Fitting distributional models with `brms`

Stefano Mezzini

2022-08-01

Contents

1 Setup

Start by attaching all necessary packages and creating a custom function for plotting parameters:

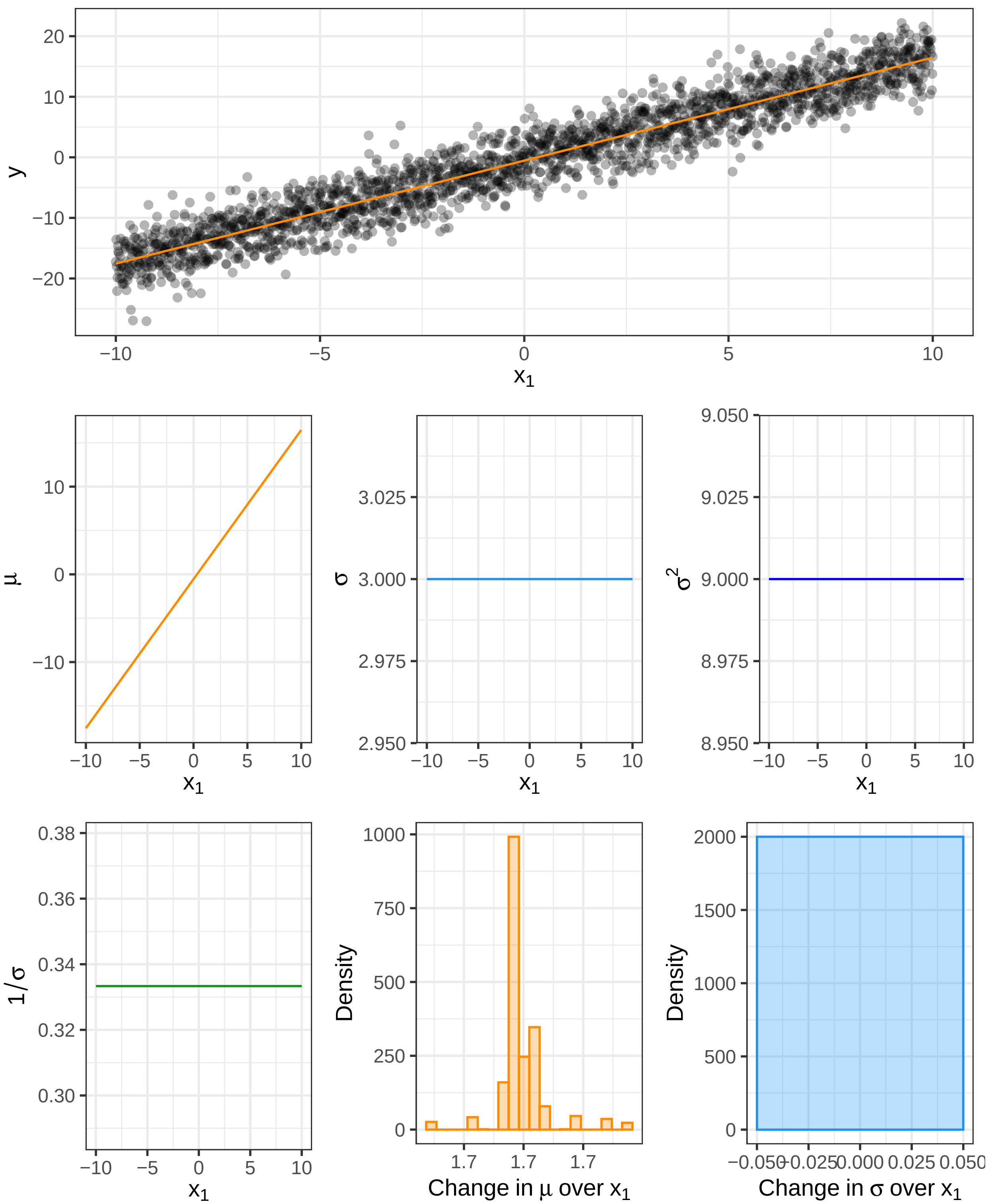
```
library('dplyr')   # for data wrangling
library('mgcv')    # for empirical Bayes modeling
library('ggplot2') # for fancy plots
library('cowplot') # for multi-panel fancy plots
library('gratia')  # for fancy GAM plots
library('brms')    # for Bayesian modeling
theme_set(theme_bw() + theme(legend.position = 'top')) # change default plot theme

# function to plot parameters and samples
gaus_plots <- function(.data) {
  plot_grid(
    ggplot(.data) +
      geom_point(aes(x1, y), alpha = 0.3) +
      geom_line(aes(x1, mu), color = 'darkorange') +
      labs(x = expression(x[1]), expression(Y~'~'~N(mu,~sigma^2))),
    plot_grid(ggplot(.data) +
                geom_line(aes(x1, mu), color = 'darkorange') +
                labs(x = expression(x[1]), y = expression(mu)),
              ggplot(.data) +
                geom_line(aes(x1, sigma), color = 'dodgerblue') +
                labs(x = expression(x[1]), y = expression(sigma)),
              ggplot(.data) +
                geom_line(aes(x1, sigma2), color = 'blue') +
                labs(x = expression(x[1]), y = expression(sigma^2)),
              ggplot(.data) +
                geom_line(aes(x1, 1 / sigma), color = 'forestgreen') +
                labs(x = expression(x[1]), y = expression(1/sigma)),
              ggplot(.data) +
                geom_histogram(aes(coef_mu), fill = 'darkorange', color = 'darkorange',
                               alpha = 0.3, na.rm = TRUE, bins = 20) +
                labs(x = expression(Change~'in'~mu~over~x[1]), y = 'Density'),
              ggplot(.data) +
                geom_histogram(aes(coef_sigma), fill = 'dodgerblue', color = 'dodgerblue',
                               alpha = 0.3, na.rm = TRUE, bins = 20) +
                labs(x = expression(Change~'in'~sigma~over~x[1]), y = 'Density'),
              ncol = 3), ncol = 1, rel_heights = 1:2)
}
```

We can start with a simple model where μ increases linearly while σ^2 is constant. In `mgcv` we estimate $\phi = 1/\sigma$ rather than σ because ϕ is more stable than σ.

```
d <-
  tibble(x1 = seq(-10, 10, by = 0.01), # predictor variable
         mu = x1 * 1.7 - 0.54, # mean
         sigma = 3, # standard deviation
         sigma2 = sigma^2, # variance
         coef_mu = c(NA, diff(mu) / diff(x1)), # changes in mu over x1
         coef_sigma = c(NA, diff(sigma) / diff(x1)), # changes in sigma over x1
         y = rnorm(n = length(x1), mean = mu, sd = sqrt(sigma2))) # samples
gaus_plots(d)
```

```
## Warning: position_stack requires non-overlapping x intervals
```


y
x1
μ
σ
σ2
1/σ
Density
Change in μ over x1
Change in σ over x1

2 Fitting distributional Gaussian models with `mgcv`

The `logb` link function in `mgcv` is a `log` link function where `b` is the minimum σ: $\eta = \log(\sigma - b)$ and $\sigma = b + \exp(\eta)$. See `?gaulss` for more info.

```
# fit model with mgcv
m_mgcv <- gam(list(y ~ x1,
                   ~ 1),
              family = gaulss(link = list('identity', 'logb'), b = 0.01),
              data = d)
knitr::kable(summary(m_mgcv)$p.table, format = 'pipe', digits = 2)
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-0.54	0.07	-7.97	0
x1	1.71	0.01	144.58	0
(Intercept).1	1.11	0.02	70.27	0

The `mgcv` model has the following estimates: $\beta_0 =$ -0.54, $\beta_1 =$ 1.71, $\eta_\phi =$ 1.11 $\implies \phi = b + \exp(\eta_\phi) =$ 3.04.

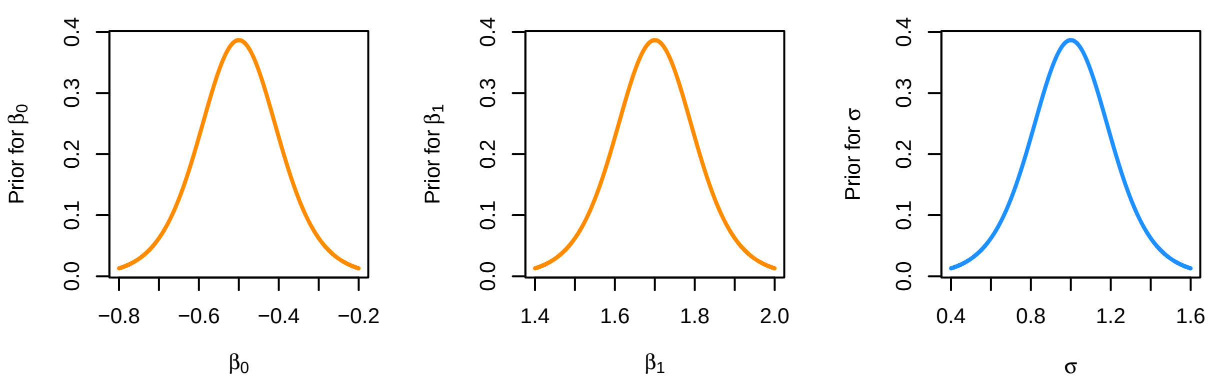
3 Fitting distributional Gaussian models with brms

```
# check what priors can be defined by printing default ones
get_prior(formula = bf(y ~ x1,
                       sigma ~ 1, # constant sd
                       alpha ~ 0, # no skew
                       family = skew_normal(link = 'identity', link_sigma = 'log')),
          data = d)
```

```
##                     prior     class coef group resp  dpar nlpar lb ub       source
##                    (flat)         b                                        default
##                    (flat)         b   x1                              (vectorized)
##  student_t(3, -0.6, 12.8) Intercept                                        default
##      student_t(3, 0, 2.5) Intercept                 sigma                  default
```

```
priors <-
  c(
    # these priors are excessively informative to reduce fitting time
    # priors are on the link scale, so lb for sigma isn't necessary (or appropriate)
    prior(student_t(8, -0.5, 0.1), class = 'Intercept'), # student_t(df, mu, scale)
    prior(student_t(8, 1.7, 0.1), class = 'b', coef = 'x1'),
    prior(student_t(8, 1, 0.2), class = 'Intercept', dpar = 'sigma')
  )

# plot priors
x <- seq(-3, 3, by = 1e-3)
layout(t(1:3))
plot(x * 0.1 - 0.5, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~beta[0]),
     xlab = expression(beta[0]), col = 'darkorange', lwd = 2)
plot(x * 0.1 + 1.7, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~beta[1]),
     xlab = expression(beta[1]), col = 'darkorange', lwd = 2)
plot(x * 0.2 + 1, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~sigma),
     xlab = expression(sigma), col = 'dodgerblue', lwd = 2)
```




```
layout(1)

# fit the model
m_brms <- brm(bf(y ~ x1,
                 sigma ~ 1,
```

```r
                  alpha ~ 0,
                  family = skew_normal(link = 'identity', link_sigma = 'log')),
               data = d,
               prior = priors,
               chains = 4,
               warmup = 100, # number of iterations to use for warmup
               iter = 2000,  # total number of iterations (for warmup + sampling)
               cores = 4)
```

