3.1 Practical: multiple predictors

Benjamin Rosenbaum

November 4, 2021

We'll now fit a linear model with 2 predictors and we'll test for an interaction. From now on, we first code the model in Stan and then use brms. The statistical model without interaction "y~x1+x2" reads

$$y_i \sim \text{normal}(\mu_i, \sigma)$$
$$\mu_i = a + b_1 \cdot x_{1,i} + b_2 \cdot x_{2,i}$$

and with an interaction term "y~x1*x2"

$$y_i \sim \text{normal}(\mu_i, \sigma)$$
$$\mu_i = a + b_1 \cdot x_{1,i} + b_2 \cdot x_{2,i} + b_3 \cdot x_{1,i} \cdot x_{2,i}$$

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 reproducability

n=100

a=1.0
b1=0.5
b2=0.4
b3=0.5
sigma=0.1

x1 = runif(n=n, min=0, max=2)
x2 = runif(n=n, min=0, max=2)
y = rnorm(n=n, mean=a+b1*x1+b2*x2+b3*x1*x2, sd=sigma)

df = data.frame(x1=x1,
                x2=x2,
```

```
               y=y)

head(df)
```

```
##          x1         x2        y
## 1 0.6155322 0.65483016 1.737940
## 2 0.5153450 0.77895739 1.906283
## 3 1.1046449 0.08210551 1.583599
## 4 0.1127663 0.72279327 1.470541
## 5 0.9370986 1.14195617 2.314595
## 6 0.9675415 1.36976048 2.654294
```

No interaction model "y~x1+x2"

We code this model straightforward using `a` for intercept and a vector `b` of length 2 for the 2 effects. Vector notation for `y, x1, x2` is used in

```
y ~ normal(a + b[1]*x1 + b[2]*x2, sigma);
```

which is short for a for-loop over

```
y[i] ~ normal(a + b[1]*x1[i] + b[2]*x2[i], sigma);
```

```
stan_code = '
data {
  int n;
  vector[n] x1;
  vector[n] x2;
  vector[n] y;
}
parameters {
  real a;
  vector[2] b;
  real<lower=0> sigma;  // standard deviation
}
model {
  // priors
  a ~ normal(0, 10);
  b ~ normal(0, 10);
  sigma ~ normal(0, 10);
  // likelihood
  y ~ normal(a + b[1]*x1 + b[2]*x2, sigma);
}
'

data = list(n=n,
            x1=df$x1,
            x2=df$x2,
            y=df$y)
```

```
stan_model = stan_model(model_code=stan_code)
fit.1  = sampling(stan_model, data=data)
```

```
print(fit.1, digits=3, probs=c(0.025, 0.975))
```

```
## Inference for Stan model: ac22f75b54ea951d67e09536e979413f.
```

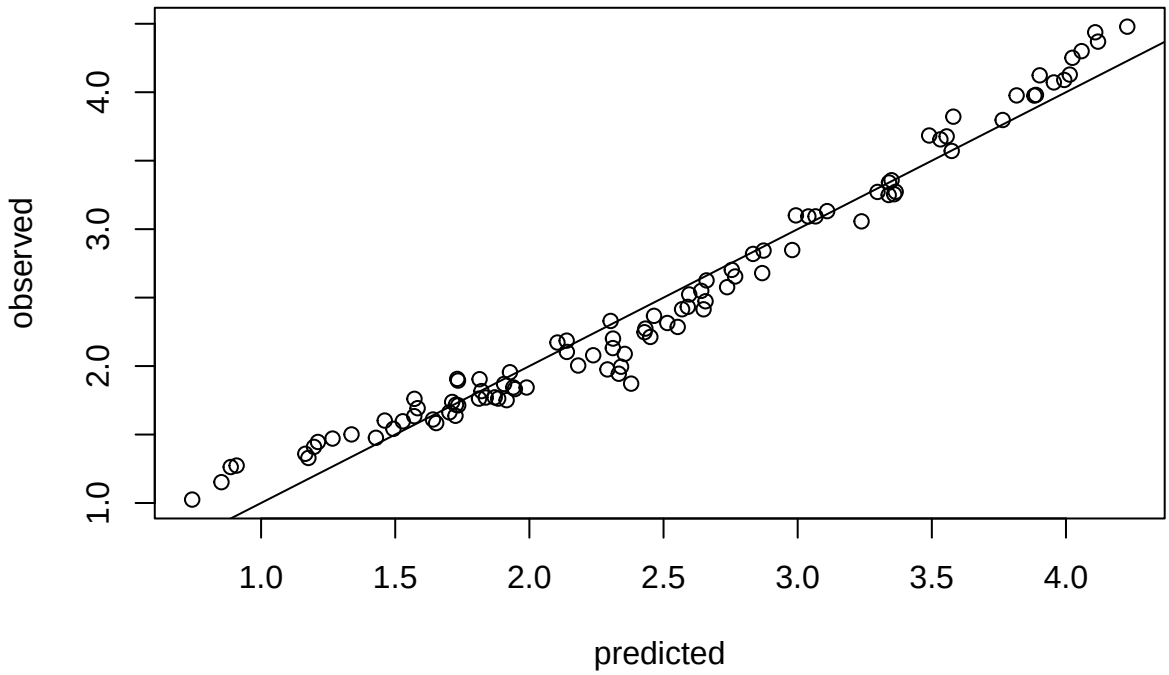
```
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean    sd    2.5%   97.5% n_eff  Rhat
## a       0.448   0.001 0.045   0.357   0.534  2297 1.000
## b[1]    1.018   0.001 0.035   0.949   1.088  2488 1.001
## b[2]    0.974   0.001 0.031   0.913   1.034  3350 1.001
## sigma   0.175   0.000 0.013   0.152   0.202  3256 1.001
## lp__  123.065   0.038 1.460 119.402 124.865  1498 1.002
##
## Samples were drawn using NUTS(diag_e) at Wed Oct  6 10:47:29 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

With multiple predictors, it's not so easy to visualize data and model predictions (see below). But it's always possible (and recommended) to look at an observed vs predicted, or better: residuals vs predicted plot.

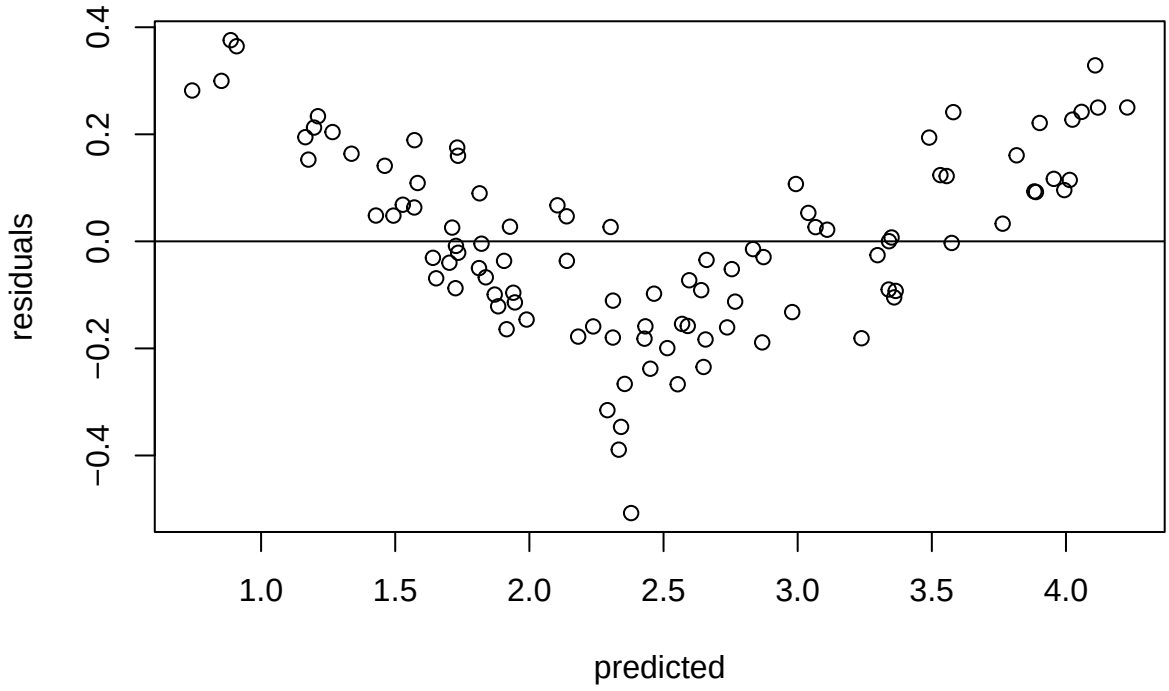
```
posterior=as.matrix(fit.1)
head(posterior)
```

```
##           parameters
## iterations         a     b[1]      b[2]     sigma     lp__
##       [1,] 0.3659536 1.065069 0.9945879 0.1630957 122.8474
##       [2,] 0.3831699 1.078593 0.9766282 0.1669266 123.2066
##       [3,] 0.3970884 1.044513 0.9844156 0.1773738 124.3066
##       [4,] 0.4194981 1.042777 0.9820669 0.1720644 124.6958
##       [5,] 0.4051255 1.061386 0.9607740 0.1881481 123.4639
##       [6,] 0.4890595 1.022006 0.9380370 0.1587657 123.4113
```

```
y.cred = matrix(0, nrow=nrow(posterior), ncol=data$n) # predictions for each posterior sample and each
for(i in 1:nrow(posterior)){ # loop over posterior samples
  y.cred[i, ] = posterior[i,"a"] +
    posterior[i,"b[1]"]*data$x1 +
    posterior[i,"b[2]"]*data$x2
}
y.cred.mean = apply(y.cred, 2, function(x) mean(x))

plot(y.cred.mean, df$y, ylab="observed", xlab="predicted")
abline(0,1)
```




```
plot(y.cred.mean, df$y-y.cred.mean, ylab="residuals", xlab="predicted")
abline(0,0)
```



