4.2 Practical: random intercepts regression

Benjamin Rosenbaum

October 27, 2022

In the previous model, we just fitted mean values to groups and there was still some unexplained variation (σ). Here, we will add a continuous predictor (covariate).

Specifically, for the same dataset of Species Richness, we add the covariate NAP (height of a sampling station compared to mean tidal level). Richness is assumed to be negatively associated with NAP.

Setup

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

setwd("~/Nextcloud/teaching Bayes 2021")

rstan_options(auto_write = TRUE)
options(mc.cores = 4)
```

Read dataset

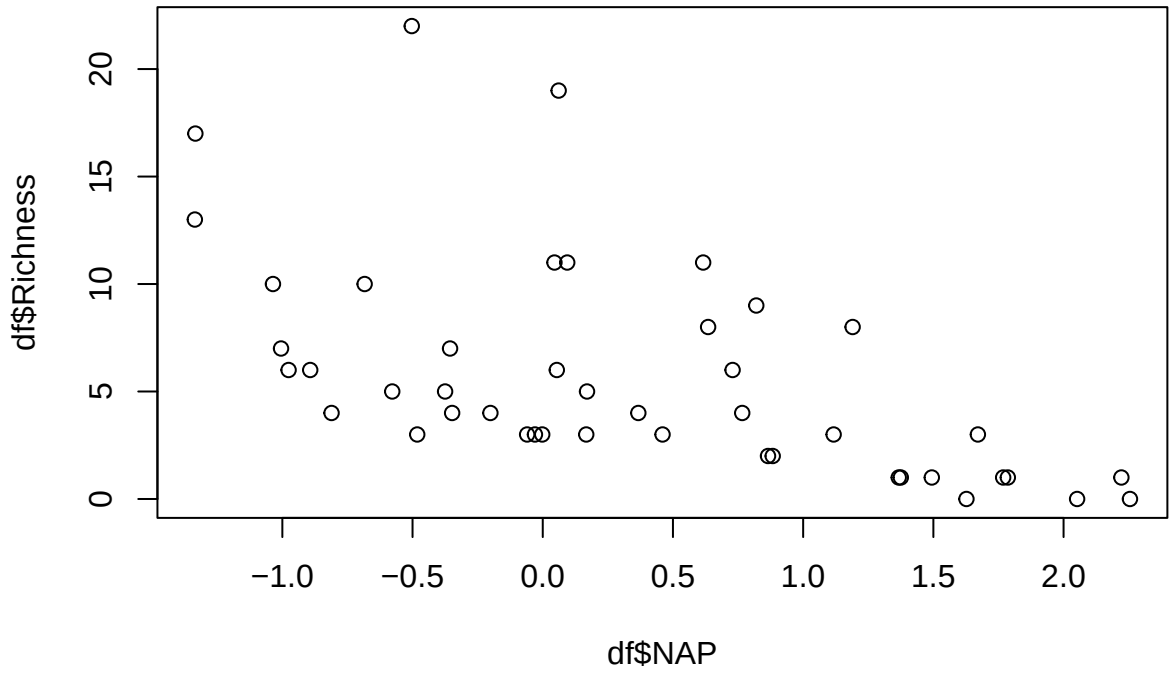
```
df = read.table("data/RIKZ.txt", header=TRUE)
head(df)
```

```
##   Sample Richness Exposure    NAP Beach
## 1      1       11       10  0.045     1
## 2      2       10       10 -1.036     1
## 3      3       13       10 -1.336     1
## 4      4       11       10  0.616     1
## 5      5       10       10 -0.684     1
## 6      6        8        8  1.190     2
```

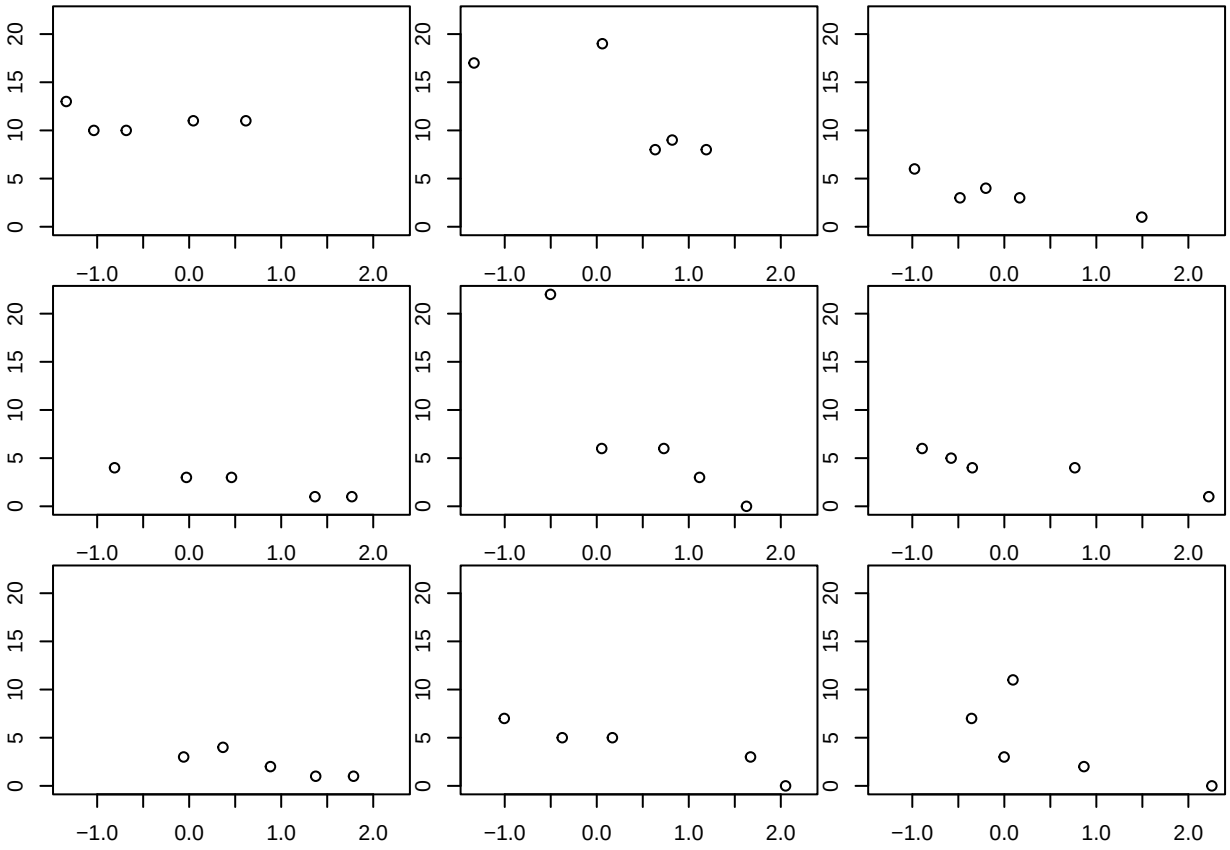
```
str(df)
```

```
## 'data.frame':    45 obs. of  5 variables:
##  $ Sample  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Richness: int  11 10 13 11 10 8 9 8 19 17 ...
##  $ Exposure: int  10 10 10 10 10 8 8 8 8 8 ...
##  $ NAP     : num  0.045 -1.036 -1.336 0.616 -0.684 ...
##  $ Beach   : int  1 1 1 1 1 2 2 2 2 2 ...
```

```
par(mfrow=c(1,1))
plot(df$NAP, df$Richness)
```




```
par(mfrow=c(3,3), mar=c(1,1,1,1), oma=c(1,1,0,0))
for (i in 1:9){
  df.sub=subset(df, df$Beach==i)
  plot(df.sub$NAP,  df.sub$Richness,
       xlim=range(df$NAP),
       ylim=range(df$Richness)  )
}
```



Random intercepts model

We will fit linear regression lines to each group (`Beach`) as follows:

$$y_i \sim \text{normal}(a_{group(i)} + b \cdot x_i, \sigma) \quad i = 1, ..., n \quad (n \text{ observations})$$
$$a_j \sim \text{normal}(\mu_a, \sigma_a) \quad j = 1, ..., m \quad (m \text{ groups})$$

Here, a_j is a group-level intercept, which are allowed to vary (partial pooling). But we assume identical slope b for all groups (complete pooling). So this is a random intercepts linear regression, lm-formulation would be `y ~ (1|group) + x`.

The approach is also similar to frequentist ANCOVA.

The Stan code differs from the previous model by adding the covariate x and parameter b (slope), that's it!

