

# Supplemental Materials – R and Stan Codes

Bayesian Dynamic Borrowing of Historical Information  
With Applications to the Analysis of Large-Scale Assessments

## Contents

<b>Codes for Single-Level Models</b> .....	1
Functions.R .....	1
CaseStudy_SingleLevel.R .....	11
Simulation_SingleLevel.R .....	14
<b>Codes for Multilevel Models</b> .....	18
CaseStudy_Multilevel.R .....	18
Simulation_Multilevel.R .....	21
Stan Codes for Multilevel models .....	23

## Codes for Single-Level Models

### Functions.R

```
##### Packages - Starts #####
library(ggmcmc)
library(robustHD)
library(ggpubr)
library(MASS)
library(psych)
library(xlsx)
library(rstan)
library(loo)
options(mc.cores = 4)
parallel:::setDefaultClusterOptions(setup_strategy = "sequential")
##### Packages - Ends #####

#Functions
my.bin <- function(obs.data, syn.data){
  #This function put continuous PARED values into bins. Bins are the same with
  original PISA cycle
  values <- as.numeric(levels(factor(obs.data))) #check what are the values in the
  original PISA cycle
  break.point <- rep(0, length(values)) #generate a break point vector
  for(i in 1:(length(values))){ #prepare the break points
    break.point[i] <- (values[i]+values[i+1])/2} #each break point is the mean of
  itself and the next value
}
```

```

    break.point[is.na(break.point)] <- Inf #in case any missing produced, it will be
converted to max value, which is positive infinity.
    break.point <- c(0, break.point) #add initial value as 0
    data <- cut(syn.data, breaks=break.point, include.lowest=TRUE, right=FALSE,
labels=values) #cut is a base function
    return(as.numeric(as.character(data))) #save from being a factor and turn into
numeric values
}

generate.PISA <- function(PISA.data, n=n, cont.var = c(2,3), res.std.err = 84, coef) {
  ## This function generates PISA cycles with "Female" "LangAtHome" "PARED" "CULTPOSS"
"HEDRES" "HOMEPOS" "PV1MATH" by taking into account of their correlations and their
structure
  Sig_s <- cov(PISA.data[,cont.var]) #cont.var = continuous variables - produce PARED
as continuous and turn into bins
  S1<-mvrnorm(n,c(colMeans(PISA.data)[cont.var]),Sig_s) #mean values are coming from
the original dataset
  Female<-rbinom(n,1,mean(PISA.data$Female))#binary variable probabilities coming from
the original dataset percentages
  IMMIG<-rbinom(n,1,mean(PISA.data$IMMIG))#binary variable probabilities coming from
the original dataset percentages
  S1[, "PARED"] <- my.bin(obs.data=PISA.data$PARED, syn.data=S1[, "PARED"]) #binning
  resi<-rnorm(n,0,res.std.err) #add some residual standard error to PV1MATH
  syn.dat <- as.matrix(cbind(1, Female, S1, IMMIG)) #save the data as matrix and 1 for
intercept
  PV1MATH <- syn.dat %*% coef + resi #Generate PV1MATH based on the historical-data
based coefficients
  final.syn.dat <- data.frame(Female, S1, IMMIG, PV1MATH) #generated dataset with
PV1MATH
  return(final.syn.dat)
}

```

## #Evaluation

```

MSE <- function(y.true, y.pred){(y.true-y.pred)^2}
bias <- function(y.true, y.pred){(y.pred-y.true)}
abs.bias <- function(y.true, y.pred){abs(y.pred-y.true)}
per.bias <- function(y.true, y.pred){(y.pred-y.true)/y.true}
abs.per.bias <- function(y.true, y.pred){abs((y.pred-y.true)/y.true)}

data.stan <- function(dat_combined) {
  dat <- dat_combined[rank(dat_combined$cycle, ties.method = 'first'), ]
  x <- cbind(Intercept = 1, dat[, 1:4])
  y <- dat$PV1MATH
  cycle <- dat$cycle

```

```

N <- nrow(x)
M <- ncol(x)
C <- length(unique(cycle))
NN <- sum(cycle == 1)
list(N = N, M = M, C = C, NN = NN, x = x, y = y, cycle = cycle)
}

##### Bayesian Linear Model with Non-informative Prior

modelstring <- '
data {
  int<lower = 1> N;           // number of students
  int<lower = 1> M;           // number of covariates
  int<lower = 1> C;           // number of cycles
  int<lower = 1> NN;          // number of students in the current cycle
  matrix[N, M] x;           // covariates
  vector[N] y;               // outcomes
  int<lower = 1, upper = C> cycle[N]; // cycles
}

transformed data {
  vector[M] mu_x;
  vector[M] sd_x;
  matrix[N, M] x_std;

  real mu_y = mean(y);
  real sd_y = sd(y);
  vector[N] y_std = (y - mu_y) / sd_y;

  // x[, 1] is the intercept
  x_std[, 1] = x[, 1];
  for (m in 2:M) {
    mu_x[m] = mean(x[, m]);
    sd_x[m] = sd(x[, m]);
    x_std[, m] = (x[, m] - mu_x[m]) / sd_x[m];
  }
}

parameters {
  vector[M] beta_std;
  real<lower = 0> sigma_y_std;
}

model {
  beta_std ~ normal(0, 100);

```

```

sigma_y_std ~ cauchy(0, 2);

y_std ~ normal(x_std * beta_std, sigma_y_std);
}

generated quantities {
  vector[M] beta;
  real<lower = 0> sigma_y = sigma_y_std * sd_y;
  vector[NN] log_lik;

  beta[1] = sd_y * beta_std[1] + mu_y;
  for (m in 2:M) {
    beta[m] = sd_y / sd_x[m] * beta_std[m];
    beta[1] -= beta[m] * mu_x[m];
  }

  for (n in 1:NN)
    log_lik[n] = normal_lpdf(y[n] | x[n, ] * beta, sigma_y);
}
'

my.blr <- function(dat_combined, n.chains = 4) {
  writeLines(modelstring, con = 'modelBLR.stan')
  data <- data.stan(subset(dat_combined, cycle == 1))
  stan('modelBLR.stan', data = data, iter = 20000, chains = 4)
}

my.blr.pool <- function(dat_combined, n.chains = 4) {
  writeLines(modelstring, con = 'modelBLR.stan')
  data <- data.stan(dat_combined)
  stan('modelBLR.stan', data = data, iter = 20000, chains = 4)
}

##### Bayesian Linear Model with Informative Prior (Aggregated Data-Dependent
##### Prior - AGDP)
modelstring.inf <- '
data {
  int<lower = 1> N;                // number of students
  int<lower = 1> M;                // number of covariates
  int<lower = 1> C;                // number of cycles
  int<lower = 1> NN;              // number of students in the current cycle
  matrix[N, M] x;                // covariates
  vector[N] y;                   // outcomes
  int<lower = 1, upper = C> cycle[N]; // cycles
  vector[M] mu;                  // prior mean of beta
  vector<lower = 0>[M] sigma;     // prior sd of beta