There is clearly something going on in the residuals, which means that the model does not explain the variation in the data appropriately.

Interaction model "y~x1*x2"

We add an interaction term `x1*x2` with effect size `b[3]`. To use vector notation, this reads `b[3] * x1 .* x2`, because `.*` multiplies vectors element-wise. As always, alternatively a for loop over

```
y[i] ~ normal(a + b[1]*x1[i] + b[2]*x2[i] + b[3]*x1[i]*x2[i], sigma);
```

can be used.

```
stan_code = '
data {
  int n;
  vector[n] x1;
  vector[n] x2;
  vector[n] y;
}
parameters {
  real a;
  vector[3] b;
  real<lower=0> sigma;  // standard deviation
}
model {
  // priors
  a ~ normal(0, 10);
  b ~ normal(0, 10);
  sigma ~ normal(0, 10);
  // likelihood
  y ~ normal(a + b[1]*x1 + b[2]*x2 + b[3] * x1 .* x2, sigma);
}
'
```

```
stan_model = stan_model(model_code=stan_code)
fit.2  = sampling(stan_model, data=data)
```

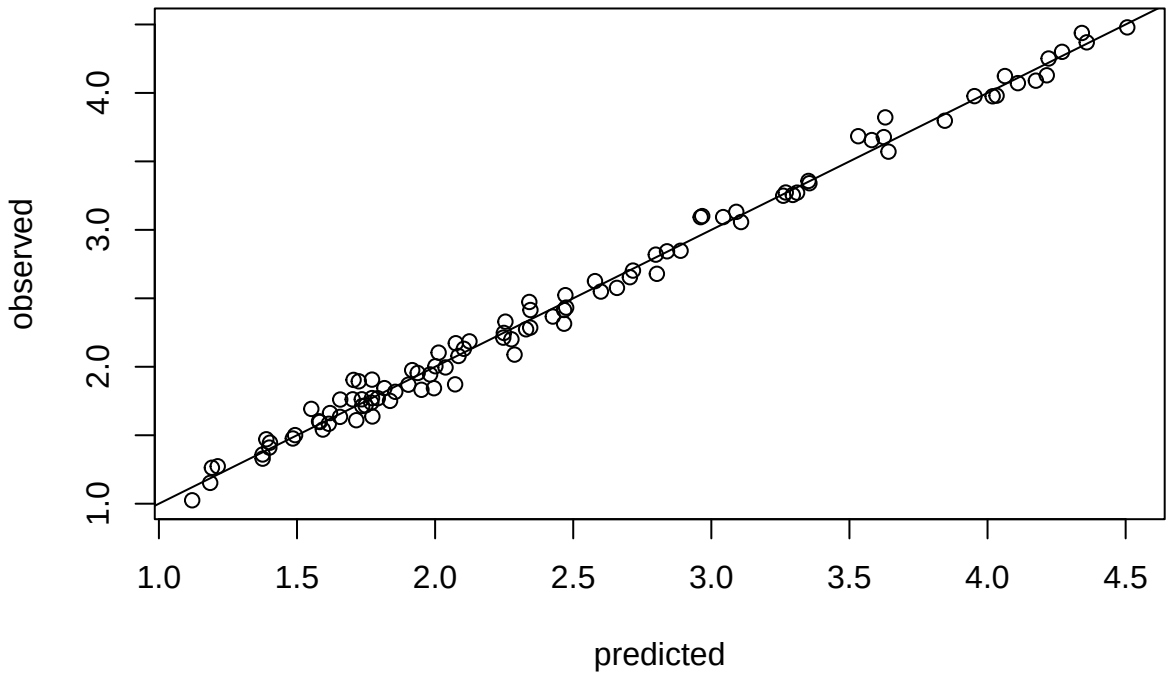
```
print(fit.2, digits=3, probs=c(0.025, 0.975))
```

```
## Inference for Stan model: 528ea9887fd787a171d6645b9591e985.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean    sd    2.5%   97.5% n_eff  Rhat
## a       0.979   0.001 0.036   0.907   1.050   914 1.007
## b[1]    0.504   0.001 0.032   0.439   0.569   929 1.007
## b[2]    0.434   0.001 0.033   0.372   0.499   978 1.007
## b[3]    0.486   0.001 0.026   0.433   0.538   915 1.008
## sigma   0.081   0.000 0.006   0.071   0.094  1763 1.002
## lp__  199.543   0.047 1.642 195.609 201.754  1203 1.005
##
## Samples were drawn using NUTS(diag_e) at Wed Oct  6 10:48:19 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

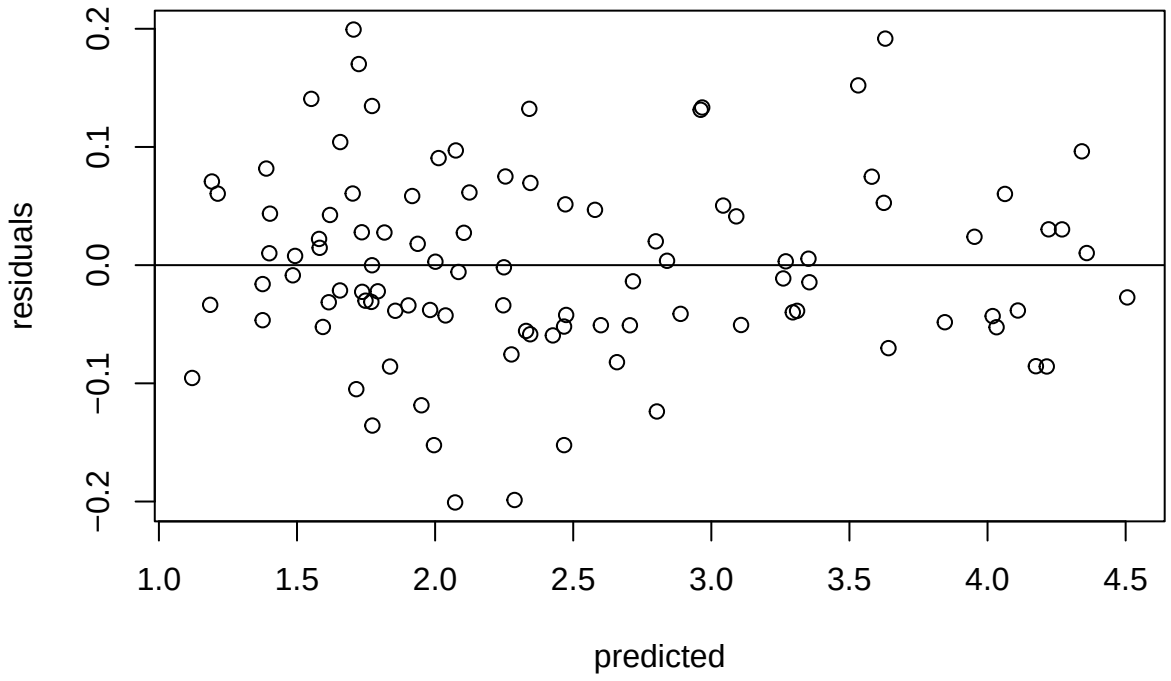
A positive interaction effect `b[3]` is found. Also, residual plots look much better.

