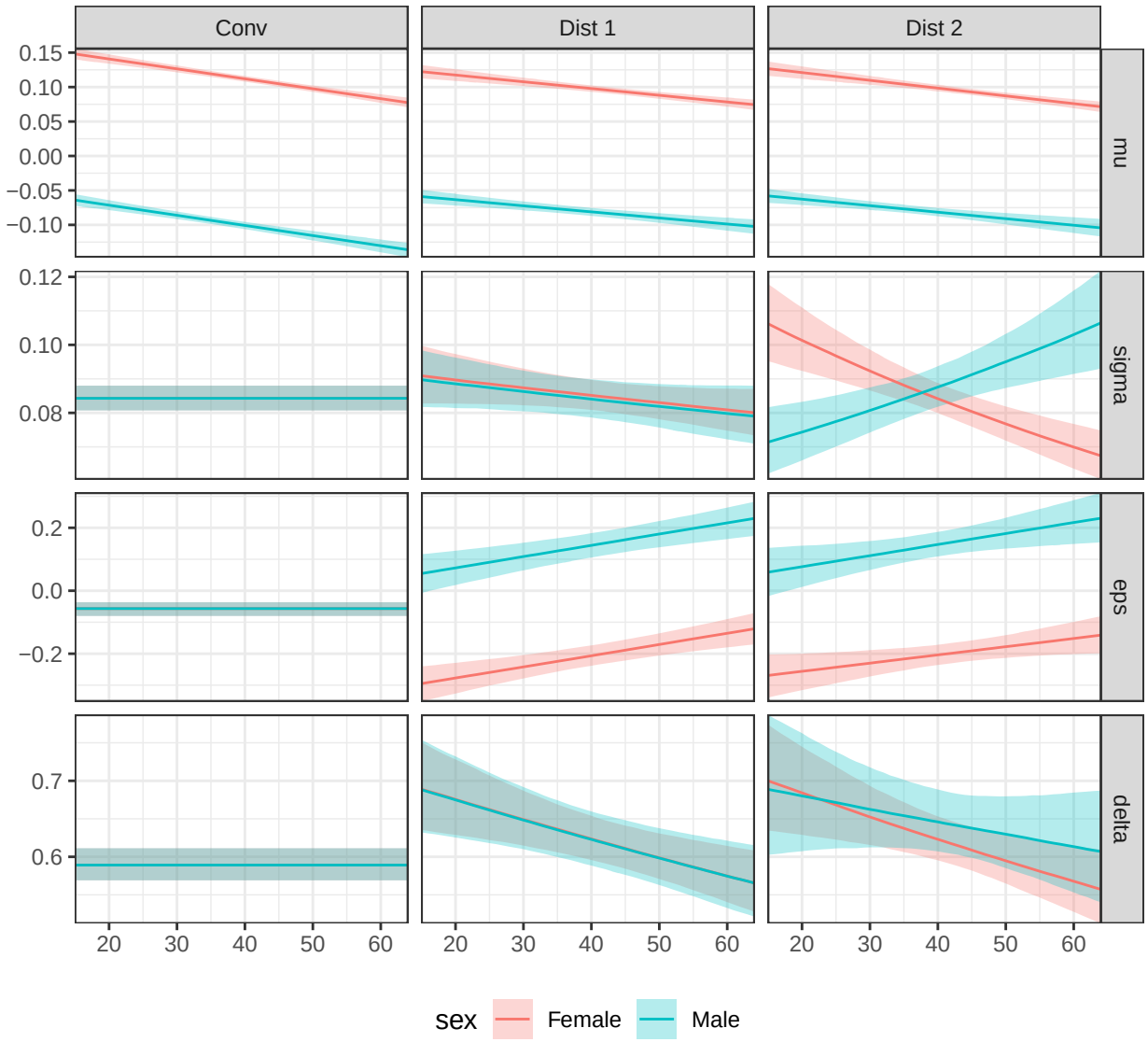
Now, we will extract estimates of the distributional parameters.

```
# Get all dpar predictions
prediction_l <- lapply(fit_l, prepare_predictions, newdata=pred_df)
```

```
# A data.table-y way to get posterior summaries of dpars
dpar_l <- lapply(c('mu', 'sigma', 'eps', 'delta'), function(d) {
  lapply(prediction_l, function(x) {
    cbind(data.table(pred_df), t(apply(brms:::get_dpar(x, dpar = d), 2,
                                       quantile,
                                       probs=c(0.025, 0.5, 0.975))))
  })
})
names(dpar_l) <- c('mu', 'sigma', 'eps', 'delta')

dpar_df <- data.frame(rbindlist(lapply(dpar_l, rbindlist, idcol = 'Model'), idcol = 'dpar'))
dpar_df$dpar <- factor(dpar_df$dpar, levels = c('mu', 'sigma', 'eps', 'delta'))
```

We can plot the posterior CIs to see the effects of adding distributional models.

```
ggplot(dpar_df,
       aes(x = age,
           ymin = `X2.5.`,
           y = `X50.`,
           ymax = `X97.5.`,
           fill = sex,
           color = sex)) +
  geom_ribbon(alpha=0.3, color=NA) +
  geom_line() +
  labs(x=NULL, y=NULL) +
  coord_cartesian(expand = 0) +
  facet_grid(dpar ~ Model, scales='free') +
  theme_bw() + theme(legend.position = 'bottom')
```



From here, we should be able to use any further `brms` processing tools we would like to.

