4.1 Practical: hierarchical models

Benjamin Rosenbaum

October 27, 2022

Aim: learn what a random effect is.

Learn about **hierarchical** levels of parameters. Bayesian statistics is great for hierarchical models! Random / mixed effects models (LMM, GLMM) can be more robust in a Bayesian framework than frequentist / maximum likelihood. Even nonlinear models with random / mixed effects are possible.

Think in terms of **no pooling**, **partial pooling** and **complete pooling** of parameters rather than **random effects** and **fixed effects**.

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

We load an example dataset from Zuur et al. 2009 “Mixed Effects Models and Extensions in Ecology with R”. It contains Species Richness data for five samples each taken on different beaches. Additional covariate NAP is the height of a sampling station compared to mean tidal level.

We want to estimate mean species richness, and in this first session we’ll ignore the predictor NAP and concentrate on fitting means only.

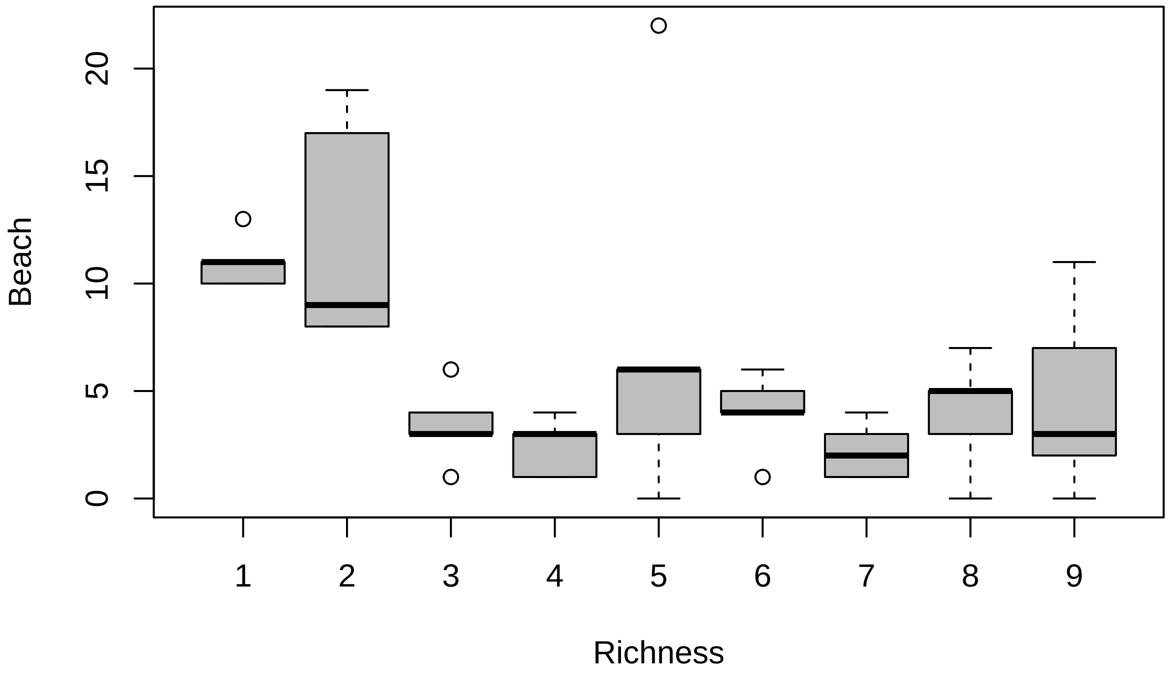
```
df = read.table("data/RIKZ.txt", header=TRUE)
head(df)
```

```
##   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
```

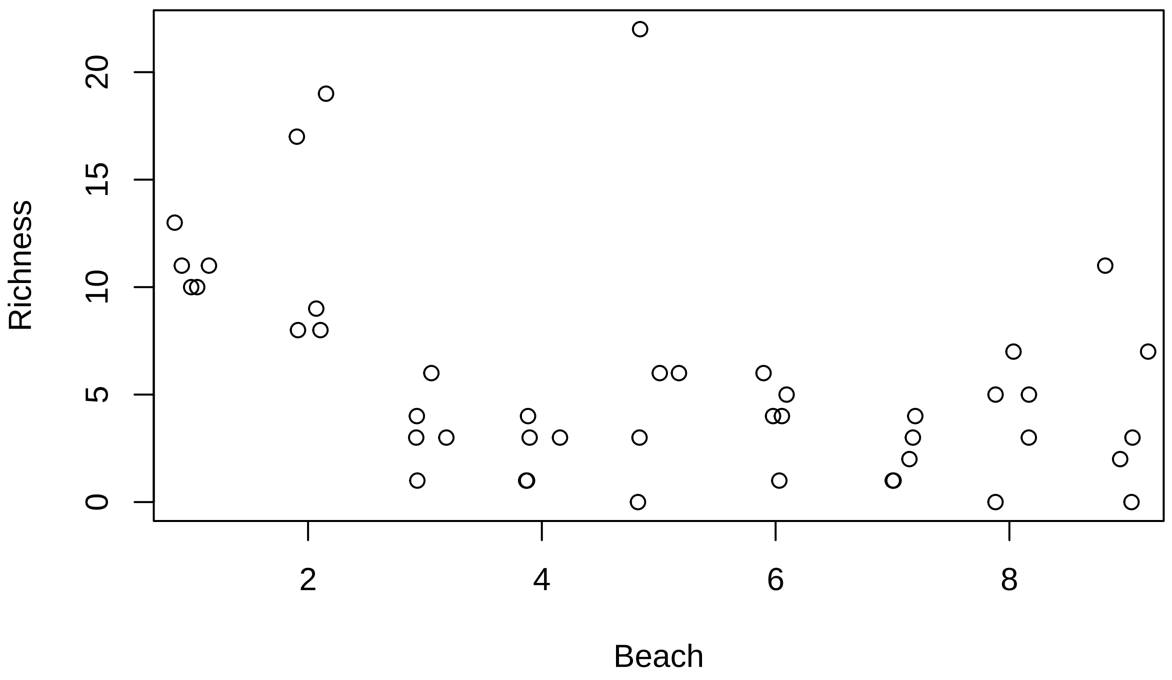
```
str(df)
```

```
## '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 ...
```

```
boxplot(Richness ~ Beach, data=df,
        ylab="Beach",
        xlab="Richness",
        col="grey")
```



```
plot(0, 0, xlim =c(1,9), ylim = range(df$Richness), type = "n",
     ylab="Richness",
     xlab="Beach")
points(Richness ~ jitter(as.numeric(Beach), factor=1), data=df)
```