```

```

}

transformed data {
  vector[M] mu_x;
  vector[M] sd_x;
  matrix[N, M] x_std;

  real mu_y = mean(y);
  real sd_y = sd(y);
  vector[N] y_std = (y - mu_y) / sd_y;

  vector[M] mu_std;
  vector<lower = 0>[M] sigma_std;

  // x[, 1] is the intercept
  x_std[, 1] = x[, 1];
  for (m in 2:M) {
    mu_x[m] = mean(x[, m]);
    sd_x[m] = sd(x[, m]);
    x_std[, m] = (x[, m] - mu_x[m]) / sd_x[m];
  }

  mu_std[1] = mu[1] - mu_y;
  sigma_std[1] = sigma[1] ^ 2;
  for (m in 2:M) {
    real r = sd_x[m] / sd_y;
    mu_std[m] = r * mu[m];
    sigma_std[m] = r * sigma[m];
    mu_std[1] += mu_x[m] * mu[m];
    sigma_std[1] += (mu_x[m] * sigma[m]) ^ 2;
  }
  mu_std[1] /= sd_y;
  sigma_std[1] = sqrt(sigma_std[1]) / sd_y;
}

parameters {
  vector[M] beta_std;
  real<lower = 0> sigma_y_std;
}

model {
  beta_std ~ normal(mu_std, sigma_std);
  sigma_y_std ~ cauchy(0, 2);

  y_std ~ normal(x_std * beta_std, sigma_y_std);
}

```

```

generated quantities {
  vector[M] beta;
  real<lower = 0> sigma_y = sigma_y_std * sd_y;
  vector[NN] log_lik;

  beta[1] = sd_y * beta_std[1] + mu_y;
  for (m in 2:M) {
    beta[m] = sd_y / sd_x[m] * beta_std[m];
    beta[1] -= beta[m] * mu_x[m];
  }

  for (n in 1:NN)
    log_lik[n] = normal_lpdf(y[n] | x[n, ] * beta, sigma_y);
}

my.blr.inf <- function(dat_combined, n.chains = 4, prior.mean = mu, prior.sd = sigma)
{
  writeLines(modelstring.inf, con = 'modelINF.stan')
  data <- c(data.stan(subset(dat_combined, cycle == 1)), list(mu = prior.mean, sigma =
prior.sd))
  stan('modelINF.stan', data = data, iter = 20000, chains = 4)
}

# Power Priors
modelstring.pp <- '
data {
  int<lower = 1> N;           // number of students
  int<lower = 1> M;           // number of covariates
  int<lower = 1> C;           // number of cycles
  int<lower = 1> NN;          // number of students in the current cycle
  matrix[N, M] x;           // covariates
  vector[N] y;               // outcomes
  int<lower = 1, upper = C> cycle[N]; // cycles
  vector<lower = 0, upper = 1>[C] a; // weight for each cycle
}

transformed data {
  vector[M] mu_x;
  vector[M] sd_x;
  matrix[N, M] x_std;

  real mu_y = mean(y);
  real sd_y = sd(y);

```

```

vector[N] y_std = (y - mu_y) / sd_y;

// x[, 1] is the intercept
x_std[, 1] = x[, 1];
for (m in 2:M) {
  mu_x[m] = mean(x[, m]);
  sd_x[m] = sd(x[, m]);
  x_std[, m] = (x[, m] - mu_x[m]) / sd_x[m];
}
}

parameters {
  vector[M] beta_std;
  real<lower = 0> sigma_y_std;
}

model {
  beta_std ~ normal(0, 100);
  sigma_y_std ~ cauchy(0, 2);

  for (n in 1:N)
    target += a[cycle[n]] * normal_lpdf(y_std[n] | x_std[n, ] * beta_std,
sigma_y_std);
}

generated quantities {
  vector[M] beta;
  real<lower = 0> sigma_y = sigma_y_std * sd_y;
  vector[NN] log_lik;

  beta[1] = sd_y * beta_std[1] + mu_y;
  for (m in 2:M) {
    beta[m] = sd_y / sd_x[m] * beta_std[m];
    beta[1] -= beta[m] * mu_x[m];
  }

  for (n in 1:NN)
    log_lik[n] = normal_lpdf(y[n] | x[n, ] * beta, sigma_y);
}
'

my.pp <- function(dat_combined, n.chains = 4, a = 0.5) {
  writeLines(modelstring.pp, con = 'modelPP.stan')
  data <- data.stan(dat_combined)
  data$a <- c(1, rep(a, data$C - 1))
  stan('modelPP.stan', data = data, iter = 20000, chains = 4)
}

```

```

}

# Bayesian Dynamic Borrowing (BDB)
modelstring.bdb <- '
data {
  int<lower = 1> N;           // number of students
  int<lower = 1> M;           // number of covariates
  int<lower = 1> C;           // number of cycles
  int<lower = 1> NN;          // number of students in the current cycle
  matrix[N, M] x;           // covariates
  vector[N] y;               // outcomes
  int<lower = 1, upper = C> cycle[N]; // cycles
  real<lower = 0> nu1;         // prior: tau2_beta ~ inv_gamma(nu1, nu2)
  real<lower = 0> nu2;         // prior: tau2_beta ~ inv_gamma(nu1, nu2)
}

transformed data {
  vector[M] mu_x;
  vector[M] sd_x;
  matrix[N, M] x_std;

  real mu_y = mean(y);
  real sd_y = sd(y);
  vector[N] y_std = (y - mu_y) / sd_y;

  // x[, 1] is the intercept
  x_std[, 1] = x[, 1];
  for (m in 2:M) {
    mu_x[m] = mean(x[, m]);
    sd_x[m] = sd(x[, m]);
    x_std[, m] = (x[, m] - mu_x[m]) / sd_x[m];
  }
}

parameters {
  vector[M] beta_std[C];
  vector[M] mu_std;
  vector<lower = 0>[M] tau2_beta;

  real<lower = 0> sigma_y_std;
}

model {
  for (c in 1:C)
    beta_std[c] ~ normal(mu_std, sqrt(tau2_beta));
  mu_std ~ normal(0, 100);