```
## Compiling Stan program...
```

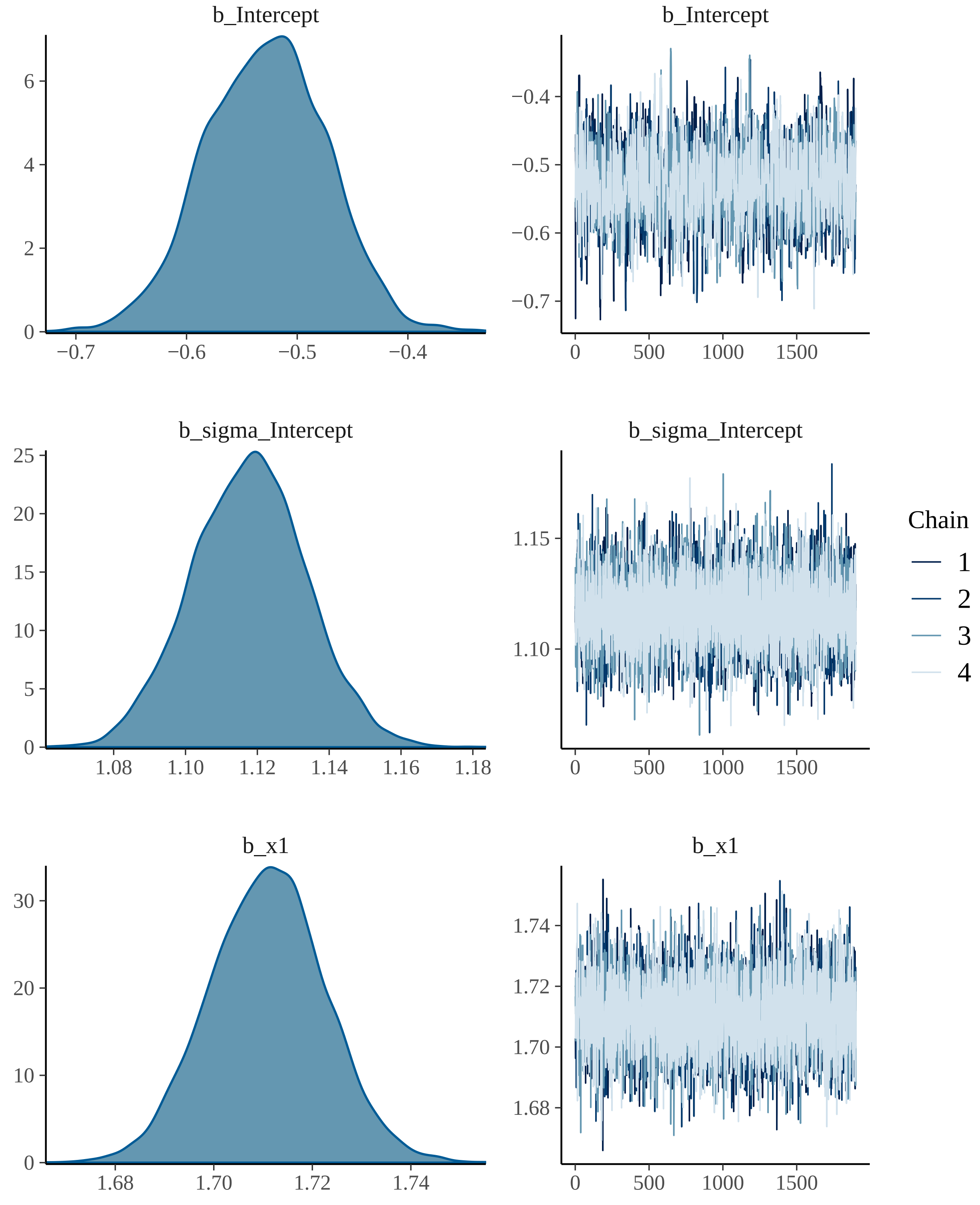
```
## Start sampling
```

Note: if `brms` models fail with the error `Error in sink(type = "output") : invalid connection`, try installing the following packages:

```r
install.packages(c("StanHeaders","rstan"),type="source")
```

The posteriors are on the link scale, and β_0 is the value of y when $x_1 = 0$, not when $x_1 = \bar{x}_1$:

```r
plot(m_brms, N = 3) # plot posteriors for all parameters
```



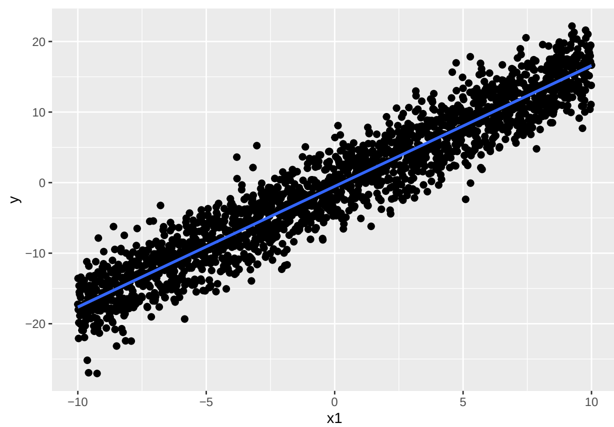
$\beta_0 = 0.54 \approx$ -0.5283928 $= \hat{\beta}_0$

$\beta_1 = 1.7 \approx$ -0.5283928 $= \hat{\beta}_1$

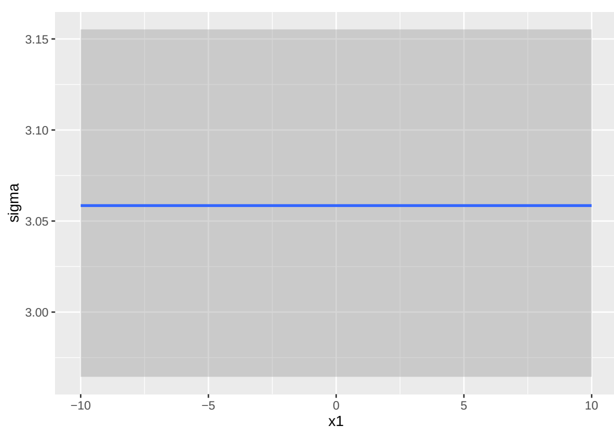
$\sigma = 3 \approx 3.0578806 = \hat{\sigma}$

We can plot the parameters using `conditional_effects()`. For the trend in the mean, we can add the data using `plot()` and specifying `points = TRUE`:

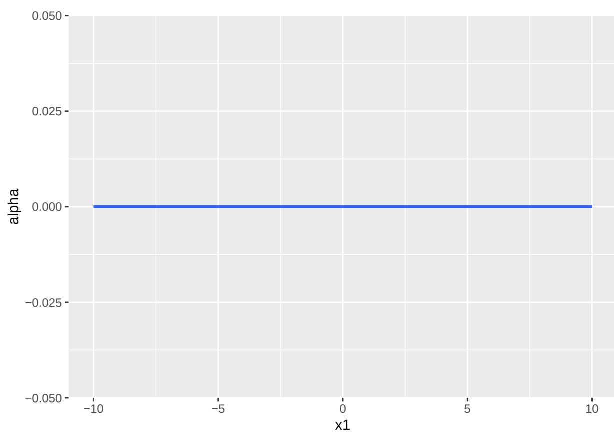
```
plot(conditional_effects(m_brms), points = TRUE)
```




```
conditional_effects(m_brms, dpar = 'sigma')
```




```
conditional_effects(m_brms, dpar = 'alpha')
```



We can extract the estimated parameters for different values of the predictor(s) using `fitted()` (or equivalently `posterior_epred()`):

```
newd <- tibble(x1 = c(-10, -5, 0, 5, 10))
fitted(m_brms, newdata = newd) # estimated mean (mu) changes with x1
```

```
##         Estimate  Est.Error        Q2.5       Q97.5
## [1,] -17.6372852 0.12756257 -17.8868497 -17.3855098
## [2,]  -9.0828390 0.07874653  -9.2360956  -8.9288351
## [3,]  -0.5283928 0.05476538  -0.6365995  -0.4242169
## [4,]   8.0260535 0.08165108   7.8650028   8.1868177
## [5,]  16.5804997 0.13116392  16.3248822  16.8335212
```

```
fitted(m_brms, newdata = newd, dpar = 'sigma') # constant standard deviation (sigma)
```

```
##      Estimate  Est.Error     Q2.5    Q97.5
## [1,] 3.058266 0.04854206 2.964451 3.155248
## [2,] 3.058266 0.04854206 2.964451 3.155248
## [3,] 3.058266 0.04854206 2.964451 3.155248
## [4,] 3.058266 0.04854206 2.964451 3.155248
## [5,] 3.058266 0.04854206 2.964451 3.155248
```

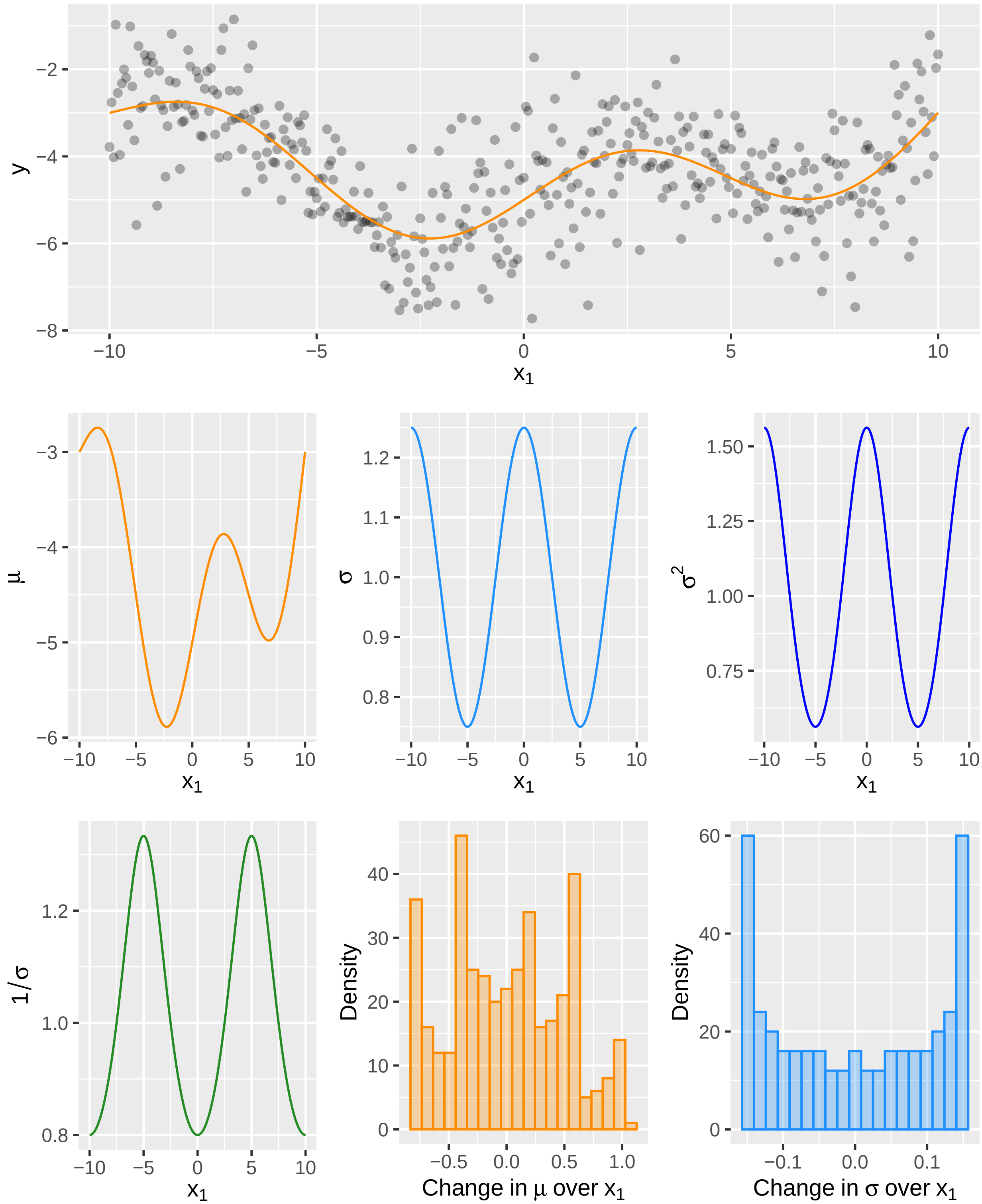
```
fitted(m_brms, newdata = newd, dpar = 'alpha') # skew (alpha) set to 0 (not estimated)
```