```
posterior=as.matrix(fit.2)

y.cred = matrix(0, nrow=nrow(posterior), ncol=data$n)
for(i in 1:nrow(posterior)){
  y.cred[i, ] = posterior[i,"a"] +
    posterior[i,"b[1]"]*data$x1 +
    posterior[i,"b[2]"]*data$x2 +
```

```
    posterior[i,"b[3]"]*data$x1*data$x2
}
y.cred.mean = apply(y.cred, 2, function(x) mean(x))

plot(y.cred.mean, df$y, ylab="observed", xlab="predicted")
abline(0,1)
```




```
plot(y.cred.mean, df$y-y.cred.mean, ylab="residuals", xlab="predicted")
abline(0,0)
```

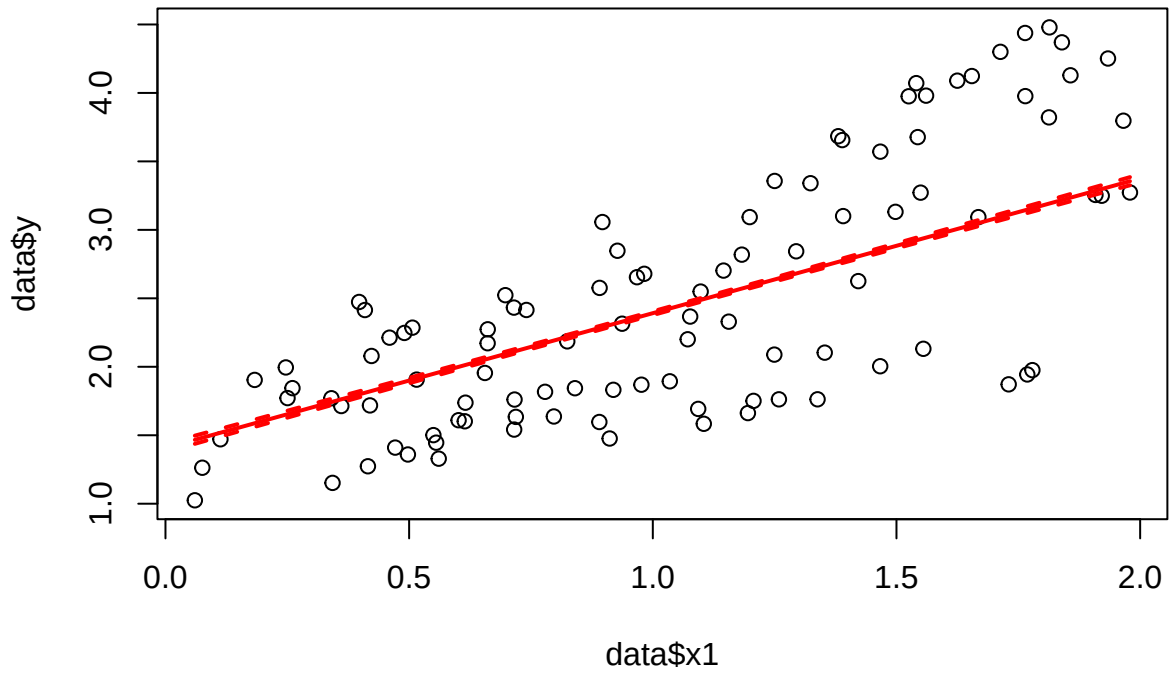


Marginal effects plot or conditional effects plot show the effect of single predictors. If there is an interaction

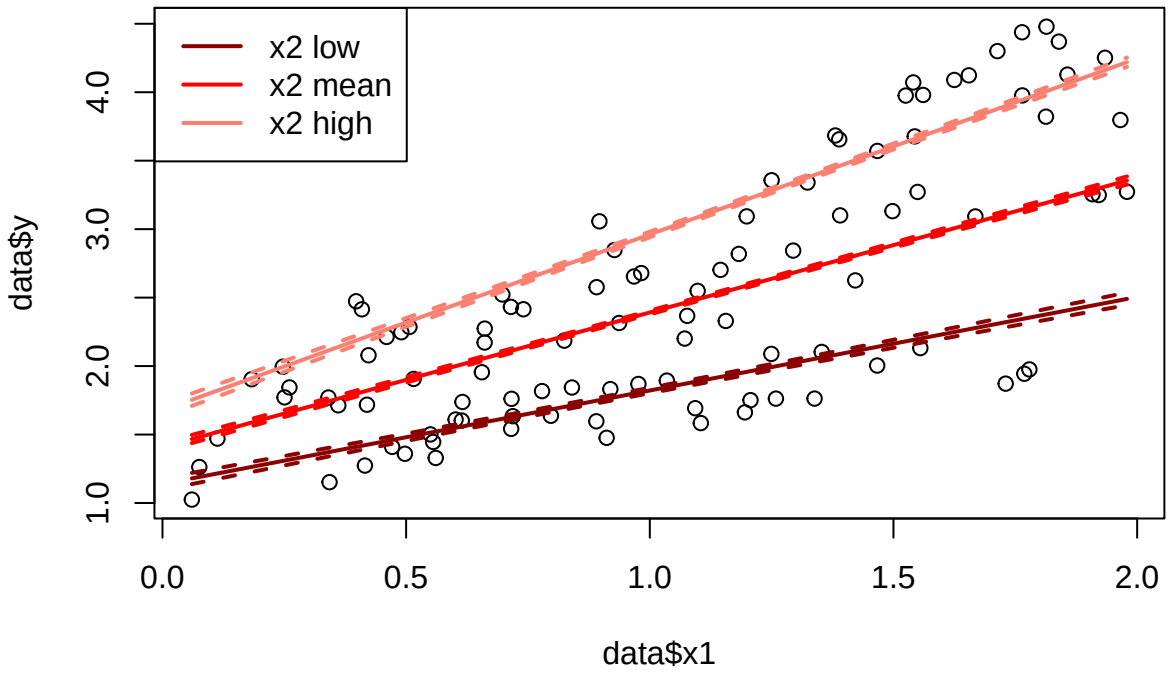
between predictors, a marginal effect of `x1` is not independent of `x2`. So we can only plot the effect of `x1`, conditional `x2=...` So for different levels of `x2`, marginal effects of `x1` can be plotted.

Predictions are generated for the whole range of `x1`, while `x2` is fixed at its mean.

```
plot(data$x1, data$y)

x1.pred = seq(from=min(df$x1), to=max(df$x1), length.out=100)
x2.pred = mean(df$x2)

y.cred = matrix(0, nrow=nrow(posterior), ncol=length(x1.pred))
for(i in 1:nrow(posterior)){
  y.cred[i, ] = posterior[i,"a"] +
    posterior[i,"b[1]"]*x1.pred +
    posterior[i,"b[2]"]*x2.pred +
    posterior[i,"b[3]"]*x1.pred*x2.pred
}

y.cred.mean = apply(y.cred, 2, function(x) mean(x))
lines(x1.pred, y.cred.mean, col="red", lwd=2)

y.cred.q05 = apply(y.cred, 2, function(x) quantile(x, probs=0.05))
lines(x1.pred, y.cred.q05, col="red", lwd=2, lty=2)

