# 2.5 Practical: introduction to brms

Benjamin Rosenbaum

November 3, 2021

The brms package is a frontend for Stan. It allows specifying models similar to "lm", "glm", "lmer" etc, and automatically translates them into Stan code. It has some additional functionalities like nonlinear models etc, but its possibilities in statistical modeling are not limitless and you might have to switch to Stan if your models get too complex.

In addition to automatic generation of Stan models, brms has some default priors for intercepts and standard deviation. The package also includes some neat functions for dealing with a fitted model, i.e. predictions, plotting, and model selection!

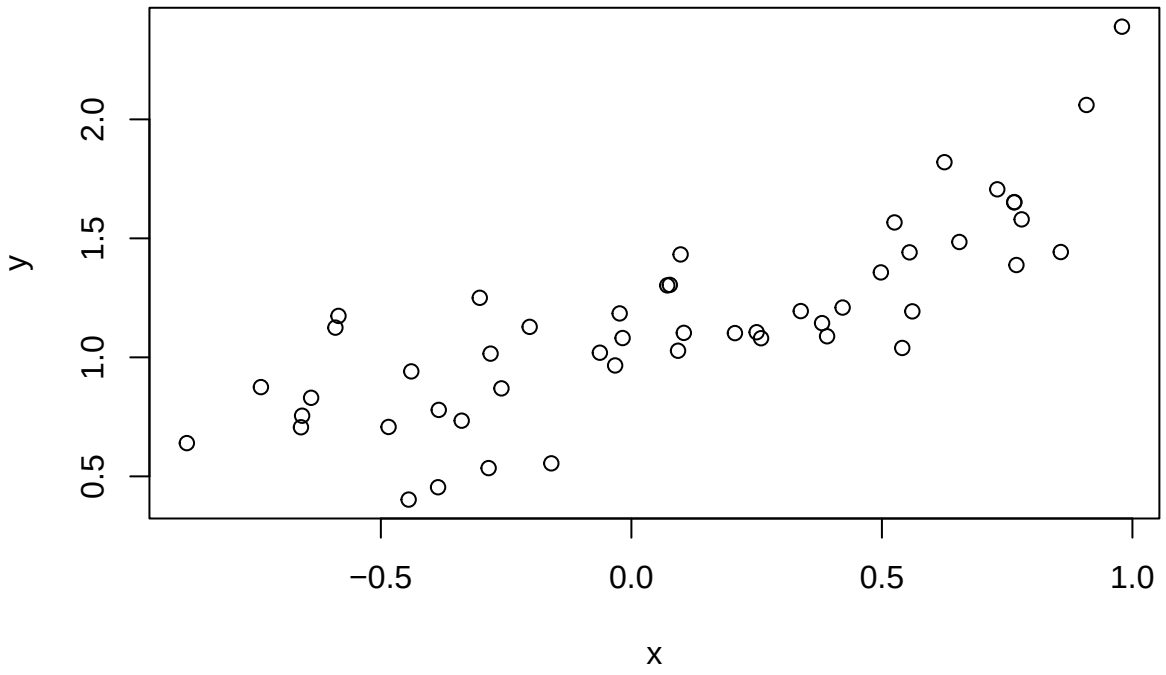
In this session, we will learn some standard brms functionalities with the example of linear regression from the last session

## Setup

```
rm(list=ls())
library(rstan)
library(coda)
library(BayesianTools)
library(brms)

rstan_options(auto_write = TRUE)
options(mc.cores = 4) # number of CPU cores
```

## Generate data

```
set.seed(100) # initiate random number generator for reproducibility

n=50

a=1.0
b=0.5
c=0.4
sigma=0.2

x = runif(n=n, min=-1, max=1)
y = rnorm(n=n, mean=a+b*x+c*x^2, sd=sigma)

df = data.frame(x=x,
                y=y)

plot(df)
```




## First brms fit

A standard linear regression using frequentist `lm()` can be fitted in 1 line, where `data=` is a data frame or a named list.

```
fit.l1 = lm(y ~ x, data=df)

summary(fit.l1)
```

```
##
## Call:
## lm(formula = y ~ x, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4427 -0.1505 -0.0146  0.1385  0.6781
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.09748    0.03456  31.757  < 2e-16 ***
## x            0.62692    0.06725   9.322 2.39e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2409 on 48 degrees of freedom
## Multiple R-squared:  0.6442, Adjusted R-squared:  0.6368
## F-statistic: 86.91 on 1 and 48 DF,  p-value: 2.392e-12
```

With the `brm()` function from the brms package, a Bayesian Stan model can be fitted with just 1 line, too. The deterministic model is defined by `y~x` and the default is normal distributed residuals. `brm()` chooses `chains=4` with `warmup=1000` and `iter=2000` by default. Other values can be provided just as with Stan.

```
fit.b1 = brm(y ~ x, data = df)
```

Variable names are generated automatically. `b_...` is for effect, here `b_Intercept` and `b_x` is slope (variable name was `x`). Residual standard deviation is always named `sigma`.

Convergence is checked by effective number of samples, here `ESS`, and `Rhat`.

```
fit.b1
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ x
##    Data: df (Number of observations: 50)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     1.10      0.03     1.03     1.16 1.00     3712     3062
## x             0.63      0.07     0.49     0.77 1.00     3298     2607
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.25      0.03     0.20     0.30 1.00     3400     3012
##
## 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).
```

But what about priors?

```
fit.b1$prior
```

```
##                   prior     class coef group resp dpar nlpar bound       source
##                  (flat)         b                                       default
##                  (flat)         b    x                             (vectorized)
##  student_t(3, 1.1, 2.5) Intercept                                       default
##    student_t(3, 0, 2.5)     sigma                                       default
```

Flat priors are used for effects (slopes), but priors for intercept and residual sd are generated from the data. *Technically* this is "double dipping" (i.e. using the data for prior *and* likelihood), but it just constrains intercept and sd broadly from going to values under which the range of the observations would be highly improbable.

## Fit with all priors specified

Priors can be specified for parameter classess (for which parameters they are used). `class=b` means for effects (`b` always means effect in brms), if there are multiple predictors, `coef=` speciefies predictor name. These are the same priors as used in our Stan model before.

```
priors = c(prior(normal(0,10), class=b, coef=x),
           prior(normal(0,10), class=Intercept),
           prior(normal(0,10), class=sigma)
)
```

```
fit.b2 = brm(y ~ x, data = df, prior = priors)
```

These priors affect the model fit only marginally, results are pretty much the same as `fit.b1`:

```
fit.b2
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ x
##    Data: df (Number of observations: 50)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     1.10      0.04     1.03     1.17 1.00     3324     2741
## x             0.63      0.07     0.48     0.76 1.00     3376     2532
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.25      0.03     0.20     0.30 1.00     3220     2611
##
## 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).
```

## Best practice

We specify priors for effects instead of brms flat priors. But no specification for intercept and sigma is provided, brms default is OK and convenient!

```
priors = c(prior(normal(0,10), class=b))  # coef not specified, for all effects
```

```
fit.b3 = brm(y ~ x, data = df, prior = priors)
```

```
fit.b3
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ x
##    Data: df (Number of observations: 50)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     1.10      0.04     1.03     1.17 1.00     3390     2632
## x             0.63      0.07     0.49     0.77 1.00     3540     2777
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.25      0.03     0.20     0.30 1.00     3587     2971
##
## 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).
```