The following approach is similar to frequentist ANOVA.

No pooling / Fixed effects model

We want to estimate the mean species richness per beach (=group), and test if there are differences between groups.

The statistical model is identical to yesterday's "t-test" model, but now there are more than 2 groups:

$$y_i \sim \text{normal}(a_{group(i)}, \sigma), \quad i = 1, ..., n \quad (n \text{ observations})$$

Note that there are no assumptions on a_j, i.e. the means of the different groups are estimated independently. They are assigned prior distributions, e.g.

$$a_j \sim \text{normal}(0, 10), \quad j = 1, ..., m \quad (m \text{ groups})$$

We use (very) weakly informative prior information on the a_j, but note that the priors for a_j are assigned independently. They could also have an individual prior distribution each if we want to.

"group" is treated as a fixed effect. Because there can be misunderstandings when using the terms "fixed effect" / "random effects",this is also called **"no pooling"**. No information on the a_j is pooled across the groups. Keep that in mind for later!

Here, we used the same variance across all groups (σ), but we could also use different sigmas.

When preparing the data for Stan, note that we use `as.integer()` to code the factorial variable `group`, so we can use it as an index.

```
data = list(y = df$Richness,
            group = as.integer(df$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
##
## $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
```

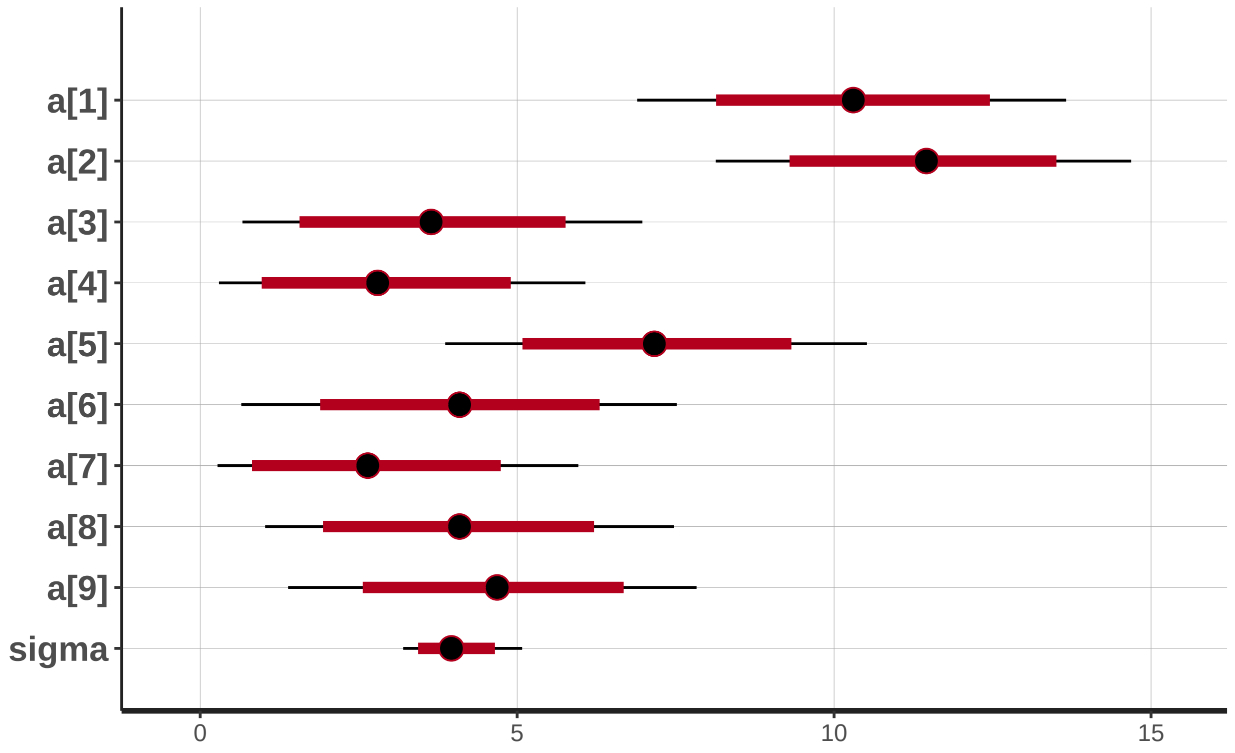
```
stan_code_nopool = '
data {
  int n;
  int n_group;
  real y[n];
  int group[n];
}
parameters {
  real<lower=0> a[n_group];
  real<lower=0> sigma;
}
model {
  // priors
  for (j in 1:n_group){
    a[j] ~ normal(5,5);
  }
  sigma ~ normal(0,10);
  // likelihood
  for(i in 1:n){
    y[i] ~ normal(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