y.cred.q95 = apply(y.cred, 2, function(x) quantile(x, probs=0.95))
lines(x1.pred, y.cred.q95, col="red", lwd=2, lty=2)
```



We repeat this with `x2.pred = mean(df$x2)-sd(df$x2)` (level=low) and `x2.pred = mean(df$x2)+sd(df$x2)` (level=high), to visualize the interaction effect.



No interaction model in brms

In brms, the model without interaction is coded exactly as in `lm()`, but we provide priors for all effect sizes.

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

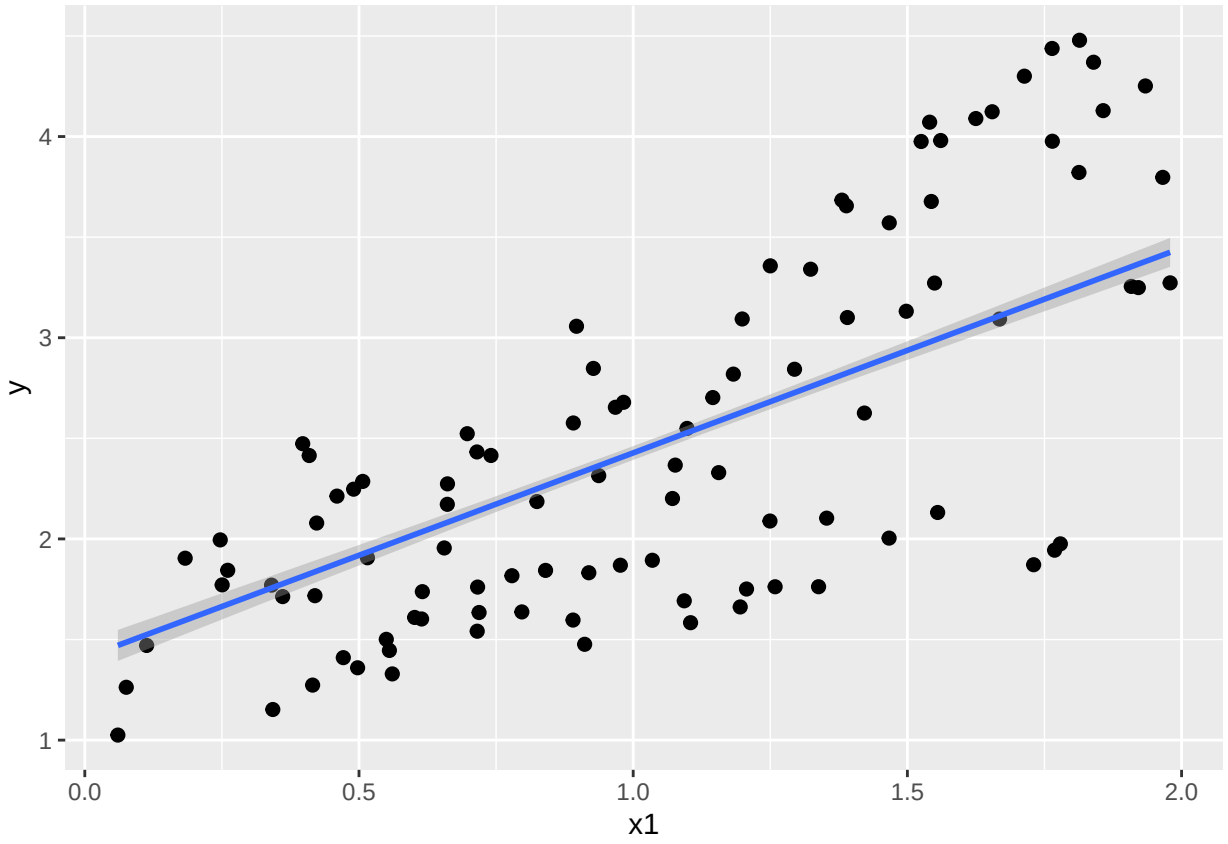
```
fit.b1 = brm(y ~ x1 + x2,
             data = df,
             prior = priors)
```

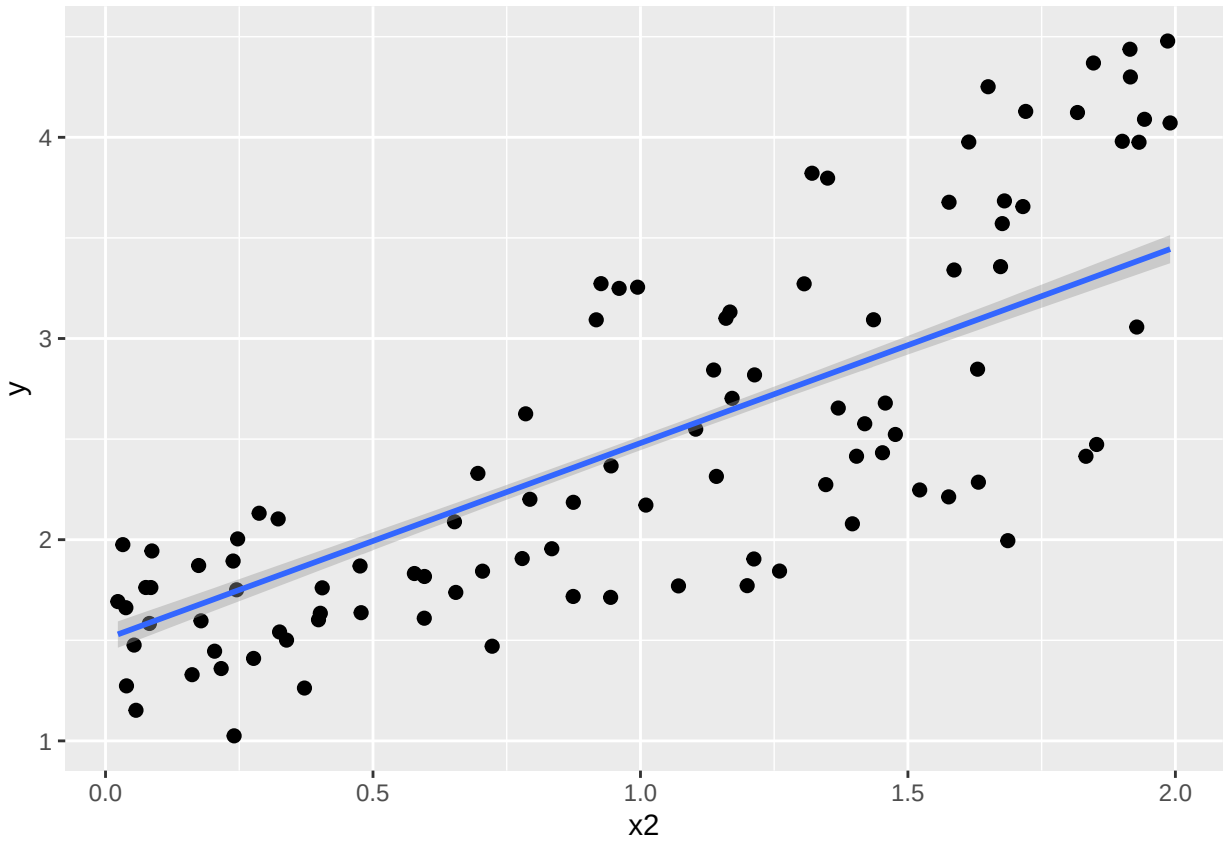
```
fit.b1
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ x1 + x2
##    Data: df (Number of observations: 100)
##   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     0.45      0.04     0.36     0.54 1.00     