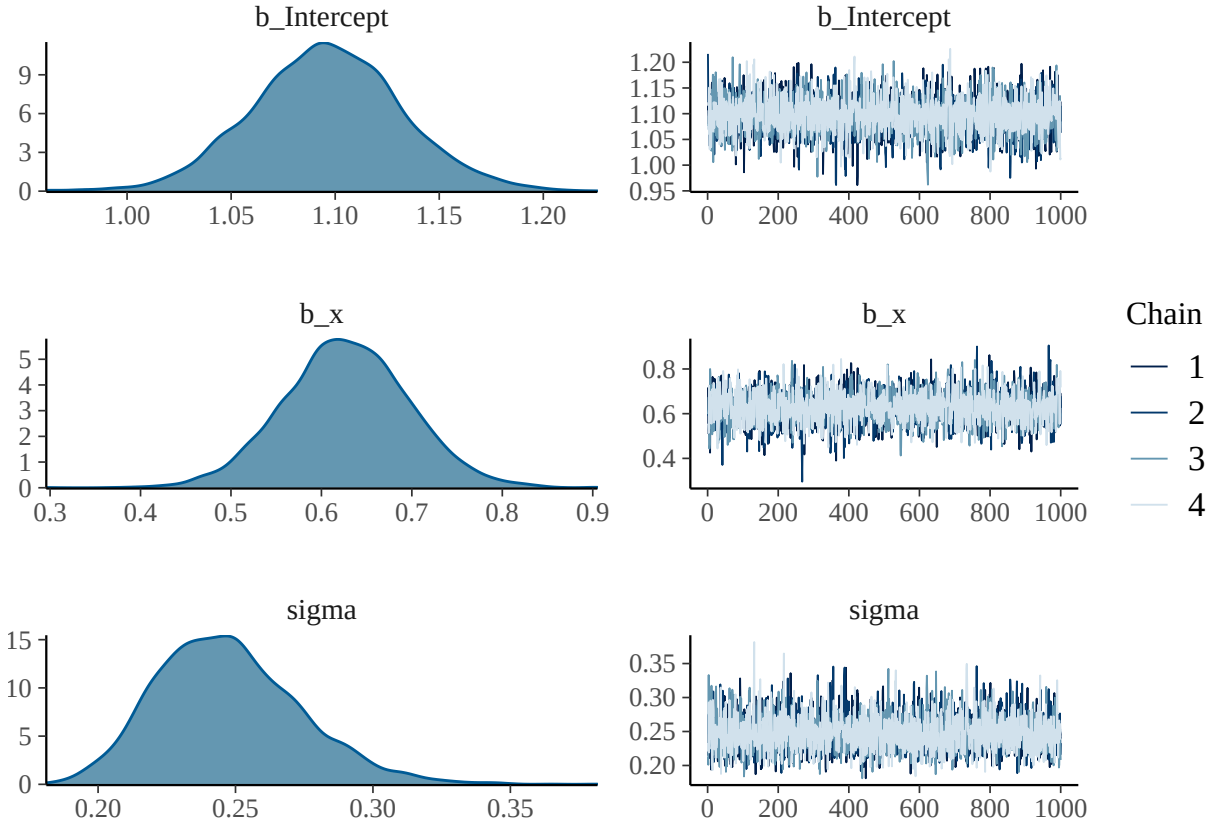
```
fit.b3$prior
```

```
##                   prior     class coef group resp dpar nlpar bound       source
##           normal(0, 10)         b                                         user
##           normal(0, 10)         b    x                               (vectorized)
##  student_t(3, 1.1, 2.5) Intercept                                     default
##    student_t(3, 0, 2.5)     sigma                                     default
```

## Explore the fit, posterior predictions etc

Posterior density and traceplots, brms design

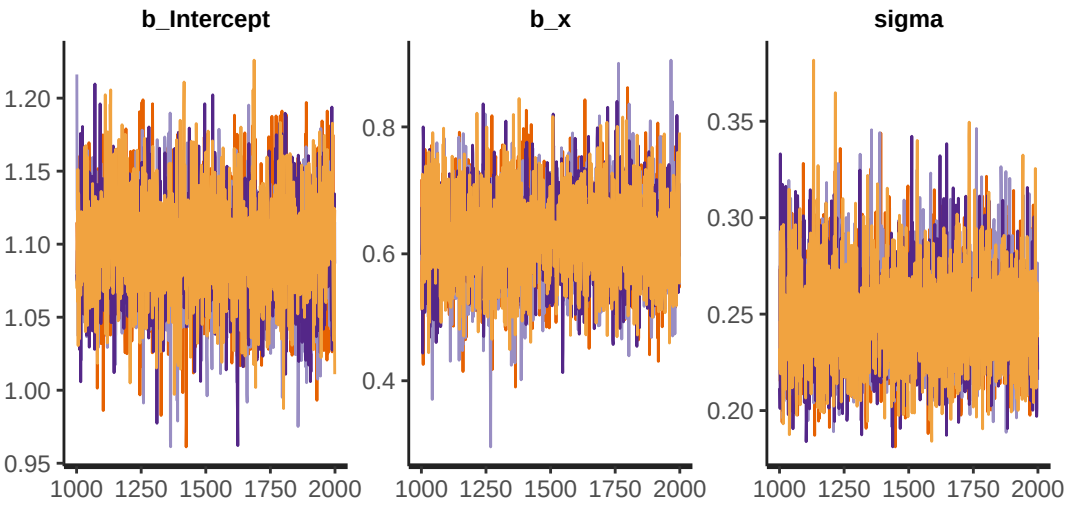
```
plot(fit.b3) # brms design
```



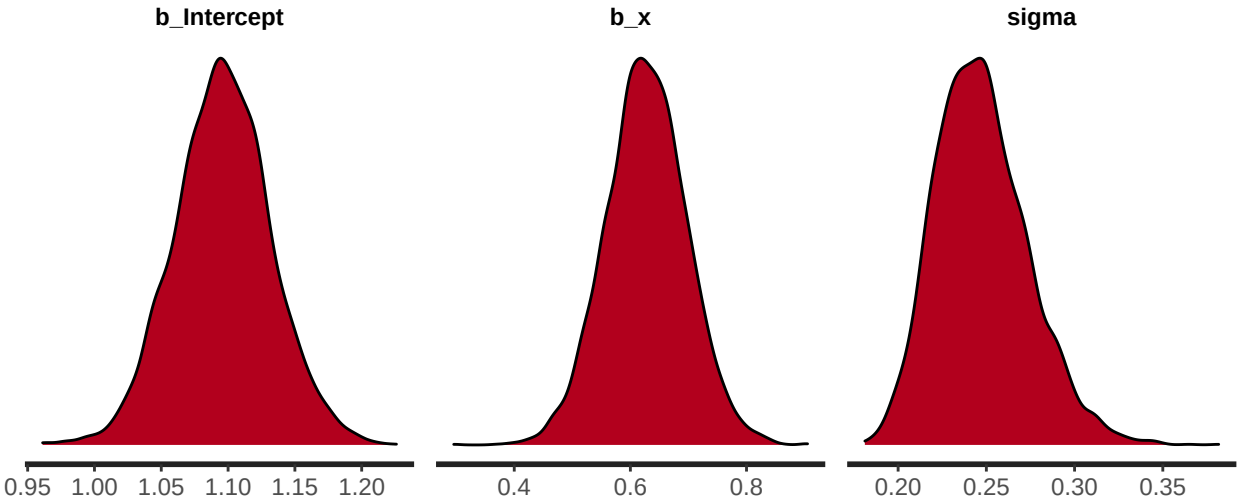

`fit.b3` is a brms object and contains a lot of things, among them the Stan fit `fit.b3$fit`

Posterior density and traceplots, Stan design

```
stan_trace(fit.b3$fit) # Stan design
```