```
data = list(y = df$Richness,
            x = df$NAP,
            group = df$Beach,
            n = nrow(df),
            n_group = 9)

data
```

```
## $y
##  [1] 11 10 13 11 10  8  9  8 19 17  6  1  4  3  3  1  3  3  1  4  3 22  6  0  6
## [26]  5  4  1  6  4  2  1  1  3  4  3  5  7  5  0  7 11  3  0  2
##
```

```
## $x
##  [1]  0.045 -1.036 -1.336  0.616 -0.684  1.190  0.820  0.635  0.061 -1.334
## [11] -0.976  1.494 -0.201 -0.482  0.167  1.768 -0.030  0.460  1.367 -0.811
## [21]  1.117 -0.503  0.729  1.627  0.054 -0.578 -0.348  2.222 -0.893  0.766
## [31]  0.883  1.786  1.375 -0.060  0.367  1.671 -0.375 -1.005  0.170  2.052
## [41] -0.356  0.094 -0.002  2.255  0.865
##
## $group
##  [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6 6 6 6 6 7 7 7 7 7 8 8 8
## [39] 8 8 9 9 9 9 9
##
## $n
## [1] 45
##
## $n_group
## [1] 9
```

```
stan_code_partpool = '
data {
  int n;
  int n_group;
  real y[n];
  real x[n];
  int group[n];
}
parameters {
  real a[n_group];
  real b;
  real<lower=0> sigma;
  real mu_a;
  real<lower=0> sigma_a;
}
model {
  // priors
  mu_a ~ normal(5,5);
  sigma_a ~ cauchy(0,10);
  for (j in 1:n_group){
    a[j] ~ normal(mu_a,sigma_a);
  }
  b ~ normal(0,10);
  sigma ~ normal(0,10);
  // likelihood
  for(i in 1:n){
    y[i] ~ normal(a[ group[i] ]+b*x[i], sigma);
  }
}
'
```

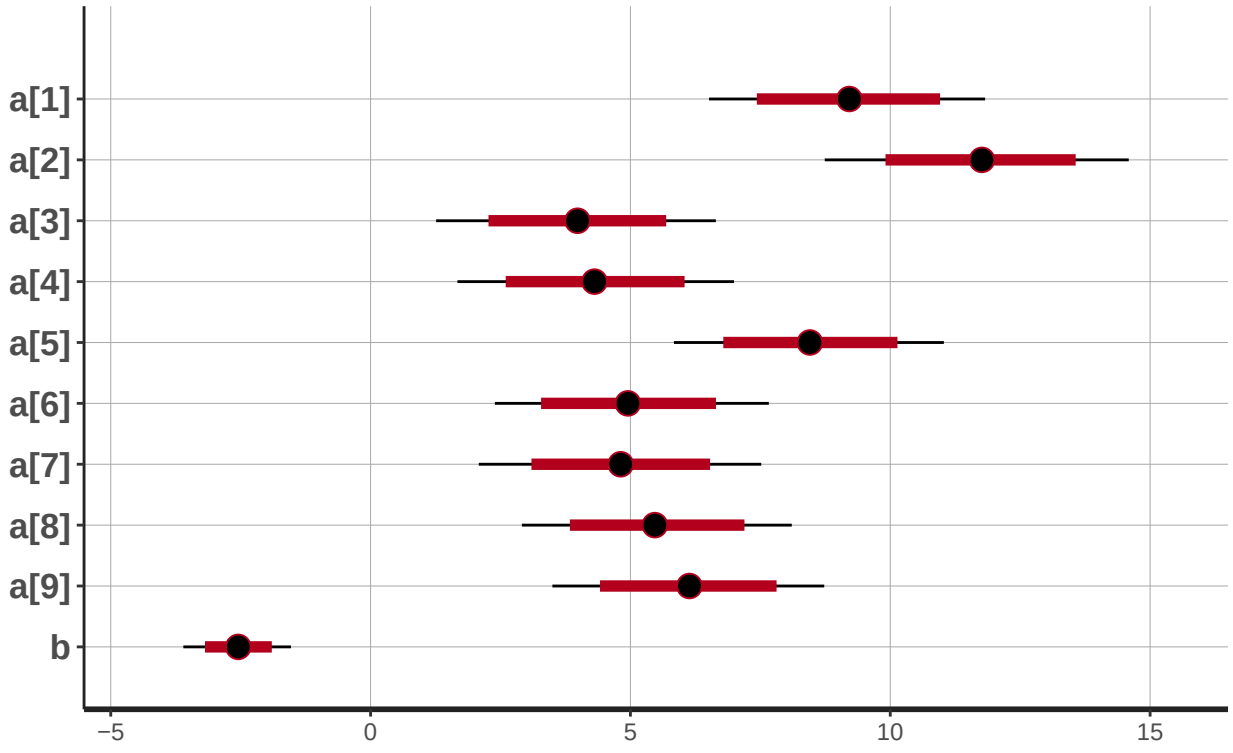
```
stan_model_partpool = stan_model(model_code=stan_code_partpool)
fit_partpool  = sampling(stan_model_partpool, data=data)
```