5149     3361
## x1            1.02      0.04     0.95     1.09 1.00     4134     3244
## x2            0.97      0.03     0.92     1.03 1.00     4170     2786
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.18      0.01     0.15     0.20 1.00     4056     3058
##
## 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).
```

`conditional_effects()` plots predictions for each predictor separately, the other predictor is fixed at its mean. Different levels (higher or lower) of the respective other predictor would only affect intercept, not the slope, in this `y~x1+x2` model.

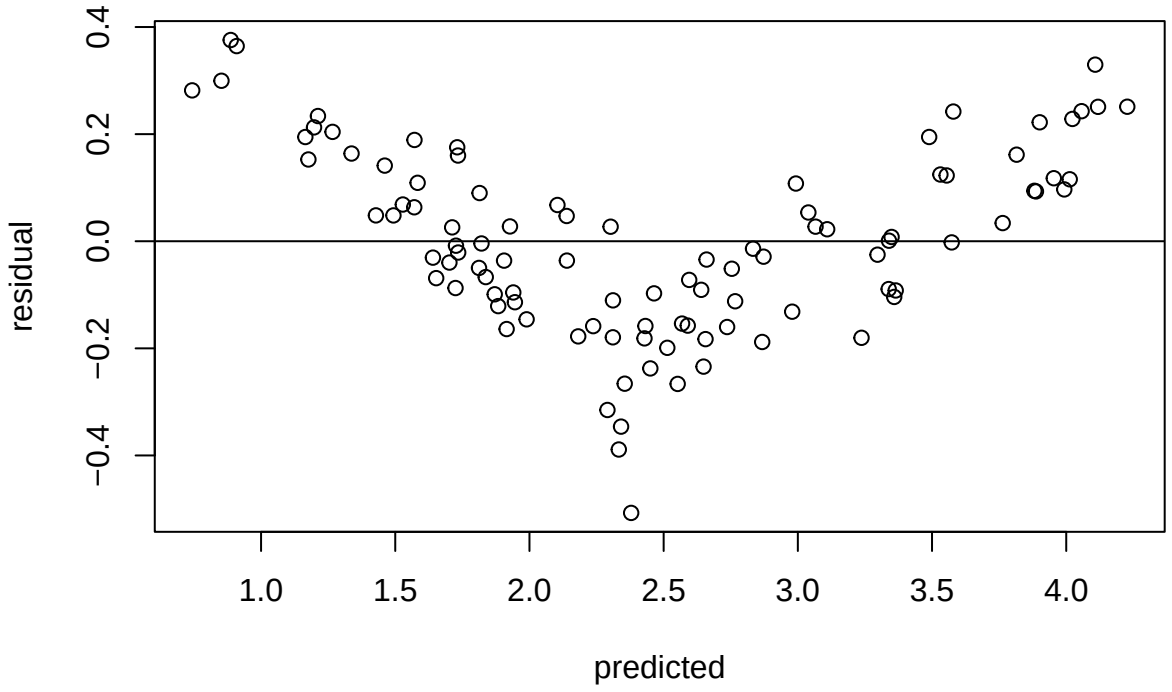
```
plot(conditional_effects(fit.b1),
     points=TRUE,
     ask=FALSE)
```





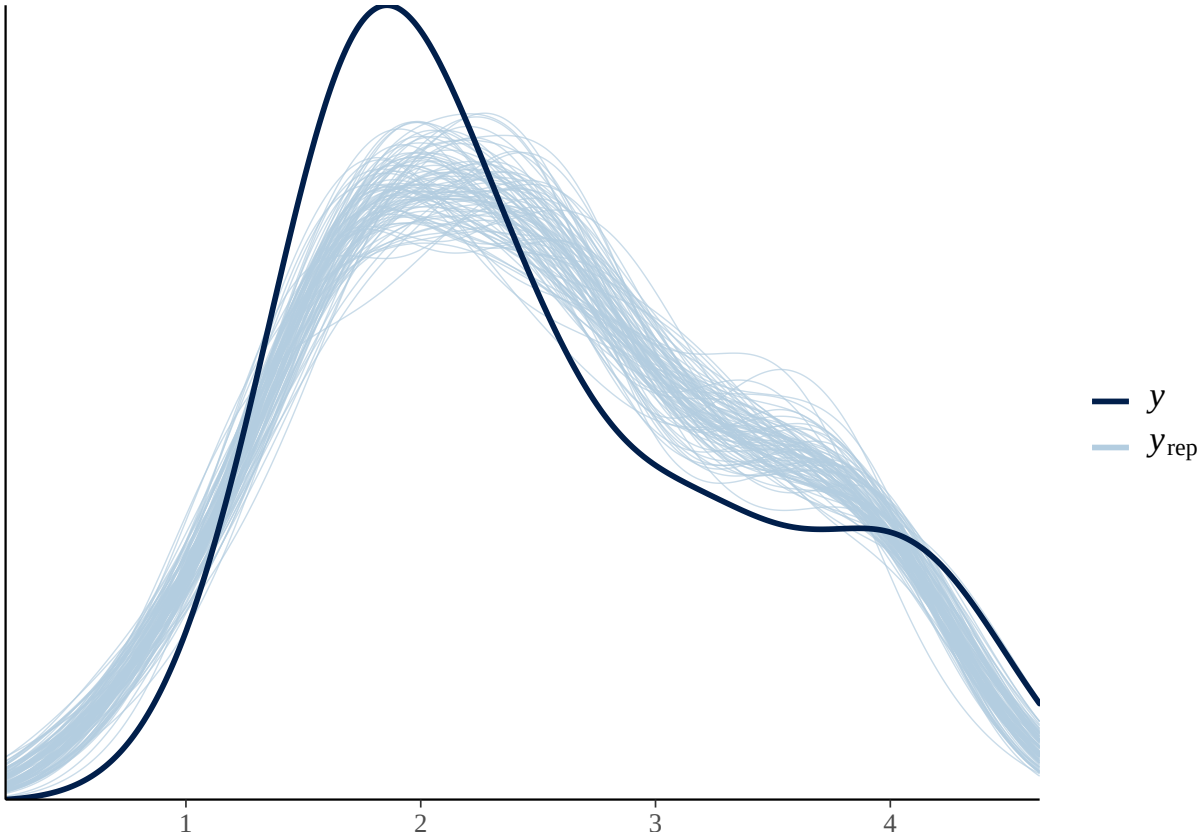
Predictions for the deterministic model are easily generated with `fitted()` for all datapoints to plot the residuals.