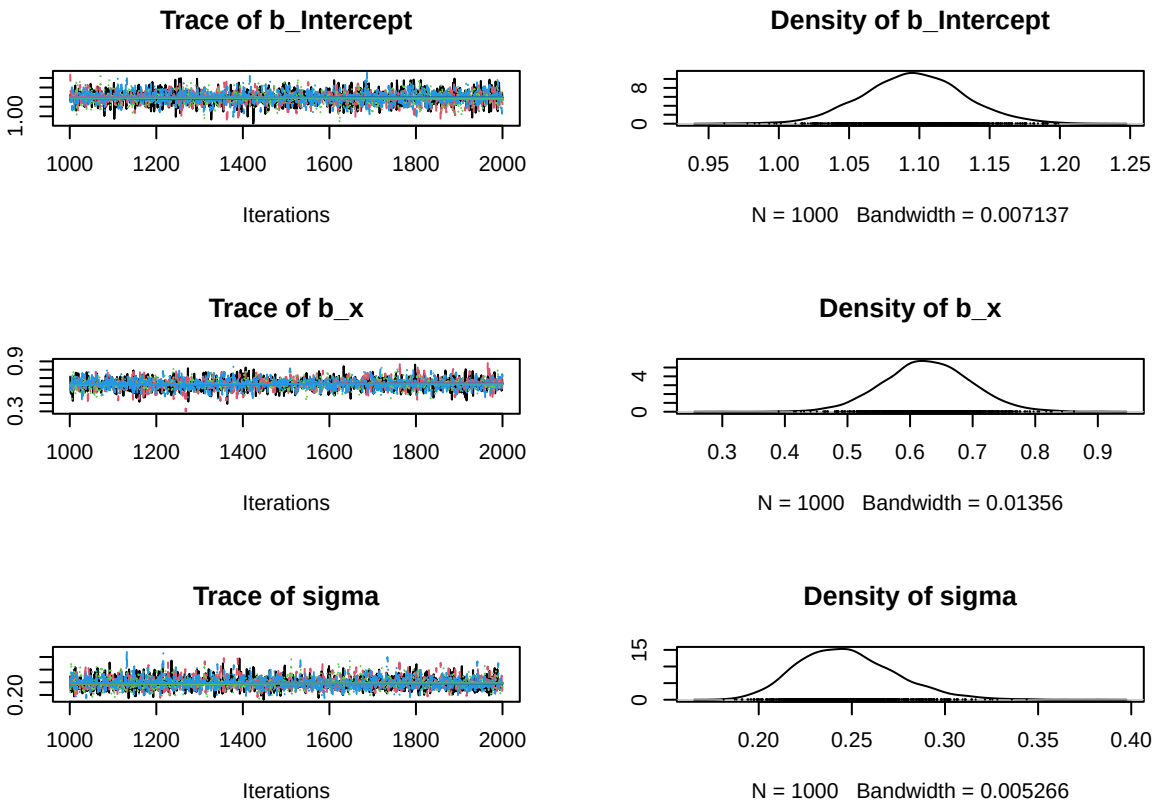



```
stan_dens(fit.b3$fit)
```



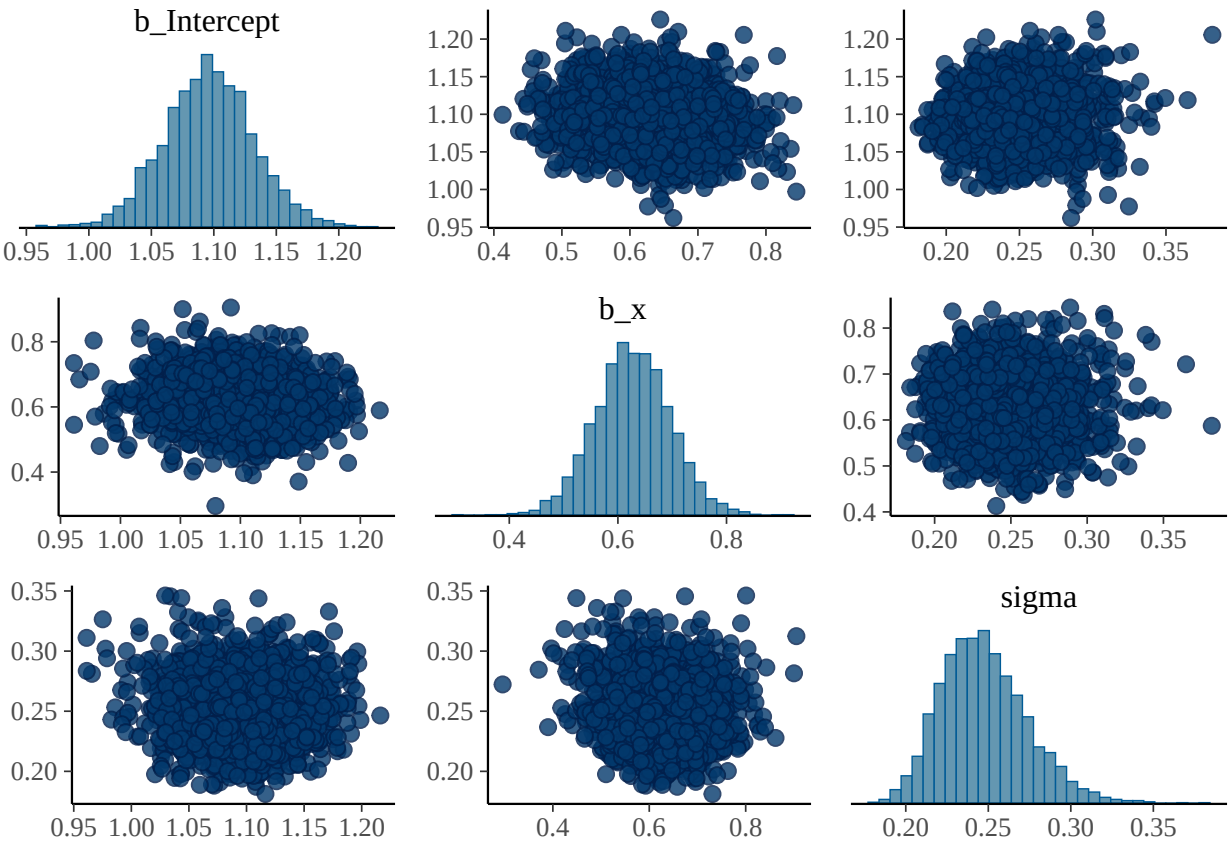

or use coda package

```
samples = As.mcmc.list(fit.b3$fit)
plot(samples[, 1:3]) # coda design
```