```
stan_model_nopool = stan_model(model_code=stan_code_nopool)
fit_nopool  = sampling(stan_model_nopool, data=data)
```

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

```
## Inference for Stan model: adfaa9670b959b5ad1ecaa70c59a9e09.
## 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]   10.307   0.034 1.708  6.894 13.660  2544 1.000
## a[2]   11.438   0.029 1.665  8.134 14.685  3389 1.000
## a[3]    3.673   0.030 1.617  0.667  6.974  2864 1.000
## a[4]    2.892   0.037 1.518  0.296  6.076  1656 1.001
## a[5]    7.193   0.029 1.664  3.865 10.517  3300 1.000
## a[6]    4.108   0.044 1.716  0.648  7.519  1509 1.003
## a[7]    2.752   0.031 1.504  0.274  5.966  2398 1.000
## a[8]     4.114   0.035 1.651   1.024   7.477  2245 1.001
## a[9]     4.656   0.035 1.615   1.387   7.841  2089 1.000
## sigma    4.015   0.010 0.484   3.201   5.078  2561 1.000
## lp__   -71.644   0.106 3.029 -78.963 -67.214   817 1.002
##
## Samples were drawn using NUTS(diag_e) at Mon Oct 11 14:30:01 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_nopool)
```



As in the “t-test” example from yesterday, we can look at the individual differences between groups, also called “contrasts”. E.g., the posterior distribution of $a_4 - a_5$.

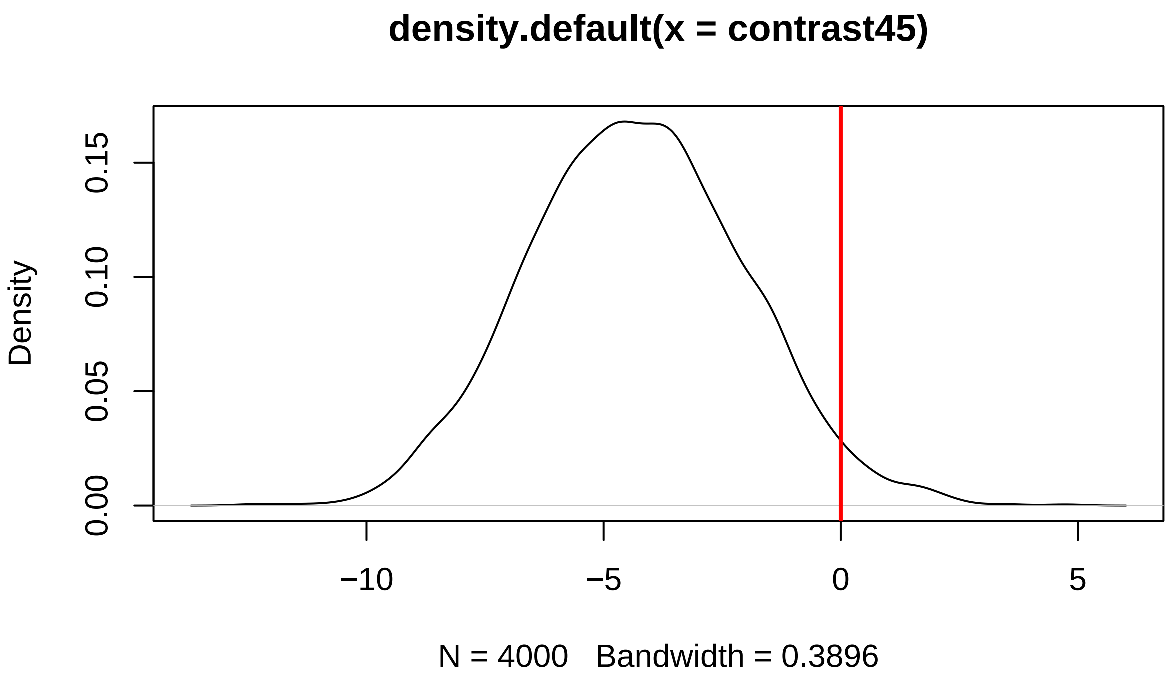
```
posterior_nopool = as.matrix(fit_nopool)

contrast45 = posterior_nopool[,"a[4]"]-posterior_nopool[,"a[5]"]