```
##      Estimate Est.Error Q2.5 Q97.5
## [1,]        0         0    0     0
## [2,]        0         0    0     0
## [3,]        0         0    0     0
## [4,]        0         0    0     0
## [5,]        0         0    0     0
```

4 Fitting smooth distributional Gaussian models

We can create a more complex "true" model where both μ and σ^2 change nonlinearly with x_1:

```
set.seed(1) # for consistent results
d <-
  tibble(x1 = seq(-10, 10, by = 0.05), # predictor variable
         mu = sinpi(x1 / 5) + x1^2 * 0.02 - 5, # mean
         sigma = cospi(x1 / 5) * 0.25 + 1, # standard deviation
         sigma2 = sigma^2, # variance
         coef_mu = c(NA, diff(mu) / diff(x1)), # changes in mu over x1
         coef_sigma = c(NA, diff(sigma) / diff(x1)), # changes in sigma over x1
         y = rnorm(n = length(x1), mean = mu, sd = sqrt(sigma2))) # samples
gaus_plots(d)
```



We start by fitting a model with mgcv as before:

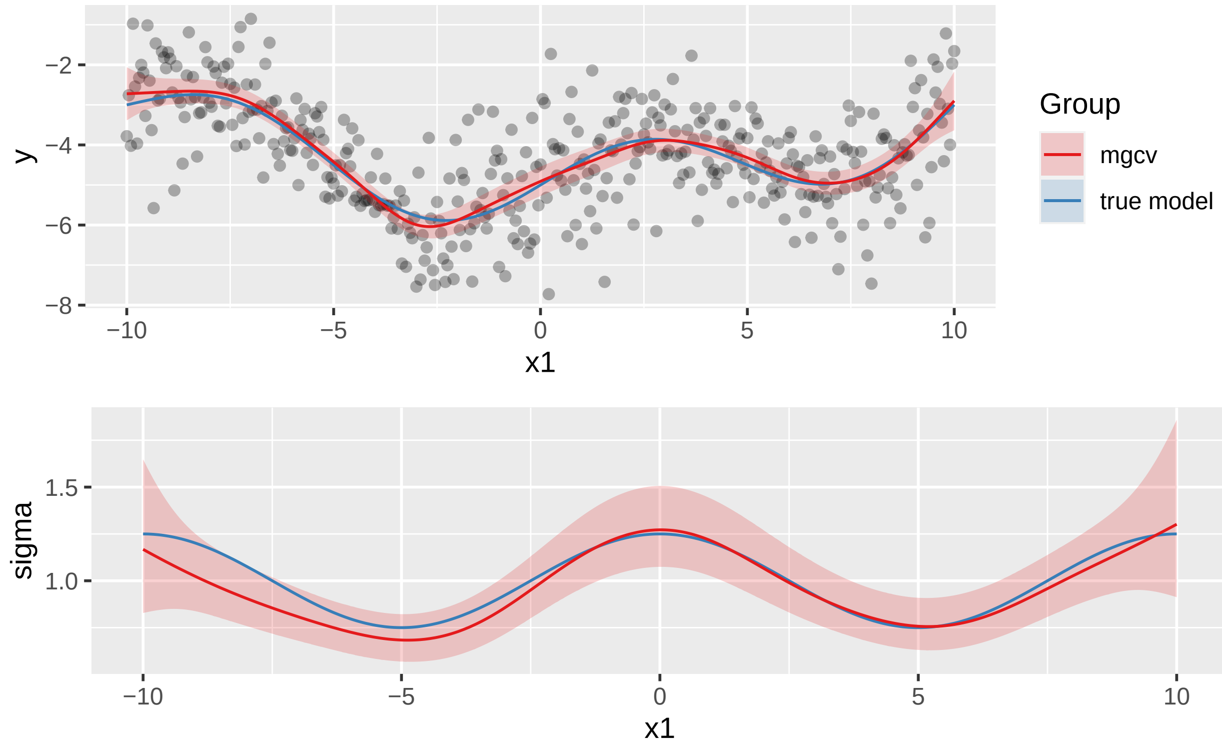
```
# fit model with mgcv
m_mgcv <- gam(list(y ~ s(x1, k = 20),
                     ~ s(x1, k = 15)),
               family = gaulss(link = list('identity', 'logb'), b = 0.01),
```

```
              data = d,
              method = 'REML') # restricted marginal likelihood
summary(m_mgcv)
```

```
##
## Family: gaulss
## Link function: identity logb
##
## Formula:
## y ~ s(x1, k = 20)
## ~s(x1, k = 15)
##
## Parametric coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -4.29302    0.04860 -88.337   <2e-16 ***
## (Intercept).1 -0.07531    0.03600  -2.092   0.0365 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##             edf Ref.df Chi.sq  p-value
## s(x1)    10.289  12.55 438.75  < 2e-16 ***
## s.1(x1)   6.222   7.65  33.42 4.28e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Deviance explained =   53%
## -REML = 581.32  Scale est. = 1         n = 401
```

```
d <- bind_cols(d,
               predict(m_mgcv, type = 'link', se.fit = TRUE) %>% # mu & log(sigma-b)
                 data.frame() %>%
                 transmute(mu_mgcv = fit.1, # mean from mgcv model w 95% CIs
                           mu_mgcv_lwr = fit.1 - 1.96 * se.fit.1,
                           mu_mgcv_upr = fit.1 + 1.96 * se.fit.1,
                           sigma_mgcv = exp(fit.2) + 0.01, # sigma w 95% CIs
                           sigma_mgcv_lwr = exp(fit.2 - 1.96 * se.fit.2) + 0.01,
                           sigma_mgcv_upr = exp(fit.2 + 1.96 * se.fit.2) + 0.01))
plot_grid(ggplot(d) +
            geom_point(aes(x1, y), alpha = 0.3) +
            geom_ribbon(aes(x1, ymin = mu_mgcv_lwr, ymax = mu_mgcv_upr, fill = 'mgcv'),
                        alpha = 0.2) +
            geom_line(aes(x1, mu, color = 'true model')) +
            geom_line(aes(x1, mu_mgcv, color = 'mgcv')) +
            scale_color_brewer('Group', type = 'qual', palette = 6,
                               aesthetics = c('color', 'fill')),
          ggplot(d) +
            geom_ribbon(aes(x1, ymin = sigma_mgcv_lwr, ymax = sigma_mgcv_upr,
                            fill = 'mgcv'), alpha = 0.2) +
            geom_line(aes(x1, sigma, color = 'true model')) +
            geom_line(aes(x1, sigma_mgcv, color = 'mgcv')) +
            scale_color_brewer('Group', type = 'qual', palette = 6,
                               aesthetics = c('color', 'fill')) +
            theme(legend.position = 'none'),
```

```
         ncol = 1, rel_heights = c(1.1, 1))
```



Note that the `s.1(x1)` for σ is on the log-link scale.

We can now fit the model with `brms` and compare results. Let's start from checking the default priors:

```
# printing default priors
get_prior(formula = bf(y ~ s(x1), # smooth change in mu
                       sigma ~ s(x1), # smooth change in sigma
                       alpha ~ 0, # no skew
                       family = skew_normal(link = 'identity', link_sigma = 'log')),
          data = d)
```

```
##                  prior     class  coef group resp  dpar nlpar lb ub       source
##                 (flat)         b                                         default
##                 (flat)         b sx1_1                              (vectorized)
##  student_t(3, -4.2, 2.5) Intercept                                       default
##    student_t(3, 0, 2.5)       sds                              0         default
##    student_t(3, 0, 2.5)       sds s(x1)                        0    (vectorized)
##                 (flat)         b                  sigma                  default
##                 (flat)         b sx1_1            sigma             (vectorized)
##    student_t(3, 0, 2.5) Intercept                 sigma                  default
##    student_t(3, 0, 2.5)       sds                 sigma        0         default
##    student_t(3, 0, 2.5)       sds s(x1)           sigma        0    (vectorized)
```

The population-level effects (`class = b`) for the effect of x_1 (`coef = sx1_1`) are the coefficients of the functions used to build the splines, which we can get from the `mgcv` model using `coef()`:

```
coef(m_mgcv)
```

```
##   (Intercept)       s(x1).1       s(x1).2       s(x1).3       s(x1).4
##   -4.29302062    0.48620408   -2.59642819   -3.56705721   -1.63468022
##       s(x1).5       s(x1).6       s(x1).7       s(x1).8       s(x1).9
##    1.19959412   -2.14919943   -0.35504812   -1.41799195   -0.84037024
##      s(x1).10      s(x1).11      s(x1).12      s(x1).13      s(x1).14
##   -1.47807834    0.32902004    1.69584764   -0.50902477   -1.52757023
##      s(x1).15      s(x1).16      s(x1).17      s(x1).18      s(x1).19
##   -0.64900969   -1.42082787    0.45107539   -5.73382116    3.36324822
## (Intercept).1     s.1(x1).1     s.1(x1).2     s.1(x1).3     s.1(x1).4
##   -0.07531434    0.03326451   -1.64913199    0.12870659   -0.02527821
##     s.1(x1).5     s.1(x1).6     s.1(x1).7     s.1(x1).8     s.1(x1).9
##   -0.16464095    0.31752956   -0.06054677   -0.42088673   -0.10861466
##    s.1(x1).10    s.1(x1).11    s.1(x1).12    s.1(x1).13    s.1(x1).14
##   -0.39873199   -0.16365821    0.38895508   -1.45478851   -0.05086436
```

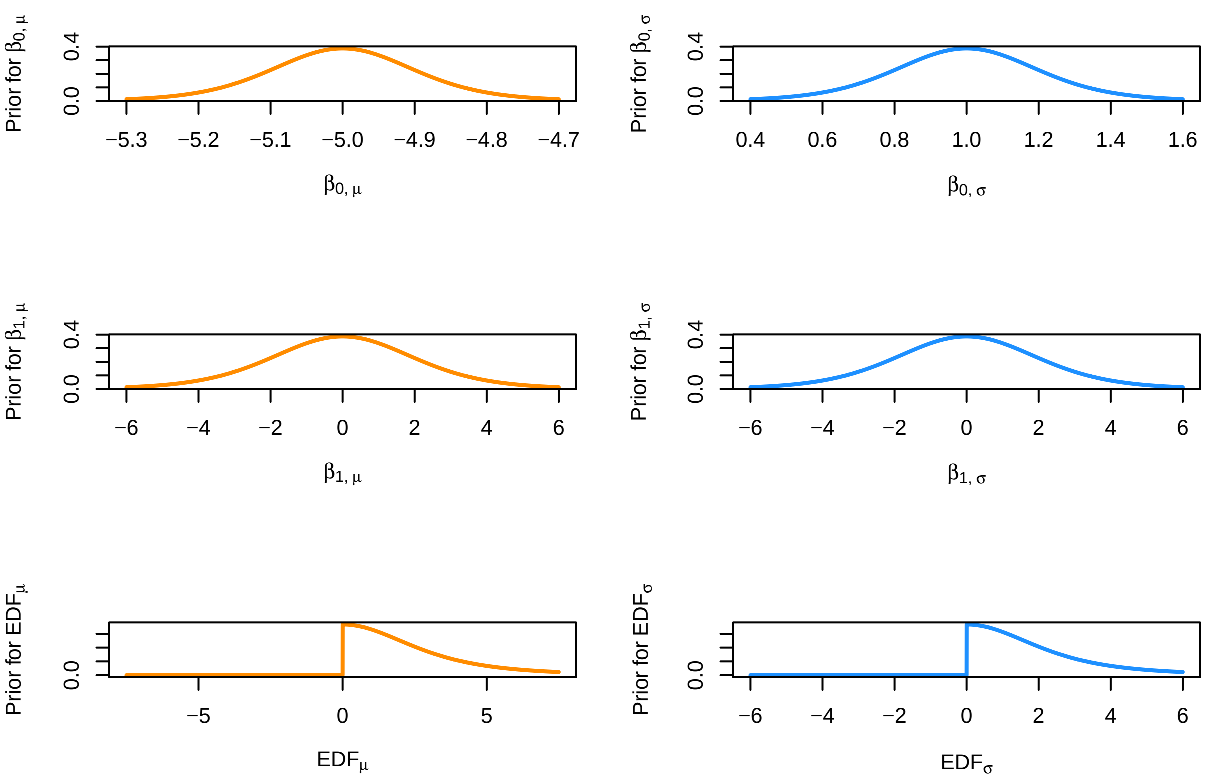
The coefficients for both smooths (`s(x1)` and `s.1(x1)`) range between -3 and +2.

We can set priors for the intercepts as before, but now we need to also set priors for the smooth terms (instead of the linear terms). We now have coefficients for the intercept terms ($\beta_{0,\ \mu}$ and $\beta_{0,\ \sigma}$, specified with `class = 'Intercept'`), for the coefficients of the smooths (`class = 'b', coef = 'sx1_1'`), and for the standard deviations of the wiggliness parameters, which are treated as random effects (`class = 'sds'`):