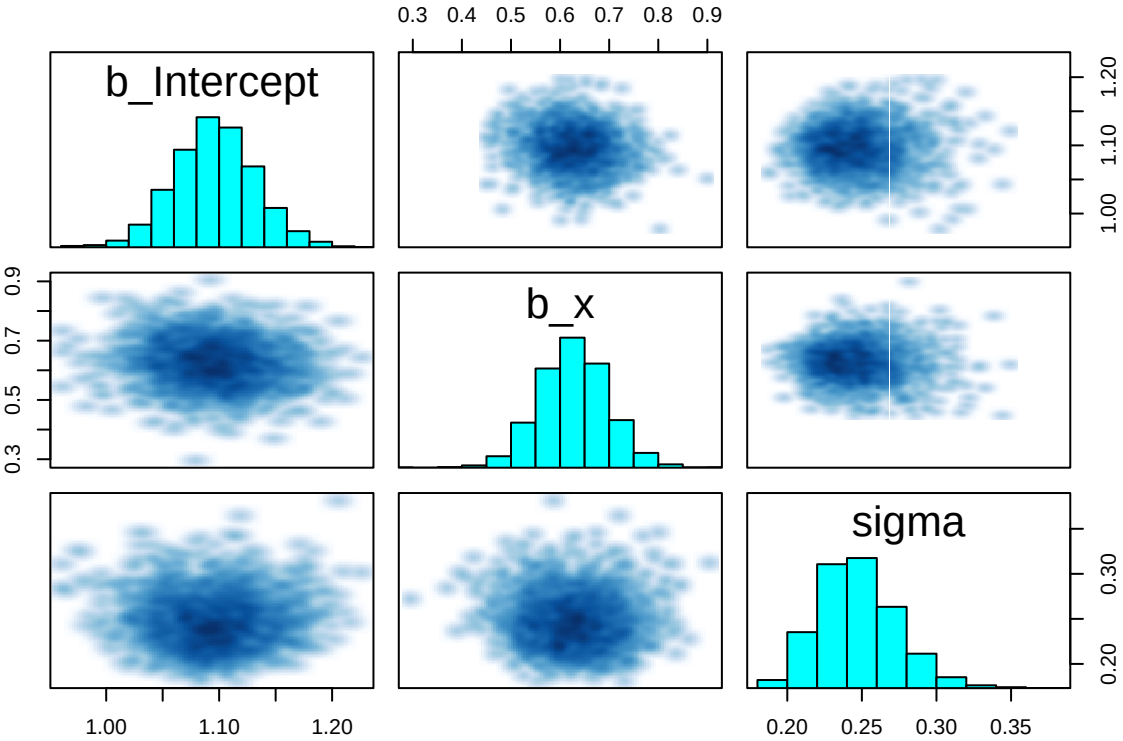



Pairs plots can be generated with different commands

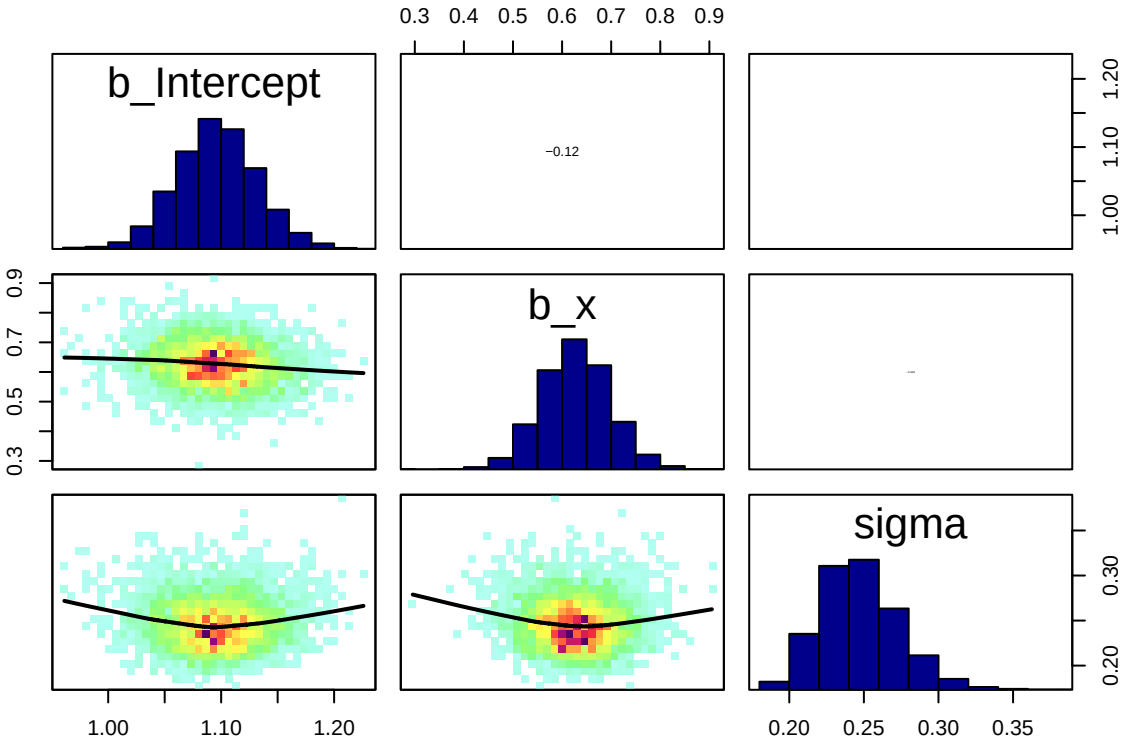
```
pairs(fit.b3) # brms design
```




```
pairs(fit.b3$fit, verInd=1:3, horInd=1:3) # Stan design
```



```
correlationPlot(as.matrix(fit.b3$fit)[, 1:3], thin=1) # BayesianTools design
```




We can also look at the Stan code which brms generated. It's heavily optimized, often not intuitively readable.

```
stancode(fit.b3)
```

```
## // generated with brms 2.16.1
## functions {
## }
## data {
##   int<lower=1> N;  // total number of observations
##   vector[N] Y;  // response variable
##   int<lower=1> K;  // number of population-level effects
##   matrix[N, K] X;  // population-level design matrix
##   int prior_only;  // should the likelihood be ignored?
## }
## transformed data {
##   int Kc = K - 1;
##   matrix[N, Kc] Xc;  // centered version of X without an intercept
##   vector[Kc] means_X;  // column means of X before centering
##   for (i in 2:K) {
##     means_X[i - 1] = mean(X[, i]);
##     Xc[, i - 1] = X[, i] - means_X[i - 1];
##   }
## }
## parameters {
##   vector[Kc] b;  // population-level effects
##   real Intercept;  // temporary intercept for centered predictors
##   real<lower=0> sigma;  // dispersion parameter
## }
## transformed parameters {
```

```
## }
## model {
##   // likelihood including constants
##   if (!prior_only) {
##     target += normal_id_glm_lpdf(Y | Xc, Intercept, b, sigma);
##   }
##   // priors including constants
##   target += normal_lpdf(b | 0, 10);
##   target += student_t_lpdf(Intercept | 3, 1.1, 2.5);
##   target += student_t_lpdf(sigma | 3, 0, 2.5)
##     - 1 * student_t_lccdf(0 | 3, 0, 2.5);
## }
## generated quantities {
##   // actual population-level intercept
##   real b_Intercept = Intercept - dot_product(means_X, b);
## }
```

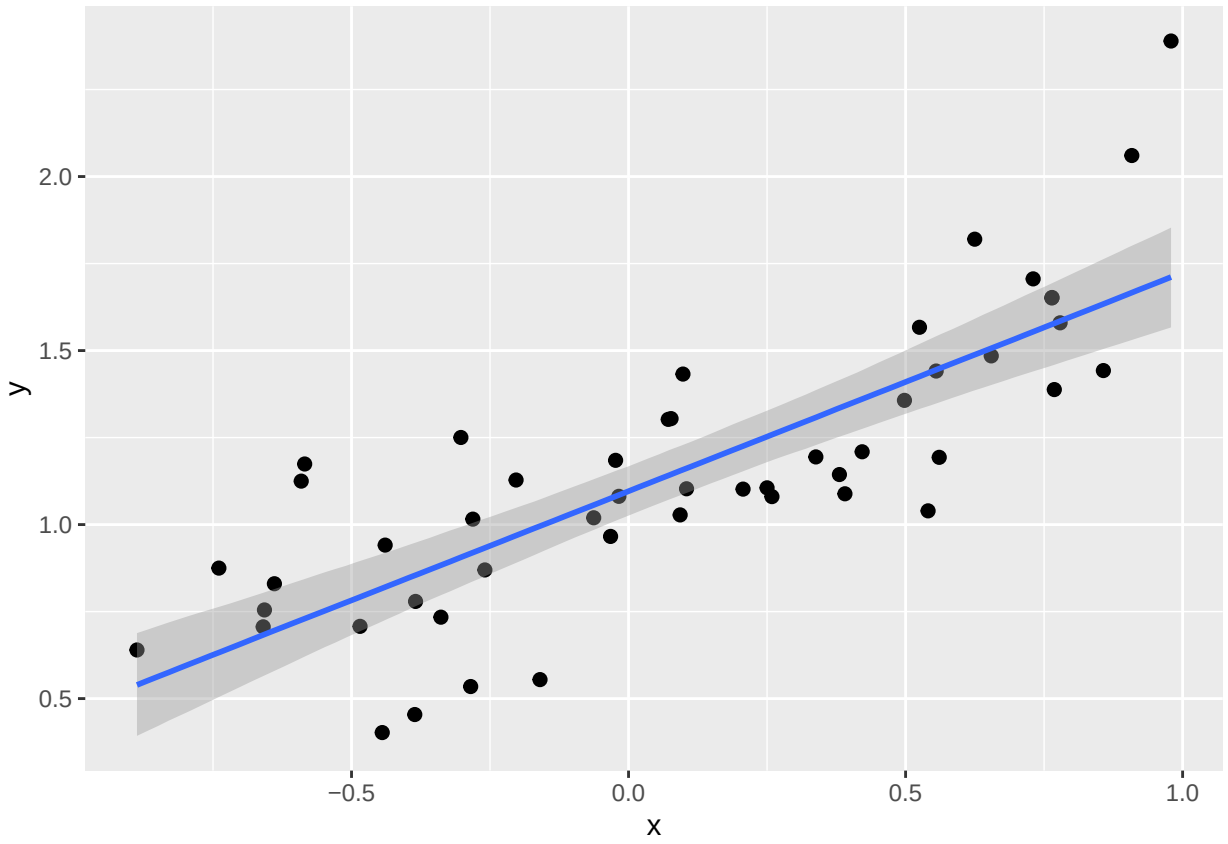
There is a brms function for one-sided hypothesis testing. E.g. test if slope `b_x` is positive. `Post.Prob` is the posterior probability $P(b\_x > 0)$.

```
hypothesis(fit.b3, "x>0", class="b")
```

```
## Hypothesis Tests for class b:
##   Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio Post.Prob Star
## 1    (x) > 0     0.63      0.07     0.51     0.74        Inf         1    *
## ---
## 'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
## '*': For one-sided hypotheses, the posterior probability exceeds 95%;
## for two-sided hypotheses, the value tested against lies outside the 95%-CI.
## Posterior probabilities of point hypotheses assume equal prior probabilities.
```

