reliability analysis

Julia Haaf

2023-11-06

```
library("acdcquery")
library("BayesFactor")
```

```
## Loading required package: coda
```

```
## Loading required package: Matrix
```

```
## ************
## Welcome to BayesFactor 0.9.12-4.5. If you have questions, please contact Richard Morey (richarddmorey
##
## Type BFManual() to open the manual.
## ************
```

```
library("MCMCpack")
```

```
## Loading required package: MASS
```

```
## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)
```

```
## ## Copyright (C) 2003-2023 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
```

```
## ##
## ## Support provided by the U.S. National Science Foundation
```

```
## ## (Grants SES-0350646 and SES-0350613)
## ##
```

```
library("splithalf")
```

```
## This is splithalf 0.8.2
```

```
## splithalf is BETA software! Please report any bugs.
```

```
## The (unofficial) version name is: 'I eat stickers all the time, dude!'
```

```
## For documentation, questions, and issues, please see github.com/sdparsons/splithalf
```

```
## or find my email at https://sdparsons.github.io/
```

```
library("tidyverse")
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x dplyr::select() masks MASS::select()
## x tidyr::unpack() masks Matrix::unpack()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library("papaja")
```

```
## Loading required package: tinylabels
```

```
library("ggplot2")
```

```
library("acdcquery")

conn <- connect_to_db("../acdc.db")
# db_file <- base::system.file("extdata", "acdc.db", package = "acdcquery")
# conn <- connect_to_db(db_file)

arguments <-  list()
arguments <- add_argument(
 list = arguments,
 conn = conn,
 variable = "study_id",
 operator = "greater",
 values = c(-1)
)

query_results <- query_db(
 conn = conn,
 arguments = arguments,
 target_table = "dataset_table",
 target_vars = c("publication_id", "study_id", "publication_code", "default", "between_id")
)
write.csv2(query_results, "dataset_table.csv")
```

```
conn <- connect_to_db("../acdc.db")
# db_file <- base::system.file("extdata", "acdc.db", package = "acdcquery")
# conn <- connect_to_db(db_file)

arguments <-  list()