```



```

tau2_beta ~ inv_gamma(nu1, nu2);

for (n in 1:N)
  y_std[n] ~ normal(x_std[n, ] * beta_std[cycle[n]], sigma_y_std);
sigma_y_std ~ cauchy(0, 2);
}

generated quantities {
  vector[M] beta;
  real<lower = 0> sigma_y = sigma_y_std * sd_y;
  vector[NN] log_lik;

  beta[1] = sd_y * beta_std[1, 1] + mu_y;
  for (m in 2:M) {
    beta[m] = sd_y / sd_x[m] * beta_std[1, m];
    beta[1] -= beta[m] * mu_x[m];
  }

  for (n in 1:NN)
    log_lik[n] = normal_lpdf(y[n] | x[n, ] * beta, sigma_y);
}
'

my.bdb <- function(dat_combined, n.chains = 4, nu1 = 0.001, nu2 = 0.001) {
  writeLines(modelstring.bdb, con = 'modelBDB.stan')
  data <- data.stan(dat_combined)
  stan('modelBDB.stan', data = data, iter = 20000, chains = 4)
}

extract.stan <- function(fit, M = 5, coef = cf) {
  est <- summary(fit)$summary
  beta <- est[grep('^beta\\[', rownames(est)), ][1:M, ]
  list(beta = beta,
    waic = waic(extract_log_lik(fit))$estimates[2:3, 1],
    looic = loo(fit)$estimates[2:3, 1],
    MSE = MSE(coef, beta[, 1]),
    bias = bias(coef, beta[, 1]),
    abs.bias = abs.bias(coef, beta[, 1]),
    per.bias = per.bias(coef, beta[, 1]),
    abs.per.bias = abs.per.bias(coef, beta[, 1]))
}

extract.bdb <- function(fit, M = 5, coef = cf) {
  est <- summary(fit)$summary
  beta <- est[grep('^beta\\[', rownames(est)), ][1:M, ]
  list(beta = beta,

```

```

mu_std = est[grep('^mu_std\\[', rownames(est)), 1],
sigma_beta_std = matrix(est[grep('^sigma_beta\\[', rownames(est)), 1], nrow =
M),
waic = waic(extract_log_lik(fit))$estimates[2:3, 1],
looic = loo(fit)$estimates[2:3, 1],
MSE = MSE(coef, beta[, 1]),
bias = bias(coef, beta[, 1]),
abs.bias = abs.bias(coef, beta[, 1]),
per.bias = per.bias(coef, beta[, 1]),
abs.per.bias = abs.per.bias(coef, beta[, 1]))
}

```

## CaseStudy\_SingleLevel.R

```
# Single-level case study code
source("Functions.R")
all.dt <- read.csv("multilevel.all.cycles.csv")
all.dt[,c(1:5,7)] <- NULL
head(all.dt)

get.beta.loo <- function(fit, M = 5) {
  est <- summary(fit)$summary
  beta <- est[grep('^beta\\[', rownames(est)), ][1:M, ]
  list(beta = beta,
        looic = loo(fit))
}

#prepare all.dt
blr.non.inf <- my.blr(all.dt)
blr.non.inf.res <- get.beta.loo(blر.non.inf)

#Informative Prior (AGDP)
pri.mn <- c(407.385619, -11.418260, 5.149994, 25.584788, 3.617521)
pri.sd <- c(6.652371, 2.253203, 0.493518, 1.189917, 3.069737)

blr.inf <- my.blr.inf(all.dt, prior.mean = pri.mn, prior.sd = pri.sd)
blr.inf.res <- get.beta.loo(blر.inf)

#Full Borrowing - Pooling
blr.pool <- my.blr.pool(all.dt)
blr.pool.res <- get.beta.loo(blر.pool)

#BDB
bdb_.001 <- my.bdb(all.dt, nu1 = .001, nu2 = .001)
bdb_.001.res <- get.beta.loo(bdb_.001)

bdb_.01 <- my.bdb(all.dt, nu1 = .01, nu2 = .01)
bdb_.01.res <- get.beta.loo(bdb_.01)

bdb_.1 <- my.bdb(all.dt, nu1 = .1, nu2 = .1)
bdb_.1.res <- get.beta.loo(bdb_.1)

bdb_1_1 <- my.bdb(all.dt, nu1 = 1, nu2 = 1)
bdb_1_1.res <- get.beta.loo(bdb_1_1)

bdb_1_.1 <- my.bdb(all.dt, nu1 = 1, nu2 = .1)
bdb_1_.1.res <- get.beta.loo(bdb_1_.1)
```

```

bdb_1_.01 <- my.bdb(all.dt, nu1 = 1, nu2 = .01)
bdb_1_.01.res <- get.beta.loo(bdb_1_.01)

bdb_1_.001 <- my.bdb(all.dt, nu1 = 1, nu2 = .001)
bdb_1_.001.res <- get.beta.loo(bdb_1_.001)

pp.25 <- my.pp(all.dt, a = 0.25)
pp.25.res <- get.beta.loo(pp.25)

pp.50 <- my.pp(all.dt, a = 0.5)
pp.50.res <- get.beta.loo(pp.50)

pp.75 <- my.pp(all.dt, a = 0.75)
pp.75.res <- get.beta.loo(pp.75)

save.image("case_study.RData")

est <- rbind(blr.non.inf.res$beta,
blr.inf.res$beta,
blr.pool.res$beta,
bdb_.001.res$beta,
bdb_.01.res$beta,
bdb_.1.res$beta,
bdb_1_1.res$beta,
bdb_1_.1.res$beta,
bdb_1_.01.res$beta,
bdb_1_.001.res$beta,
bdb_1_3.res$beta,
pp.0.res$beta,
pp.25.res$beta,
pp.50.res$beta,
pp.75.res$beta,
pp.1.res$beta)

blr.non.inf.res$looic$se_looic

loo <- rbind(blr.non.inf.res$looic,
            blr.inf.res$looic,
            blr.pool.res$looic,
            bdb_.001.res$looic,
            bdb_.01.res$looic,
            bdb_.1.res$looic,
            bdb_1_1.res$looic,
            bdb_1_.1.res$looic,

