Lab 10: Multilevel Modeling

```
library(rstan)
```

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
library(rethinking)
```

```
## Loading required package: cmdstanr

## This is cmdstanr version 0.5.1.9000

## - CmdStanR documentation and vignettes: mc-stan.org/cmdstanr

## - CmdStan path: /Users/alexziyujiang/.cmdstan/cmdstan-2.29.2

## - CmdStan version: 2.29.2

## Loading required package: parallel

## rethinking (Version 2.21)

##
## Attaching package: 'rethinking'

## The following object is masked from 'package:rstan':
##
##     stan

## The following object is masked from 'package:stats':
##
##     rstudent
```

Motivating example: tadpoles mortality data

The example we will be looking at today is the mortality data for Reed frog. You can think of each row as a `tank`, we put a bunch of tadpoles into each of them and see how many of them survives. We will be especially looking at the following variables

- `surv`: the number of tadpoles surviving in each tank
- `density`: initial number of tadpoles

Visualize the Cluster effect

We then take a look at the data. One thing to notice is that the tanks have different sizes. The first few are small, then medium and finally large.

```r
data(reedfrogs)
data("reedfrogs")
d <- reedfrogs
str(d)
```

```
## 'data.frame':    48 obs. of  5 variables:
##  $ density : int  10 10 10 10 10 10 10 10 10 10 ...
##  $ pred    : Factor w/ 2 levels "no","pred": 1 1 1 1 1 1 1 1 2 2 ...
##  $ size    : Factor w/ 2 levels "big","small": 1 1 1 1 2 2 2 2 1 1 ...
##  $ surv    : int  9 10 7 10 9 9 10 9 4 9 ...
##  $ propsurv: num  0.9 1 0.7 1 0.9 0.9 1 0.9 0.4 0.9 ...
```

We first try to visualize the data. One crude estimate we can use without fitting any model is the empirical survival proportion:

$$\hat{p}_i = \left(\frac{y_i}{N_i}\right), i = 1, ..., K$$

- **Side remark**: we do not use it here, but if you're interested in empirical estimate of the logit ratios, we can use

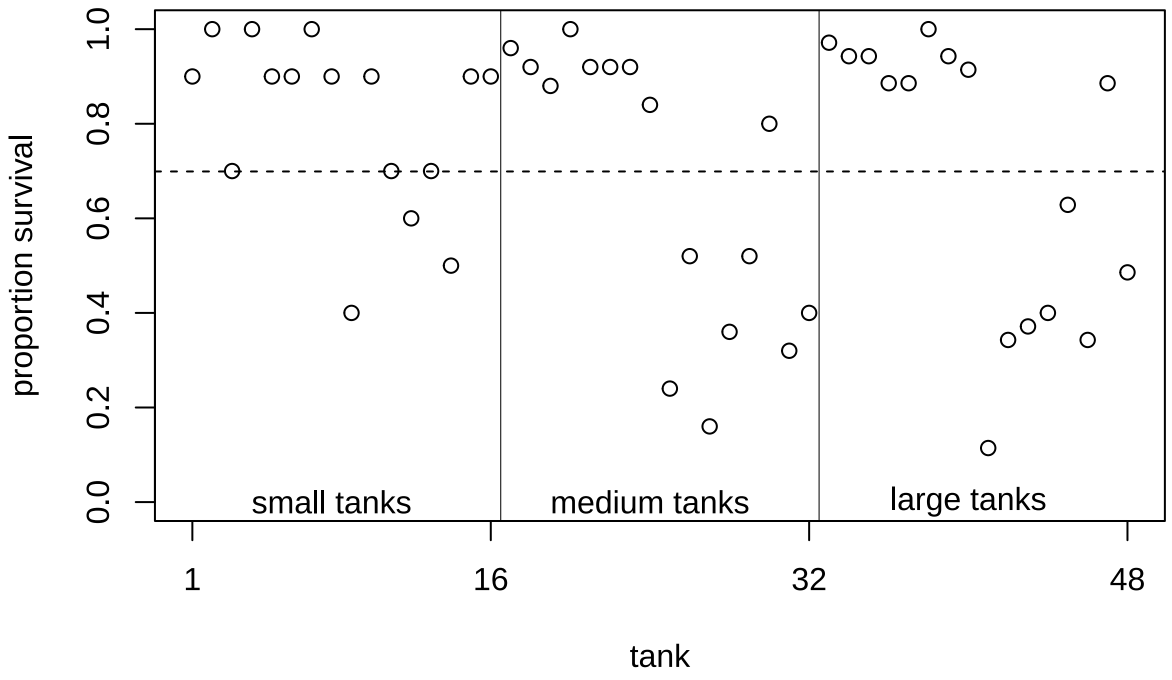
$$\hat{\alpha}_i = \log\left(\frac{y_i + 0.5}{N_i - y_i + 0.5}\right)$$

The 0.5 adjust for cases where $\hat{p}_i = 0$ or 1.

There are two ways to compute it, depending on whether we want to use data from other tanks

- 'Pooling' data from all tanks to compute the empirical estimates: $\hat{p}_i = \hat{p} = \frac{\sum_{i=1}^K y_i}{\sum_{i=1}^K N_i}$, if K is the number of tanks. This is shown as the dashed line in the graph.
- Or we don't do any pooling, just use data from tank i to compute $\hat{p}_i = \left(\frac{y_i}{N_i}\right)$. This is shown as the white dots in the following figure.