```
print(fit_partpool, digits=3, probs=c(0.025, 0.975))
```

```
## Inference for Stan model: 22022d02afacd5e109664cf4b0256974.
## 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[1]      9.206   0.022 1.369   6.514  11.823  3927 1.000
## a[2]     11.747   0.025 1.459   8.740  14.589  3507 1.000
## a[3]      3.982   0.022 1.352   1.262   6.644  3880 1.000
## a[4]      4.308   0.025 1.356   1.673   6.993  3043 1.000
## a[5]      8.463   0.020 1.319   5.838  11.030  4175 1.000
## a[6]      4.954   0.021 1.334   2.392   7.663  4178 0.999
## a[7]      4.792   0.023 1.367   2.082   7.517  3478 1.001
## a[8]      5.481   0.021 1.331   2.901   8.104  3868 1.000
## a[9]      6.124   0.019 1.327   3.499   8.728  4636 1.000
## b        -2.555   0.009 0.514  -3.603  -1.532  3248 1.000
## sigma     3.177   0.008 0.403   2.514   4.071  2640 1.002
## mu_a      6.481   0.022 1.243   3.901   8.908  3254 1.000
## sigma_a   3.390   0.023 1.152   1.686   6.264  2515 1.000
## lp__    -86.309   0.079 2.929 -92.987 -81.691  1390 1.002
##
## Samples were drawn using NUTS(diag_e) at Tue Oct 12 11:23:00 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).
```

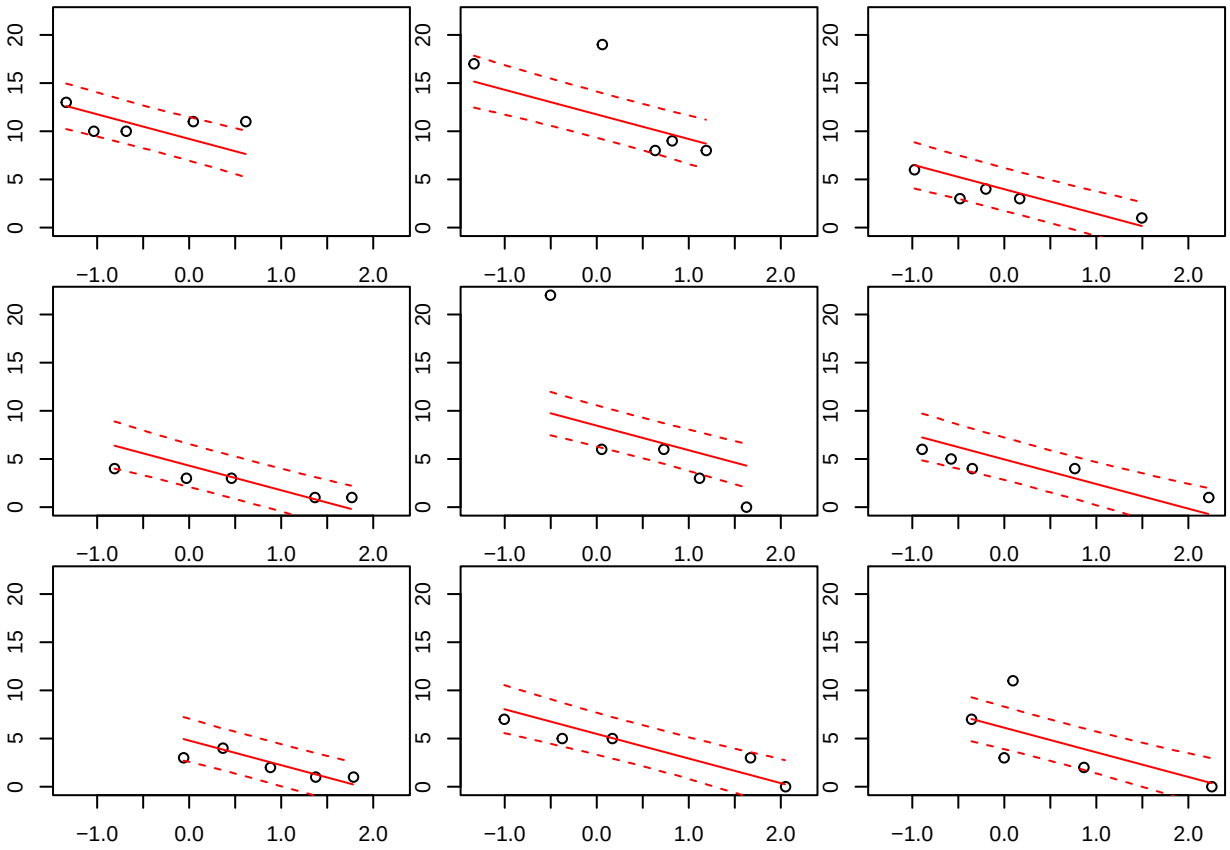
```
plot(fit_partpool)
```



Predictions / credible intervals

Again, we can generate predictions and compute credible intervals (for the deterministic part of the model). Here: 90% credible intervals.

```r
posterior=as.matrix(fit_partpool)

par(mfrow=c(3,3), mar=c(1,1,1,1), oma=c(1,1,0,0))
for (i in 1:9){
  df.sub=subset(df, df$Beach==i)
  x.pred = seq(from=min(df.sub$NAP), to=max(df.sub$NAP), by=0.01)
  plot(df.sub$NAP,  df.sub$Richness,
       xlim=range(df$NAP),
       ylim=range(df$Richness)  )

  y.cred = matrix(0, nrow=nrow(posterior), ncol=length(x.pred))
  for(j in 1:nrow(posterior)){
    # column i in posterior corresponds to a_i, alternatively reference by name:
    # posterior[j,paste0("a[",i,"]")]
    y.cred[j, ] = posterior[j,i] + posterior[j,"b"]*x.pred
  }

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

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

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



brms complete pooling

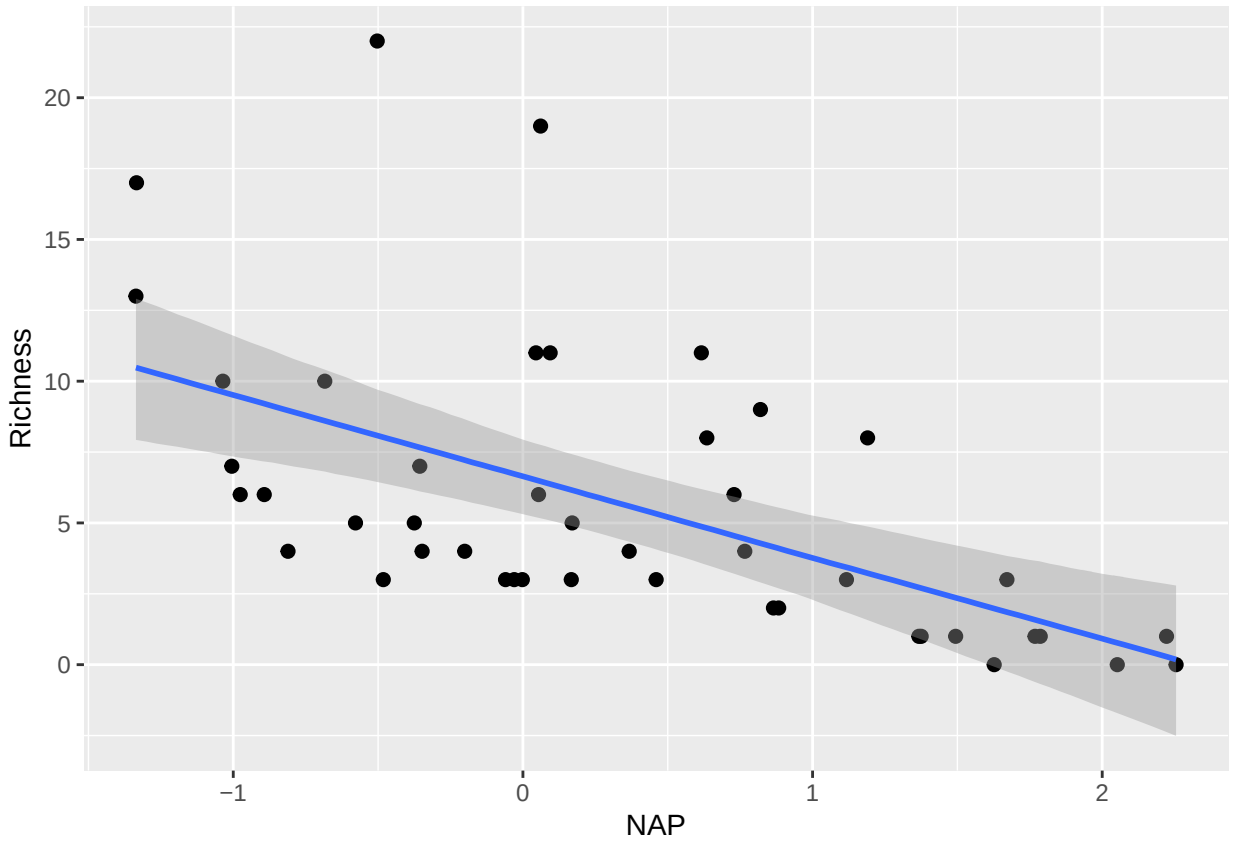
We start with the "complete pooling" model in `brms`, which fits just a linear regression on `NAP`, ignoring the effect of predictor `Beach` on the intercept.

```
fit.b.compl = brm( Richness ~ NAP,
                   data=df )
```

```
fit.b.compl
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: Richness ~ NAP
##    Data: df (Number of observations: 45)
##   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     6.64      0.67     5.31     7.94 1.00     3382     2541
## NAP          -2.87      0.64    -4.13    -1.60 1.00     3834     2667
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.24      0.46     3.46     5.25 1.00     3193     2526
##
## 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).
```

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



brms no pooling

Next, "no pooling" fits different intercepts for all level of the categorical predictor `Beach`, but same effect of `NAP`. So the term "no pooling" isn't exactly right here since all groups share a slope. Here it just means that intercepts are independent.