```

```

bdb_1_.01.res$looic,
bdb_1_.001.res$looic,
bdb_1_3.res$looic,
pp.0.res$looic,
pp.25.res$looic,
pp.50.res$looic,
pp.75.res$looic,
pp.1.res$looic)

loo_se <- rbind(blr.non.inf.res$looic$se_looic,
               blr.inf.res$looic$se_looic,
               blr.pool.res$looic$se_looic,
               bdb_.001.res$looic$se_looic,
               bdb_.01.res$looic$se_looic,
               bdb_.1.res$looic$se_looic,
               bdb_1_1.res$looic$se_looic,
               bdb_1_.1.res$looic$se_looic,
               bdb_1_.01.res$looic$se_looic,
               bdb_1_.001.res$looic$se_looic,
               pp.25.res$looic$se_looic,
               pp.50.res$looic$se_looic,
               pp.75.res$looic$se_looic)

row.names(loo) <- c("blr.non.inf.res", "blr.inf.res", "blr.pool.res", "bdb_.001.res",
                  "bdb_.01.res",
                  "bdb_.1.res", "bdb_1_1.res", "bdb_1_.1.res", "bdb_1_.01.res", "bdb_1_.001.res",
                  "bdb_1_3.res", "pp.0.res",
                  "pp.25.res", "pp.50.res", "pp.75.res", "pp.1.res")

write.xlsx(est, paste0("CaseResults.xlsx"), sheetName = "Estimates",
           col.names = TRUE, row.names = TRUE, append = FALSE)

write.xlsx(loo, paste0("CaseResults.xlsx"), sheetName = "Loo",
           col.names = TRUE, row.names = TRUE, append = TRUE)

```

## Simulation\_SingleLevel.R

```
# Single level simulation study code
```

```
a <- "cond1" #can go up to cond10
```

```
source("Functions.R")
```

```
rep=1 #1:500
```

```
set.seed(2926100)
```

```
seed.number <- round(runif(1000,10000000, 99999999),0)
```

```
my.seed <- seed.number[rep]
```

```
set.seed(my.seed)
```

```
cond <- read.csv("cond.100.csv") #conditions - read *n.500.csv or *.n.2000.csv for  
other sample size conditions
```

```
ps03 <- read.csv("ps03.n100.csv") #scaled data - read *n.500.csv or *.n.2000.csv for  
other sample size conditions
```

```
ps06 <- read.csv("ps06.n100.csv") #scaled data - read *n.500.csv or *.n.2000.csv for  
other sample size conditions
```

```
ps09 <- read.csv("ps09.n100.csv") #scaled data - read *n.500.csv or *.n.2000.csv for  
other sample size conditions
```

```
ps12 <- read.csv("ps12.n100.csv") #scaled data - read *n.500.csv or *.n.2000.csv for  
other sample size conditions
```

```
ps15 <- read.csv("ps15.n100.csv") #scaled data - read *n.500.csv or *.n.2000.csv for  
other sample size conditions
```

```
ps18 <- read.csv("ps18.csv") #not scaled
```

```
ps18[,c(1:5,7,12)] <- NULL
```

```
ps03[,c(1:5,7)] <- NULL
```

```
ps06[,c(1:5,7)] <- NULL
```

```
ps09[,c(1:5,7)] <- NULL
```

```
ps12[,c(1:5,7)] <- NULL
```

```
ps15[,c(1:5,7)] <- NULL
```

```
cf <- cond[,a]
```

```
ps18.syn <- generate.PISA(ps18, n =100, coef=cf, res.std.err = 81) #generate non-  
scaled - change n = 100 accordingly
```

```
#prepare all.dt
```

```
ps18.syn$cycle <- 1
```

```
all.dt <- rbind(ps03, ps06, ps09, ps12, ps15, ps18.syn)
```

```
#Non-informative Prior
```

```
blr.non.inf <- my.blr(all.dt)
```

```
est.blr.non.inf <- extract.stan(blr.non.inf)
```

```

#Informative Prior (AGDP)
pri.mn <- cond[,"cond5"]
pri.sd <- cond[,"cf.sd"]
blr.inf <- my.blr.inf(all.dt, prior.mean = pri.mn, prior.sd = pri.sd)
est.blr.inf <- extract.stan(blr.inf)

#Full Borrowing - Pooling
blr.pool <- my.blr.pool(all.dt)
est.blr.pool <- extract.stan(blr.pool)

#BDB
bdb_.001 <- my.bdb(all.dt, nu1 = .001, nu2 = .001)
bdb_.01 <- my.bdb(all.dt, nu1 = .01, nu2 = .01)
bdb_.1 <- my.bdb(all.dt, nu1 = .1, nu2 = .1)
bdb_1_1 <- my.bdb(all.dt, nu1 = 1, nu2 = 1)
bdb_1_.1 <- my.bdb(all.dt, nu1 = 1, nu2 = .1)
bdb_1_.01 <- my.bdb(all.dt, nu1 = 1, nu2 = .01)
bdb_1_.001 <- my.bdb(all.dt, nu1 = 1, nu2 = .001)
bdb_1_3 <- my.bdb(all.dt, nu1 = 1, nu2 = 3)

est.bdb_.001 <- extract.bdb(bdb_.001)
est.bdb_.01 <- extract.bdb(bdb_.01)
est.bdb_.1 <- extract.bdb(bdb_.1)
est.bdb_1_1 <- extract.bdb(bdb_1_1)
est.bdb_1_.1 <- extract.bdb(bdb_1_.1)
est.bdb_1_.01 <- extract.bdb(bdb_1_.01)
est.bdb_1_.001 <- extract.bdb(bdb_1_.001)
est.bdb_1_3 <- extract.bdb(bdb_1_3)

# Power Prior
pp.0 <- my.pp(all.dt, a = 0)
pp.25 <- my.pp(all.dt, a = 0.25)
pp.50 <- my.pp(all.dt, a = 0.5)
pp.75 <- my.pp(all.dt, a = 0.75)
pp.1 <- my.pp(all.dt, a = 1)

est.pp.0 <- extract.stan(pp.0)
est.pp.25 <- extract.stan(pp.25)
est.pp.50 <- extract.stan(pp.50)
est.pp.75 <- extract.stan(pp.75)
est.pp.1 <- extract.stan(pp.1)

# Results

rmse <- cbind(est.blr.non.inf$MSE, est.blr.inf$MSE, est.blr.pool$MSE,
est.bdb_.001$MSE, est.bdb_.01$MSE,