arguments <- add_argument(
 list = arguments,
 conn = conn,
 variable = "study_id",
 operator = "greater",
 values = c(-1)
)

query_results <- query_db(
 conn = conn,
 arguments = arguments,
 target_vars = c("default", "publication_id", "publication_code", "between_id")
```

```
)
```

```
## Warning: Column `block`: mixed type, first seen values of type integer,
## coercing other values of type string
```

```
# head(query_results)
(datID <- length(unique(query_results$dataset_id)))
```

```
## [1] 42
```

```
length(unique(query_results$publication_id))
```

```
## [1] 9
```

```
length(unique(query_results$between_id)) # Rey-Mermet data has two groups, older and younger participan
```

```
## [1] 32
```

```
table(query_results$dataset_id[query_results$between_id == 28], query_results$between_id[query_results$
```

```
##
##         28
##   33 41280
##   34 41132
##   35 41280
##   36 41280
```

```
query_results$dataset_id <- ifelse(query_results$between_id == 28
                                   , ifelse(query_results$dataset_id == 33, datID + 1
                                            , ifelse(query_results$dataset_id == 34, datID + 2
                                                     , ifelse(query_results$dataset_id == 35, datID +
                                                              , datID + 4)))
                                   , query_results$dataset_id)

table(query_results$dataset_id)
```

```
##
##      1      2      3      4      5      6      7      8      9     10     11
##  13920  80760  80760  67586  67586  86280  86280  10269  18144   8694  12348
##     12     13     14     15     16     17     18     19     20     21     22
##   5859   7371   8001   9135   8316   5166   6363   7497   6363   7560  39123
##     23     24     25     26     27     28     29     30     31     32     33
##   6111  11403   5292   3213  15876   8127  19151  19152  13680  13679  50560
##     34     35     36     37     38     39     40     41     42     43     44
##  49920  50560  50240  89351  89397 294538  17424  24200  17424  41280  41132
##     45     46
##  41280  41280
```