But first we must code `Beach` as a factor, otherwise it would be interpreted as a continuous predictor.

```
str(df)
```

```
## 'data.frame':    45 obs. of  5 variables:
##  $ Sample  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Richness: int  11 10 13 11 10 8 9 8 19 17 ...
##  $ Exposure: int  10 10 10 10 10 8 8 8 8 8 ...
##  $ NAP     : num  0.045 -1.036 -1.336 0.616 -0.684 ...
##  $ Beach   : int  1 1 1 1 1 2 2 2 2 2 ...
```

```
df$Beach = as.factor(df$Beach)
```

```
fit.b.no = brm( Richness ~ NAP + Beach,
                data=df )
```

As usual with categorical predictors, `lm()` or `brm()` uses dummy coding. I.e. the presented effects are not the different intercepts for all levels of `Beach`. The intercept for the first level of `Beach` is `Intercept`, but `Beach2` etc are the differences in intercept for the other levels. `NAP` (effect name `b_NAP`) is the slope of `Richness` vs `NAP` for all levels of `Beach`.

```
fit.b.no
```

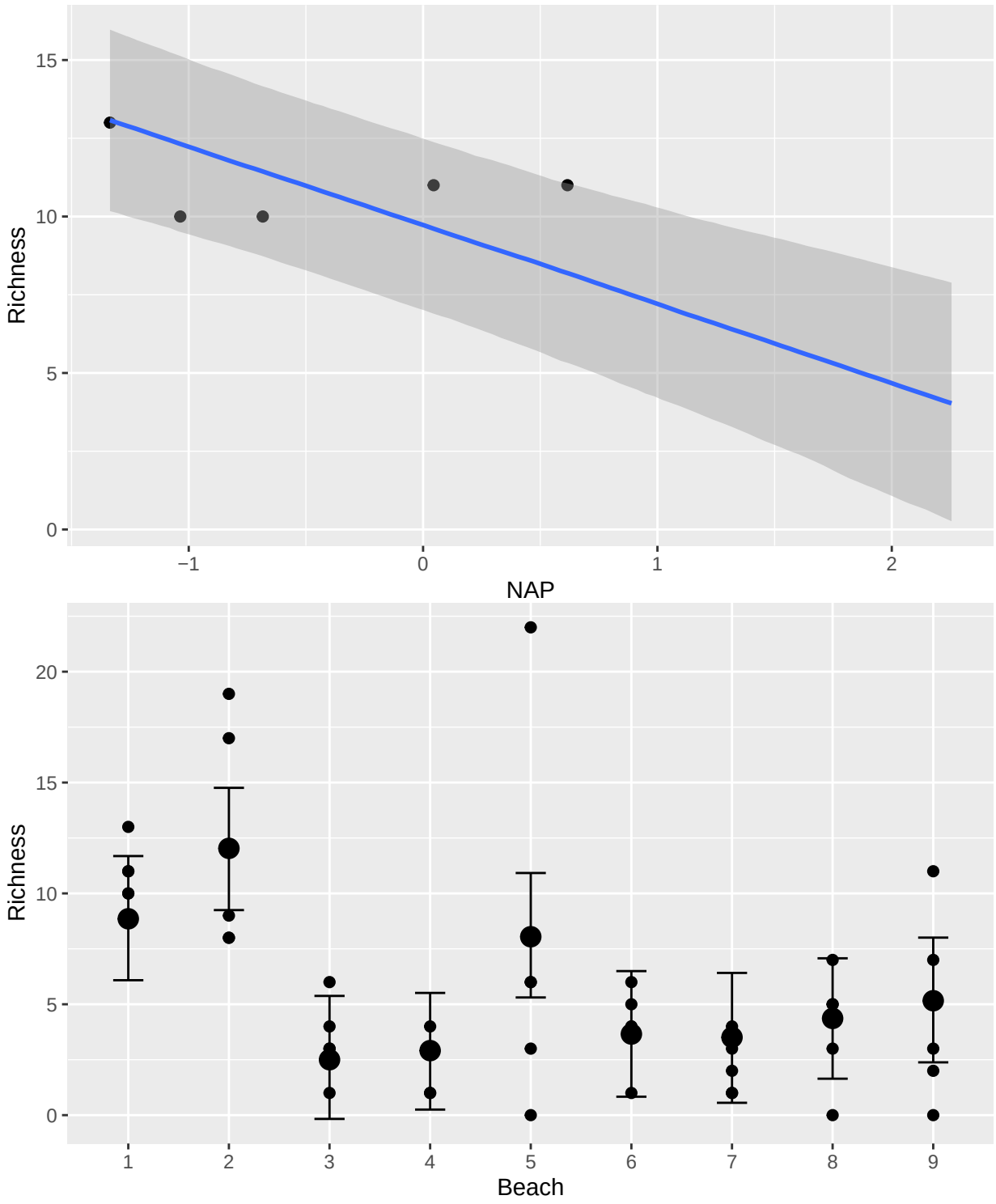
```
##  Family: gaussian
```

```
##   Links: mu = identity; sigma = identity
## Formula: Richness ~ NAP + Beach
##    Data: df (Number of observations: 45)
##   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     9.72      1.41     7.01    12.48 1.00      908     1601
## NAP          -2.52      0.52    -3.52    -1.46 1.00     2847     2508
## Beach2        3.17      1.99    -0.78     7.11 1.00     1312     2119
## Beach3       -6.31      1.95   -10.02    -2.50 1.00     1331     2225
## Beach4       -5.93      2.01    -9.98    -2.08 1.00     1258     2061
## Beach5       -0.79      2.05    -4.83     3.04 1.00     1167     2009
## Beach6       -5.17      2.00    -9.20    -1.23 1.00     1319     2114
## Beach7       -5.33      2.11    -9.45    -1.33 1.00     1206     2183
## Beach8       -4.47      2.01    -8.40    -0.56 1.00     1253     2317
## Beach9       -3.66      2.07    -7.67     0.38 1.00     1286     1955
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     3.15      0.38     2.51     3.98 1.00     3334     3108
##
## 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).
```

The convenient function `conditional_effects()` plots fitted values, separately for each predictor. In the first plot, the effect of `NAP` is shown (whole range), but the categorical predictor `Beach` is fixed at its first level. So this figure isn't particularly helpful if we want to compare all observed and predicted values. See below how to fix this.

In the second plot, the effect of `Beach` is shown (all levels), with the continuous predictor `NAP` fixed at its mean.