```

```

        est.bdb_.1$MSE, est.bdb_1_1$MSE, est.bdb_1_.1$MSE, est.bdb_1_.01$MSE,
est.bdb_1_.001$MSE,
        est.bdb_1_3$MSE, est.pp.0$MSE, est.pp.25$MSE, est.pp.50$MSE,
est.pp.75$MSE, est.pp.1$MSE)

rbias <- cbind(est.blr.non.inf$bias, est.blr.inf$bias, est.blr.pool$bias,
est.bdb_.001$bias, est.bdb_.01$bias,
        est.bdb_.1$bias, est.bdb_1_1$bias, est.bdb_1_.1$bias,
est.bdb_1_.01$bias, est.bdb_1_.001$bias,
        est.bdb_1_3$bias, est.pp.0$bias, est.pp.25$bias, est.pp.50$bias,
est.pp.75$bias, est.pp.1$bias)

rbias <- cbind(est.blr.non.inf$abs.bias, est.blr.inf$abs.bias, est.blr.pool$abs.bias,
est.bdb_.001$abs.bias, est.bdb_.01$abs.bias,
        est.bdb_.1$abs.bias, est.bdb_1_1$abs.bias, est.bdb_1_.1$abs.bias,
est.bdb_1_.01$abs.bias, est.bdb_1_.001$abs.bias,
        est.bdb_1_3$abs.bias, est.pp.0$abs.bias, est.pp.25$abs.bias,
est.pp.50$abs.bias, est.pp.75$abs.bias, est.pp.1$abs.bias)

rpbias <- cbind(est.blr.non.inf$per.bias, est.blr.inf$per.bias, est.blr.pool$per.bias,
est.bdb_.001$per.bias, est.bdb_.01$per.bias,
        est.bdb_.1$per.bias, est.bdb_1_1$per.bias, est.bdb_1_.1$per.bias,
est.bdb_1_.01$per.bias, est.bdb_1_.001$per.bias,
        est.bdb_1_3$per.bias, est.pp.0$per.bias, est.pp.25$per.bias,
est.pp.50$per.bias, est.pp.75$per.bias, est.pp.1$per.bias)

rapbias <- cbind(est.blr.non.inf$abs.per.bias, est.blr.inf$abs.per.bias,
est.blr.pool$abs.per.bias, est.bdb_.001$abs.per.bias,
        est.bdb_.01$abs.per.bias, est.bdb_.1$abs.per.bias,
est.bdb_1_1$abs.per.bias, est.bdb_1_.1$abs.per.bias,
        est.bdb_1_.01$abs.per.bias, est.bdb_1_.001$abs.per.bias,
est.bdb_1_3$abs.per.bias, est.pp.0$abs.per.bias,
        est.pp.25$abs.per.bias, est.pp.50$abs.per.bias,
est.pp.75$abs.per.bias, est.pp.1$abs.per.bias)

rw <- cbind(est.blr.non.inf$waic, est.blr.inf$waic, est.blr.pool$waic,
est.bdb_.001$waic, est.bdb_.01$waic, est.bdb_.1$waic,
        est.bdb_1_1$waic, est.bdb_1_.1$waic, est.bdb_1_.01$waic,
est.bdb_1_.001$waic, est.bdb_1_3$waic, est.pp.0$waic,
        est.pp.25$waic, est.pp.50$waic, est.pp.75$waic, est.pp.1$waic)

rl <- cbind(est.blr.non.inf$looic, est.blr.inf$looic, est.blr.pool$looic,
est.bdb_.001$looic, est.bdb_.01$looic, est.bdb_.1$looic,
        est.bdb_1_1$looic, est.bdb_1_.1$looic, est.bdb_1_.01$looic,
est.bdb_1_.001$looic, est.bdb_1_3$looic, est.pp.0$looic,
        est.pp.25$looic, est.pp.50$looic, est.pp.75$looic, est.pp.1$looic)

```



```

rest <- rbind(est.blr.non.inf$beta, est.blr.inf$beta, est.blr.pool$beta,
est.bdb_.001$beta, est.bdb_.01$beta, est.bdb_.1$beta,
             est.bdb_1_1$beta, est.bdb_1_.1$beta, est.bdb_1_.01$beta,
est.bdb_1_.001$beta, est.bdb_1_3$beta, est.pp.0$beta,
             est.pp.25$beta, est.pp.50$beta, est.pp.75$beta, est.pp.1$beta)

r1 <- as.vector(rmse)
r2 <- as.vector(rbias)
r3 <- as.vector(rabias)
r4 <- as.vector(rpbias)
r5 <- as.vector(rapbias)
r6 <- as.vector(rw)
r7 <- as.vector(rl)
r8 <- as.vector(rest)

res <- c(r1, r2, r3, r4, r5, r6, r7, r8)

write.table(t(res), paste0("results.100.",a,".",rep,".csv"), row.names = FALSE,
col.names = FALSE)

```

# Codes for Multilevel Models

## CaseStudy\_Multilevel.R

```
# Multilevel case study code
library(tidyverse)
library(rstan)
library(loo)
options(mc.cores = 4)
rstan_options(auto_write = TRUE)
parallel::setDefaultClusterOptions(setup_strategy = 'sequential')

set.seed(98765)
set.seed(round(runif(1, 0, 1e8)))

# combine all csv data files
all <- data.frame()
for (file in sprintf('ps%02d.csv', seq(3, 18, by = 3), '.csv'))
  all <- rbind(all, read.csv(file))
C <- max(all$cycle)

# drop schools with less than 10 students
n.sch <- group_by(all, cycle, SCHOOLID) %>%
  summarize(n = n())
data <- inner_join(all, n.sch, by = c('cycle', 'SCHOOLID')) %>%
  filter(n >= 10) %>%
  select(-n) %>%
  rename(schid = SCHOOLID)

# outcome model: please change it if needed!
# MM is the number of level 1 parameters that have random effects (including the
# intercept)
# make sure that variables 1:MM are those have random effects (that is, put those
# variables first)
model <- ~ Female + PARED + HOMEPOS + IMMIG + TCSHORT + STRATIO + Female:TCSHORT
MM <- 2

# fit noninformative prior models to historical cycles to find the informative prior
# for the current cycle
estimate <- list()
fits <- list()
rhat <- rep(NA, C)
for (c in 2:C) {
  print(c)
  dat <- filter(data, cycle == c) %>%
    mutate(schid = as.integer(as.factor(schid)), cycle = 1)
```

```

x <- as.data.frame(model.matrix(model, dat))
fit <- stan('reg.stan', chains = 4, iter = 30000, thin = 10, save_warmup = F,
          data = list(N = nrow(dat), NN = 1, M = ncol(x), MM = MM, S =
max(dat$schid), C = 1, x = x, y = dat$PV1MATH, sch = dat$schid, cycle = dat$cycle))
est <- summary(fit)$summary
rhat[c] <- max(est[, 'Rhat'], na.rm = T)
name <- rownames(est)
beta <- est[grep('^beta\\[', name), c(1, 3)]
names(beta) <- colnames(x)[1:nrow(beta)]
print(fit, 'beta')
estimate[[c]] <- list(beta = beta[, 1], sigma.beta = beta[, 2],
                    sigma.e = est[grep('^sigma_e$', name), 1],
                    #tau.u = est[grep('^tau_u\\[', name)],
                    sigma.u = as.matrix(est[grep('^sigma_u\\[', name), 1], nrow
= 1, byrow = T))
fits[[c]] <- fit
}
# check convergence
print(rhat)

coefs <- do.call(rbind, lapply(estimate, unlist))
betas <- coefs[, 1:ncol(x)]
sigmas.beta <- coefs[, (ncol(x) + 1):(ncol(x) * 2)]
sigmas <- coefs[, (ncol(x) * 2 + 1):ncol(coefs)]
save(list = c('data', 'fits', 'betas', 'sigmas.beta', 'sigmas'), file =
paste0('input.RData'))

load('input.RData')

methods <- c('reg_noninf', 'reg_pooling', 'inf', 'pp_0.25', 'pp_0.5', 'pp_0.75',
            sort(unlist(lapply(c(1, 20), function(nu) {
                paste0('bdb_0.001_0.001', 'bdb_0.01_0.01', 'bdb_0.1_0.1',
'bdb_1_1', 'bdb_1_0.001', 'bdb_1_0.01', 'bdb_1_0.1'),
                '_ ', nu)
            }))))
all <- data

output <- list()
for (method in 1:length(methods)) {
    print(methods[method])
    s <- strsplit(methods[method], '_')[[1]]
    data <- if (s[2] == 'noninf') subset(all, cycle == 1) else all[rank(all$cycle,
ties.method = 'first'), ]
    data$schid <- as.integer(as.factor(data$schid))
    x <- as.data.frame(model.matrix(model, data))