```
head(query_results)
```

```
##   publication_id       publication_code between_id observation_id dataset_id
## 1              1 chetverikov_2017_blame          1              1          1
## 2              1 chetverikov_2017_blame          1              2          1
## 3              1 chetverikov_2017_blame          1              3          1
## 4              1 chetverikov_2017_blame          1              4          1
## 5              1 chetverikov_2017_blame          1              5          1
## 6              1 chetverikov_2017_blame          1              6          1
##   subject block trial condition_id congruency accuracy        rt
```

```
## 1       1     1     1            1          1        1 0.4498339
## 2       1     1     2            1          2        1 0.4500146
## 3       1     1     3            1          1        1 0.4333159
## 4       1     1     4            1          2        1 0.4500789
## 5       1     1     5            1          2        1 0.4333531
## 6       1     1     6            1          2        1 0.4668802
```

```
dat <- query_results[order(query_results$dataset_id), ]
head(dat)
```

```
##   publication_id       publication_code between_id observation_id dataset_id
## 1              1 chetverikov_2017_blame          1              1          1
## 2              1 chetverikov_2017_blame          1              2          1
## 3              1 chetverikov_2017_blame          1              3          1
## 4              1 chetverikov_2017_blame          1              4          1
## 5              1 chetverikov_2017_blame          1              5          1
## 6              1 chetverikov_2017_blame          1              6          1
##   subject block trial condition_id congruency accuracy        rt
## 1       1     1     1            1          1        1 0.4498339
## 2       1     1     2            1          2        1 0.4500146
## 3       1     1     3            1          1        1 0.4333159
## 4       1     1     4            1          2        1 0.4500789
## 5       1     1     5            1          2        1 0.4333531
## 6       1     1     6            1          2        1 0.4668802
```