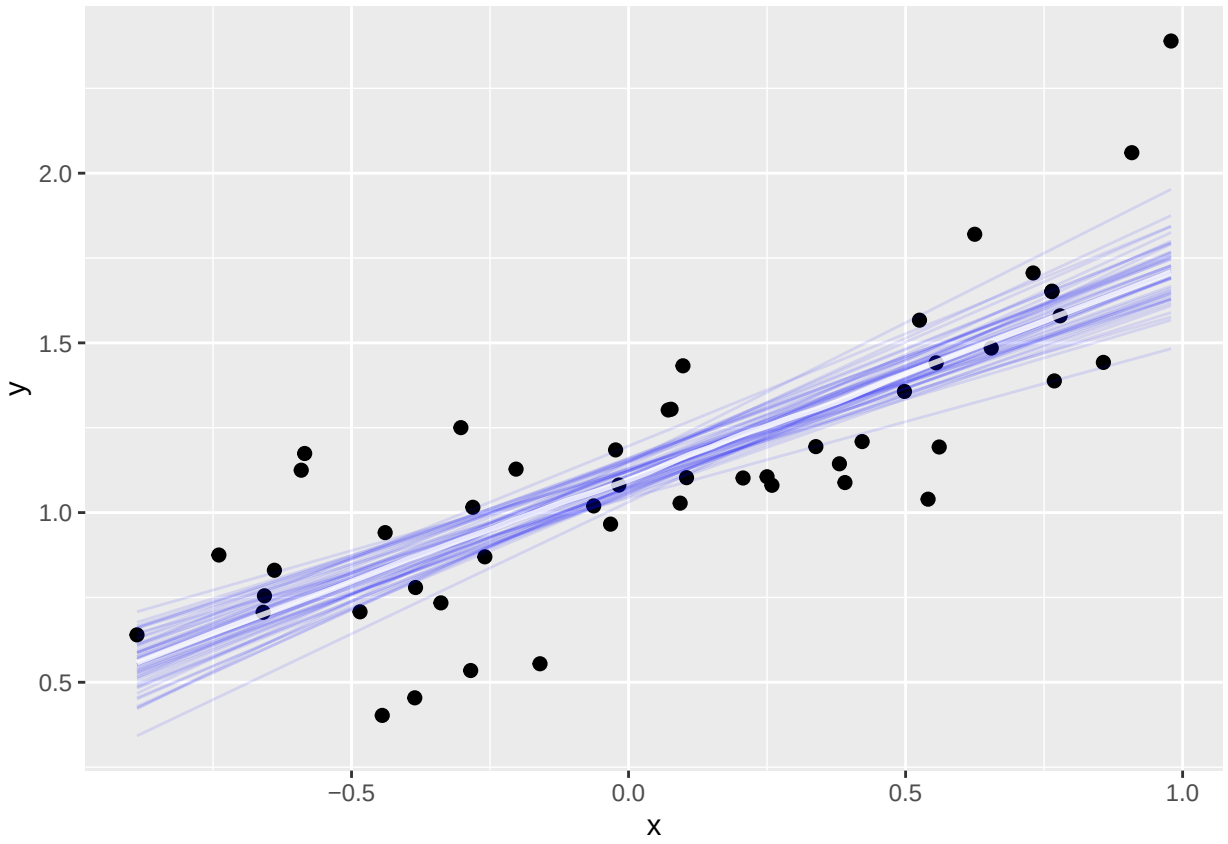
When plotting posterior predictions, `conditional_effects` is a convenient tool to visualize (fixed) effects of all predictors. Here there is only one (x), so credible / confidence intervals of fitted **deterministic model** `b_Intercept + b_x * x` are plotted.

```
plot(conditional_effects(fit.b3),
     points=TRUE)
```



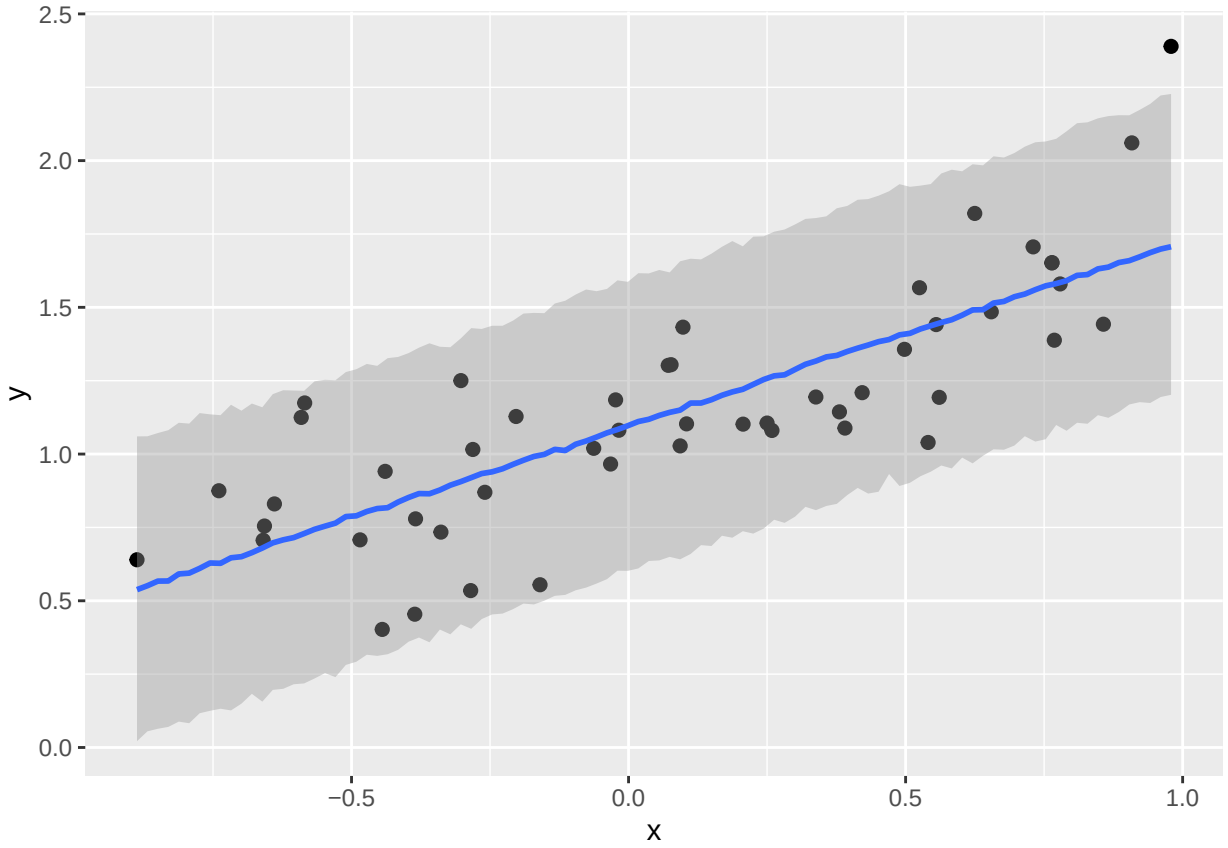

Single lines, each representing one possible regression line from one posterior sample:

```
plot(conditional_effects(fit.b3,
                         spaghetti=TRUE,
                         ndraws=50),
     points=TRUE
)
```




Plot prediction intervals: includes **stochastic** part of the model, too (`sigma`)

```
plot(conditional_effects(fit.b3,
                         method = "posterior_predict"),
     points=TRUE)
```




We compute credible intervals with `fitted()` (deterministic model) or prediction intervals with `predict()` (whole statistical model including stochastic part). If no `newdata=` is specified, original data is used for predictors. 95% intervals are default, but can be changed with `probs=c( , )`.

```
cred.fit.b3 = fitted(fit.b3)
head(cred.fit.b3)
```