```

```

dat <- list(N = nrow(x), NN = sum(data$cycle == 1), M = ncol(x), MM = MM,
           S = max(data$schid), C = max(data$cycle), x = x, y =
as.numeric(data$PV1MATH),
           sch = data$schid, cycle = data$cycle, cycle_sch =
unlist(by(data$cycle, data$schid, median, simplify = F)))
  if (s[1] == 'inf') {
    dat$mu <- colMeans(betas[-1, ])
    dat$sigma <- colMeans(sigmas.beta[-1, ])
  } else if (s[1] == 'pp') {
    dat$a <- c(1, rep(as.numeric(s[2]), dat$C - 1))
  } else if (s[1] == 'bdb') {
    dat$nu1 <- as.numeric(s[2])
    dat$nu2 <- as.numeric(s[3])
    dat$nu <- as.numeric(s[4])
  }

fit <- stan(paste0(s[1], '.stan'), data = dat, iter = 30000, chains = 4, thin =
10,
           save_warmup = F, pars = c('beta', 'sigma_e', 'sigma_u', 'log_lik'),
include = T)
  est <- summary(fit)$summary
  ic <- list(waic = waic(extract_log_lik(fit))$estimates, looic =
loo(fit)$estimates)
  max.rhat <- max(est[, 'Rhat'], na.rm = T)
  if (max.rhat > 1.1)
    print(paste0('Error: ', method, ', ', max.rhat))
  print(fit, 'beta')
  print(max.rhat)
  output[[method]] <- list(fit = fit, est = est, ic = ic, method = methods[method])
  rm(list = c('fit', 'est', 'ic'))
  gc(F)
}

save(list = c('output'), file = 'output.RData')

```

## Simulation\_Multilevel.R

```
# Multilevel simulation study code
library(rstan)
library(loo)
options(mc.cores = 4)

condition <- 1 #1 to 10
replication <- 1 #1 to 500

set.seed(condition ^ 2)
repeat {
  seeds <- unique(round(runif(3000, 0, 1e8)))
  if (length(seeds) >= 1000) break
}
set.seed(seeds[replication])

methods <- c('reg_noninf', 'reg_pooling', 'inf', 'pp_0.25', 'pp_0.5', 'pp_0.75',
             sort(unlist(lapply(c(1, 20), function(nu) {
               paste0(c('bdb_0.001_0.001', 'bdb_0.01_0.01', 'bdb_0.1_0.1',
                         'bdb_1_1', 'bdb_1_0.001', 'bdb_1_0.01', 'bdb_1_0.1'),
                     '_ ', nu)
             }))))
load('input-multilevelSimulation_10sch-40Students.RData')

beta <- colMeans(betas[-1, ])
sigma <- colMeans(sigmas[-1, ])
beta <- c(beta[1],
          if (condition == 10)
            -beta[-1]
          else
            beta[-1] + abs(beta[-1]) * c(-0.8, -0.5, -0.2, -0.1, 0, 0.1, 0.2, 0.5,
0.8)[condition]
          )

dat <- subset(data, cycle == 1)
u <- rnorm(max(dat$schid), 0, sqrt(sigma[2]))
dat$PVIMATH <- as.matrix(cbind(1, dat[1:(ncol(dat) - 3)])) %*% beta + u[dat$schid] +
rnorm(nrow(dat), 0, sigma[1])

all <- rbind(dat, subset(data, cycle != 1))

output <- list()
for (method in 1:length(methods)) {
  print(methods[method])
```

```

s <- strsplit(methods[method], '_')[[1]]
data <- if (s[1] == 'inf' || s[2] == 'noninf')
  subset(all, cycle == 1)
else
  all[rank(all$cycle, ties.method = 'first'), ]
x <- as.data.frame(model.matrix(~ Female + PARED + HOMEPOS + IMMIG + TCSHORT +
STRATIO, data))
dat <- list(N = nrow(x), NN = sum(data$cycle == 1), M = ncol(x), MM = 1,
  S = max(data$schid), C = max(data$cycle), x = x, y =
as.numeric(data$PV1MATH),
  sch = data$schid, cycle = data$cycle, cycle_sch =
unlist(by(data$cycle, data$schid, median, simplify = F)))
if (s[1] == 'inf') {
  dat$mu <- colMeans(betas[-1, ])
  dat$sigma <- colMeans(sigmatas.beta[-1, ])
} else if (s[1] == 'pp') {
  dat$a <- c(1, rep(as.numeric(s[2]), dat$C - 1))
} else if (s[1] == 'bdb') {
  dat$nu1 <- as.numeric(s[2])
  dat$nu2 <- as.numeric(s[3])
  dat$nu <- as.numeric(s[4])
}

fit <- stan(paste0(s[1], '.stan'), data = dat, iter = 30000, chains = 4, thin =
10,
  save_warmup = F, pars = c('beta', 'sigma_e', 'sigma_u', 'log_lik'),
include = T)
est <- summary(fit)$summary
ic <- list(waic = waic(extract_log_lik(fit))$estimates, looic =
loo(fit)$estimates)
max.rhat <- max(est[, 'Rhat'], na.rm = T)
if (max.rhat > 1.1)
  write.table(c(method, max.rhat), paste0('error_', condition, '_', replication,
'.log'), T)
print(fit, 'beta')
print(max.rhat)
output[[method]] <- list(est = est, ic = ic, method = methods[method])
rm(list = c('fit', 'est', 'ic'))
gc(F)
}

save(list = c('beta', 'sigma', 'output'), file = paste0('output_', condition, '_',
replication, '.RData'))