```
cred.fit.b1 = fitted(fit.b1)
plot(cred.fit.b1[,1], df$y-cred.fit.b1[,1], xlab="predicted", ylab="residual")
abline(0,0)
```



This and the PPC (posterior predictive check) reveal that the model misses something.

```
pp_check(fit.b1, ndraws=100)
```



Interaction model in brms

As in `lm()`, `y~x1*x2` models all lower order effects (main effects and intercept), too.

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

```
fit.b2 = brm(y ~ x1 * x2,
             data = df,
             prior = priors)
```

```
fit.b2
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ x1 * x2
##    Data: df (Number of observations: 100)
##   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     0.98      0.04     0.91     1.05 1.00     1438     2025
## x1            0.50      0.03     0.44     0.57 1.00     1439     1855
## x2            0.43      0.03     0.37     0.50 1.00     1398     1959
## x1:x2         0.49      0.03     0.43     0.54 1.00     1312     1700
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.08      0.01     0.07     0.09 1.00     2731     2184
##
## 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).
```

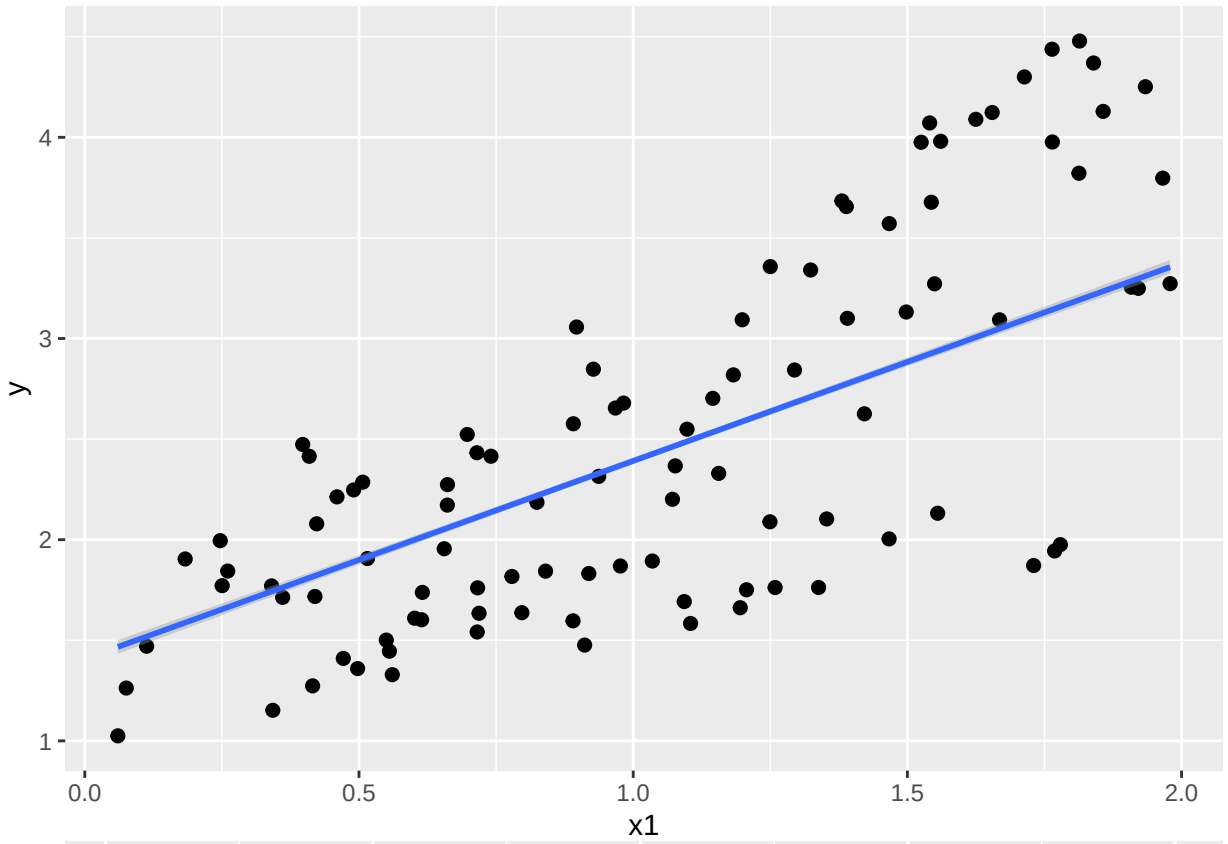
We observe a positive interaction term in this model.

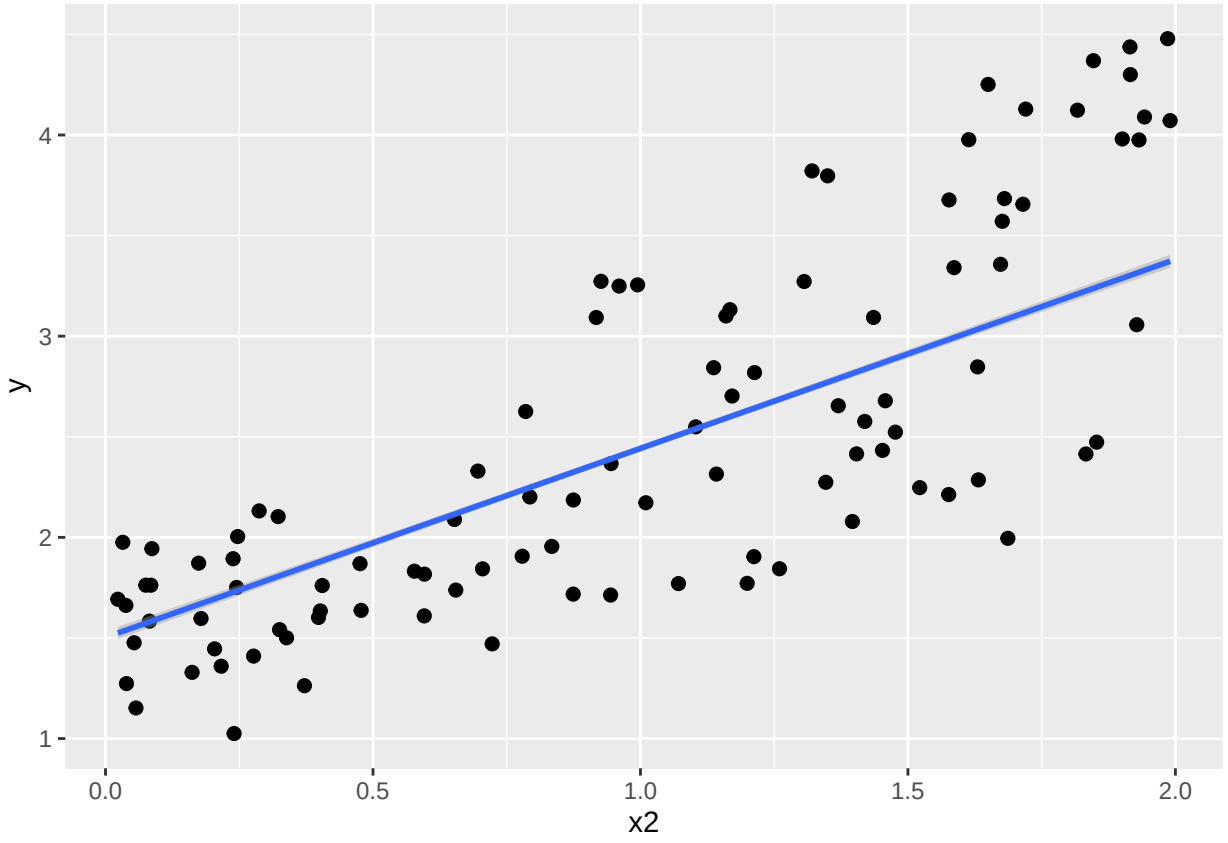
```
hypothesis(fit.b2, "x1:x2>0", class="b")
```

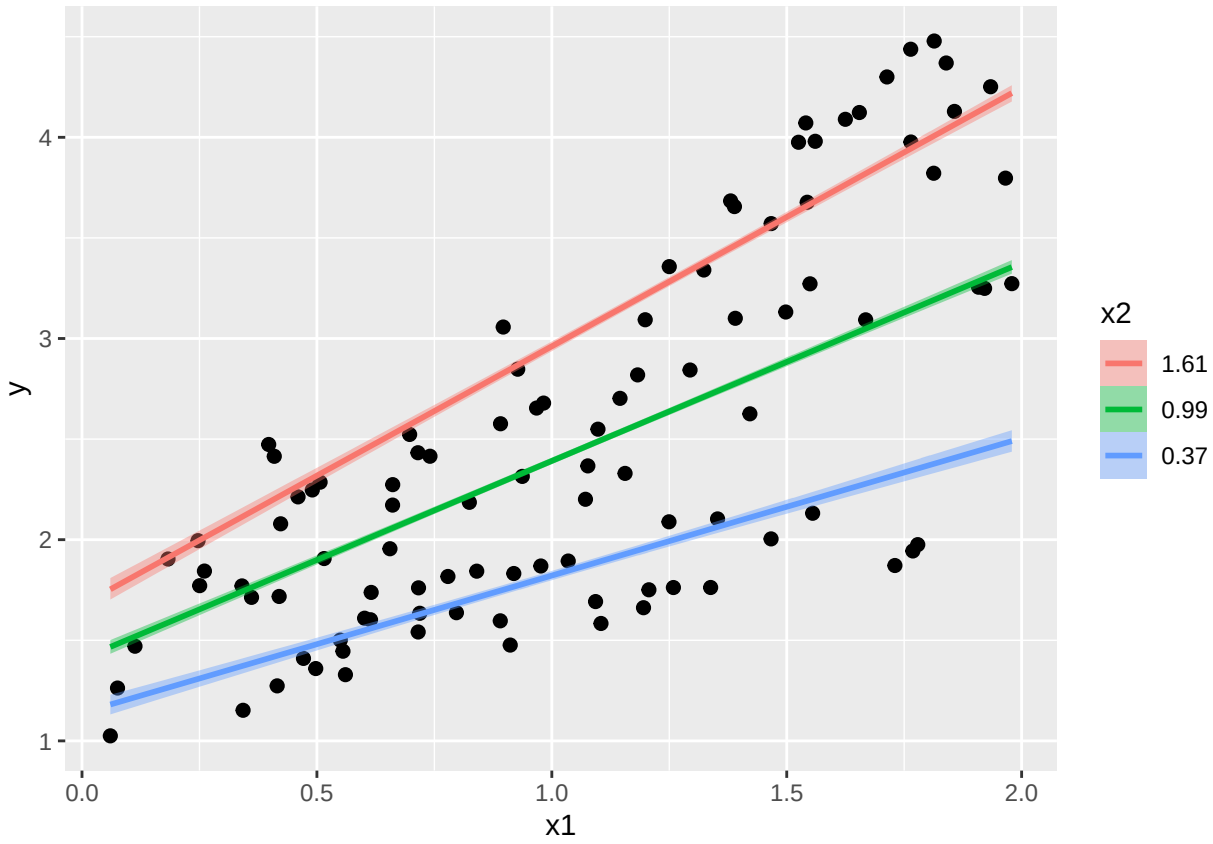
```
## Hypothesis Tests for class b:
##    Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio Post.Prob Star
## 1 (x1:x2) > 0     0.49      0.03     0.44     0.53        Inf         1    *
## ---
## 'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
## '*': For one-sided hypotheses, 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.
```

In this interaction model, `conditional_effects()` plots predictions for each predictor separately, the other predictor is fixed at its mean, in the first two figures. Because we now have an interaction, slopes depend on the level of the other predictor. In the third figure, the interaction is depicted: slope in `x1` changes with levels of `x2` (levels chosen are mean-sd, mean, mean+sd)