mean(contrast45)
```

```
## [1] -4.301041
```

```
sum(contrast45<0)/length(contrast45)
```

```
## [1] 0.97025
```

```
plot(density(contrast45))
abline(v=0, col="red", lwd=2)
```



Partial pooling / Random effects model

Until now, we assumed that the different group means a_j are completely independent and we fitted them all individually.

Assume we want to estimate overall species richness and we're not really interested in the exact richness of each beach. Then we can assume a joint mean μ_a across all groups (beaches) and each group mean a_j differs from this joint mean with a group-level standard deviation σ_a.

And we can explicitely model that:

$$\begin{aligned} y_i &\sim \text{normal}(a_{group(i)}, \sigma), \quad i = 1, ..., n \quad (n \text{ observations}) \\ a_j &\sim \text{normal}(\mu_a, \sigma_a), \quad j = 1, ..., m \quad (m \text{ groups}) \end{aligned}$$

This looks similar to the equations above, but instead of the prior distributions for a_j, we have a joint normal distribution with parameters mean μ_a and sdev σ_a. These are free parameters to be estimated, and get their own prior distributions, e.g.

$$\begin{aligned} \mu_a &\sim \text{normal}(0, 10) \\ \sigma_a &\sim \text{cauchy}(0, 1)^+ \end{aligned}$$

For the joint mean μ_a, we can use the same expectation we used for the group means before.

For the between-groups standard deviation σ_a, it is standard procedure to use positive half-Chauchy distributions (student-t distribution with 1 degree of freedom, more heavy-tailed than a normal distribution).

σ describes within-groups variation of response values as before and also gets its own prior.

Here, group is treated as a random effect. Some information is shared / pooled across groups. Each group data informs μ_a, σ_a, which then informs the other group means a_j and vice versa. This is also called **"partial pooling"**

Because a_j's distribution depends on μ_a and σ_a, this is a **hierarchical model**. μ_a is a **population-level** parameter (first in hierarchy), while a_j are **group-level** parameters (second in hierarchy).

The partial pooling model differs from the no pooling model above only in the joint distribution of the a_j and additional parameters μ_a and σ_a (with their own priors).

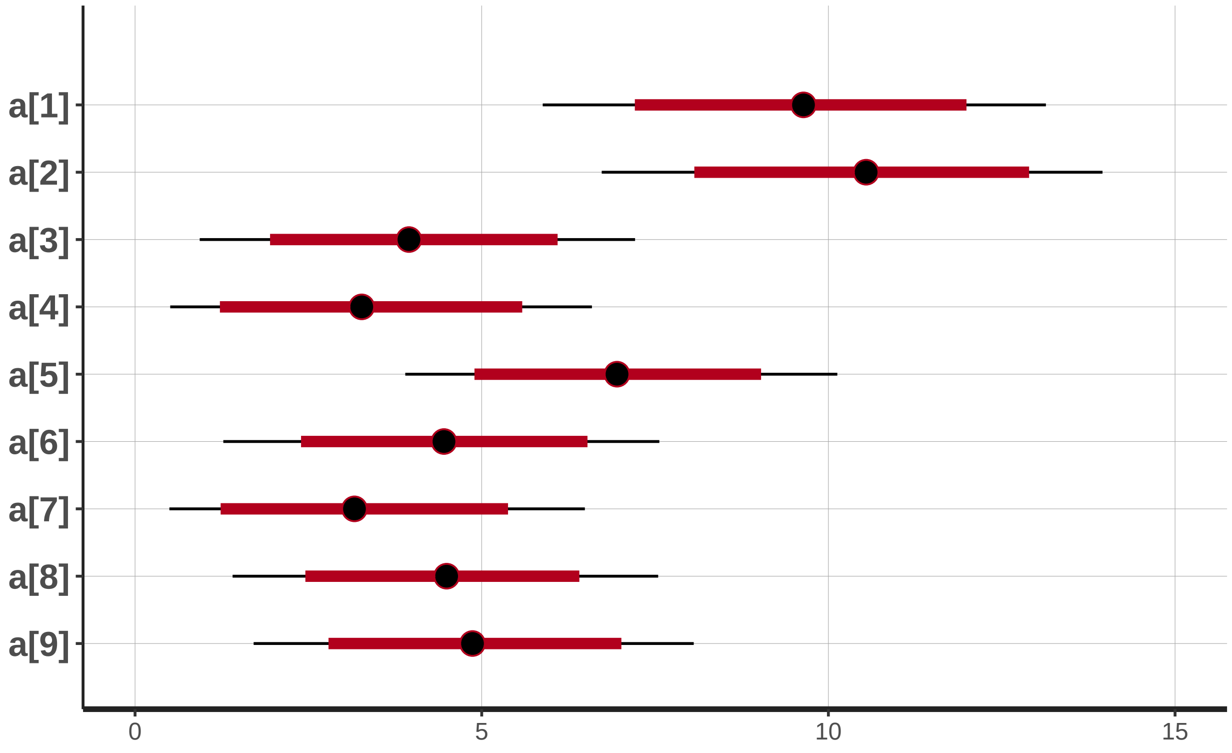
```
stan_code_partpool = '
data {
  int n;
  int n_group;
  real y[n];
  int group[n];
}
parameters {
  real<lower=0> a[n_group];
  real<lower=0> sigma;
  real<lower=0> mu_a;
  real<lower=0> sigma_a;
}
model {
  // priors
  mu_a ~ normal(5, 5);
  sigma_a ~ cauchy(0, 10);

  for (j in 1:n_group){
    a[j] ~ normal(mu_a, sigma_a);
  }
  sigma ~ normal(0,10);

  // likelihood
  for(i in 1:n){
    y[i] ~ normal(a[ group[i] ], sigma);
  }
}
'
```

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

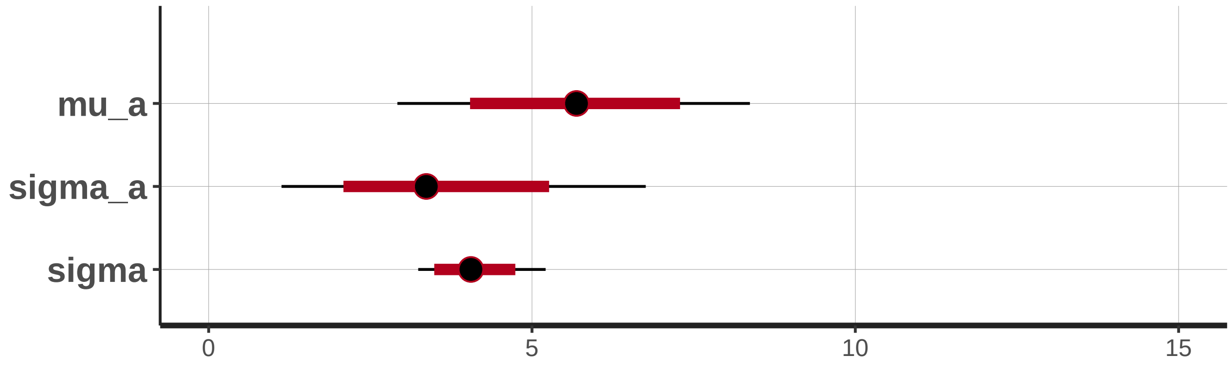
```
print(fit_partpool, digits=3, probs=c(0.025, 0.975))
```