```

## Stan Codes for Multilevel models `bdb.stan`

```
// Bayesian dynamic borrowing for multilevel models
data {
  int<lower = 1> N;           // number of students
  int<lower = 1> NN;          // number of students in the current cycle
  int<lower = 1> M;           // number of covariates
  int<lower = 1> MM;          // number of covariates with random
effects
  int<lower = 1> S;           // number of schools
  int<lower = 1> C;           // number of cycles
  matrix[N, M] x;            // independent variables
  vector[N] y;               // dependent variables
  int<lower = 1, upper = S> sch[N]; // schools
  int<lower = 1, upper = C> cycle[N]; // cycles of students
  int<lower = 1, upper = C> cycle_sch[S]; // cycles of schools
  real<lower = 0> nu1;         // parameter for tau_beta ~ inv_gamma(nu1,
nu2)
  real<lower = 0> nu2;         // parameter for tau_beta ~ inv_gamma(nu1,
nu2)
  real<lower = 0> nu;          // parameter for prec_u ~ wishart(nu, nu *
sigma)
}

transformed data {
  vector[M] mu_x;
  vector[M] sd_x;
  matrix[N, M] x_std;

  real mu_y = mean(y);
  real sd_y = sd(y);
  vector[N] y_std = (y - mu_y) / sd_y;

  // x[, 1] is the intercept
  x_std[, 1] = x[, 1];
  for (m in 2:M) {
    mu_x[m] = mean(x[, m]);
    sd_x[m] = sd(x[, m]);
    x_std[, m] = (x[, m] - mu_x[m]) / sd_x[m];
  }
}

parameters {
  // fixed effects
  vector[M] beta_std[C];
```

```

vector[M] mu_std;
vector<lower = 0>[M] tau2_beta_std;

// random effects
vector[MM] u_std[S];
cholesky_factor_corr[MM] L_omega_u;
vector<lower = 0>[MM] tau_u_std;
cov_matrix[MM] prec_u_std[C];

real<lower = 0> sigma_e_std;
}

model {
  for (c in 1:C)
    beta_std[c] ~ normal(mu_std, sqrt(tau2_beta_std));
  mu_std ~ normal(0, 10);
  for (m in 1:M)
    tau2_beta_std[m] ~ inv_gamma(nu1, nu2);

  for (s in 1:S)
    u_std[s] ~ multi_normal_prec(rep_vector(0, MM), prec_u_std[cycle_sch[s]]);
  for (c in 1:C)
    prec_u_std[c] ~ wishart(nu, nu * inverse_spd(quad_form_diag(L_omega_u,
tau_u_std)));
  L_omega_u ~ lkj_corr_cholesky(3);
  tau_u_std ~ cauchy(0, 1);

  for (n in 1:N)
    y_std[n] ~ normal(x_std[n, ] * beta_std[cycle[n]] + x_std[n, 1:MM] *
u_std[sch[n]], sigma_e_std);
  sigma_e_std ~ cauchy(0, 1);
}

generated quantities {
  vector[M] beta;
  real<lower = 0> sigma_e = sigma_e_std * sd_y;
  vector[NN] log_lik;

  cov_matrix[MM] sigma_u[C];

  for (c in 1:C)
    sigma_u[c] = inverse_spd(prec_u_std[c]) * (sd_y ^ 2);

  beta[1] = sd_y * beta_std[1, 1] + mu_y;
  for (m in 2:M) {
    beta[m] = sd_y / sd_x[m] * beta_std[1, m];

```



```

    beta[1] -= beta[m] * mu_x[m];
}

for (n in 1:NN)
    log_lik[n] = normal_lpdf(y[n] | x[n, ] * beta + x[n, 1:MM] * (u_std[sch[n]] *
sd_y), sigma_e);
}

```

## inf.stan

```
// Informative priors (AGDP) for multilevel models
data {
  int<lower = 1> N; // number of students
  int<lower = 1> NN; // number of students in the current cycle
  int<lower = 1> M; // number of covariates
  int<lower = 1> MM; // number of covariates with random
}

effects {
  int<lower = 1> S; // number of schools
  int<lower = 1> C; // number of cycles
  matrix[N, M] x; // independent variables
  vector[N] y; // dependent variables
  int<lower = 1, upper = S> sch[N]; // schools
  int<lower = 1, upper = C> cycle[N]; // cycles
  vector[M] mu; // prior mean of beta
  vector<lower = 0>[M] sigma; // prior sd of beta
}

transformed data {
  vector[M] mu_x;
  vector[M] sd_x;
  matrix[N, M] x_std;

  real mu_y = mean(y);
  real sd_y = sd(y);
  vector[N] y_std = (y - mu_y) / sd_y;

  vector[M] mu_std;
  vector<lower = 0>[M] sigma_std;

  // x[, 1] is the intercept
  x_std[, 1] = x[, 1];
  for (m in 2:M) {
    mu_x[m] = mean(x[, m]);
    sd_x[m] = sd(x[, m]);
    x_std[, m] = (x[, m] - mu_x[m]) / sd_x[m];
  }

  mu_std[1] = mu[1] - mu_y;
  sigma_std[1] = sigma[1] ^ 2;
  for (m in 2:M) {
    real r = sd_x[m] / sd_y;
    mu_std[m] = r * mu[m];
    sigma_std[m] = r * sigma[m];
    mu_std[1] += mu_x[m] * mu[m];
  }
}
```

```

    sigma_std[1] += (mu_x[m] * sigma[m]) ^ 2;
  }
  mu_std[1] /= sd_y;
  sigma_std[1] = sqrt(sigma_std[1]) / sd_y;
}

parameters {
  // fixed effects
  vector[M] beta_std;

  // random effects
  vector[MM] u_std[S];
  cholesky_factor_corr[MM] L_omega_u;
  vector<lower = 0>[MM] tau_u_std;

  real<lower = 0> sigma_e_std;
}

model {
  beta_std ~ normal(mu_std, sigma_std);

  for (s in 1:S)
    u_std[s] ~ multi_normal_cholesky(rep_vector(0, MM), diag_pre_multiply(tau_u_std,
L_omega_u));
  L_omega_u ~ lkj_corr_cholesky(3);
  tau_u_std ~ cauchy(0, 1);

  for (n in 1:N)
    y_std[n] ~ normal(x_std[n, ] * beta_std + x_std[n, 1:MM] * u_std[sch[n]],
sigma_e_std);
  sigma_e_std ~ cauchy(0, 1);
}

generated quantities {
  vector[M] beta;
  real<lower = 0> sigma_e = sigma_e_std * sd_y;
  vector[NN] log_lik;

  vector[MM] u[S];
  vector<lower = 0>[MM] tau_u = tau_u_std * sd_y;
  corr_matrix[MM] omega_u = multiply_lower_tri_self_transpose(L_omega_u);
  cov_matrix[MM] sigma_u = quad_form_diag(omega_u, tau_u);

  for (s in 1:S)
    u[s] = u_std[s] * sd_y;
}