```
get.K <- function(dat){
  dat <- subset(dat, accuracy == 1)
  dat <- subset(dat, rt > .2)
  dat <- subset(dat, rt < 2.5)
  dat <- subset(dat, congruency %in% 1:2)
  issue <- issue.sub(dat)
  dat <- subset(dat, !(subject %in% issue))
  I <- length(unique(dat$subject))
  K <- table(dat$subject)/2
  Kall <- round(mean(K, na.rm = T))
  return(Kall)
}

get.I <- function(dat){
  dat <- subset(dat, accuracy == 1)
  dat <- subset(dat, rt > .2)
  dat <- subset(dat, rt < 2.5)
  dat <- subset(dat, congruency %in% 1:2)
  issue <- issue.sub(dat)
  dat <- subset(dat, !(subject %in% issue))
  return(length(unique(dat$subject)))
}

issue.sub <- function(dats){
  tmp <- table(dats$subject, dats$congruency)
  tmp <- cbind(tmp, as.numeric(rownames(tmp)))
  return(tmp[tmp[, 1] < 10 | tmp[, 2] < 10, 3])
}
```

Let's try with normal model

```
genModOneTask <- function(dats, M = 3000, b0 = .030, a0 = 2, b1 = 1, a1 = 2){
    if (mean(dats$congruency %in% 1:2)<1) stop("Conditions must be 1 and 2")
    sub <- as.integer(as.factor(dats$subject))
    I <- max(sub)
    N <- dim(dats)[1]
    keep <- (M/10 + 1) : M

    K <- table(dats$subject, dats$congruency)
    Kall <- rowSums(K)
    mn <- tapply(dats$rt, list(dats$subject,dats$congruency), mean)
    sd <- tapply(dats$rt, list(dats$subject,dats$congruency), sd)

    theta = alpha = matrix(nrow = M, ncol = I, 0)
    s2 <- 1:M
    muTheta = s2Theta = muAlpha = s2Alpha = 1:M
    theta[1,] <- rep(0, I)
  s2[1] <- .3^2
  muAlpha[1] <- .8
  muTheta[1] <- 0
  muTheta.m <- .05
  muTheta.v <- .1^2
  muAplha.m <- .8
  muAlpha.v <- 1^2
  s2Theta[1] <- .2^2
  s2Alpha[1] <- .3^2


    x <- matrix(nrow = I, ncol = 2)
    x[,1] <- rep(0,I)
    x[,2] <- rep(1,I)

    for (m in 2:M){
      #alpha
      c <- apply(K*(mn - x*theta[m-1,]), 1, sum) / s2[m-1] + muAlpha[m-1] / s2Alpha[m-1]
      v <- 1/(Kall/s2[m-1] + 1/s2Alpha[m-1])
      alpha[m,] <- rnorm(I, c*v, sqrt(v))
      #theta
      c <- K[,2]*(mn[,2] - alpha[m,]) / s2[m-1] + muTheta[m-1] / s2Theta[m-1]
      v <- 1 / (K[,2] / s2[m-1] + 1 / s2Theta[m-1])
      theta[m,] <- rnorm(I, c*v, sqrt(v))