```
## Inference for Stan model: c64ee8fef431865984092735c87e8060.
## 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]      9.621   0.065 1.853  5.880 13.150   807 1.006
## a[2]     10.518   0.058 1.861  6.731 13.954  1032 1.004
## a[3]      4.006   0.046 1.618  0.933  7.212  1235 1.004
## a[4]      3.333   0.052 1.621  0.506  6.589   975 1.006
## a[5]      6.964   0.033 1.617  3.898 10.132  2458 1.001
## a[6]      4.456   0.041 1.608  1.274  7.562  1540 1.004
## a[7]      3.258   0.050 1.583  0.495  6.488  1013 1.004
## a[8]      4.480   0.033 1.552  1.407  7.545  2251 1.000
## a[9]      4.891   0.034 1.620  1.711  8.057  2318 1.003
## sigma     4.099   0.013 0.503  3.242  5.211  1579 1.002
## mu_a      5.678   0.034 1.339  2.918  8.370  1517 1.004
## sigma_a   3.557   0.060 1.388  1.132  6.759   531 1.008
## lp__    -81.918   0.150 3.311 -89.339 -76.523   488 1.014
##
## Samples were drawn using NUTS(diag_e) at Mon Oct 11 14:31:47 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, pars="a") + xlim(c(0,15))
```



```
plot(fit_partpool, pars=c("mu_a", "sigma_a", "sigma")) + xlim(c(0,15))
```



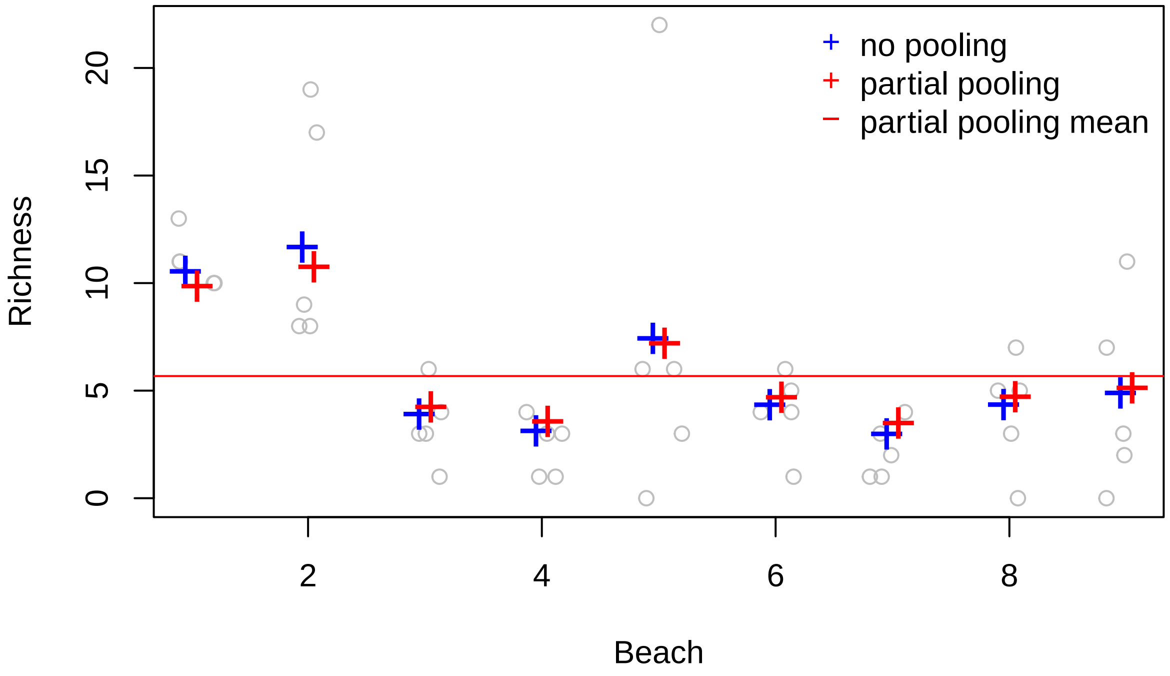
Comparison

Now we compare the results of the "no pooling" and the "partial pooling" models.

The mean estimates for the parameters a_j can directly be extracted from the summary table.