```
priors <-
  c(
    # again, priors are excessively informative to reduce fitting time
    # intercepts
    prior(student_t(8, - 5, 0.1), class = 'Intercept'),
    prior(student_t(8, 1, 0.2), class = 'Intercept', dpar = 'sigma'),
    # coefficients
    prior(student_t(8, 0, 2), class = 'b', coef = 'sx1_1'),
    prior(student_t(8, 0, 2), class = 'b', coef = 'sx1_1', dpar = 'sigma'),
    # standard deviations of random effects for wiggly parameters, see ?mgcv::gamm
```

```
    prior(student_t(3, 1, 2.5), class = 'sds', lb = 0),
    prior(student_t(3, 1, 2), class = 'sds', lb = 0, dpar = 'sigma')
  )

# plot priors
layout(matrix(1:6, ncol = 2, byrow = TRUE))
plot(x * 0.1 - 5, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~beta['0,'~mu]),
     xlab = expression(beta['0,'~mu]), col = 'darkorange', lwd = 2)
plot(x * 0.2 + 1, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~beta['0,'~sigma]),
     xlab = expression(beta['0,'~sigma]), col = 'dodgerblue', lwd = 2)
plot(x * 2, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~beta['1,'~mu]),
     xlab = expression(beta['1,'~mu]), col = 'darkorange', lwd = 2)
plot(x * 2, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~beta['1,'~sigma]),
     xlab = expression(beta['1,'~sigma]), col = 'dodgerblue', lwd = 2)
plot(x * 2.5, dt(x, 3) * (x > 0), type = 'l',
     ylab = expression(Prior~'for'~EDF[mu]), xlab = expression(EDF[mu]),
     col = 'darkorange', lwd = 2)
plot(x * 2, dt(x, 3) * (x > 0), type = 'l',
     ylab = expression(Prior~'for'~EDF[sigma]), xlab = expression(EDF[sigma]),
     col = 'dodgerblue', lwd = 2)
```




```
layout(1)

# fit the model
m_brms <- brm(bf(y ~ s(x1), # smooth change in mu
                 sigma ~ s(x1), # smooth change in sigma
```

```r
                 alpha ~ 0, # no skew
                 family = skew_normal(link = 'identity', link_sigma = 'log')),
              data = d,
              prior = priors,
              chains = 4,
              warmup = 500,
              iter = 2500,
              cores = 4,
              control = list(adapt_delta = 0.95))
```

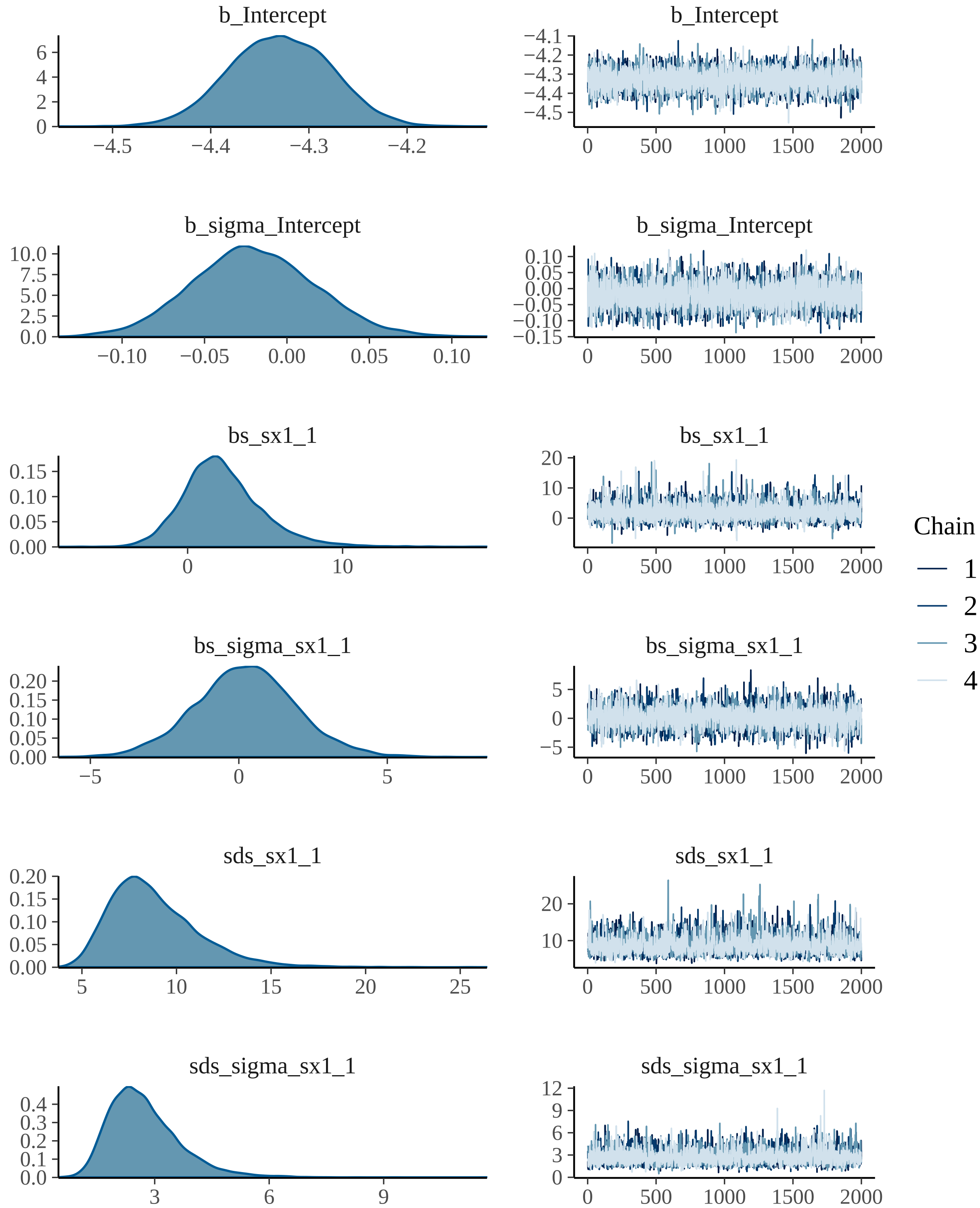
```
## Compiling Stan program...
```

```
## Start sampling
```

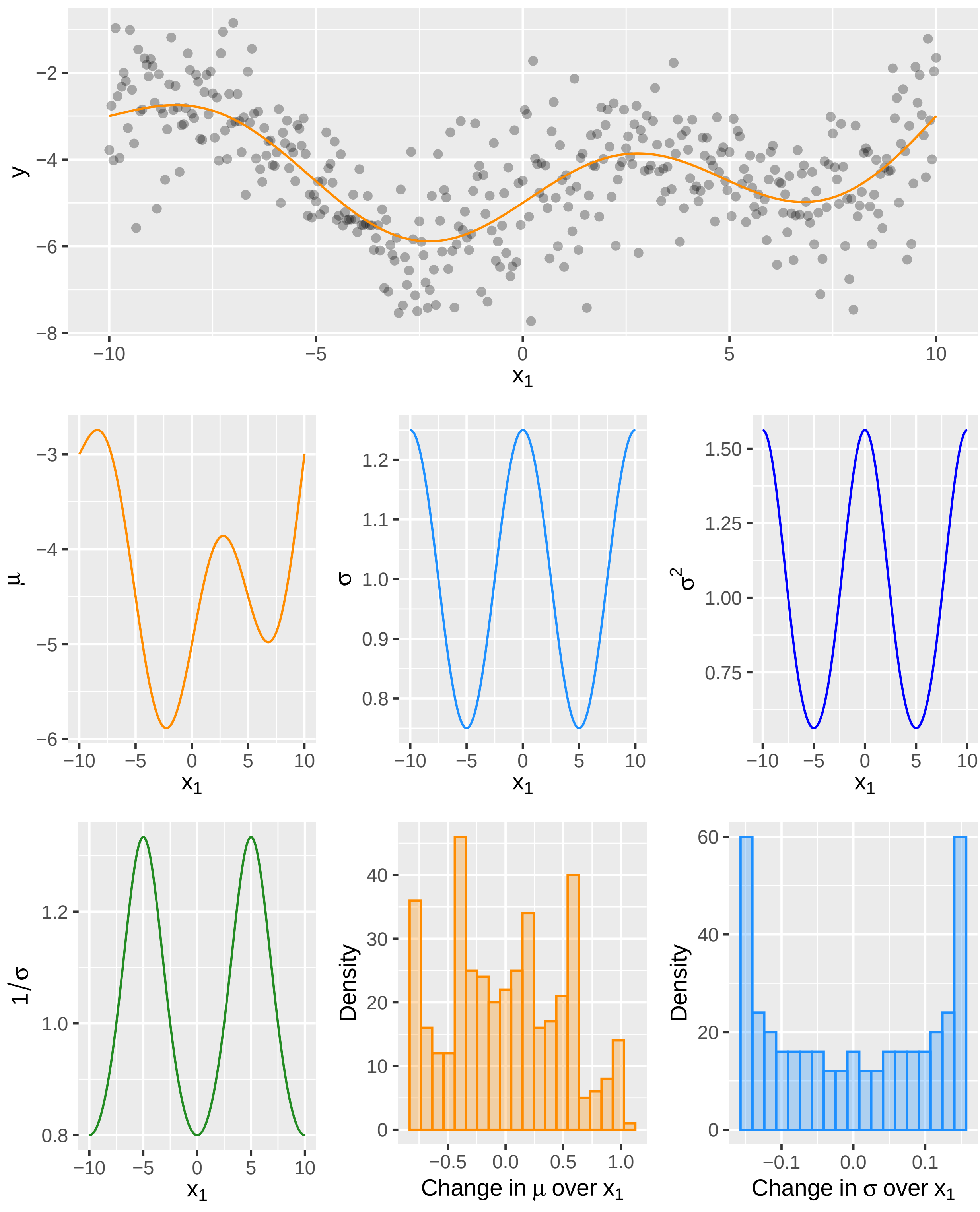
```
## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```r
plot(m_brms, N = 6) # plot posteriors for all parameters
```

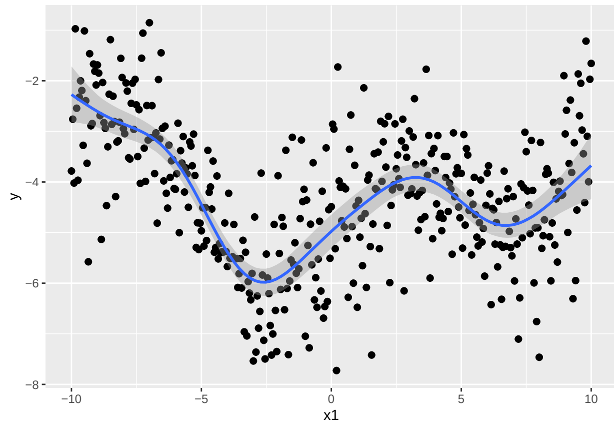