```r
# extract Stan samples
# median intercept for each tank
# also transform to probability with logistic
#d$propsurv.est <- logistic( apply( post$a , 2 , mean ) )
# display raw proportions surviving in each tank
plot( d$propsurv , ylim=c(0,1) ,  xaxt="n" ,
    xlab="tank" , ylab="proportion survival" )
axis( 1 , at=c(1,16,32,48) , labels=c(1,16,32,48) )
# overlay posterior means
#points( d$propsurv.est )
# mark posterior mean probability across tanks
abline( h=sum(d$surv)/sum(d$density) , lty=2 )
# draw vertical dividers between tank densities
abline( v=16.5 , lwd=0.5 )
abline( v=32.5 , lwd=0.5 )
text( 8 , 0 , "small tanks" )
text( 16+8 , 0 , "medium tanks" )
text( 32+8 , 0 , "large tanks" )
```



There is a lot of variation in these data. We can see from the 'crude estimate' for each tank that the survival rate estimates differs a lot across tanks. A lot of the variation in the survival rate comes from the fact that there are different things in each tank that we cannot measure, and they in turn create variation in survival across tanks. We call these tanks an example of **cluster variable**.

Modeling idea

The two ways of computing the 'crude estimates' corresponds to two philosophy in modeling clustered data.

- **complete pooling**: ignore the cluster differences, and only use one parameter α to model predictions for survival rates in each tank
  - there will be a lot of data, so the **grand mean** will be quite precise
  - however α is unlikely to exactly match the mean for each particular tank
  - this leads to **underfitting**

For frequentist model, we can compute the frequetist estimate by taking the overall empirical estimate $(\frac{\sum_{i=1}^N y_i}{\sum_{i=1}^N N_i})$.

In a Bayesian model it will look something like

$$\begin{aligned} y_i &\sim \text{Binomial}\left(N_i, p_i\right) \\ \text{logit}\left(p_i\right) &= \alpha \\ \alpha &\sim \text{Normal}(0, 1.5) \end{aligned}$$

- **no pooling**: only use data in tank i to produce estimates in that tank
  - no information is shared across tanks
  - variance estimates for individual tanks can be high due to small observations in each tank
  - in this case, we tend to overfit the data (i.e. over sensitive to observation where group size is small)

In frequentist methods we can use the tank-wise empirical estimates $(\frac{y_i}{N_i})$, under a Bayesian context it will be something like

$$\begin{aligned}
y_i &\sim \text{Binomial}\left(N_i, p_i\right) \\
\text{logit}\left(p_i\right) &= \alpha_{\text{TANK }[i]} \\
\alpha_j &\sim \text{Normal}(0, 1.5), j = 1, ..., K
\end{aligned}$$

This leads us to consider **partial pooling**, an intermediate step between complete pooling and no pooling to balance between underfitting and overfitting. Partial pooling allows us to do the following things at the same time:

- we will be able to simultaneously estimate an intercept for each tank – this is done by modeling **varying intercepts**. we call it **intercepts** because they are not mingled with covariates (at least for now)
- we will be able to draw information from other tanks while estimating intercept for a certain tank – this is done by using an **adaptive regularizing prior**
  - this is done by giving these varying intercepts a parametrized **common prior**, shared by all tanks
  - the estimation of the parameters in the **common prior** use all data

$$\begin{aligned}
y_i &\sim \text{Binomial}\left(N_i, p_i\right) \\
\text{logit}\left(p_i\right) &= \alpha_{\text{TANK}[i]} \\
\alpha_j &\sim \text{Normal}(\bar{\alpha}, \sigma) \\
\bar{\alpha} &\sim \text{Normal}(0, 1.5) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}$$

- There are two levels in this model: the tank intercept is drawn from a normal distribution with (unknown) parameters $\bar{\alpha}$ and σ, and these two parameter has their own priors. This is why we call it **multilevel modeling**.
- As $\bar{\alpha}$ and σ are essentially `parameters for parameters $\alpha_{\dot}$`, we call them hyperparameters, and their priors are called **hyperpriors**

implementing the model