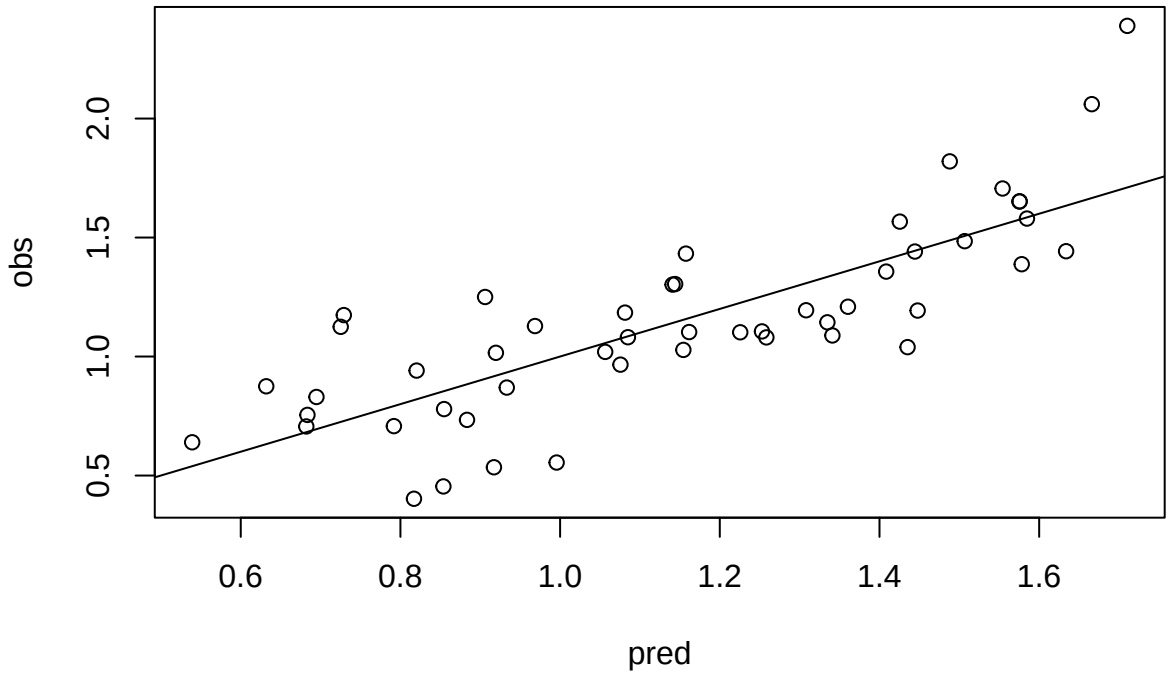
```
##       Estimate  Est.Error      Q2.5     Q97.5
## [1,] 0.8546696 0.04741393 0.7653988 0.9483279
## [2,] 0.7917986 0.05226041 0.6933601 0.8955124
## [3,] 1.1616049 0.03580822 1.0908210 1.2324466
## [4,] 0.5391664 0.07516701 0.3929751 0.6882403
## [5,] 1.0564637 0.03678038 0.9841965 1.1302325
## [6,] 1.0755677 0.03632919 1.0051693 1.1485164
```

```
pred.fit.b3 = predict(fit.b3)
head(pred.fit.b3)
```

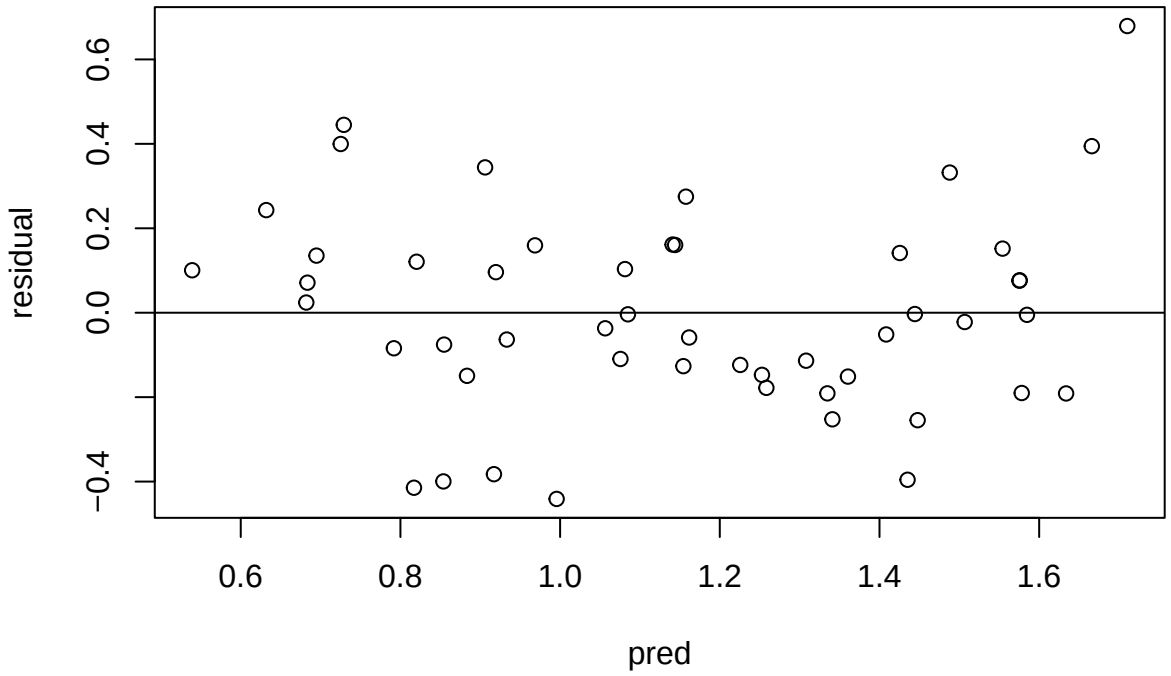
```
##       Estimate Est.Error       Q2.5    Q97.5
## [1,] 0.8487738 0.2528083 0.35562549 1.352343
## [2,] 0.7989115 0.2557686 0.30902383 1.306990
## [3,] 1.1648114 0.2498452 0.66893607 1.655400
## [4,] 0.5434871 0.2640519 0.02733332 1.061763
## [5,] 1.0546880 0.2511655 0.55571652 1.559798
## [6,] 1.0775320 0.2496146 0.58727584 1.574551
```

These can be used for “observed vs predicted” or “residual vs predicted” plots.

```
plot(cred.fit.b3[, 1], df$y, xlab="pred", ylab="obs")
abline(0,1)
```

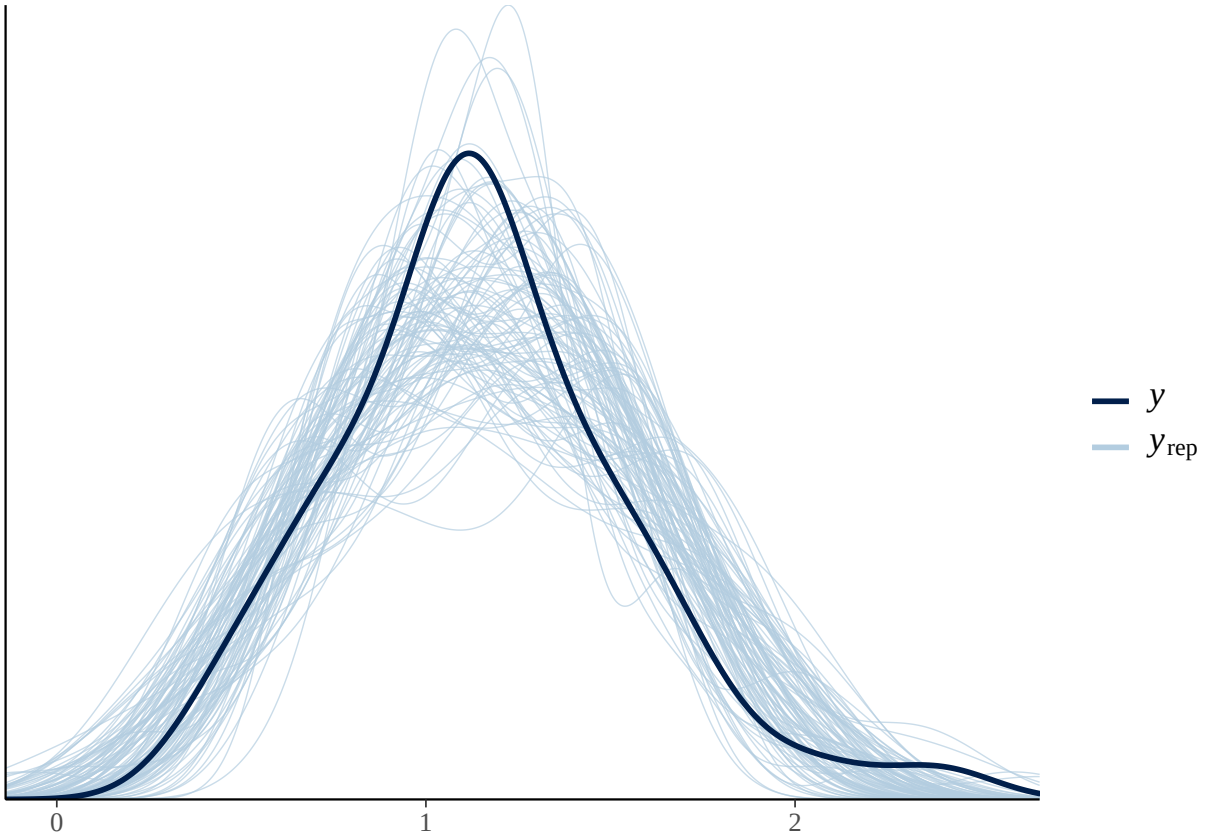



```
plot(cred.fit.b3[, 1], df$y-cred.fit.b3[, 1], xlab="pred", ylab="residual")
abline(0,0)
```