```
summary_nopool = summary(fit_nopool)$summary
summary_partpool = summary(fit_partpool)$summary
```

```
plot(0, 0, xlim =c(1,9), ylim = range(df$Richness), type = "n",
     ylab="Richness",
     xlab="Beach")
points(Richness ~ jitter(as.numeric(Beach), factor=1.0), data=df,
       col="grey")

points((1:9)-0.05, summary_nopool[1:9, "mean"], pch="+", col="blue", cex=2)
points((1:9)+0.05, summary_partpool[1:9, "mean"], pch="+", col="red", cex=2)
abline(h=summary_partpool["mu_a", "mean"], col="red")

legend("topright", legend=c("no pooling","partial pooling", "partial pooling mean"), pch=c("+","+","-")
```



The red line is overall mean μ_a, while individual groups have individual means.

The difference between "no pooling" and "partial pooling" estimates is that extreme values tend to be pulled towards the joint mean by partial pooling (**shrinkage**).

Statistical power is borrowed across groups. This can be helpful if some groups have a low number of observations.

In addition to both models, there is also **complete pooling**. Here, all information across groups would be pooled:

$$y_i \sim \text{normal}(\mu, \sigma), \quad i = 1, ...n$$

There is no effect of group included, i.e. all groups share the same mean.

"partial pooling" is a compromise between "no pooling" and "complete pooling".

Further reading

Read more about complete pooling, partial pooling, no pooling and shrinkage:

https://www.tjmahr.com/plotting-partial-pooling-in-mixed-effects-models/

brms complete pooling

We start with the "complete pooling" model in `brms`, which fits just the mean species richness (intercept only), ignoring the predictor `Beach`.

```
fit.b.compl = brm( Richness ~ 1,
                   data=df )
```

```
fit.b.compl
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: Richness ~ 1
##    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     5.63      0.73     4.14     7.01 1.00     2806     2263
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     5.07      0.54     4.14     6.27 1.00     3090     2351
##
## 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).
```

brms no pooling

Next, "no pooling" fits each level of the categorical predictor `Beach` separately. 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 ~ Beach,
                data=df )
```

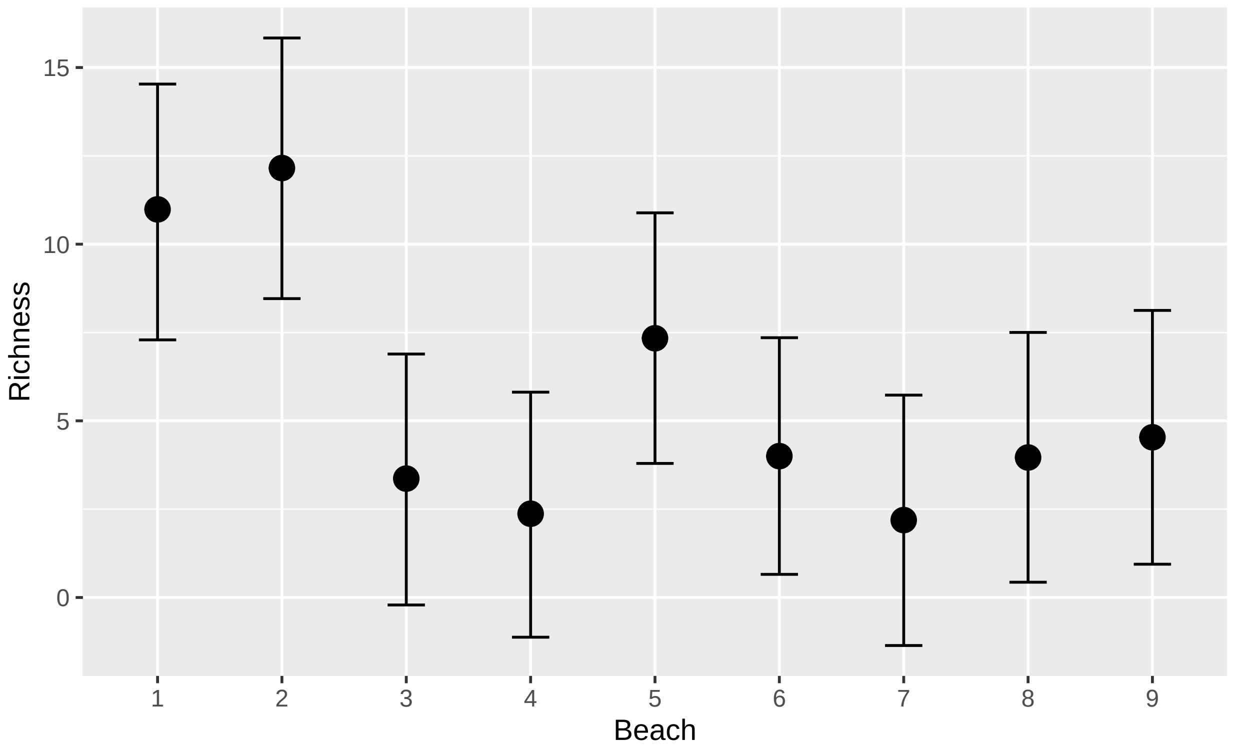
Like `lm()`, `brms` uses dummy coding for categorical predictors. I.e., it does not fit the means for each level, but an `Intercept` for the first level and an "effect" (difference to the intercept) for all other levels.

```
fit.b.no
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: Richness ~ 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    10.97      1.82     7.29    14.53 1.00     1105     1511
## Beach2        1.18      2.61    -3.80     6.36 1.00     1709     2510
## Beach3       -7.64      2.57   -12.73    -2.29 1.00     1584     2288
## Beach4       -8.63      2.58   -13.59    -3.46 1.00     1456     1716
## Beach5       -3.63      2.55    -8.47     1.53 1.00     1642     2469
## Beach6       -6.98      2.51   -11.93    -1.84 1.00     1504     2154
## Beach7       -8.80      2.60   -13.94    -3.54 1.00     1626     2095
## Beach8       -7.03      2.59   -12.03    -1.90 1.00     1684     2313
## Beach9       -6.43      2.60   -11.45    -1.11 1.00     1620     2231
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.02      0.49     3.24     5.16 1.00     3677     2558
##
## 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).
```