```
plot(conditional_effects(fit.b2),
     points=TRUE,
     ask=FALSE)
```

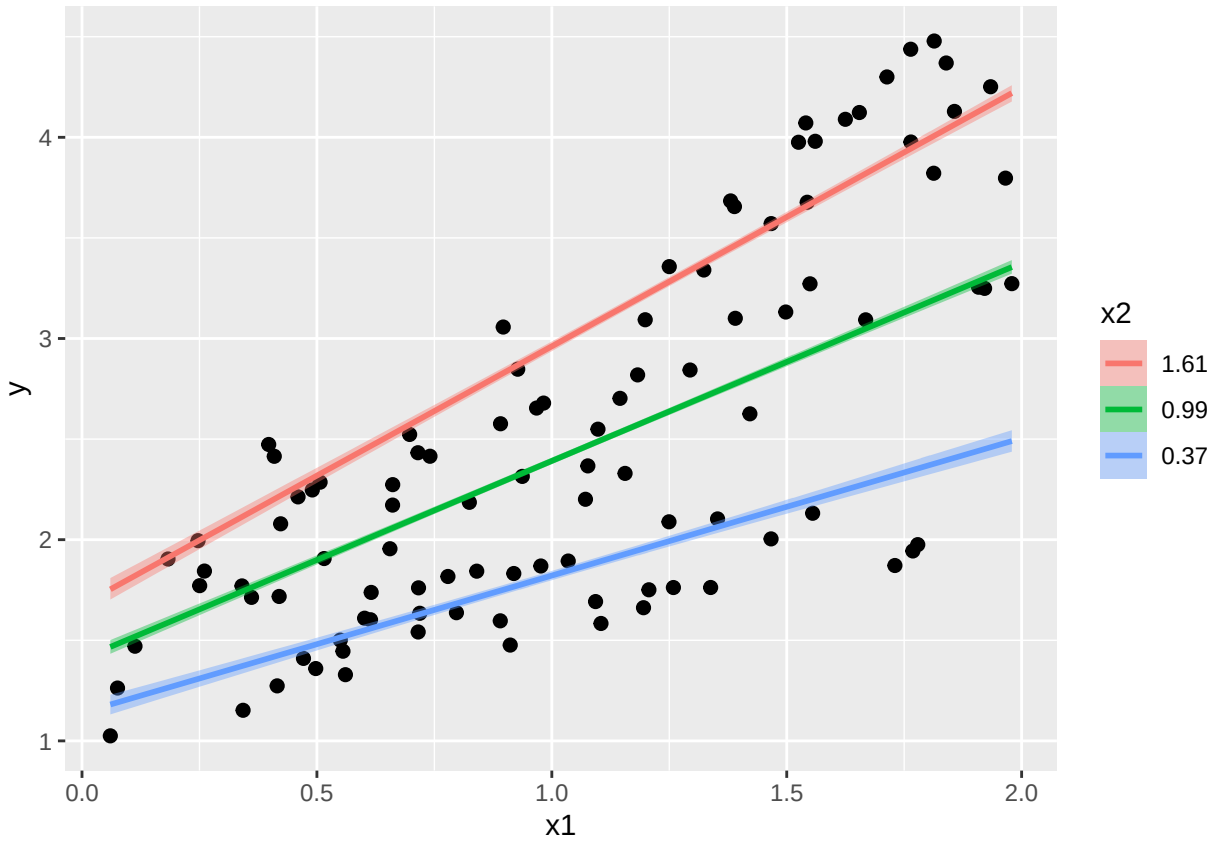

y
4
3
2
1
0.0
0.5
1.0
1.5
2.0
x1


y
4
3
2
1
0.0
0.5
1.0
1.5
2.0
x2



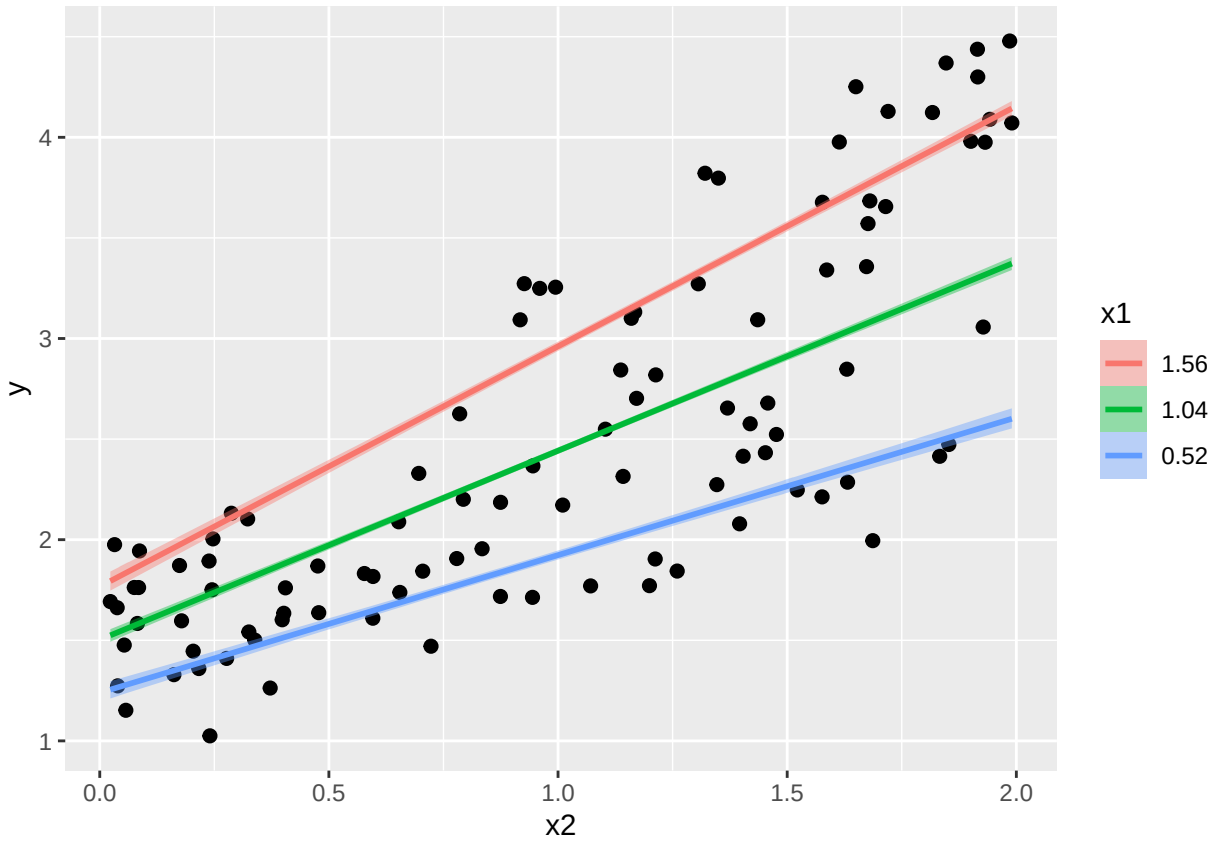
To plot this interaction only, use `effects=x1:x2`

```
plot(conditional_effects(fit.b2, effects = "x1:x2"),
     points=TRUE,
     ask=FALSE)
```



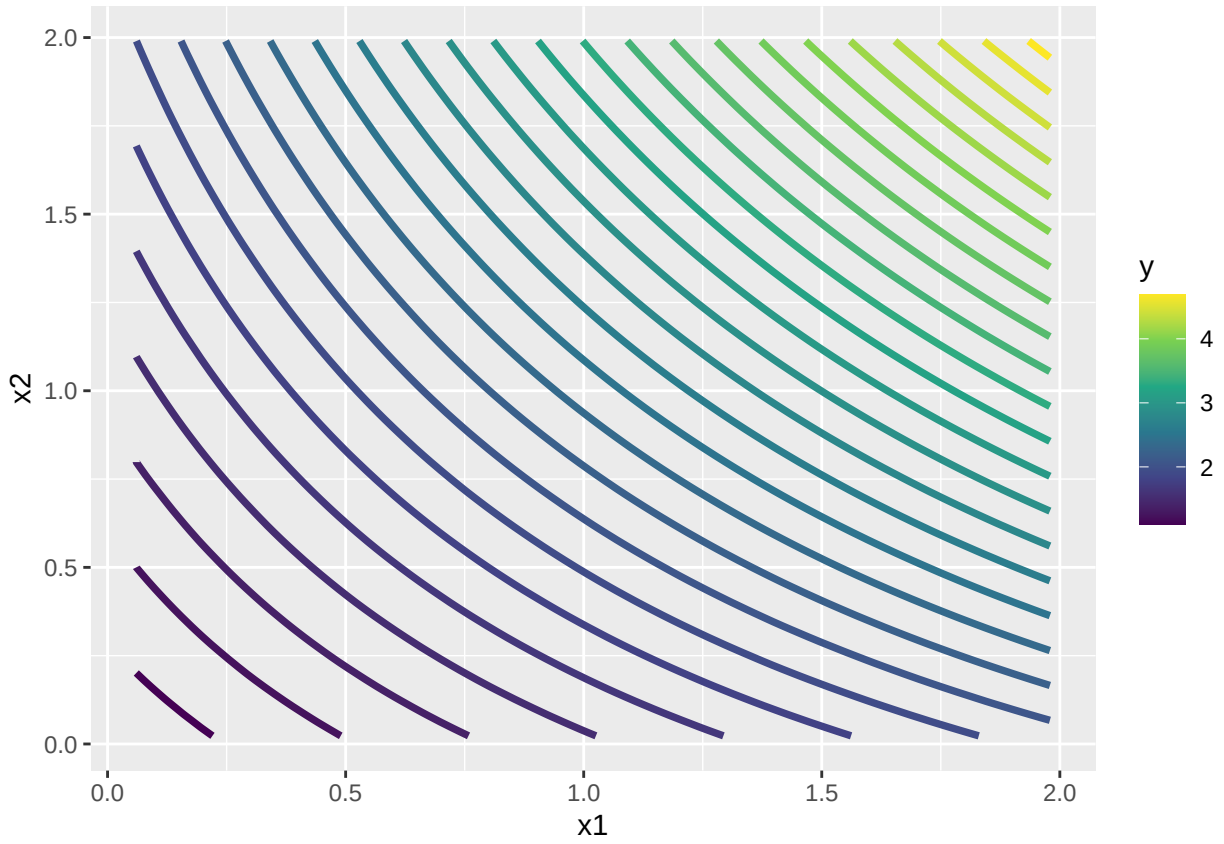
Or to see how slope in `x2` changes with levels of `x1`, switch predictors: `effects=x2:x1`