```

```

beta[1] = sd_y * beta_std[1] + mu_y;
for (m in 2:M) {
  beta[m] = sd_y / sd_x[m] * beta_std[m];
  beta[1] -= beta[m] * mu_x[m];
}

for (n in 1:NN)
  log_lik[n] = normal_lpdf(y[n] | x[n, ] * beta + x[n, 1:MM] * u[sch[n]], sigma_e);
}

```

## pp.stan

```
// Power priors for multilevel models
data {
  int<lower = 1> N; // number of students
  int<lower = 1> NN; // number of students in the current cycle
  int<lower = 1> M; // number of covariates
  int<lower = 1> MM; // number of covariates with random
}

effects {
  int<lower = 1> S; // number of schools
  int<lower = 1> C; // number of cycles
  matrix[N, M] x; // independent variables
  vector[N] y; // dependent variables
  int<lower = 1, upper = S> sch[N]; // schools
  int<lower = 1, upper = C> cycle[N]; // cycles
  vector<lower = 0, upper = 1>[C] a; // weight for each cycle
}

transformed data {
  vector[M] mu_x;
  vector[M] sd_x;
  matrix[N, M] x_std;

  real mu_y = mean(y);
  real sd_y = sd(y);
  vector[N] y_std = (y - mu_y) / sd_y;

  // x[, 1] is the intercept
  x_std[, 1] = x[, 1];
  for (m in 2:M) {
    mu_x[m] = mean(x[, m]);
    sd_x[m] = sd(x[, m]);
    x_std[, m] = (x[, m] - mu_x[m]) / sd_x[m];
  }
}

parameters {
  // fixed effects
  vector[M] beta_std;

  // random effects
  vector[MM] u_std[S];
  cholesky_factor_corr[MM] L_omega_u;
  vector<lower = 0>[MM] tau_u_std;

  real<lower = 0> sigma_e_std;
```

```

}

model {
  beta_std ~ normal(0, 10);

  for (s in 1:S)
    u_std[s] ~ multi_normal_cholesky(rep_vector(0, MM), diag_pre_multiply(tau_u_std,
L_omega_u));
  L_omega_u ~ lkj_corr_cholesky(3);
  tau_u_std ~ cauchy(0, 1);

  for (n in 1:N)
    target += a[cycle[n]] * normal_lpdf(y_std[n] | x_std[n, ] * beta_std + x_std[n,
1:MM] * u_std[sch[n]],
                                                    sigma_e_std);

  sigma_e_std ~ cauchy(0, 1);
}

generated quantities {
  vector[M] beta;
  real<lower = 0> sigma_e = sigma_e_std * sd_y;
  vector[NN] log_lik;

  vector<lower = 0>[MM] tau_u = tau_u_std * sd_y;
  corr_matrix[MM] omega_u = multiply_lower_tri_self_transpose(L_omega_u);
  cov_matrix[MM] sigma_u = quad_form_diag(omega_u, tau_u);

  beta[1] = sd_y * beta_std[1] + mu_y;
  for (m in 2:M) {
    beta[m] = sd_y / sd_x[m] * beta_std[m];
    beta[1] -= beta[m] * mu_x[m];
  }

  for (n in 1:NN)
    log_lik[n] = normal_lpdf(y[n] | x[n, ] * beta + x[n, 1:MM] * (u_std[sch[n]] *
sd_y), sigma_e);
}

```

## reg.stan

```
// Noninformative priors and pooling for multilevel models
data {
  int<lower = 1> N;                // number of students
  int<lower = 1> NN;               // number of students in the current cycle
  int<lower = 1> M;               // number of covariates
  int<lower = 1> MM;              // number of covariates with random
effects
  int<lower = 1> S;               // number of schools
  int<lower = 1> C;               // number of cycles
  matrix[N, M] x;               // independent variables
  vector[N] y;                  // dependent variables
  int<lower = 1, upper = S> sch[N]; // schools
  int<lower = 1, upper = C> cycle[N]; // cycles
}

transformed data {
  vector[M] mu_x;
  vector[M] sd_x;
  matrix[N, M] x_std;

  real mu_y = mean(y);
  real sd_y = sd(y);
  vector[N] y_std = (y - mu_y) / sd_y;

  // x[, 1] is the intercept
  x_std[, 1] = x[, 1];
  for (m in 2:M) {
    mu_x[m] = mean(x[, m]);
    sd_x[m] = sd(x[, m]);
    x_std[, m] = (x[, m] - mu_x[m]) / sd_x[m];
  }
}

parameters {
  // fixed effects
  vector[M] beta_std;

  // random effects
  vector[MM] u_std[S];
  cholesky_factor_corr[MM] L_omega_u;
  vector<lower = 0>[MM] tau_u_std;

  real<lower = 0> sigma_e_std;
}
```

```

model {
  beta_std ~ normal(0, 10);

  for (s in 1:S)
    u_std[s] ~ multi_normal_cholesky(rep_vector(0, MM), diag_pre_multiply(tau_u_std,
L_omega_u));
  L_omega_u ~ lkj_corr_cholesky(3);
  tau_u_std ~ cauchy(0, 1);

  for (n in 1:N)
    y_std[n] ~ normal(x_std[n, ] * beta_std + x_std[n, 1:MM] * u_std[sch[n]],
sigma_e_std);
  sigma_e_std ~ cauchy(0, 1);
}

generated quantities {
  vector[M] beta;
  real<lower = 0> sigma_e = sigma_e_std * sd_y;
  vector[NN] log_lik;

  vector<lower = 0>[MM] tau_u = tau_u_std * sd_y;
  corr_matrix[MM] omega_u = multiply_lower_tri_self_transpose(L_omega_u);
  cov_matrix[MM] sigma_u = quad_form_diag(omega_u, tau_u);

  beta[1] = sd_y * beta_std[1] + mu_y;
  for (m in 2:M) {
    beta[m] = sd_y / sd_x[m] * beta_std[m];
    beta[1] -= beta[m] * mu_x[m];
  }

  for (n in 1:NN)
    log_lik[n] = normal_lpdf(y[n] | x[n, ] * beta + x[n, 1:MM] * (u_std[sch[n]] *
sd_y), sigma_e);
}

```