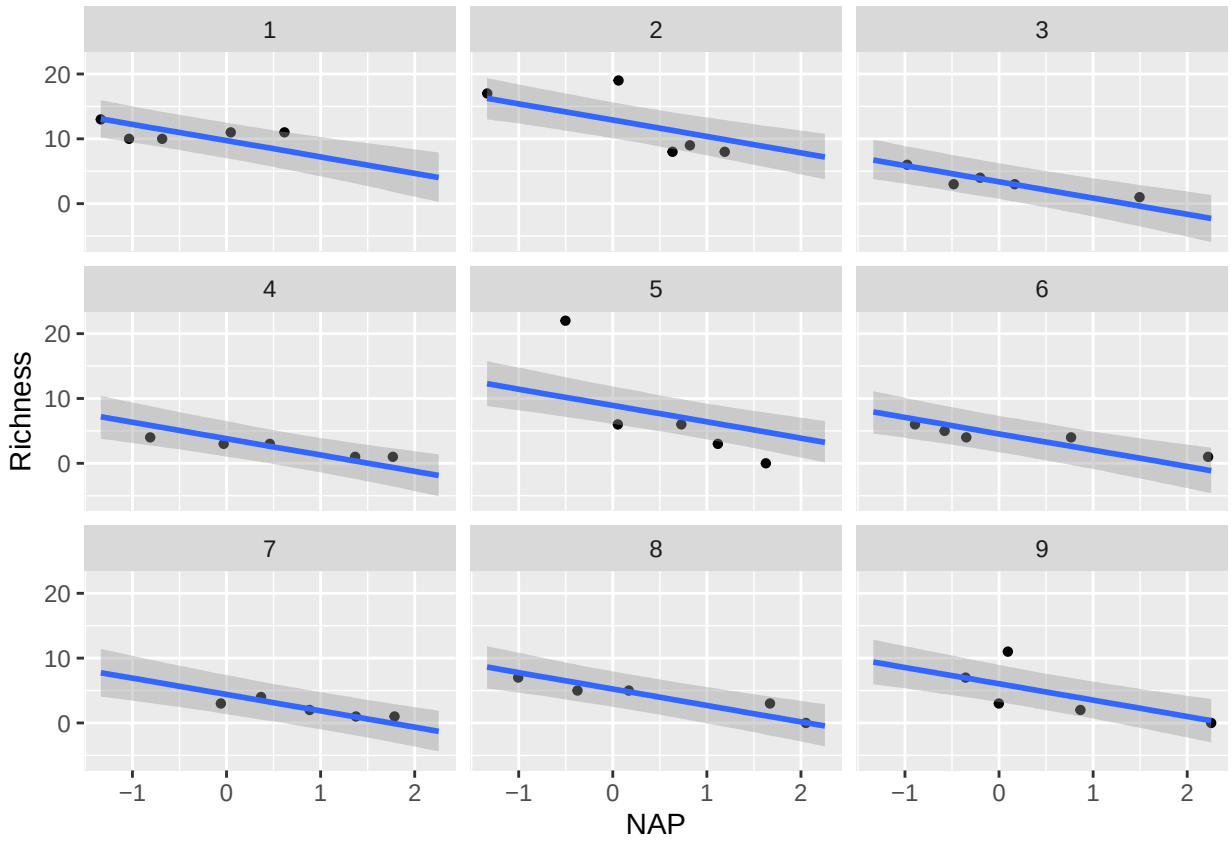
```
plot( conditional_effects(fit.b.no),
      points=TRUE,
      ask=FALSE)
```



By using `conditions=` and specifying all levels of `Beach`, effect of `NAP` is shown for all levels. Slope is identical in all levels, just the intercept changes.

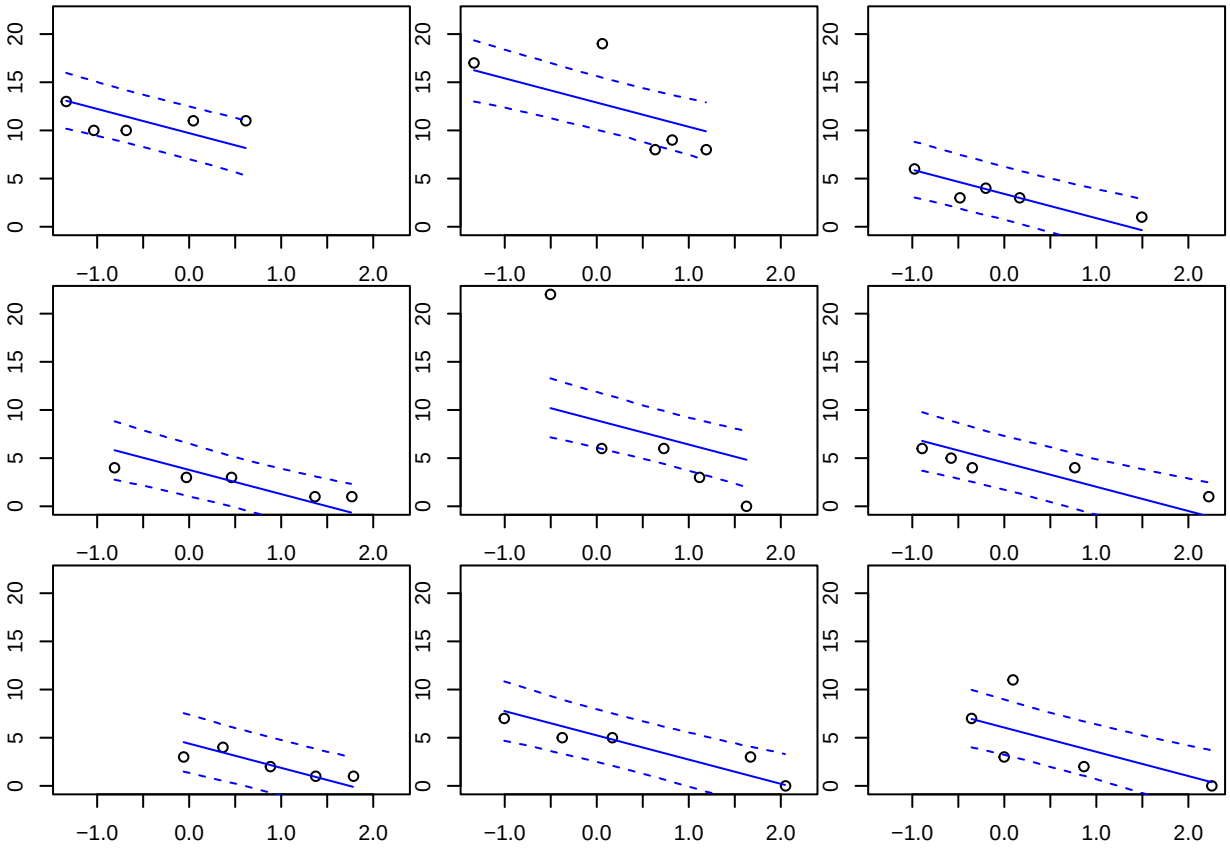
```
plot( conditional_effects(fit.b.no,
                          effects = "NAP",
                          conditions = data.frame( Beach=levels(df$Beach) ) ),
      points=TRUE,
```

```
       ask=FALSE )
```



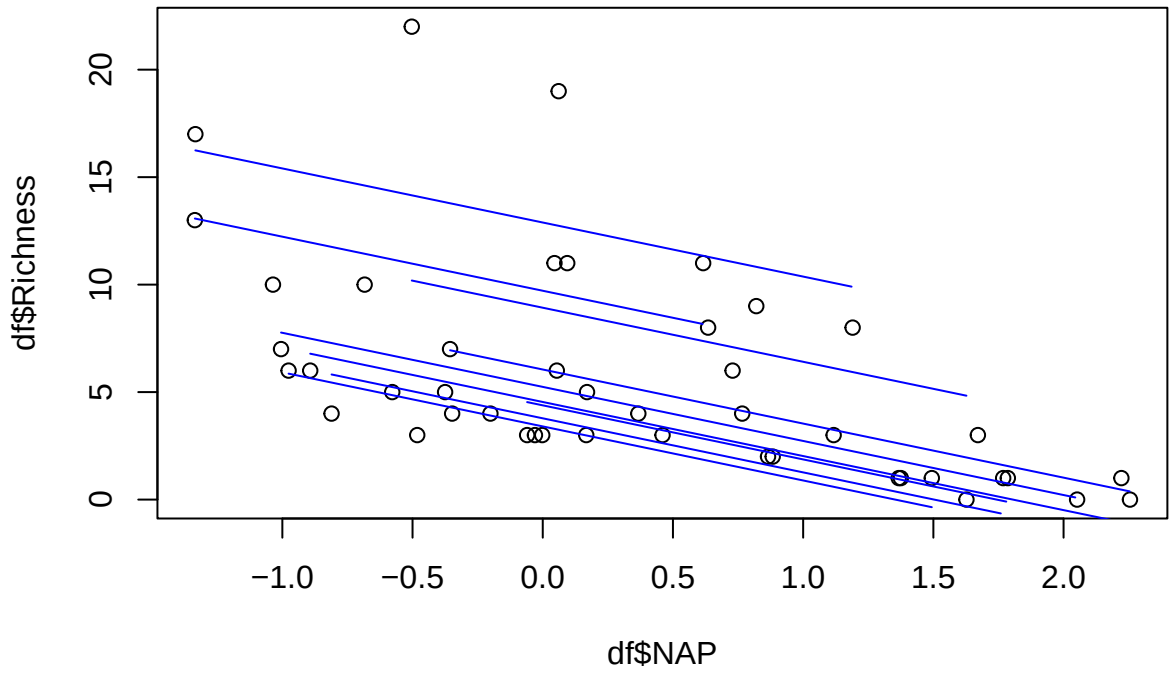
The group-level predictions can also be extracted by hand using `fitted()`. We can specify the range of `NAP` and levels of `Beach` with `newdata`. We plot 95% credible intervals.

```
levels = levels(df$Beach)
par(mfrow=c(3,3), mar=c(1,1,1,1), oma=c(1,1,0,0))

for (i in 1:9){
  df.sub=subset(df, df$Beach==i)
  x.pred = seq(from=min(df.sub$NAP), to=max(df.sub$NAP), by=0.01)
  plot(df.sub$NAP,  df.sub$Richness,
       xlim=range(df$NAP),
       ylim=range(df$Richness)  )

  y.cred = fitted(fit.b.no, newdata=data.frame(NAP=x.pred,
                                               Beach=levels[i] ) )

  lines(x.pred, y.cred[, 1], col="blue")
  lines(x.pred, y.cred[, 3], col="blue", lty=2)
  lines(x.pred, y.cred[, 4], col="blue", lty=2)
}
```