      #s2
      scale <- sum((K-1) * sd^2 + K * (((mn - alpha[m,]) - x*theta[m,])^2)) / 2 + .5
      s2[m] <- rinvgamma(1, shape = (N+1)/2, scale = scale)
      #muAlpha
      v <- 1 / (I / s2Alpha[m-1] + 1 / muAlpha.v)
      c <- sum(alpha[m,]) / s2Alpha[m-1]
      muAlpha[m] <- rnorm(1, v*c, sqrt(v))
      #s2Theta
      scale <- sum((alpha[m,] - muAlpha[m])^2) / 2 + b1^2
      s2Alpha[m] <- rinvgamma(1, shape = I/2 + a1, scale = scale)
      #muTheta
      v <- 1 / (I / s2Theta[m-1] + 1 / muTheta.v)
```

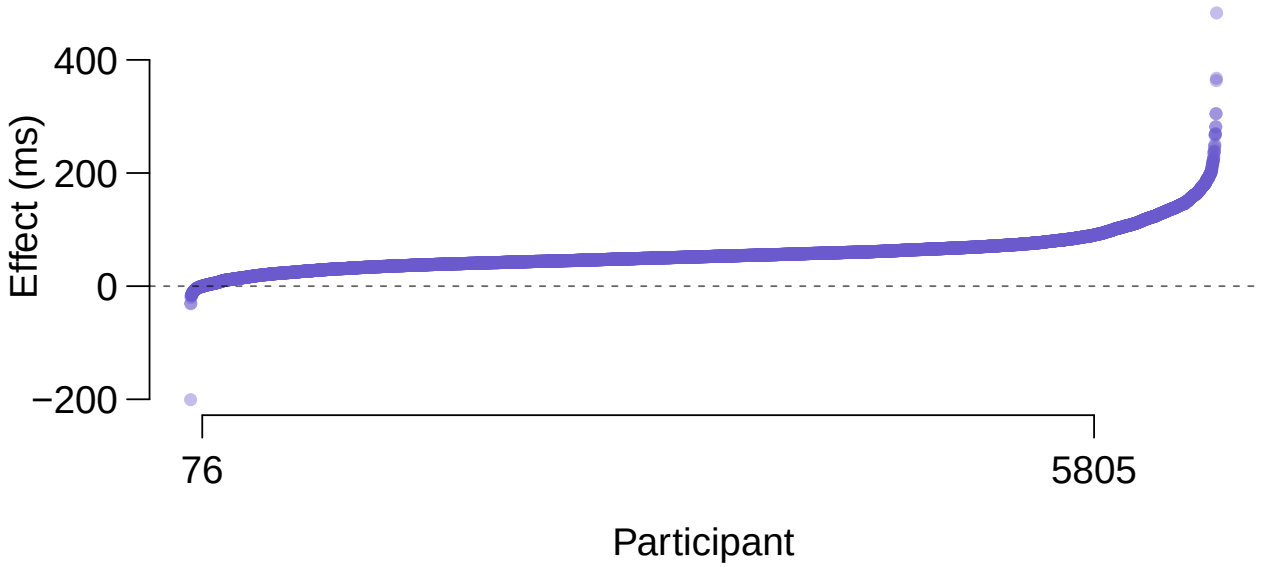
```
      c <- sum(theta[m,]) / s2Theta[m-1]
      muTheta[m] <- rnorm(1, v*c, sqrt(v))
      #s2Theta
      scale <- sum((theta[m,] - muTheta[m])^2) / 2 + b0^2
      s2Theta[m] <- rinvgamma(1, shape = I/2 + a0, scale = scale)
      }
    return(list(s2Theta = s2Theta[keep], s2Alpha = s2Alpha[keep]
                , s2 = s2[keep]
                , alpha = alpha[keep,], theta = theta[keep,]))
}
```

```
res <- list()

ids.sel <- 1:length(unique(dat$dataset_id))

for(i in ids.sel){
  dat_sub <- subset(dat, dataset_id == i)
  dat_sub <- subset(dat_sub, accuracy == 1)
  dat_sub <- subset(dat_sub, rt > .2)
  dat_sub <- subset(dat_sub, rt < 2.5)
  dat_sub <- subset(dat_sub, congruency %in% 1:2)
  issue <- issue.sub(dat_sub)
  dat_sub <- subset(dat_sub, !(subject %in% issue))

  res[[paste0("dataset_", i)]] <- genModOneTask(dat_sub)
  saveRDS(res[[paste0("dataset_", i)]], paste0("results_hierarchical_modeling/dataset", i, "_normal.RDS
  print(i)
}

saveRDS(res, paste0("results_hierarchical_modeling/all_normal.RDS"))
```

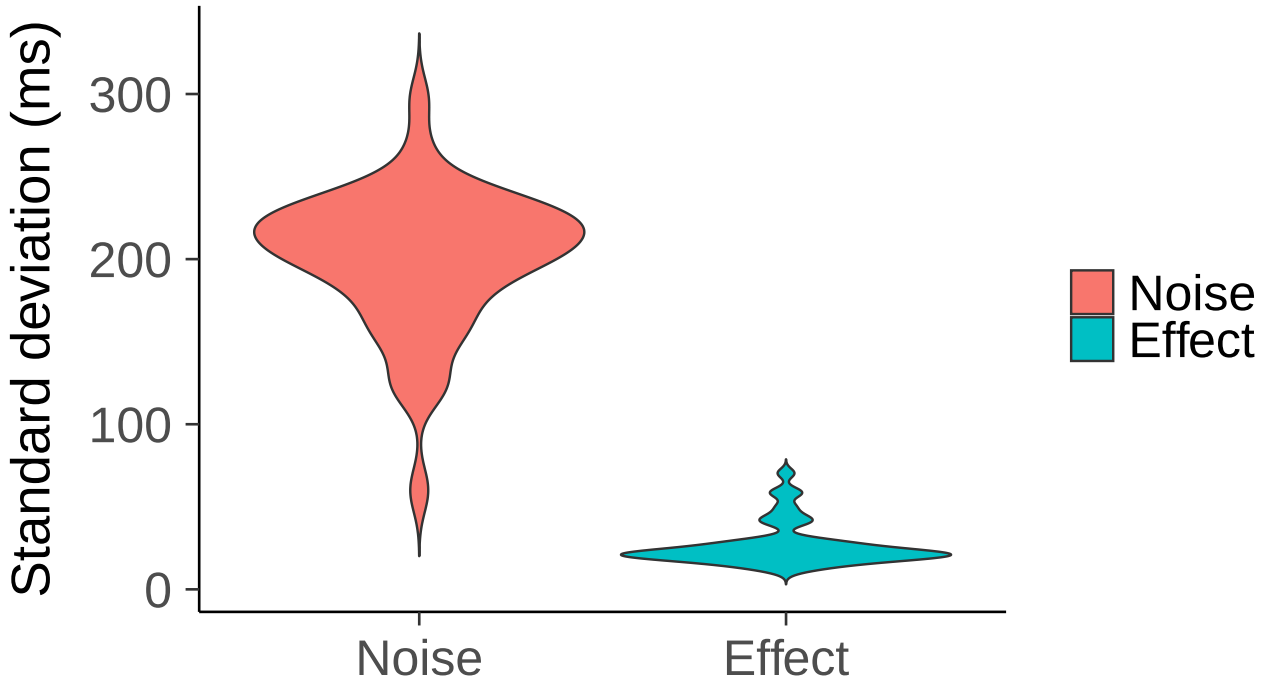
```
ids.sel <- 1:44

stn <- matrix(ncol = 3, nrow = 44)
for(i in ids.sel){
  tmp <- readRDS(paste0("results_hierarchical_modeling/dataset", i, "_normal.RDS"))
  stn[i, 1] <- mean(tmp$s2Theta / tmp$s2)
  stn[i, 2] <- get.I(subset(dat, dataset_id == i))
  stn[i, 3] <- get.K(subset(dat, dataset_id == i))
}

# plot(gamma_sum[, 1], stn[, 1])
# abline(0,1)
# cor(gamma_sum[, 1], stn[, 1])
#
# plot(sqrt(gamma_sum[, 1]), sqrt(stn[, 1]))
# abline(0,1)
```

```
allthetas <- list()
for(i in ids.sel){