```
gaus_plots(d) # plot the parameters to compare to the posteriors
```

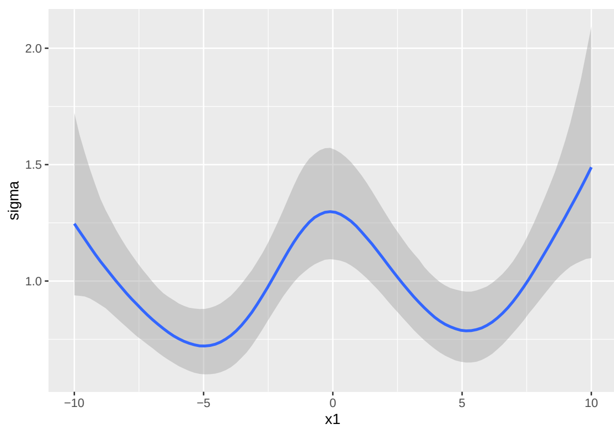


We can plot the parameters using `conditional_effects()`. For the trend in the mean, we can add the data using `plot()` and specifying `points = TRUE`:

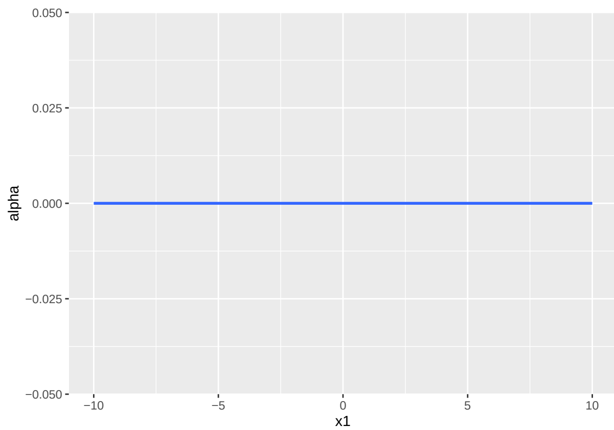
```
plot(conditional_effects(m_brms), points = TRUE)
```




```
conditional_effects(m_brms, dpar = 'sigma')
```




```
conditional_effects(m_brms, dpar = 'alpha')
```

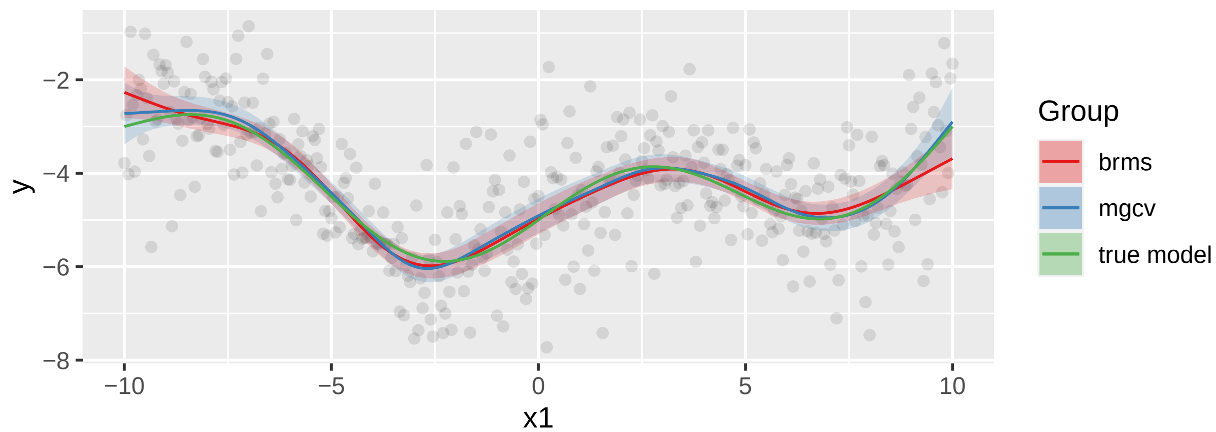


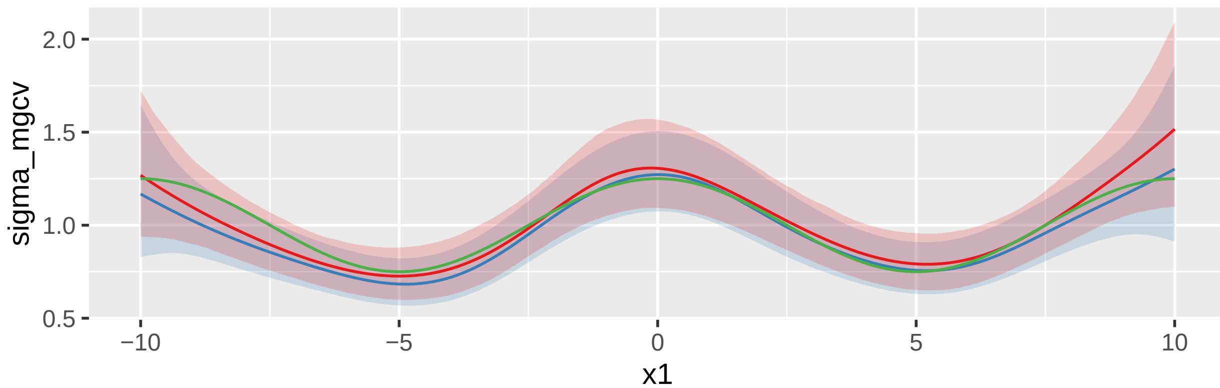
We can extract the estimated parameters for different values of the predictor(s) using `fitted()` (or equivalently `posterior_epred()`):

```
d <- bind_cols(d,
               fitted(m_brms) %>% # estimated mean
                 data.frame() %>%
```

```
                 transmute(mu_brms = Estimate, # mean from brms model w 95% CIs
                           mu_brms_lwr = Q2.5,
                           mu_brms_upr = Q97.5),
               fitted(m_brms, dpar = 'sigma') %>% # estimated SD from brms model
                 data.frame() %>%
                 transmute(sigma_brms = Estimate,
                           sigma_brms_lwr = Q2.5,
                           sigma_brms_upr = Q97.5))

plot_grid(ggplot(d) +
            geom_point(aes(x1, y), alpha = 0.1) +
            geom_ribbon(aes(x1, ymin = mu_mgcv_lwr, ymax = mu_mgcv_upr, fill = 'mgcv'),
                        alpha = 0.2) +
            geom_ribbon(aes(x1, ymin = mu_brms_lwr, ymax = mu_brms_upr,
                            fill = 'brms'), alpha = 0.2) +
            geom_line(aes(x1, mu_brms, color = 'brms')) +
            geom_line(aes(x1, mu_mgcv, color = 'mgcv')) +
            geom_line(aes(x1, mu, color = 'true model')) +
            scale_color_brewer('Group', type = 'qual', palette = 6,
                               aesthetics = c('color', 'fill')),
          ggplot(d) +
            geom_ribbon(aes(x1, ymin = sigma_mgcv_lwr, ymax = sigma_mgcv_upr,
                            fill = 'mgcv'), alpha = 0.2) +
            geom_ribbon(aes(x1, ymin = sigma_brms_lwr, ymax = sigma_brms_upr,
                            fill = 'brms'), alpha = 0.2) +
            geom_line(aes(x1, sigma_mgcv, color = 'mgcv')) +
            geom_line(aes(x1, sigma_brms, color = 'brms')) +
            geom_line(aes(x1, sigma, color = 'true model')) +
            scale_color_brewer('Group', type = 'qual', palette = 6,
                               aesthetics = c('color', 'fill')) +
            theme(legend.position = 'none'),
          ncol = 1, rel_heights = c(1.1, 1))
```





5 Fitting smooth distributional Beta models

The Beta distribution has support over the interval $(0, 1)$, so the mean and variance are not independent (since the a mean closer to 0 or 1 implies a smaller variance). Generally, the distribution is defined using parameters α and β, i.e., $Y \sim \mathrm{B}(\alpha, \beta)$. With this parameterization, the mean is

$$\mu = \frac{\alpha}{\alpha + \beta},$$

and the variance is

$$\sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}.$$

In `brms`, the distribution is defined in terms of the mean (μ, as defined above) and the scale parameter

$$\phi = \alpha + \beta$$

. With this parameterization, the variance becomes

$$\sigma^2 = \frac{\mu(1 - \mu)}{\phi + 1}.$$

We thus can write the following functions in `R` to extract the α and β parameters from μ and ϕ:

```
get_alpha <- function(.mu, .phi) return(.mu * .phi)
get_beta <- function(.mu, .phi) return((1 - .mu) * .phi)
```

The Beta distribution uses a logit (i.e. log of the odds) link function for μ and a log link function for ϕ. The log function and it's inverse function (the exponential) already exist in `R`, but the logit function and its inverse do not. We can then create them:

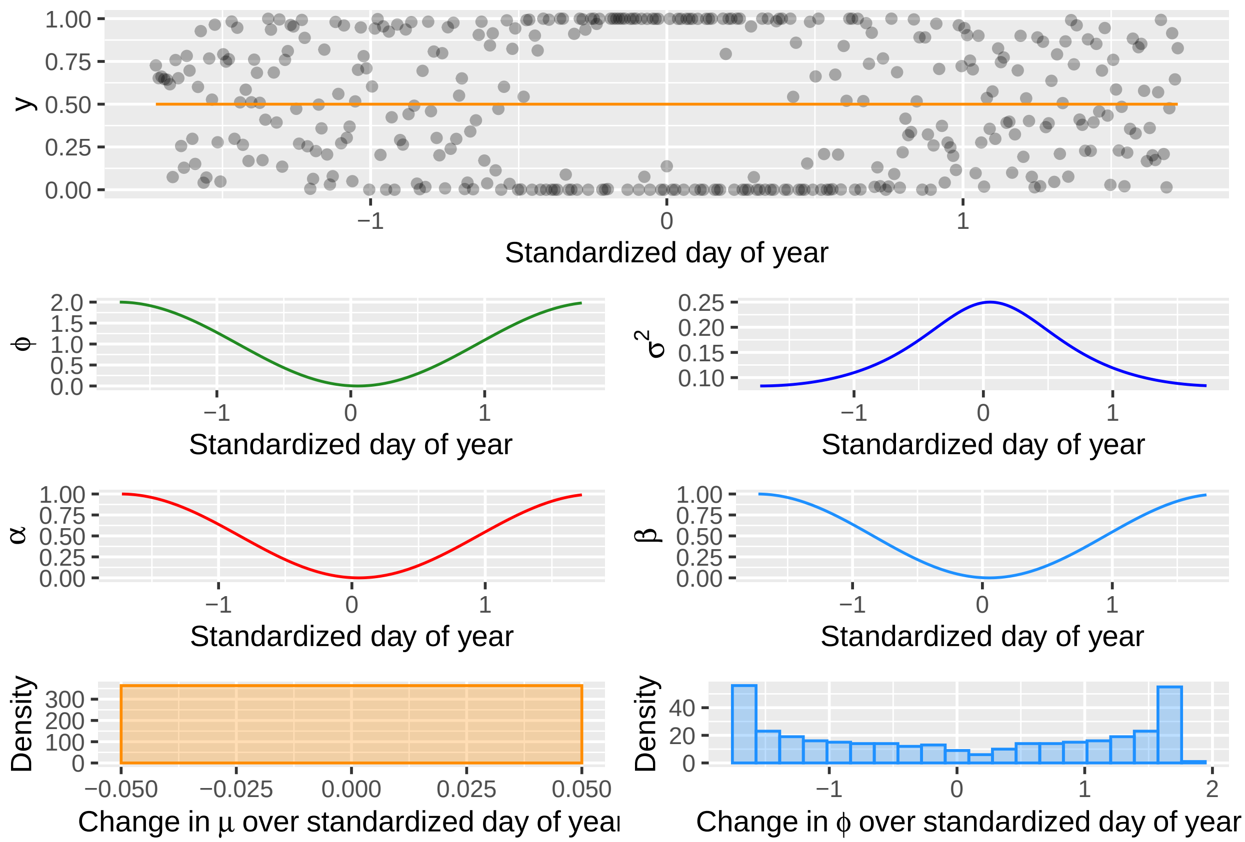
```r
# logit link function and inverse function
logit <- function(.y) log(.y / (1 - .y))
inv_logit <- function(eta) exp(eta) / (exp(eta) + 1)
```

We can also create a function to create all the plots like we did for the Gaussian distribution:

```r
# function to plot parameters and samples
beta_plots <- function(.data) {
  plot_grid(
    ggplot(.data) +
      geom_point(aes(doy_z, y), alpha = 0.3) +
      geom_line(aes(doy_z, mu), color = 'darkorange') +
      labs(x = 'Standardized day of year', expression(Y~'~'~B(mu,~phi))),
    plot_grid(ggplot(.data) +
                geom_line(aes(doy_z, phi), color = 'forestgreen') +
                labs(x = 'Standardized day of year', y = expression(phi)),
              ggplot(.data) +
                geom_line(aes(doy_z, sigma2), color = 'blue') +
                labs(x = 'Standardized day of year', y = expression(sigma^2)),
              ggplot(.data) +
                geom_line(aes(doy_z, alpha), color = 'red') +
                labs(x = 'Standardized day of year', y = expression(alpha)),
              ggplot(.data) +
                geom_line(aes(doy_z, beta), color = 'dodgerblue') +
                labs(x = 'Standardized day of year', y = expression(beta)),
              ggplot(.data) +
                geom_histogram(aes(coef_mu), fill = 'darkorange', color = 'darkorange',
                               alpha = 0.3, na.rm = TRUE, bins = 20) +
                labs(x = expression(Change~'in'~mu~over~standardized~day~of~year),
                     y = 'Density'),
              ggplot(.data) +
                geom_histogram(aes(coef_phi), fill = 'dodgerblue', color = 'dodgerblue',
                               alpha = 0.3, na.rm = TRUE, bins = 20) +
                labs(x = expression(Change~'in'~phi~over~standardized~day~of~year),
                     y = 'Density'),
              ncol = 2),
    ncol = 1, rel_heights = 1:2)
}
```

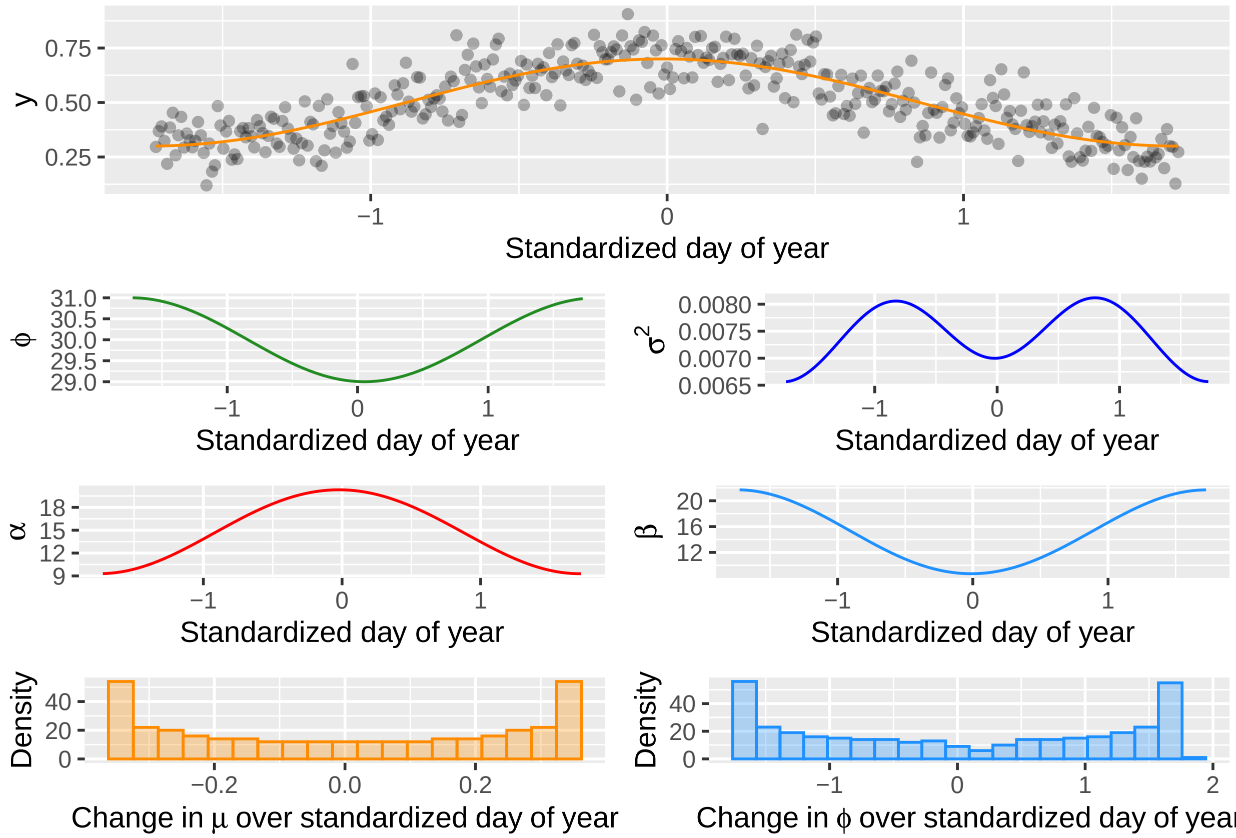
We can then create the dataset as if $Y \sim \mathrm{B}(\alpha, \beta)$ is a linear transformation of NDVI following the linear function $Y = \frac{K+1}{2}$. We start with a simple example where μ is constant and the variance peaks on day 188.

```r
tibble(doy = seq(1, 365, by = 1), # predictor variable
       doy_z = (doy - mean(doy)) / sd(doy), # standardized predictor variable
       mu = 0.5, # mean; within (0, 1)
       phi = cos(doy / 60) + 1, # precision parameter; within (0, Inf)
       alpha = get_alpha(mu, phi), # shape 1
       beta = get_beta(mu, phi), # shape 2
       sigma2 = mu * (1 - mu) / (phi + 1), # variance; within (0, 0.25)
       coef_mu = c(NA, diff(mu) / diff(doy_z)), # changes in mu over x1
       coef_phi = c(NA, diff(phi) / diff(doy_z)), # changes in sigma over x1
       y = rbeta(n = length(doy), shape1 = alpha, shape2 = beta)) %>% # samples
  mutate(y = if_else(y == 1, 0.999, y)) %>% # can't have y = 1
  beta_plots()
```



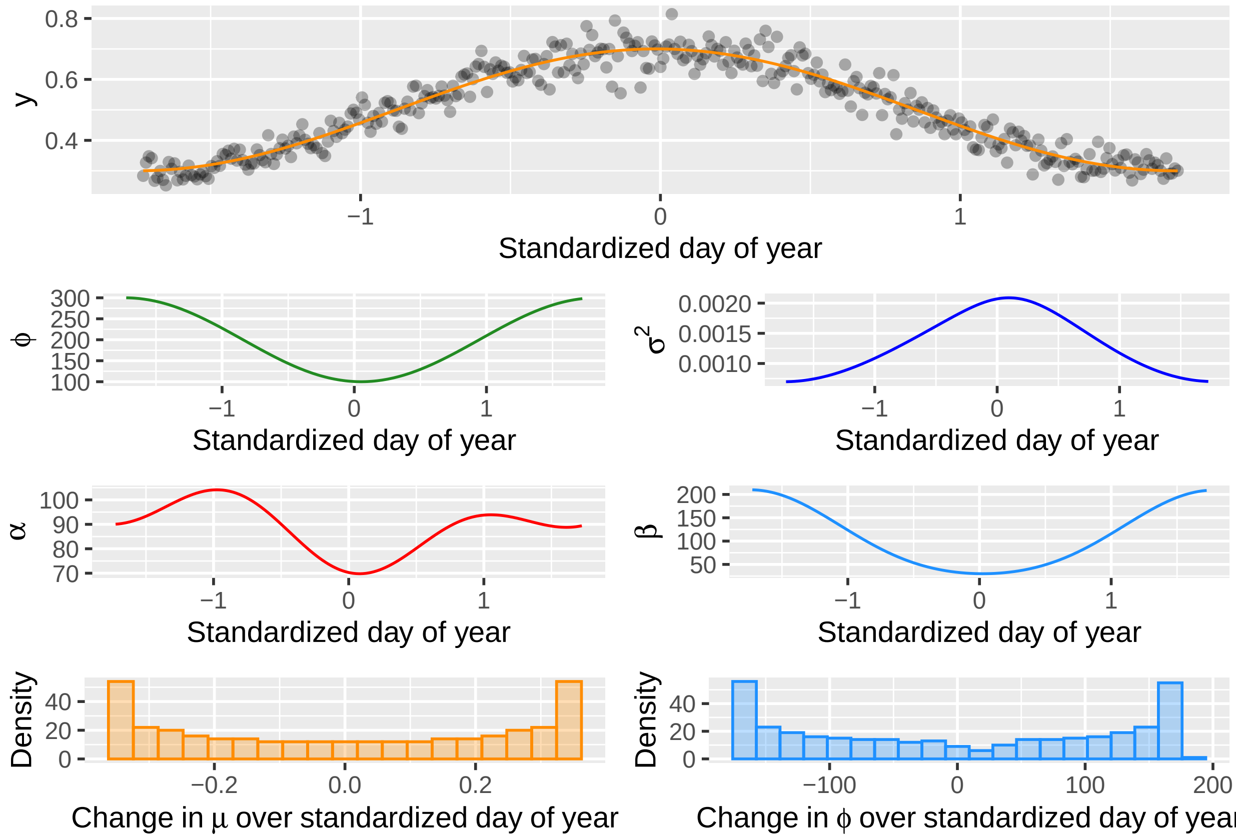
In this example, μ is highest on days 181 and 182. Note how σ^2 decreases as the mean (μ) approaches 0 or 1 because that implies σ^2 is approaching zero.