Another neat tool is `pp_check()` (posterior predictive check). It draws some figure for observed data (e.g. density or histogram) together with the figures from data predicted from the model for multiple draws from the posterior sample distribution. They should overlay for a good fit. Alternative tools can be selected by `type=`.

```
pp_check(fit.b3, ndraws=100)
```




## Quadratic regression and model comparison

As we have seen in the last session, there might be a U-shaped relationship of y with x. For including a quadratic term, `I(x^2)` is used as additional predictor (just as would be with `lm()`)

```
priors = c(prior(normal(0,10), class=b))  # coef not specified, for all effects
```

```
fit.b4 = brm(y ~ x + I(x^2), data = df, prior = priors)
```

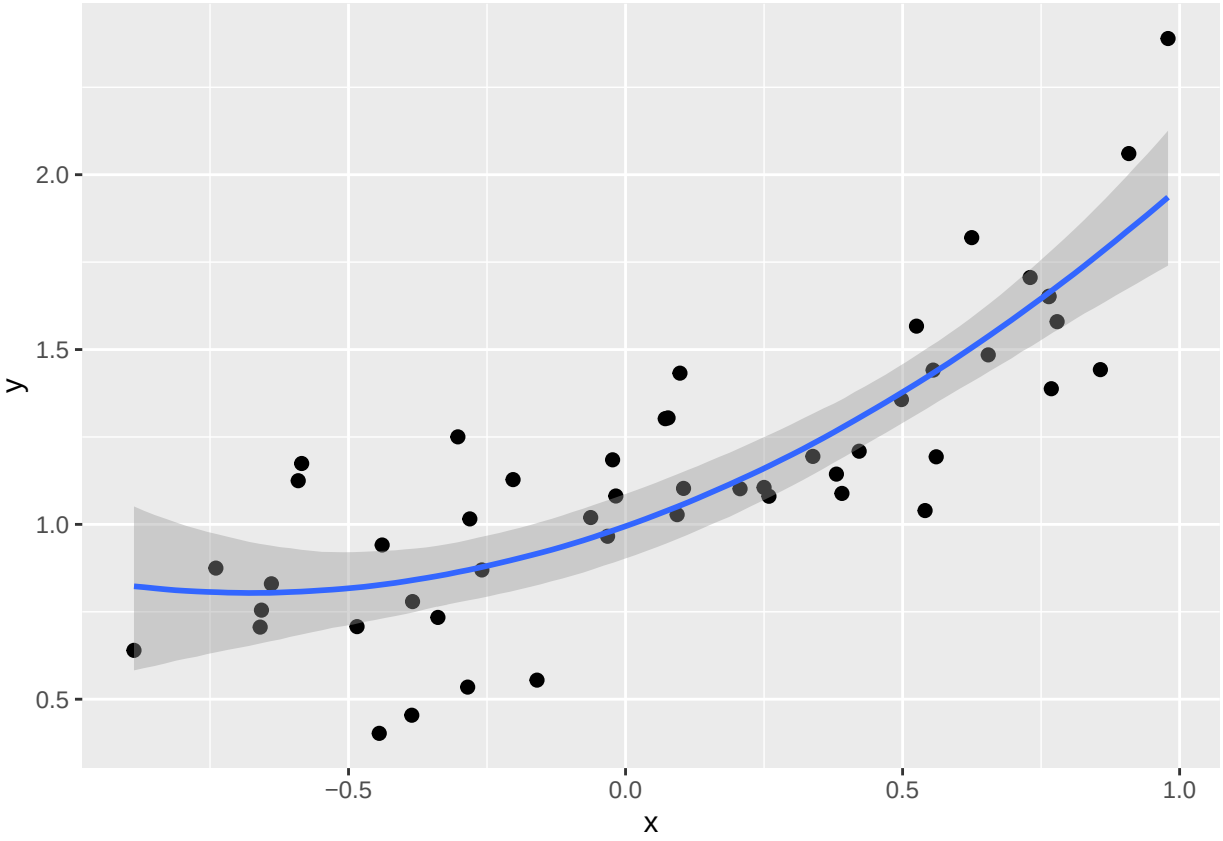
```
fit.b4
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ x + I(x^2)
##    Data: df (Number of observations: 50)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     0.99      0.05     0.90     1.09 1.00     3964     3190
## x             0.56      0.07     0.43     0.69 1.00     3930     3022
## IxE2          0.41      0.14     0.14     0.68 1.00     3885     3021
##
## Family Specific Parameters:
```

```
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.23      0.02     0.19     0.28 1.00     4009     2963
##
## 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).
```

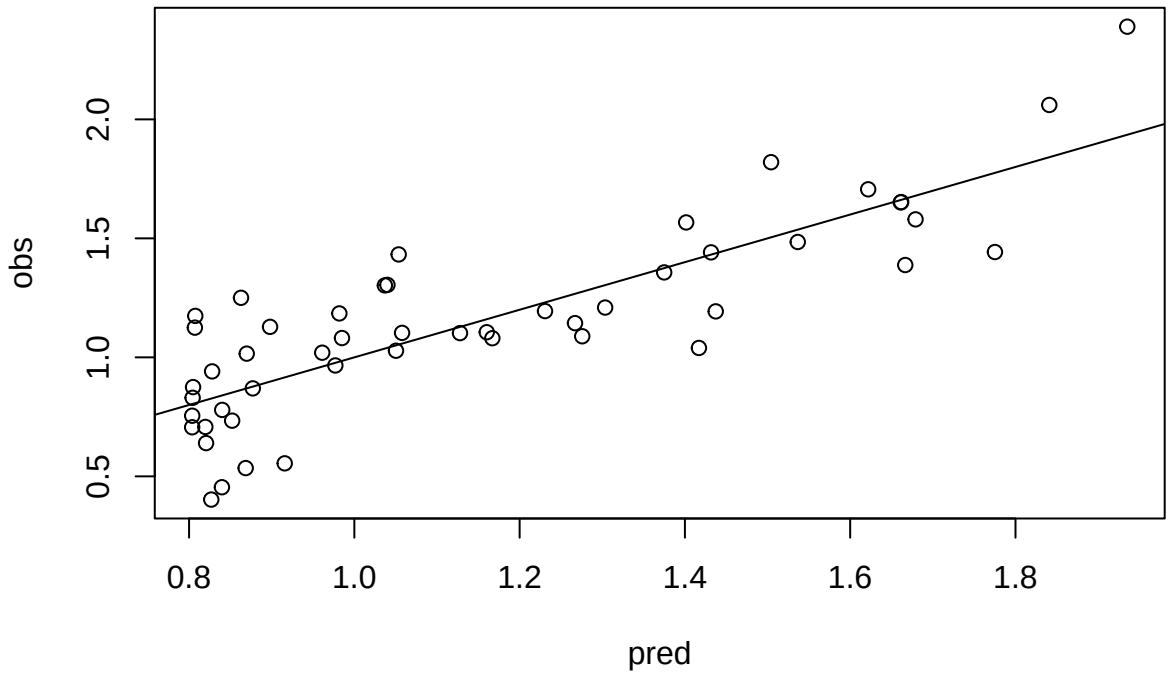
With only 1 independent variable, `conditional_effects()` plots credible intervals for prediction.

```
plot(conditional_effects(fit.b4),
     points=TRUE)
```