Or we can plot the whole dataset with all group-level mean regression lines.

```
plot(df$NAP, df$Richness)

for (i in 1:9){
  df.sub=subset(df, df$Beach==i)
  x.pred = seq(from=min(df.sub$NAP), to=max(df.sub$NAP), by=0.01)
  y.cred = fitted(fit.b.no, newdata=data.frame(NAP=x.pred,
                                               Beach=levels[i] ) )
  lines(x.pred, y.cred[, 1], col="blue")
}
```



brms partial pooling

With partial pooling, we use a random intercept for the categorical predictor `Beach`.

```
fit.b.part = brm( Richness ~ NAP + (1|Beach),
                  data=df )
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: Richness ~ NAP + (1 | Beach)
##    Data: df (Number of observations: 45)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Group-Level Effects:
## ~Beach (Number of levels: 9)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     3.23      1.09     1.68     5.85 1.00     1042     1646
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     6.44      1.19     4.00     8.81 1.00     1366     2055
## NAP          -2.57      0.51    -3.59    -1.57 1.00     3410     2742
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     3.16      0.40     2.53     4.06 1.00     2635     2314
##
## 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).
```

In the summary table, fixed effect `NAP` (slope b) and mean `Intercept` (overall mean μ_a) are presented.

`sd(Intercept)` is the variation σ_a of group-level intercepts a_j.

Again, instead of fitting all intercepts a_j, `brm()` fits the differences $\alpha_j = a_j - \mu_a$. The model reads

$$y_i \sim \text{normal}(\mu_a + \alpha_{group(i)} + b \cdot x, \sigma), \quad i = 1, ..., n \quad (n \text{ observations})$$
$$\alpha_j \sim \text{normal}(0, \sigma_a), \quad j = 1, ..., m \quad (m \text{ groups})$$

but it's equivalent to the model above from the Stan code.

Random effects (differences α_j) are extraced via:

```
ranef(fit.b.part)
```

```
## $Beach
## , , Intercept
##
##     Estimate Est.Error       Q2.5     Q97.5
## 1  2.6666167  1.709030 -0.5972960 6.1351324
## 2  5.2281726  1.749419  1.9329418 8.7760233
## 3 -2.4425661  1.644030 -5.6873020 0.7125342
## 4 -2.0762038  1.642906 -5.3980155 1.1129168
## 5  2.0413587  1.635892 -0.9929831 5.5002424
## 6 -1.4648570  1.662044 -4.9059740 1.9194479
## 7 -1.6222387  1.649032 -4.8397260 1.6130859
## 8 -0.9050957  1.628436 -4.2315553 2.2439116
## 9 -0.2856135  1.636130 -3.5087505 2.9765703
```

`brm` automatically provides defualt priors for the μ_a (`Intercept`), σ_a (`sd`) and σ (`sigma`). But no prior (aka flat prior) for the slope b of `NAP` is given.

```
print(prior_summary(fit.b.part, all = FALSE), show_df = FALSE)
```

```
## Intercept ~ student_t(3, 4, 4.4)
## <lower=0> sd ~ student_t(3, 0, 4.4)
## <lower=0> sigma ~ student_t(3, 0, 4.4)
```

We can assign our own prior for μ_a and the effect size b of `NAP`.

```
priors = c(prior(normal(5,5), class=Intercept),
           prior(normal(0,10), class=b) )
```

```
fit.b.part = brm( Richness ~ NAP + (1|Beach),
                  prior=priors,
                  data=df )
```

```
fit.b.part
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: Richness ~ NAP + (1 | Beach)
##    Data: df (Number of observations: 45)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Group-Level Effects:
## ~Beach (Number of levels: 9)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     3.19      1.03     1.61     5.63 1.00     1191     1822
```

```
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     6.58      1.18     4.13     8.85 1.00     1377     1806
## NAP          -2.57      0.51    -3.57    -1.58 1.00     3803     2847
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     3.18      0.41     2.53     4.09 1.00     2659     2843
##
## 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).
```

```
ranef(fit.b.part)
```

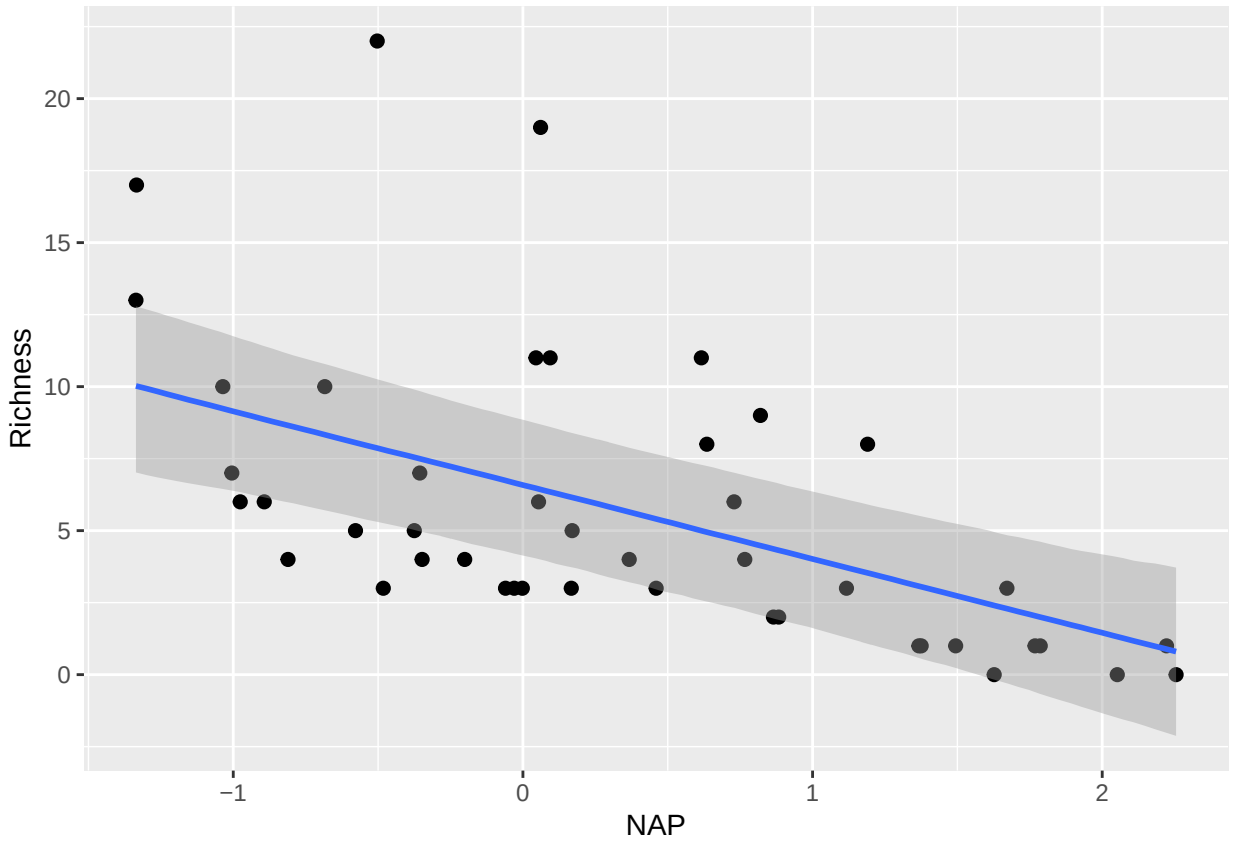
```
## $Beach
## , , Intercept
##
##     Estimate Est.Error       Q2.5     Q97.5
## 1  2.5662339  1.670542 -0.5292381 5.9356385
## 2  5.0565911  1.791404  1.6913392 8.6547675
## 3 -2.5490207  1.646364 -5.8020496 0.5908907
## 4 -2.2091771  1.657349 -5.5205343 1.0147003
## 5  1.9152395  1.658822 -1.3366014 5.3375810
## 6 -1.5797281  1.615251 -4.8895151 1.4580609
## 7 -1.6970358  1.633132 -5.0007588 1.5246646
## 8 -1.0667126  1.565259 -4.2474466 2.0261986
## 9 -0.4409244  1.610306 -3.6273263 2.7079669
```

```
print(prior_summary(fit.b.part, all = FALSE), show_df = FALSE)
```

```
## b ~ normal(0, 10)
## Intercept ~ normal(5, 5)
## <lower=0> sd ~ student_t(3, 0, 4.4)
## <lower=0> sigma ~ student_t(3, 0, 4.4)
```