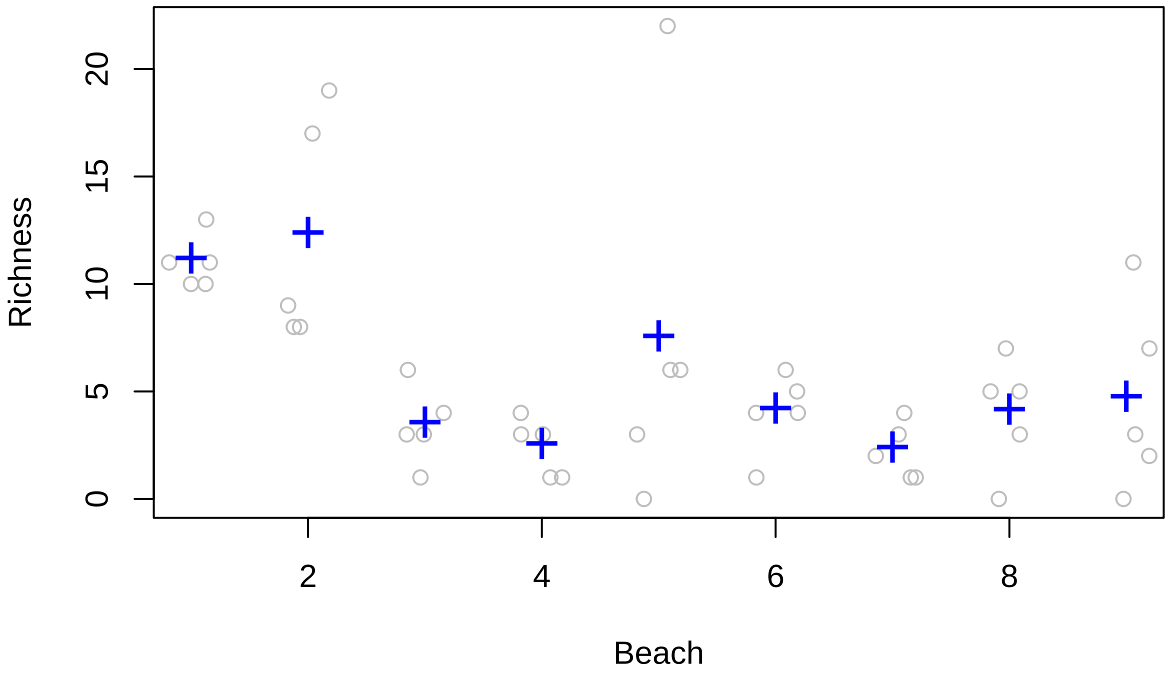
However, when using `conditional_effects()` or `fitted()`, the group means in each level are predicted.

```
plot( conditional_effects(fit.b.no) )
```



```
plot(0, 0, xlim =c(1,9), ylim = range(df$Richness), type = "n",
     ylab="Richness",
     xlab="Beach")
points(Richness ~ jitter(as.numeric(Beach), factor=1.0), data=df, col="grey")
```

```
levels = levels(df$Beach)
for(i in 1:9){
  group.mean = fitted(fit.b.no, newdata=data.frame(Beach=levels[i]) )
  points(i, group.mean[, 1], pch="+", col="blue", cex=2 )
}
```



brms partial pooling

With a partial pooling model, we're not interested in `Beach` as a fixed effect, but fit an overall intercept or mean, while each group level prediction has a random intercept.

```
fit.b.part = brm( Richness ~ (1|Beach),  # same as "Richness ~ 1+(1|Beach)"
                  data=df )
```

```
fit.b.part
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: Richness ~ (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)     3.46      1.21     1.50     6.25 1.00     1213     1602
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     5.49      1.27     2.97     7.94 1.00     1579     1522
##
## Family Specific Parameters:
```

```
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.07      0.50     3.23     5.19 1.00     2417     2548
##
## 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, population-level mean μ_a is presented as `Intercept`, while for the group-level effects only their standard deviation σ_a is presented as `sd(Intercept)`. The residual standard deviation is σ is `sigma`. brms always uses the variable names `sigma` for residuals and `sd` for random effects.

When interested in the single group-level effects, use `ranef()`

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

```
## $Beach
## , , Intercept
##
##     Estimate Est.Error       Q2.5    Q97.5
## 1  4.1001769  2.077448  0.2888656 8.323663
## 2  4.9871039  2.154849  0.8611748 9.347415
## 3 -1.5010852  1.870968 -5.2156725 2.109744
## 4 -2.2602815  1.885351 -6.0119151 1.430784
## 5  1.4486500  1.928202 -2.1606646 5.419085
## 6 -1.0941946  1.877480 -4.8437490 2.514381
## 7 -2.4196677  1.901968 -6.3006079 1.103256
## 8 -1.0637608  1.850571 -4.6964704 2.587625
## 9 -0.6486799  1.884597 -4.4507644 2.918655
```

Now we see how brms fits these. Instead of presenting the single a_j (group-level means), the difference $\alpha_j = a_j - \mu_a$ from the overall mean μ_a is shown! This is an equivalent model formulation using $a_j = \mu_a + \alpha_j$ with

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

assuming that mean difference is zero (and same standard deviation σ_a). This formulation is helpful, especially when there are more than one random effects, e.g. `y ~ 1 + (1|x1) + (1|x2)`.

But wait… what about priors, we didn't define any. What did brms choose?