```
library(rjags)
```

```
## Loading required package: coda

##
## Attaching package: 'coda'

## The following object is masked from 'package:rstan':
##
##     traceplot

## Linked to JAGS 4.3.0

## Loaded modules: basemod,bugs
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
model_code <- "
data {
  n <- length(y)
}
model {
  sigma ~ dexp(1)
  alpha_bar ~ dnorm(0, pow(1.5, -2))
  for (i in 1:n) {
    alpha[i] ~ dnorm(alpha_bar, pow(sigma, -2))
    p[i] <- ilogit(alpha[tank[i]])
    y[i] ~ dbinom(p[i], size[i])
  }
}
"
m1 <- jags.model(file = textConnection(model_code),
                 data = list(
                   y = d$surv,
                   size = d$density,
                   tank = 1:nrow(d)
                 ))
```

```
## Compiling data graph
##    Resolving undeclared variables
##    Allocating nodes
##    Initializing
##    Reading data back into data table
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 48
##    Unobserved stochastic nodes: 50
##    Total graph size: 250
##
## Initializing model
```

```
m1_res <- coda.samples(m1, variable.names = c("alpha_bar", "sigma", "alpha"), n.iter = 2000)
res <- m1_res[[1]] %>% as.data.frame()
precis(res, depth = 1) %>% data.frame() %>% select(c(-histogram))
```

```
## 48 vector or matrix parameters hidden. Use depth=2 to show them.
```

```
##                mean        sd     X5.5.   X94.5.
## alpha_bar 1.334969 0.2567066 0.9380437 1.739851
## sigma     1.601399 0.2119074 1.2896586 1.947891
```

```
# get posterior mean
alpha_means <- colMeans(res)[1:48]
alpha_bar_mean <- colMeans(res)[49]
```