Now, let's look at observed and predicted. `conditional_effects()` plots fixed effects only by default. The whole dataset is shown and predictions with joint mean b and mean intercept μ_a.

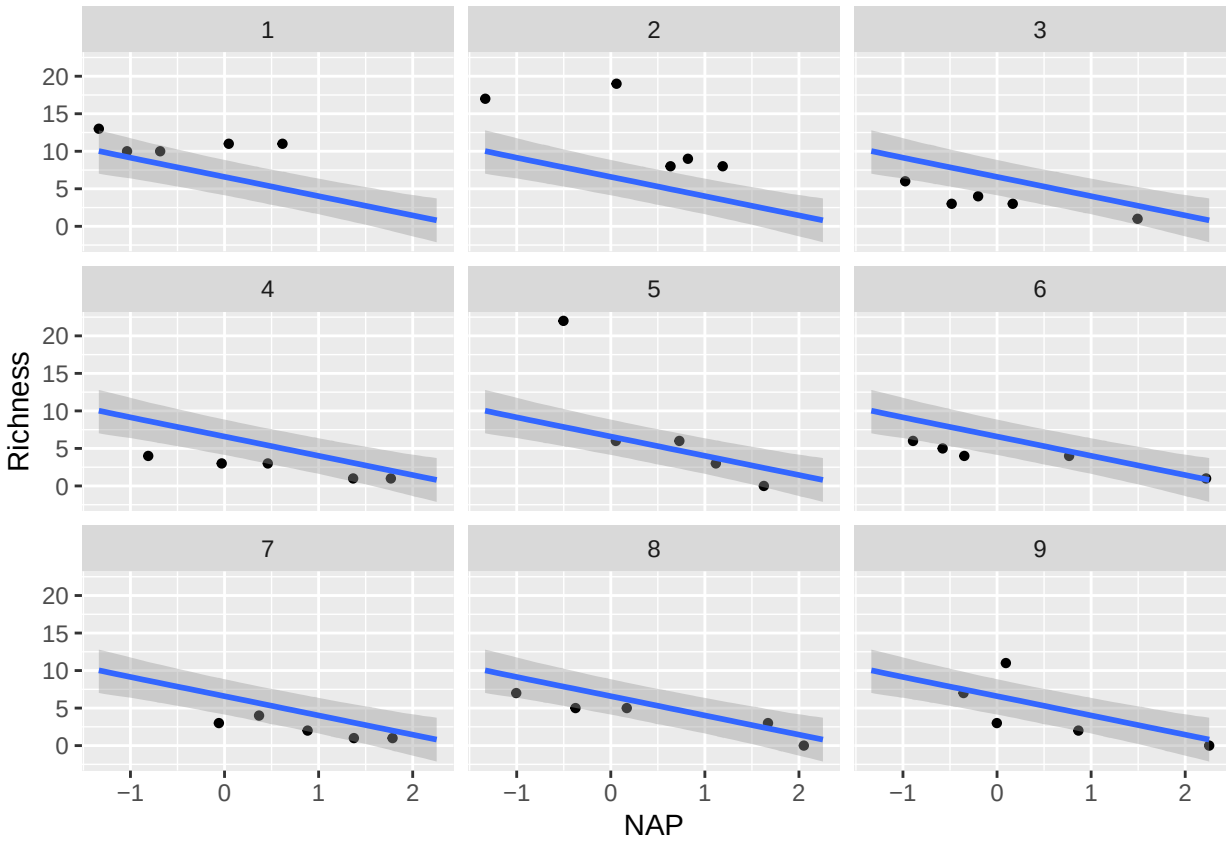
```
plot( conditional_effects(fit.b.part),
      points=TRUE,
      ask=FALSE )
```



When specifying group-level predictors (all levels) with `conditions=`, we receive a warning that they are not part of the (fixed effects) model. We see a plot for all levels of `Beach`, but the regression line is the same: predictions for fixed effects part only!

```
plot( conditional_effects(fit.b.part,
                          effects = "NAP",
                          conditions = data.frame(Beach=levels(df$Beach)) ),
      points=TRUE,
      ask=FALSE )
```

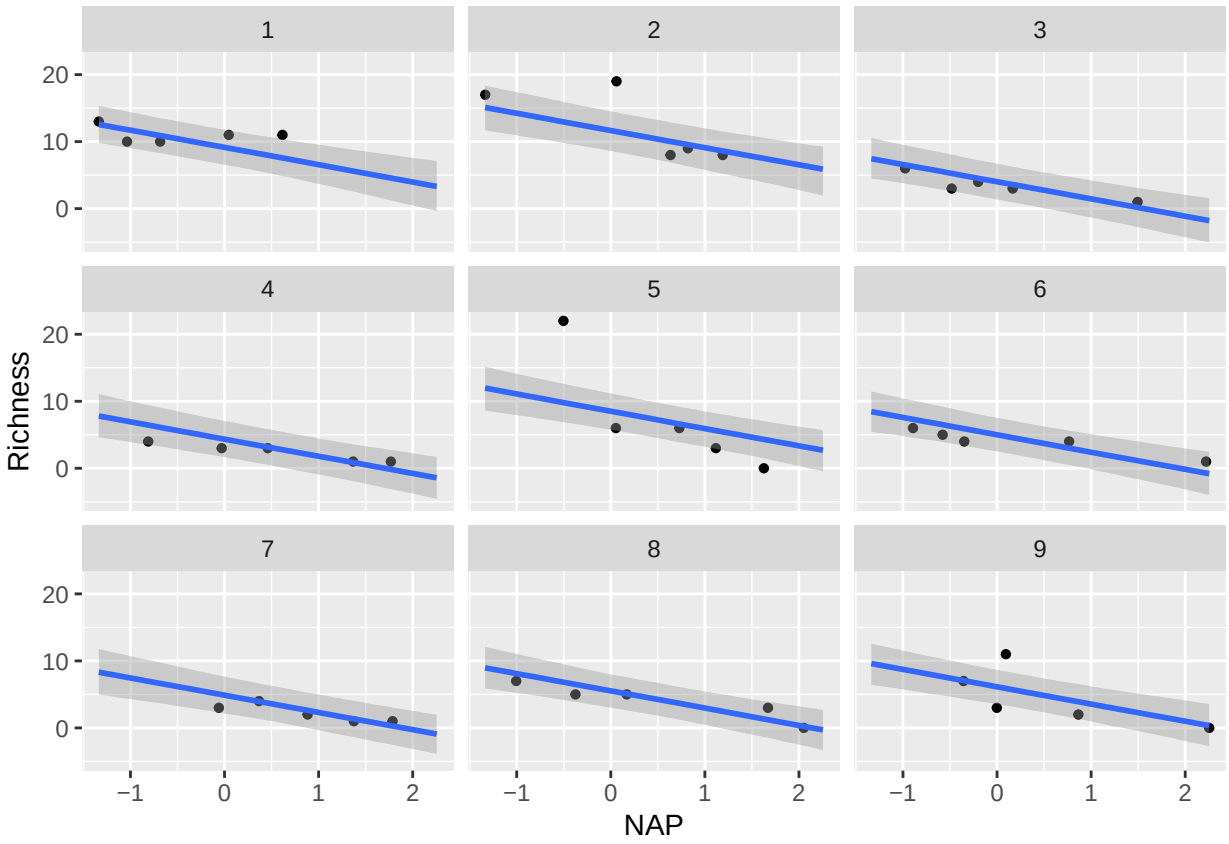
```
## Warning: The following variables in 'conditions' are not part of the model:
## 'Beach'
```