References

Bürkner, Paul-Christian. 2018. "Advanced Bayesian Multilevel Modeling with the r Package Brms." *The R Journal* 10 (1): 395411. https://doi.org/10.32614/RJ-2018-017.

Jones, M. C., and Arthur Pewsey. 2009. "Sinh-Arcsinh Distributions." *Biometrika* 96 (4): 761–80. https://doi.org/10.1093/biomet/asp053.

Vehtari, Aki, Andrew Gelman, and Jonah Gabry. 2017. "Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and WAIC." *Statistics and Computing* 27 (5): 1413–32. https://doi.org/10.1007/s11222-016-9696-4.

Session Info

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
```

```
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] data.table_1.13.2    rstan_2.21.2         ggplot2_3.3.3
## [4] StanHeaders_2.21.0-6 brms_2.14.0          Rcpp_1.0.6
## [7] kableExtra_1.2.1     knitr_1.30
##
## loaded via a namespace (and not attached):
##   [1] nlme_3.1-144         matrixStats_0.56.0   xts_0.12.1
##   [4] webshot_0.5.1        threejs_0.3.3        httr_1.4.2
##   [7] backports_1.1.10     tools_3.6.3          R6_2.5.0
##  [10] DT_0.15              colorspace_1.4-1     withr_2.3.0
##  [13] tidyselect_1.1.0     gridExtra_2.3        prettyunits_1.1.1
##  [16] processx_3.4.5       Brobdingnag_1.2-6    emmeans_1.4.8
##  [19] curl_4.3             compiler_3.6.3       cli_2.1.0
##  [22] rvest_0.3.4          xml2_1.3.2           shinyjs_2.0.0
##  [25] labeling_0.4.2       colourpicker_1.1.0   bookdown_0.20
##  [28] checkmate_2.0.0      scales_1.1.1         dygraphs_1.1.1.6
##  [31] mvtnorm_1.1-1        ggridges_0.5.2       callr_3.5.1
##  [34] stringr_1.4.0        digest_0.6.27        rmarkdown_2.5
##  [37] base64enc_0.1-3      pkgconfig_2.0.3      htmltools_0.5.0
##  [40] fastmap_1.0.1        htmlwidgets_1.5.1    rlang_0.4.8
##  [43] rstudioapi_0.13      shiny_1.5.0          farver_2.0.3
##  [46] generics_0.0.2       zoo_1.8-8            jsonlite_1.7.2
##  [49] crosstalk_1.1.0.1    gtools_3.8.2         dplyr_1.0.2
##  [52] inline_0.3.16        magrittr_2.0.1       loo_2.3.1
##  [55] bayesplot_1.7.2      Matrix_1.2-18        munsell_0.5.0
##  [58] fansi_0.4.1          abind_1.4-5          lifecycle_0.2.0
##  [61] stringi_1.5.3        yaml_2.2.1           pkgbuild_1.1.0
##  [64] plyr_1.8.6           grid_3.6.3           parallel_3.6.3
##  [67] promises_1.1.1       crayon_1.3.4         miniUI_0.1.1.1
##  [70] lattice_0.20-38      BH_1.75.0-0          ps_1.4.0
##  [73] pillar_1.4.6         igraph_1.2.5         estimability_1.3
##  [76] markdown_1.1         shinystan_2.5.0      codetools_0.2-16
##  [79] reshape2_1.4.4       stats4_3.6.3         rstantools_2.1.1
##  [82] glue_1.4.2           evaluate_0.14        V8_3.2.0
##  [85] RcppParallel_5.0.2   vctrs_0.3.4          httpuv_1.5.4
##  [88] gtable_0.3.0         purrr_0.3.4          assertthat_0.2.1
##  [91] xfun_0.19            mime_0.10            RcppEigen_0.3.3.7.0
##  [94] xtable_1.8-4         coda_0.19-3          later_1.1.0.1
##  [97] rsconnect_0.8.16     viridisLite_0.3.0    tibble_3.0.4
## [100] shinythemes_1.1.2    ellipsis_0.3.1       bridgesampling_0.7-2
```