```
d$density
```

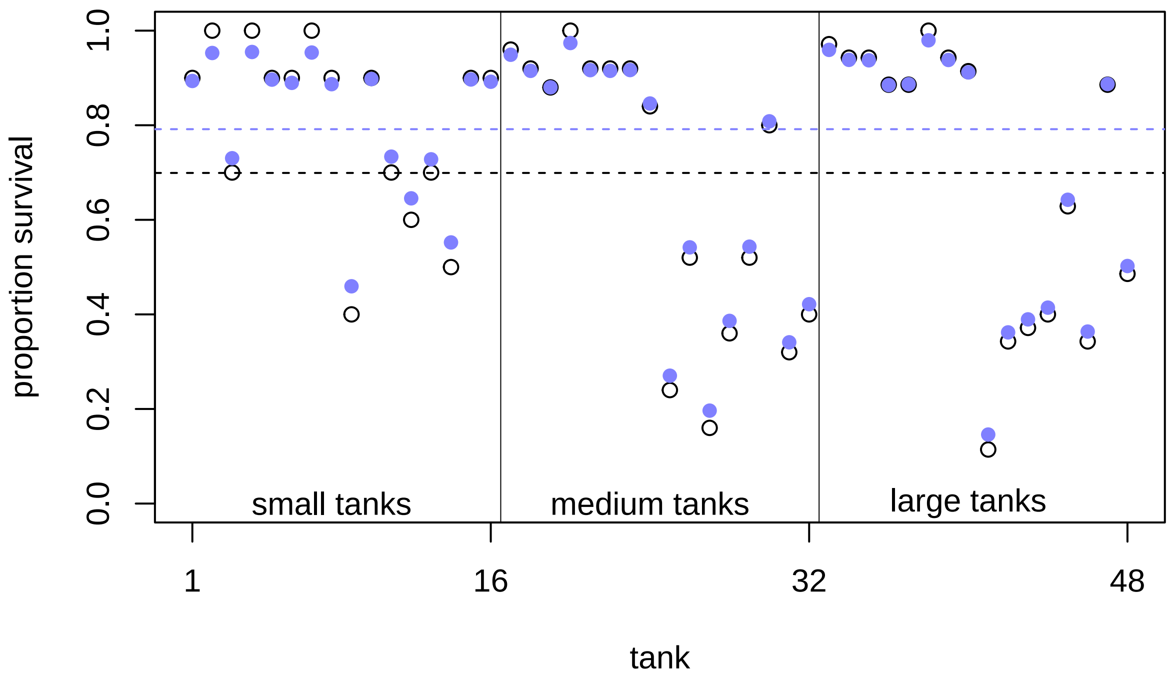
```
##  [1] 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 25 25 25 25 25 25 25 25 25
## [26] 25 25 25 25 25 25 25 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
```

```
dim(d)
```

```
## [1] 48  5
```

```
# extract Stan samples
#post <- extract.samples(m13.2)

# median intercept for each tank
# also transform to probability with logistic
d$propsurv.est <- inv_logit(alpha_means)
# display raw proportions surviving in each tank
plot( d$propsurv , ylim=c(0,1)  , xaxt="n" ,
    xlab="tank" , ylab="proportion survival" )
axis( 1 , at=c(1,16,32,48) , labels=c(1,16,32,48) )
# overlay posterior means
points( d$propsurv.est , pch = 16, col=rangi2 )
# mark posterior mean probability across tanks
abline( h=sum(d$surv)/sum(d$density) , lty=2)
abline( h=inv_logit(alpha_bar_mean) , lty=2, col = rangi2 )
# draw vertical dividers between tank densities
abline( v=16.5 , lwd=0.5 )
abline( v=32.5 , lwd=0.5 )
text( 8 , 0 , "small tanks" )
text( 16+8 , 0 , "medium tanks" )
text( 32+8 , 0 , "large tanks" )
```



We can see a few things from the graph:

- the blue dots (partial pooling estimates) tends to ‘shrink’ towards the center of the data (the blue line), away from their respective crude estimates (black dots). The phenomenom is called **shrinkage**. As the α’s are all shrinking towards there mean, we call the prior $N(\bar{\alpha}, \sigma)$ **regularization prior**
- Different from what we’ve seen in previous examples (say, the ridge regression example), the amount of regularization is learned from the data. So we call it **adaptive prior**
- for smaller tanks, the shrinkage effects is stronger. This is in line with our intuition that the no pooling estimates (black dots) are less reliable in smaller tanks, so we need to borrow information from other groups a little more to avoid overfitting.

The power of partial pooling: a simulation example

We have seen that partial pooling is a balance of complete and no pooling methods, in the hope that we can avoid overfitting and underfitting issues. Now we see a simulation example to further provide evidence to this claim. Doing simulations for models that are more complicated is very useful, because it helps us understand the inner dynamics of a model better.

Here we set the 'true values' (we call it true values, because we use it to generate data) for $\bar{\alpha}, \sigma$ to be 1.5. Here we consider four 'sizes of tanks' to illustrate the effect of cluster size a little more carefully, so we have four different size: 5, 10, 25 and 35. We simulate the data through the following process

- We sample intercepts $\alpha_1, ..., \alpha_{60}$ from the regularization prior $N(1.5, 1.5)$
- We conduct inverse logit transformation on $\alpha_1, ..., \alpha_{60}$ to get the true survival rate of tadpoles $p_1, ..., p_N$.
- Given number of tadpoles in each tank $N_1, ..., N_{60}$, we sample the survived counts from the Binomial distribution $Bin(N_i, p_i)$

We then compare the estimation performance of all three estimation techiniques: complete, partial and no pooling. The performance we will be looking at is the absolute error: $|\hat{p}_{est.} - p_{true}|$.