```
tibble(doy = seq(1, 365, by = 1), # predictor variable
       doy_z = (doy - mean(doy)) / sd(doy), # standardized predictor variable
       mu = sinpi((doy - 90) / 183) / 5 + 0.5, # mean; within (0, 1)
       phi = cos(doy / 60) + 30, # precision parameter; within (0, Inf)
       alpha = get_alpha(mu, phi), # shape 1
       beta = get_beta(mu, phi), # shape 2
       sigma2 = mu * (1 - mu) / (phi + 1), # variance; within (0, 0.25)
       coef_mu = c(NA, diff(mu) / diff(doy_z)), # changes in mu over x1
       coef_phi = c(NA, diff(phi) / diff(doy_z)), # changes in sigma over x1
       y = rbeta(n = length(doy), shape1 = alpha, shape2 = beta)) %>% # samples
  mutate(y = if_else(y == 1, 0.999, y)) %>% # can't have y = 1
  beta_plots()
```



Finally, in this example, μ peaks on day ~181.5 and ϕ is large and is lowest around day 188.5. We will be modeling the data from this last example. Since the days of year span a wide range of numbers, (0, 365), standardizing the predictors to Z scores will decrease the number of warm-up iterations and computation time.

```
d <-
  tibble(doy = seq(1, 365, by = 1), # predictor variable
         doy_z = (doy - mean(doy)) / sd(doy), # standardized predictor variable
         mu = sinpi((doy - 90) / 183) / 5 + 0.5, # mean; within (0, 1)
         phi = cos(doy / 60) * 100 + 200, # precision parameter; within (0, Inf)
         alpha = get_alpha(mu, phi), # shape 1
         beta = get_beta(mu, phi), # shape 2
         sigma2 = mu * (1 - mu) / (phi + 1), # variance; within (0, 0.25)
         coef_mu = c(NA, diff(mu) / diff(doy_z)), # changes in mu over x1
         coef_phi = c(NA, diff(phi) / diff(doy_z)), # changes in sigma over x1
         y = rbeta(n = length(doy), shape1 = alpha, shape2 = beta)) %>% # samples
  mutate(y = if_else(y == 1, 0.999, y)) # can't have y = 1
beta_plots(d)
```



Although mgcv does not support location-scale Beta models, we can fit a location Beta model to have some approximate results:

```
m_mgcv <- gam(y ~ s(doy_z, k = 10),
              family = betar(link = 'logit'),
              data = d,
              method = 'REML')
```

```
# check what priors can be defined by printing default ones
get_prior(formula = bf(y ~ s(doy_z),
                       phi ~ s(doy_z),
                       family = Beta(link = 'logit', link_phi = 'log')),
          data = d)
```

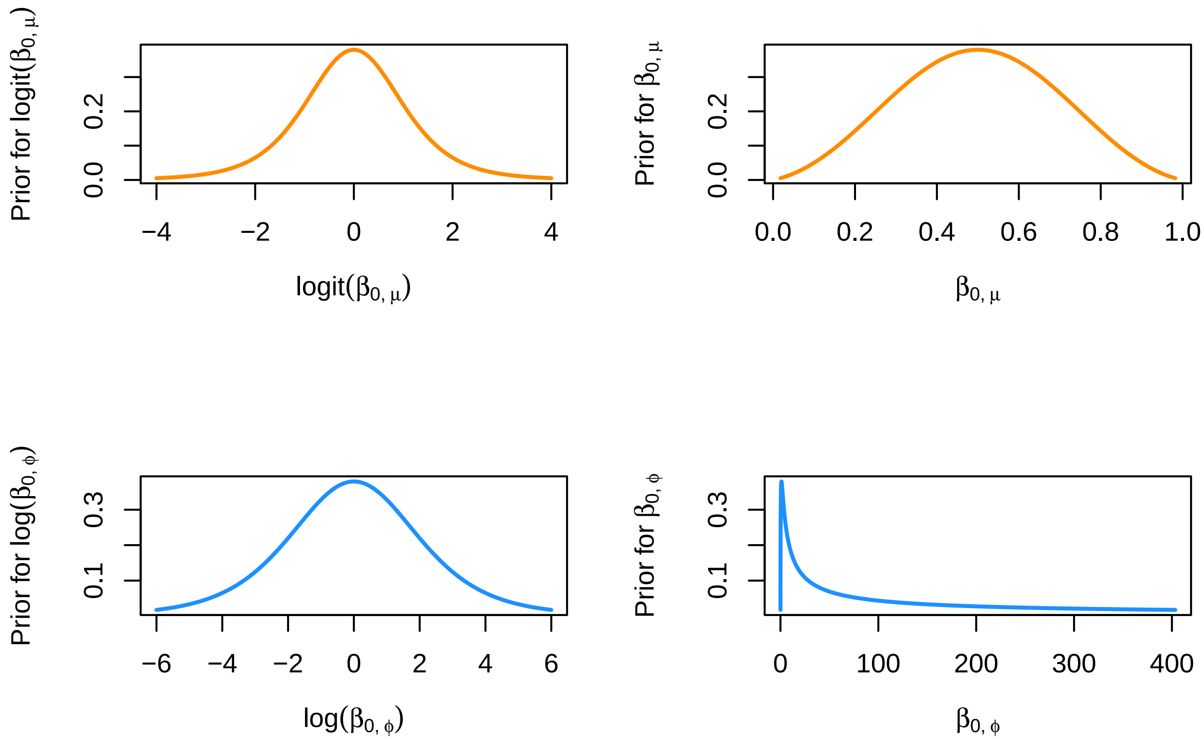
```
##                  prior     class      coef group resp dpar nlpar lb ub       source
##                 (flat)         b                                            default
##                 (flat)         b sdoy_z_1                              (vectorized)
##   student_t(3, 0, 2.5) Intercept                                            default
##   student_t(3, 0, 2.5)       sds                                0           default
##   student_t(3, 0, 2.5)       sds s(doy_z)                       0      (vectorized)
##                 (flat)         b                     phi                    default
##                 (flat)         b sdoy_z_1            phi               (vectorized)
##   student_t(3, 0, 2.5) Intercept                     phi                    default
##   student_t(3, 0, 2.5)       sds                     phi        0           default
##   student_t(3, 0, 2.5)       sds s(doy_z)            phi        0      (vectorized)
```

Based on "Introduction to Bayesian Statistics" by W.M. Bolstad & J.M. Curran (2017), we can find the equivalent sample sizes (n_{eq}) of each prior as $n_{eq} = \alpha + \beta + 1$ for a $\mathrm{B}(\alpha, \beta)$ prior and $n_{eq} \sim \mathrm{Pois}(\nu)$ for a $\Gamma(r, \nu)$ prior with shape r and rate ν.

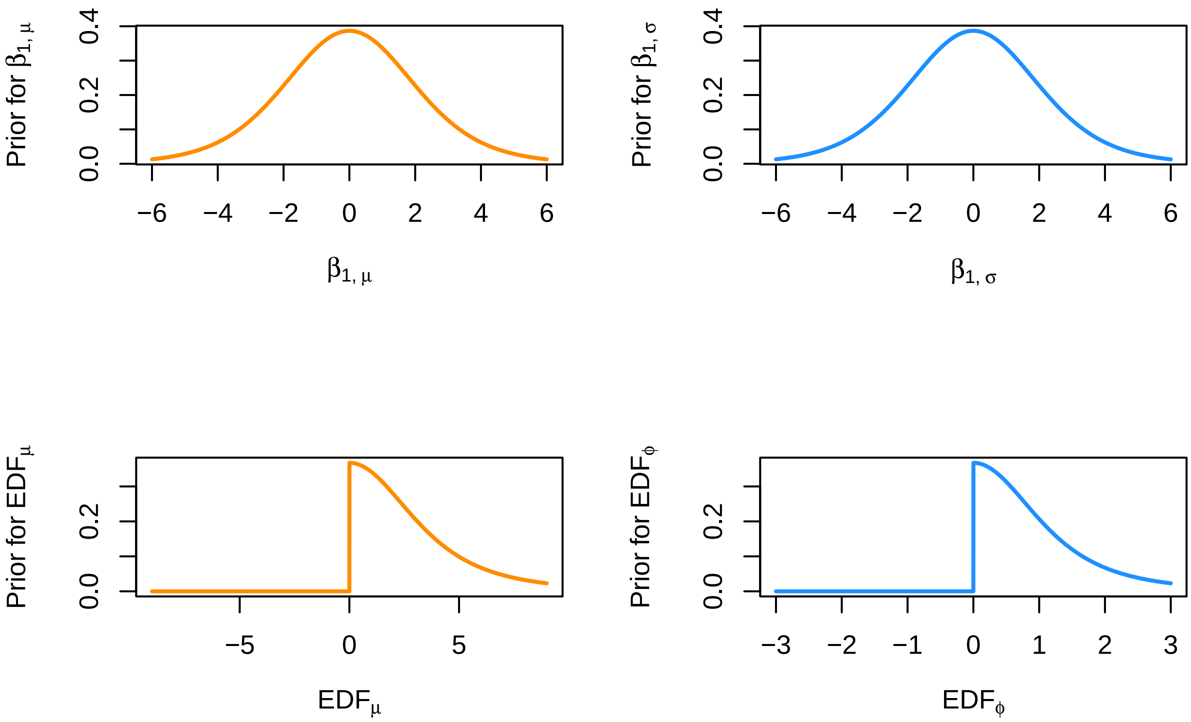
```r
priors <-
  c(
    # intercepts
    ## logit(mu) spans all reals, but mu is unlikely to be very near 0 or 1
    prior(student_t(5, 0, 1), class = 'Intercept'), # similar beta(3, 3) prior (n_eq = 7)
    ## log(phi) spans all reals; keep the prior for phi fairly wide
    prior(student_t(5, 0, 3), class = 'Intercept', dpar = 'phi'),
    # coefficients
    prior(student_t(8, 0, 2), class = 'b', coef = 'sdoy_z_1'),
    prior(student_t(8, 0, 2), class = 'b', coef = 'sdoy_z_1', dpar = 'phi'),
    # standard deviations of random effects for wiggly parameters, see ?mgcv::gamm
    prior(student_t(3, 1, 3), class = 'sds', lb = 0),
    prior(student_t(3, 1, 1), class = 'sds', lb = 0, dpar = 'phi')
  )
```

We can plot the priors to make sure they're reasonable:

```r
layout(matrix(1:4, ncol = 2, byrow = TRUE))
plot(seq(-4, 4, by = 0.01), dt(seq(-4, 4, by = 0.01), df = 5), type = 'l',
     ylab = expression(Prior~'for'~logit(beta['0,'~mu])),
     xlab = expression(logit(beta['0,'~mu])), col = 'darkorange', lwd = 2)
plot(inv_logit(seq(-4, 4, by = 0.01)), dt(seq(-4, 4, by = 0.01), 5), type = 'l',
     ylab = expression(Prior~'for'~beta['0,'~mu]),
     xlab = expression(beta['0,'~mu]), col = 'darkorange', lwd = 2)
plot(seq(-3, 3, by = 0.01) * 2, dt(seq(-3, 3, by = 0.01), df = 5), type = 'l',
     ylab = expression(Prior~'for'~log(beta['0,'~phi])),
     xlab = expression(log(beta['0,'~phi])), col = 'dodgerblue', lwd = 2)
plot(exp(seq(-3, 3, by = 0.01) * 2), dt(seq(-3, 3, by = 0.01), df = 5), type = 'l',
     ylab = expression(Prior~'for'~beta['0,'~phi]),
     xlab = expression(beta['0,'~phi]), col = 'dodgerblue', lwd = 2)
```




```
plot(x * 2, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~beta['1,'~mu]),
     xlab = expression(beta['1,'~mu]), col = 'darkorange', lwd = 2)
plot(x * 2, dt(x, 8), type = 'l', ylab = expression(Prior~'for'~beta['1,'~sigma]),
     xlab = expression(beta['1,'~sigma]), col = 'dodgerblue', lwd = 2)
plot(x * 3, dt(x, 3) * (x > 0), type = 'l',
     ylab = expression(Prior~'for'~EDF[mu]), xlab = expression(EDF[mu]),
     col = 'darkorange', lwd = 2)
plot(x, dt(x, 3) * (x > 0), type = 'l',
     ylab = expression(Prior~'for'~EDF[phi]), xlab = expression(EDF[phi]),
     col = 'dodgerblue', lwd = 2)
```




```
layout(1)
```

We fit the model using `doy_z` instead of `doy` to decrease the fitting time:

```
m_brms <- brm(bf(y ~ s(doy_z),
                 phi ~ s(doy_z),
                 family = Beta(link = 'logit', link_phi = 'log')),
              data = d,
              prior = priors,
              chains = 4,
              warmup = 500,
              iter = 2000,
              cores = 4)
```

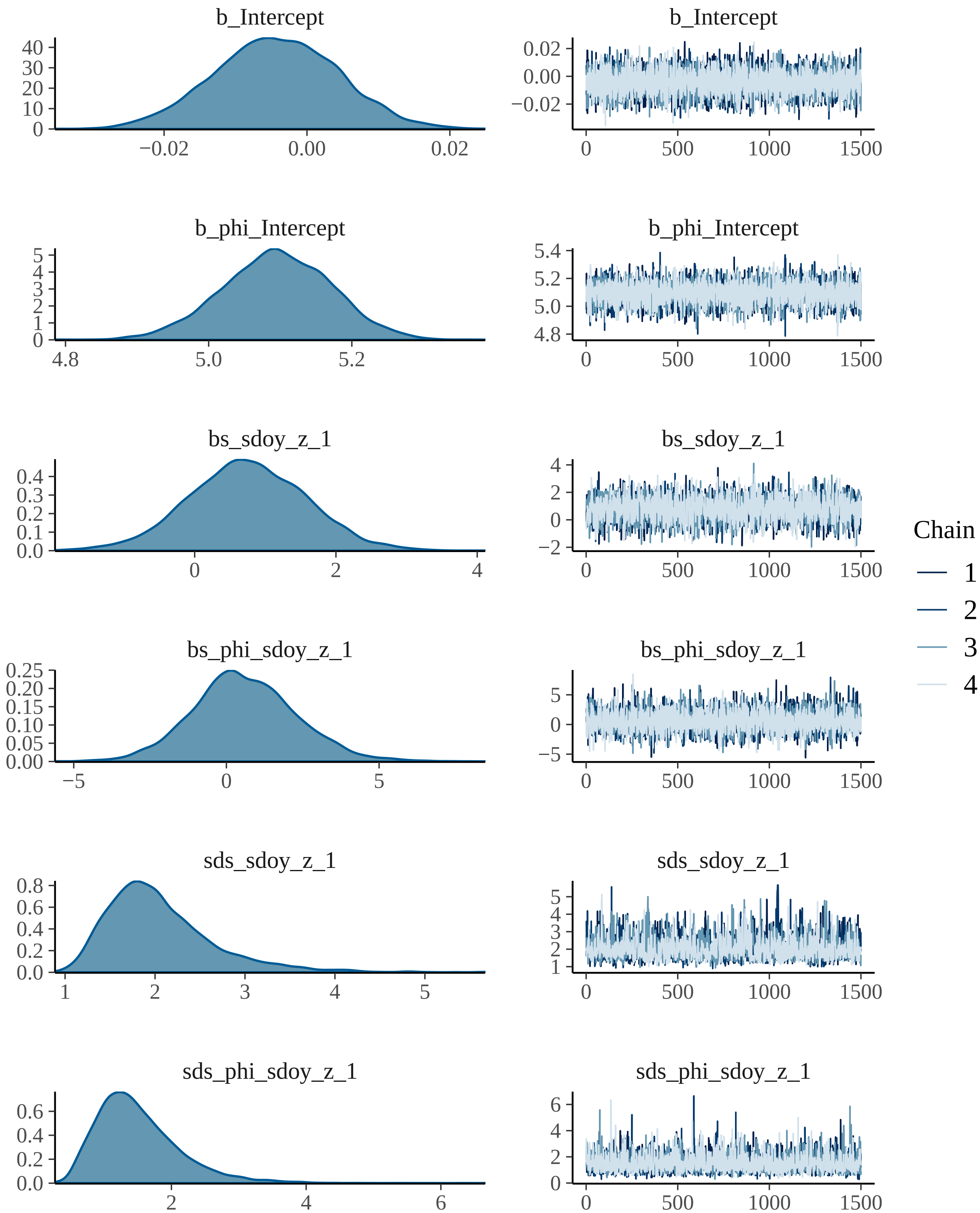
```
## Compiling Stan program...

## Start sampling

## Warning: There were 11 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems
```

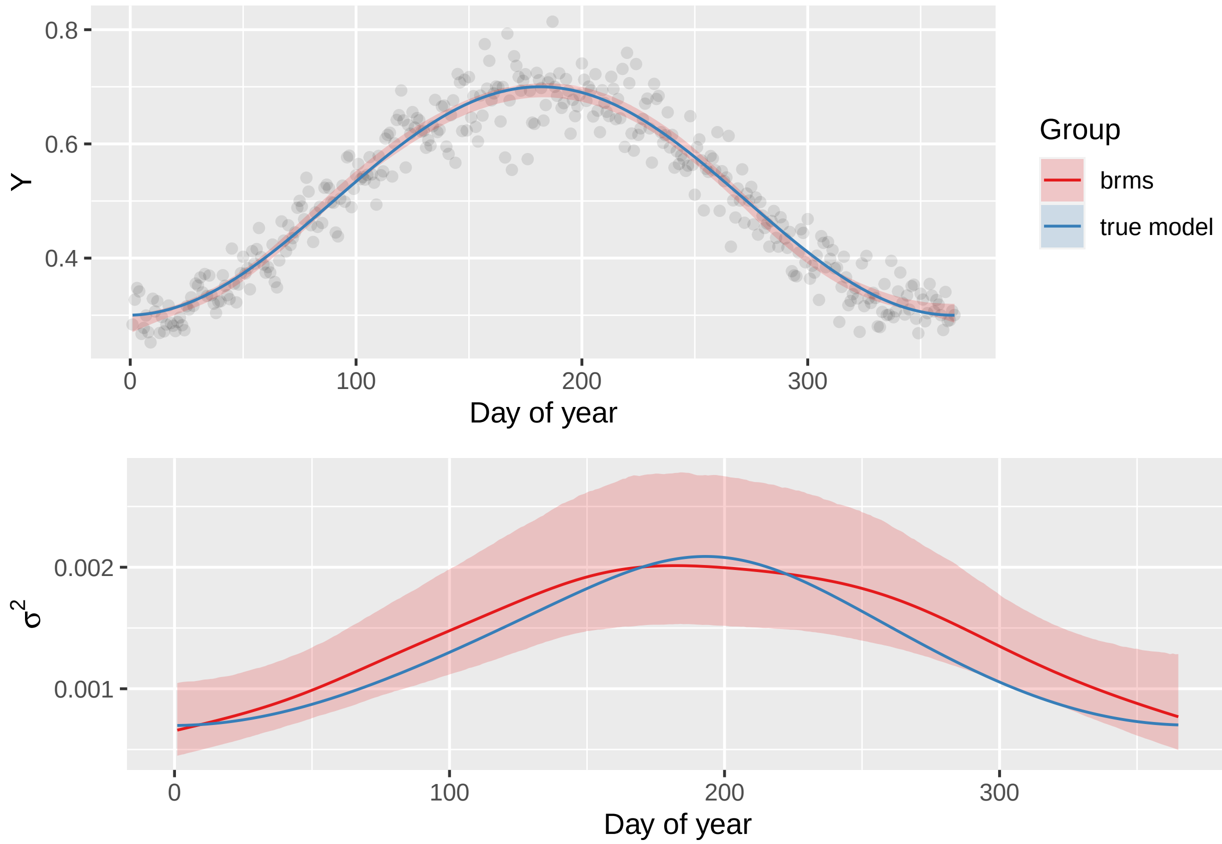
```r
plot(m_brms, N = 6) # plot posteriors for all 6 parameters
```



Finally, we can plot the predictions from the models:

```r
d <- bind_cols(d,
               fitted(m_brms) %>% # estimated mean
                 data.frame() %>%
                 transmute(mu_brms = Estimate, # mean from brms model w 95% CIs
                           mu_brms_lwr = Q2.5,
                           mu_brms_upr = Q97.5),
               fitted(m_brms, dpar = 'phi') %>% # estimated scale from brms model
                 data.frame() %>%
                 transmute(phi_brms = Estimate,
                           phi_brms_lwr = Q2.5,
                           phi_brms_upr = Q97.5)) %>%
  mutate(sigma2_brms = mu_brms * (1 - mu_brms) / (phi_brms + 1),
         sigma2_brms_lwr = mu_brms_lwr * (1 - mu_brms_lwr) / (phi_brms_lwr + 1),
         sigma2_brms_upr = mu_brms_upr * (1 - mu_brms_upr) / (phi_brms_upr + 1))

plot_grid(ggplot(d) +
            geom_point(aes(doy, y), alpha = 0.1) +
            geom_ribbon(aes(doy, ymin = mu_brms_lwr, ymax = mu_brms_upr,
                            fill = 'brms'), alpha = 0.2) +
            geom_line(aes(doy, mu, color = 'true model')) +
            scale_color_brewer('Group', type = 'qual', palette = 6,
                               aesthetics = c('color', 'fill')) +
            labs(x = 'Day of year', y = 'Y'),
          ggplot(d) +
            geom_ribbon(aes(doy, ymin = sigma2_brms_lwr, ymax = sigma2_brms_upr,
                            fill = 'brms'), alpha = 0.2) +
            geom_line(aes(doy, sigma2_brms, color = 'brms')) +
            geom_line(aes(doy, sigma2, color = 'true model')) +
            scale_color_brewer('Group', type = 'qual', palette = 6,
                               aesthetics = c('color', 'fill')) +
            theme(legend.position = 'none') +
            labs(x = 'Day of year', y = expression(sigma^2)),
          ncol = 1, rel_heights = c(1.1, 1))
```



0.8
0.6
0.4
Y
0
100
200
300
Day of year
Group
brms
true model
0.002
0.001
σ2
0
100
200
300
Day of year