To include also the random effects for model predictions, `re_formula=NULL` must be specified. This reads a little weird (I would have expected something like `re_formula=TRUE`), but the default for no random effects as above is `re_formula=NA` and `NULL` is the command for using random effects for prediction here.

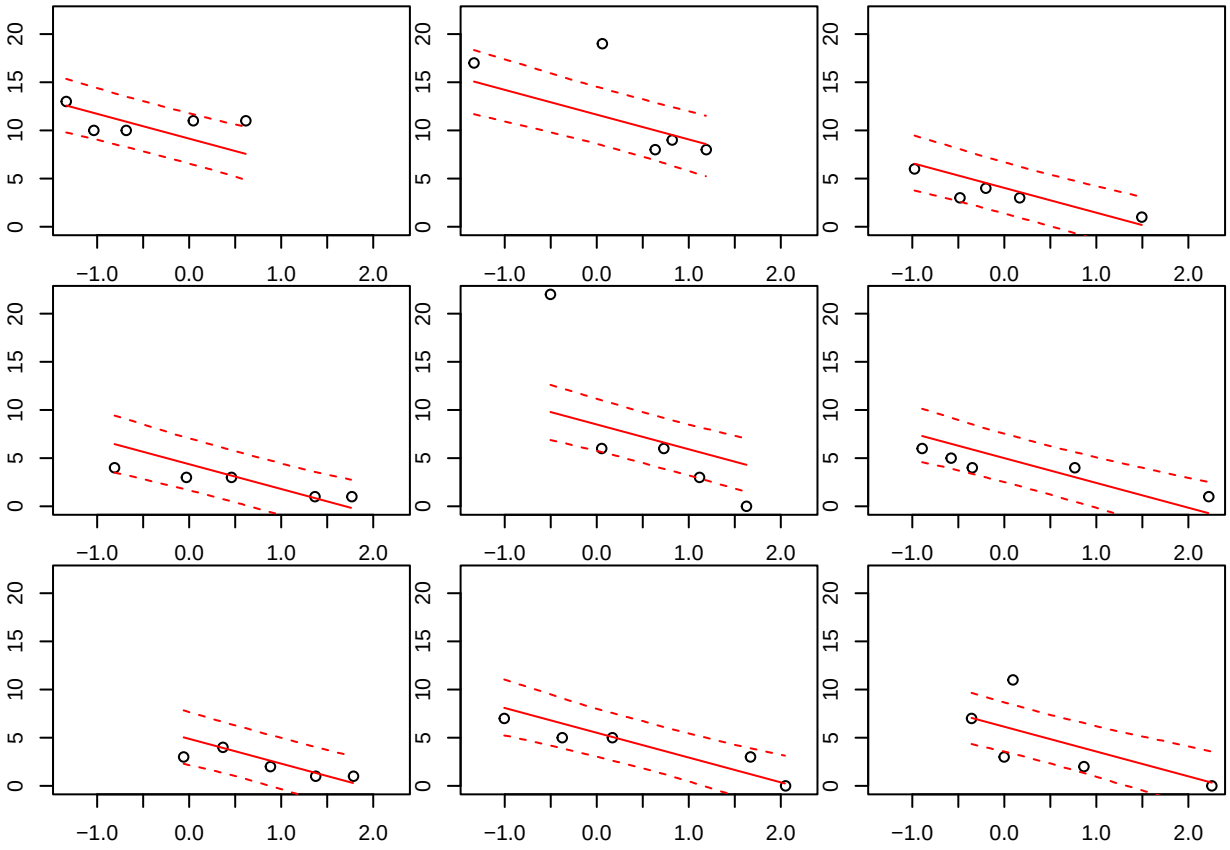
Slopes are identical, but intercepts vary between groups.

```
plot( conditional_effects(fit.b.part,
                          effects = "NAP",
                          re_formula = NULL,
                          conditions = data.frame(Beach=levels(df$Beach)) ),
      points=TRUE,
      ask=FALSE )
```



Same as in the "no pooling" model, group-level predictions can also be extracted by hand using `fitted()`.

```
levels = levels(df$Beach)

par(mfrow=c(3,3), mar=c(1,1,1,1), oma=c(1,1,0,0))
for (i in 1:9){
  df.sub=subset(df, df$Beach==i)
  x.pred = seq(from=min(df.sub$NAP), to=max(df.sub$NAP), by=0.01)
  plot(df.sub$NAP,  df.sub$Richness,
       xlim=range(df$NAP),
       ylim=range(df$Richness)  )

  y.cred = fitted(fit.b.part, newdata=data.frame(NAP=x.pred,
                                                 Beach=levels[i] ) )

  lines(x.pred, y.cred[, 1], col="red")
  lines(x.pred, y.cred[, 3], col="red", lty=2)
  lines(x.pred, y.cred[, 4], col="red", lty=2)
}
```



We can plot the whole dataset with all group-level mean regression lines, both for the previous "no pooling" model and the current "partial pooling" model. With partial pooling, intercepts a_j are drawn towards the overall mean intercept μ_a compared to no pooling.

```
plot(df$NAP, df$Richness)

# partial pooling
for (i in 1:9){
  df.sub=subset(df, df$Beach==i)
  x.pred = seq(from=min(df.sub$NAP), to=max(df.sub$NAP), by=0.01)
  y.cred = fitted(fit.b.part, newdata=data.frame(NAP=x.pred,
                                                 Beach=levels[i] ) )
  lines(x.pred, y.cred[, 1], col="red")
}

x.pred = seq(from=min(df$NAP), to=max(df$NAP), by=0.01)
y.cred = fitted(fit.b.part, newdata=data.frame(NAP=x.pred,
                                               Beach=NA) )

# partial pooling mean
lines(x.pred, y.cred[, 1], col="red", lwd=3, lty=3)

# no pooling (previous model)
for (i in 1:9){
  df.sub=subset(df, df$Beach==i)
  x.pred = seq(from=min(df.sub$NAP), to=max(df.sub$NAP), by=0.01)
  y.cred = fitted(fit.b.no, newdata=data.frame(NAP=x.pred,
```

```
                                                Beach=levels[i] ) )
  lines(x.pred, y.cred[, 1], col="blue")
}

legend("topright",
       legend=c("no pooling","partial pooling", "partial pooling mean"),
       lwd=c(1,1,3),
       lty=c(1,1,3),
       col=c("blue","red","red"),
       bty="n")
```