```
# simulate data
a_bar <- 1.5
sigma <- 1.5
nponds <- 60
Ni <- as.integer( rep( c(5,10,25,35) , each=15 ) )
set.seed(5005)
a_pond <- rnorm( nponds , mean=a_bar , sd=sigma )
dsim <- data.frame( pond=1:nponds , Ni=Ni , true_a=a_pond )
dsim$Si <- rbinom( nponds , prob=logistic(dsim$true_a) , size=dsim$Ni )
# no pooling
dsim$p_nopool <- dsim$Si / dsim$Ni
# partial pooling
dat <- list( Si=dsim$Si , Ni=dsim$Ni , pond=dsim$pond )
m2 <- jags.model(file = textConnection(model_code),
                 data = list(
                   y = dat$Si,
                   size = dat$Ni,
                   tank = 1:length(dat$Si)
                 ))
```

```
## Compiling data graph
##    Resolving undeclared variables
##    Allocating nodes
##    Initializing
##    Reading data back into data table
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 60
##    Unobserved stochastic nodes: 62
##    Total graph size: 310
##
## Initializing model
```

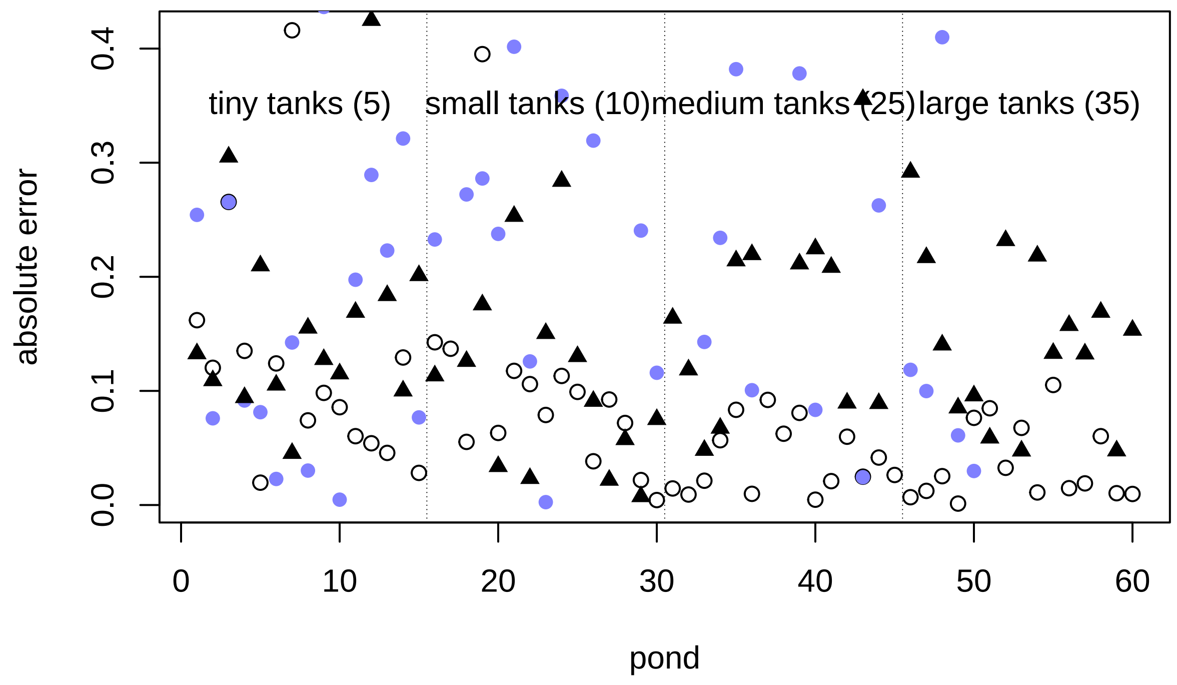
```
m2_res <- coda.samples(m1, variable.names = c("alpha_bar", "sigma", "alpha"), n.iter = 2000)
res <- m2_res[[1]] %>% as.data.frame()
precis(res, depth = 1) %>% data.frame() %>% select(c(-histogram))
```