```
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)
```

brms automatically took care on the random effects structure for the a_j / or rather the α_j (not shown here). Default priors (based on the values of y) are assigned for the μ_a (`Intercept`), σ_a (`sd`) and σ (`sigma`).

If we want to assign priors e.g. for the overall mean, we can do so:

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

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

```r
fit.b.part
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: Richness ~ (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)     3.46      1.22     1.45     6.28 1.00     1183     1915
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     5.67      1.35     3.09     8.44 1.01     1079     1396
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.07      0.51     3.23     5.20 1.00     2609     2982
##
## 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).
```

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

```
## $Beach
## , , Intercept
##
##     Estimate Est.Error       Q2.5    Q97.5
## 1  3.9208242  2.068261  0.1170315 8.194879
## 2  4.8630510  2.147823  0.8017441 9.340667
## 3 -1.6731544  1.910771 -5.5480820 1.964943
## 4 -2.4141567  1.946965 -6.4995070 1.240531
## 5  1.2436709  1.912178 -2.4964078 5.030099
## 6 -1.2714226  1.946873 -5.2263237 2.426271
## 7 -2.5599606  2.015151 -6.5917169 1.219517
## 8 -1.2367727  1.907484 -4.9749819 2.365029
## 9 -0.7832589  1.904452 -4.6440013 2.985740
```

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

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

Next we want to plot some predictions.

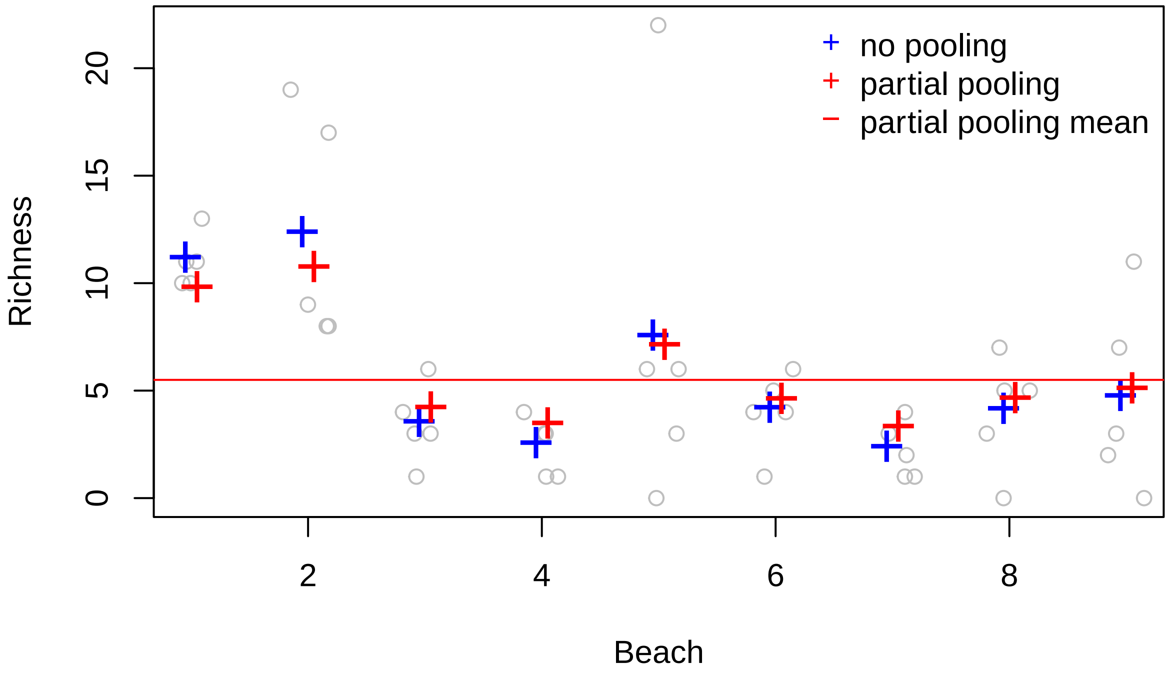
```r
plot( conditional_effects(fit.b.part) )
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument 'x' in selecting a method for fu
```

The convenient function `conditional_effects()` prints an error message. It only plots fixed effects, and since there are none (except for the intercept μ_a), it doesn't plot anything.

So we have to do this manually using `fitted()` and specifying the group-level a_j `newdata=data.frame(Beach=...)`, or specifying no group level `newdata=data.frame(Beach=NA)` which predicts on the population level μ_a.

```
plot(0, 0, xlim =c(1,9), ylim = range(df$Richness), type = "n",
     ylab="Richness",
     xlab="Beach")
points(Richness ~ jitter(as.numeric(Beach), factor=1.0), data=df, col="grey")

levels = levels(df$Beach)
for(i in 1:9){
  group.mean = fitted(fit.b.no, newdata=data.frame(Beach=levels[i]) )
  points(i-0.05, group.mean[, 1], pch="+", col="blue", cex=2 )
}
for(i in 1:9){
  group.mean = fitted(fit.b.part, newdata=data.frame(Beach=levels[i]) )
  points(i+0.05, group.mean[, 1], pch="+", col="red", cex=2 )
}

total.mean = fitted(fit.b.part, newdata=data.frame(Beach=NA) )

abline(h = mean(total.mean), col="red")

legend("topright",
       legend=c("no pooling","partial pooling", "partial pooling mean"),
       pch=c("+","+","-"),
       col=c("blue","red","red"),
       bty="n")
```


Richness
Beach
no pooling
partial pooling
partial pooling mean