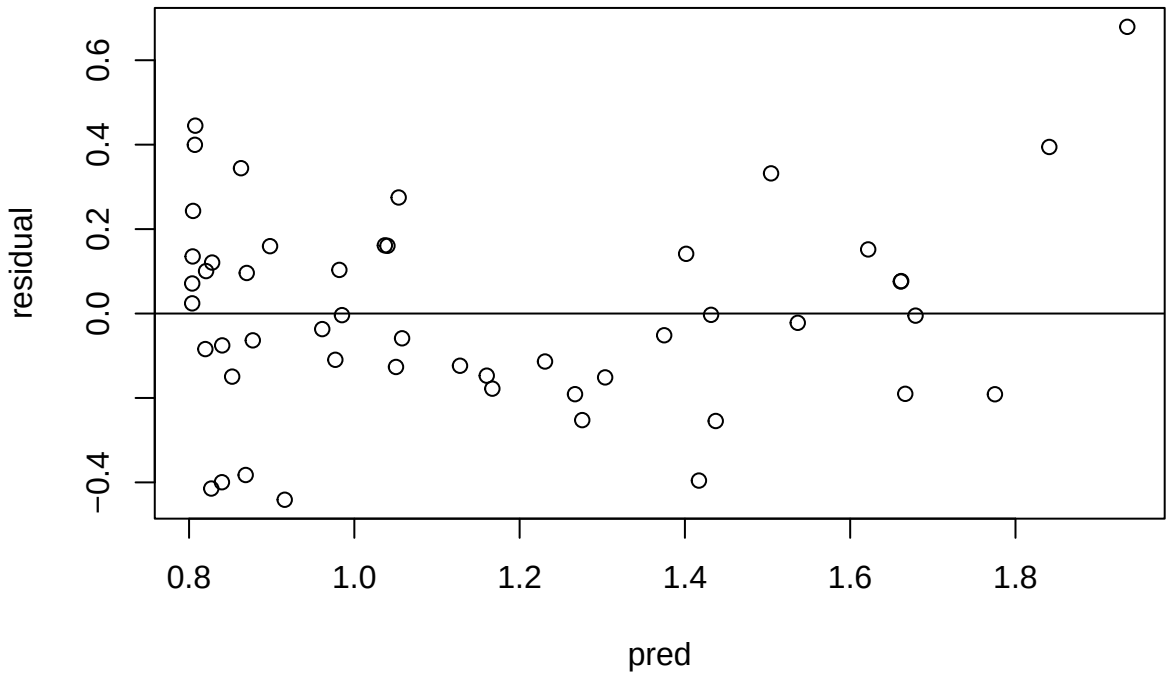



Compute credible intervals for the data to further check model fit, also PPC (posterior predictive check)

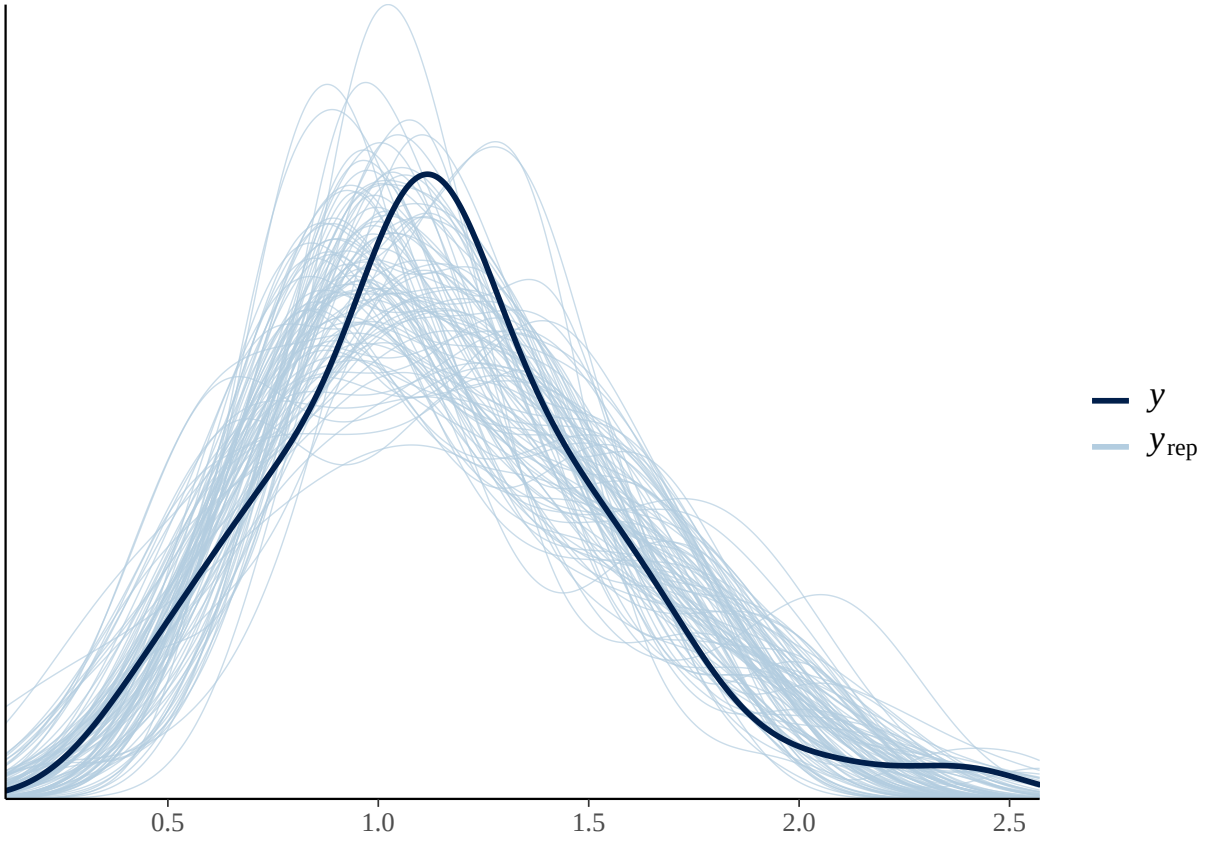
```
cred.fit.b4 = fitted(fit.b4)

plot(cred.fit.b4[, 1], df$y, xlab="pred", ylab="obs")
abline(0,1)
```




```
plot(cred.fit.b4[, 1], df$y-cred.fit.b3[, 1], xlab="pred", ylab="residual")
abline(0,0)
```




```
pp_check(fit.b4, ndraws=100)
```




We can check the probability of having a quadratic effect

```
hypothesis(fit.b4, "IxE2>0", class="b")
```

```
## Hypothesis Tests for class b:
##   Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio Post.Prob Star
## 1 (IxE2) > 0     0.41      0.14     0.18     0.63        999         1    *
## ---
## 'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
## '*': For one-sided hypotheses, the posterior probability exceeds 95%;
## for two-sided hypotheses, the value tested against lies outside the 95%-CI.
## Posterior probabilities of point hypotheses assume equal prior probabilities.
```

For assessing the quality of each fit, $R^2$ values are computed with `bayes_R2()`. It computes a goodness-of-fit $R^2$ for each sample of the posterior, so the Bayesian $R^2$ is a distribution of $R^2$-values.

```
bayes_R2(fit.b3)
```

```
##     Estimate  Est.Error      Q2.5     Q97.5
## R2 0.6353807 0.05376745 0.5064381 0.7125153
```

```
bayes_R2(fit.b4)
```

```
##    Estimate  Est.Error      Q2.5     Q97.5
## R2 0.691813 0.04334477 0.5894547 0.7518826
```

For model comparison, an information criterion based on leave-one-out cross validation (LOO-CV) can be easily computed: loo IC.

```
LIC.3 = loo(fit.b3)
LIC.4 = loo(fit.b4)
```

```
LIC.3
```

```
##
## Computed from 4000 by 50 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo     -2.2  5.6
## p_loo         3.4  1.2
## looic         4.4 11.2
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
LIC.4
```

```
##
## Computed from 4000 by 50 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo      1.5 4.5
## p_loo         3.9 1.0
## looic        -2.9 9.0
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     49    98.0%   2338
##  (0.5, 0.7]   (ok)        1     2.0%   790
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

In Bayesian statistics, everything is a distribution, even the information criterion. I.e., from every sample from the posterior distribution, a sample for the IC can be computed. Therefore, we get a posterior distribution of the IC. This is super helpful, because we also get a distribution for the difference (mean and sd) and get to check if this mean difference sufficiently nonzero.

(e.g. `elpd_diff>2*se_diff` would mean that we're sure that this model performs worse than the best model)

```
loo_compare(LIC.3, LIC.4)
```

```
##        elpd_diff se_diff
## fit.b4  0.0       0.0
## fit.b3 -3.7       2.9
```