```
## 48 vector or matrix parameters hidden. Use depth=2 to show them.

##                mean        sd     X5.5.   X94.5.
## alpha_bar 1.354337 0.2538651 0.9525231 1.761164
## sigma     1.633345 0.2128497 1.3168233 2.000817
```

```
alpha_means <- colMeans(res)[1:60]
# partial pool estimates
dsim$p_partpool <- inv_logit(alpha_means)
# complete pool
dsim$p_completepool <- rep(sum(dsim$Si)/sum(dsim$Ni),60)
# true values
dsim$p_true <- inv_logit(a_pond)


# calculate absolute error
nopool_error <- abs( dsim$p_nopool - dsim$p_true )
partpool_error <- abs( dsim$p_partpool - dsim$p_true )
completepool_error <- abs(dsim$p_completepool - dsim$p_true)
# plot results
plot( 1:60 , nopool_error , xlab="pond" , ylab="absolute error"
    )
points( 1:60 , partpool_error,col=rangi2 , pch=16)
points(1:60, completepool_error, pch =17)
abline( v=15.5 , lwd=0.5, lty = 3 )
abline( v=30.5 , lwd=0.5, lty = 3 )
abline( v=45.5 , lwd=0.5, lty = 3 )
text( 7.5 , 0.35 , "tiny tanks (5)" )
text( 15+7.5 , 0.35 , "small tanks (10)" )
text( 30+8 , 0.35 , "medium tanks (25)" )
text( 45+8.5 , 0.35 , "large tanks (35)" )
```



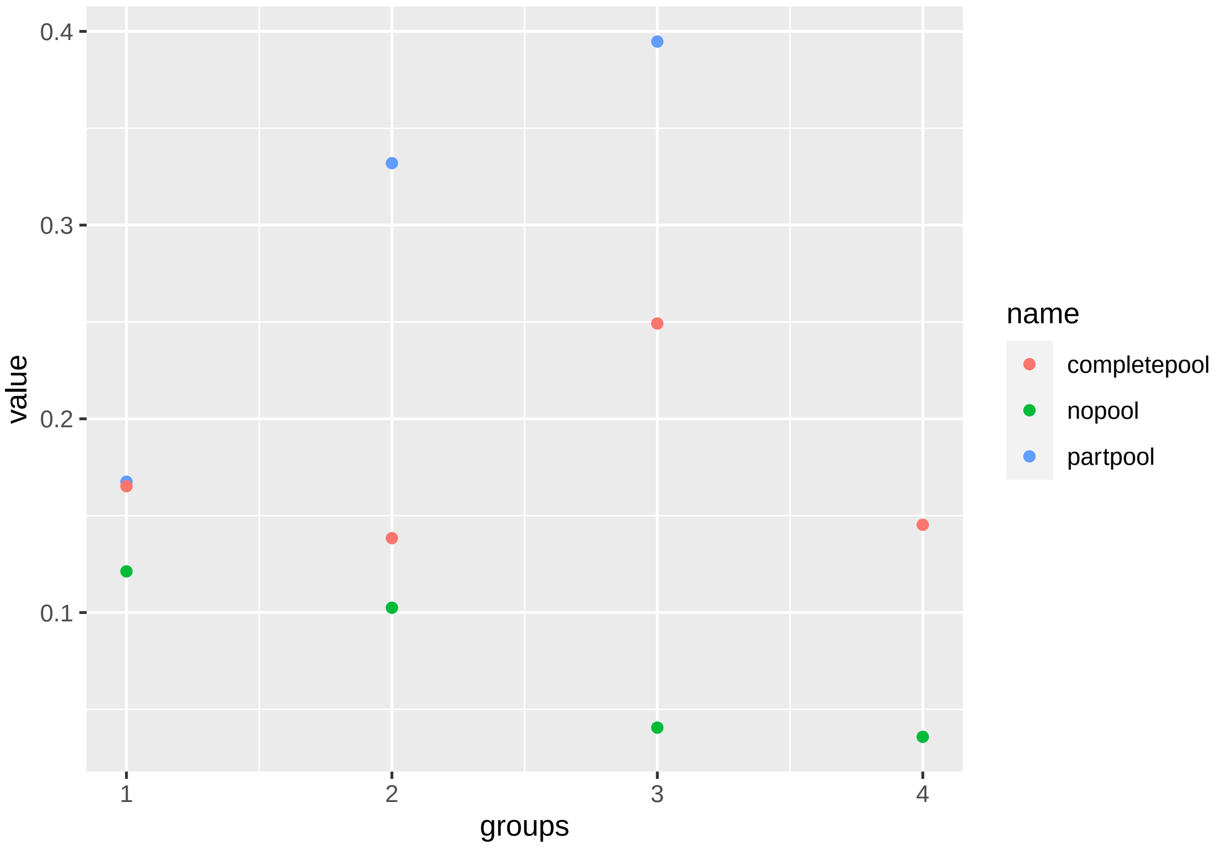
```
nopool_avg <- aggregate(nopool_error,list(dsim$Ni),mean)
partpool_avg <- aggregate(partpool_error,list(dsim$Ni),mean)
```

```
completepool_avg <- aggregate(completepool_error,list(dsim$Ni),mean)
```

We can also plot the averaged absolute errors, grouped over different tank sizes to compare between these methods:

```
df <- data.frame(groups = 1:4, nopool = nopool_avg$x, partpool = partpool_avg$x, completepool = complete
ggplot(df, aes(x = groups, y = value)) + geom_point(aes(col = name))
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



There are a few things we could glean from the results:

- The complete pooling method performs the worst uniformly across all four groups with different tank sizes. This makes sense as complete pooling method tend to underfit since it ignores individual differences across clusters.
- Looking at no pooling and partial pooling, we see that for medium and large tanks, the results are pretty similar and closer to zero. This is because we already have many data, and both kinds of estimates are much more accurate for large datasets.
- For smaller tanks, the estimation error is larger, especially for the tiny size. This is because the amount of data we have is so small that even by pooling together information, we could not gain enough.
- Partial pooling does not necessarily do better in every scenario, but overall it outperforms the no pooling method in most tanks, especially when datasets are smaller.
- **Side remark:** Finally, this is just results from one particular simulation. To mitigate sampling error caused by simulating a particular dataset, what we usually do in research is to replicate this process many times, and store the absolute errors for each replication. What we can do then is to plot grouped box plots for the absolute errors (grouped by different methods), to compare between these methods.

Concluding remarks

- Partially pooling performs better on average. It adjusts cluster-level estimates to balance between underfitting and overfitting, by placing an adaptive regularization prior
- When cluster sizes are small, partial pooling shows great improvement in estimation compared to no pooling estimates
- When cluster sizes are large, the difference is very tiny from the no-pooling estimates