```
plot(conditional_effects(fit.b2, effects = "x2:x1"),
     points=TRUE,
     ask=FALSE)
```



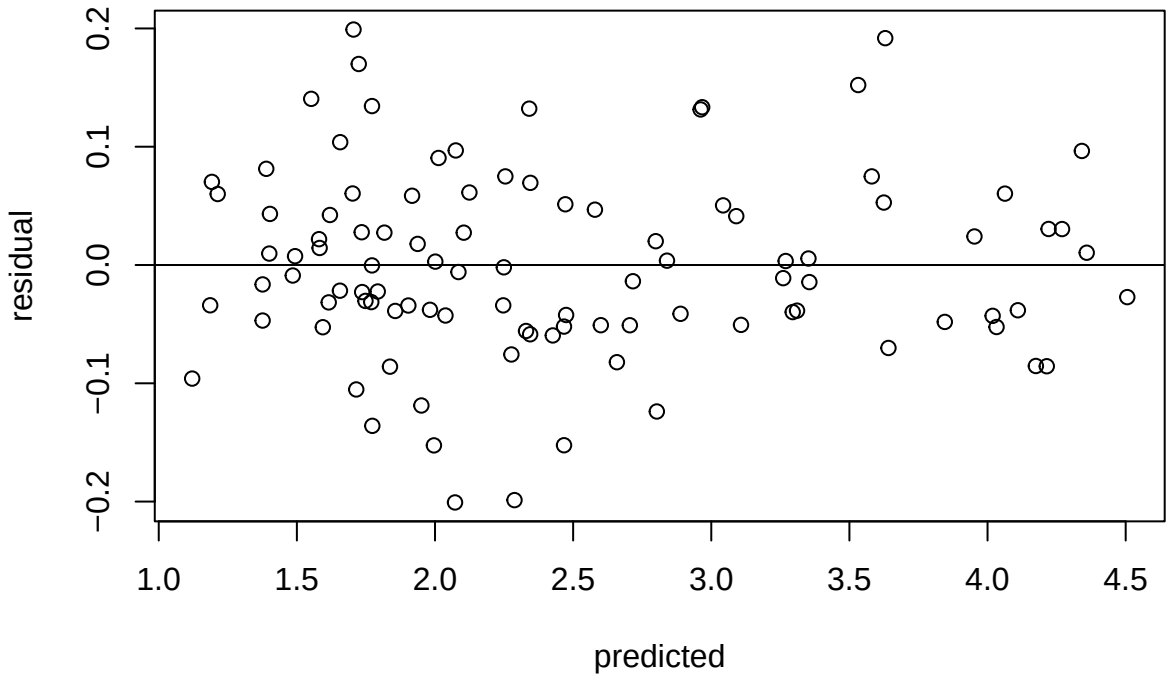
Since there are two predictors, predictions can be visualized in a countour plot

```
plot(conditional_effects(fit.b2, effects = "x1:x2", surface = TRUE))
```

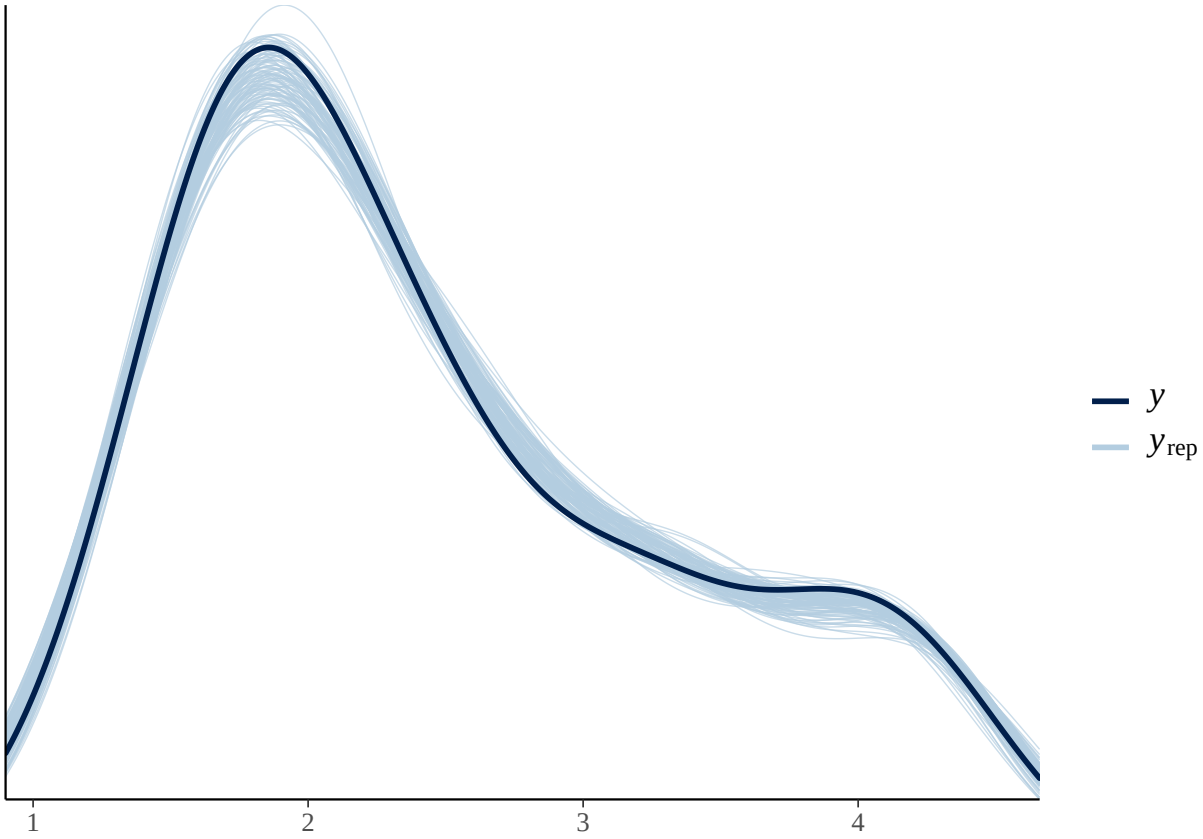


The residual plot and the PPC (posterior predictive check) look good here.

```
cred.fit.b2 = fitted(fit.b2)
plot(cred.fit.b2[,1], df$y-cred.fit.b2[,1], xlab="predicted", ylab="residual")
abline(0,0)
```



```
pp_check(fit.b2, ndraws=100)
```



We look at R^2 values to assess goodness-of-fit.

```
bayes_R2(fit.b1)
```

```
##    Estimate   Est.Error      Q2.5    Q97.5
## R2 0.963971 0.001593267 0.9599037 0.965828
```

```
bayes_R2(fit.b2)
```

```
##     Estimate    Est.Error      Q2.5     Q97.5
## R2 0.9923192 0.0002434424 0.9916861 0.9925984
```

Finally, we test the interaction effect model against the simpler model using the LOOIC. Model comparison shows a clear preference for the interaction model, with a significant difference.

```
LIC.1 = loo(fit.b1)
LIC.2 = loo(fit.b2)

loo_compare(LIC.1, LIC.2)
```

```
##        elpd_diff se_diff
## fit.b2   0.0       0.0
## fit.b1 -77.3       8.7
```