  tmp <- readRDS(paste0("results_hierarchical_modeling/dataset", i, "_normal.RDS"))
  tmp2 <- colMeans(tmp$theta)
  allthetas[[paste0("dataset_", i)]] <- tmp2
}
```

```r
tt <- unlist(allthetas)

par(mgp = c(2.3, .7, 0), cex = 1.3)
plot(1:length(tt), sort(tt) * 1000
     , pch = 20, col = adjustcolor("slateblue", .4)
     , axes = F
     , ylab = "Effect (ms)"
     , xlab = "Participant")
axis(1, c(76, 5805))
axis(2, seq(-.2, .4, .2)*1000, las = 2)
abline(h=0, col = adjustcolor(1, .6), lty = 2)
```

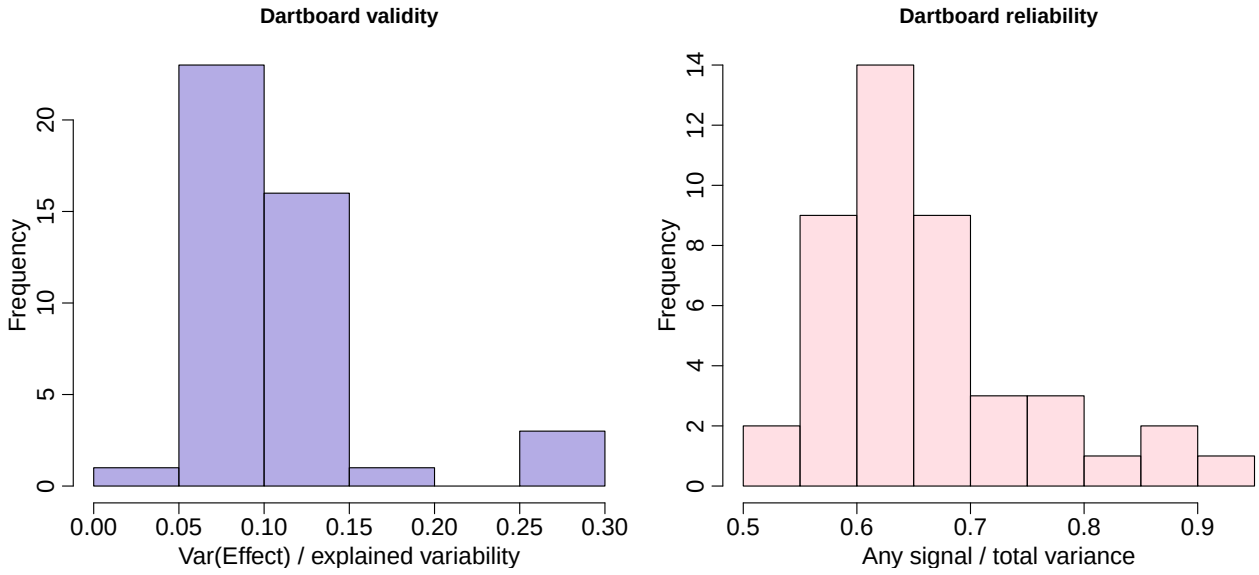



```r
alldatas <- matrix(nrow = length(ids.sel), ncol = 2)
for(i in ids.sel){
  tmp <- readRDS(paste0("results_hierarchical_modeling/dataset", i, "_normal.RDS"))
  tmp2 <- c(mean(tmp$s2Theta), mean(tmp$s2))
  alldatas[i, ] <- tmp2
}
datf <- data.frame(data = factor(c(rep(0, length(ids.sel)), rep(2, length(ids.sel))), labels = c("Noise
                   , rt = c(sqrt(alldatas[, 2])*1000, sqrt(alldatas[, 1])*1000))

p<-ggplot(datf, aes(x=data, y=rt, fill=data)) +
  geom_violin(trim=FALSE, scale = "width") +
   xlab("") +
   ylab("Standard deviation (ms)") +
   theme_apa() +
   theme(legend.title = element_blank()
         , axis.text=element_text(size=22)
        , axis.title=element_text(size=24)
        , legend.text=element_text(size=22))

p
```

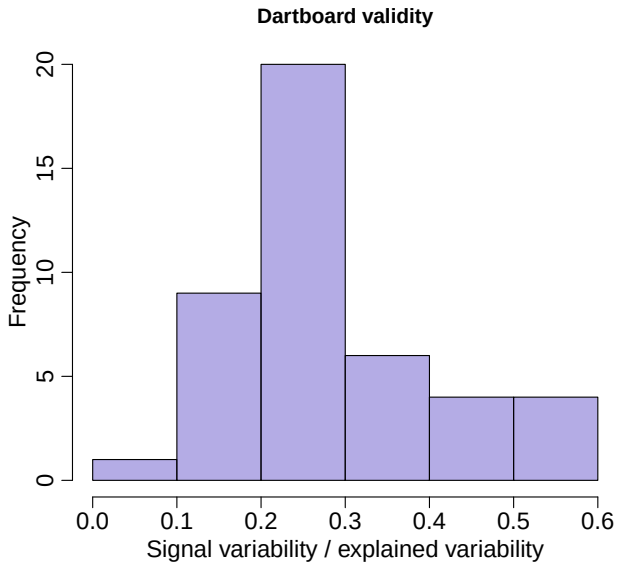


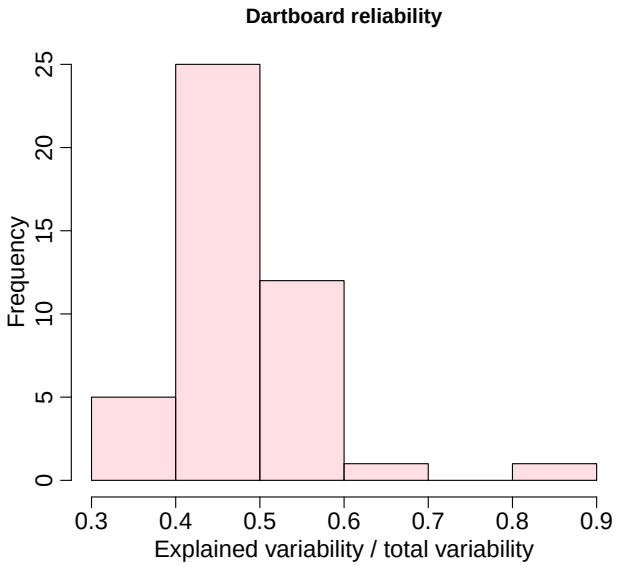
Dartboard validity

```
ids.sel <- 1:44

allvariances <- matrix(ncol = 2, nrow = 44)
for(i in ids.sel){
  tmp <- readRDS(paste0("results_hierarchical_modeling/dataset", i, "_normal.RDS"))
  allvariances[i, 1] <- mean((tmp$s2Theta) / (2* tmp$s2Alpha + tmp$s2Theta))
  allvariances[i, 2] <- mean((tmp$s2Alpha + .5*tmp$s2Theta) / (tmp$s2Alpha + .5*tmp$s2Theta + tmp$s2))
}

layout(matrix(1:2, ncol = 2))
par(mgp = c(2, .7, 0), mar = c(3,3.5,3,1), cex.lab = 1.3, cex.axis = 1.3)

hist(sqrt(allvariances[, 1])
     , main = "Dartboard validity"
     , xlab = "Var(Effect) / explained variability"
     , col = adjustcolor("slateblue", .5))
hist(sqrt(allvariances[, 2])
     , main = "Dartboard reliability"
     , xlab = "Any signal / total variance"
     , col = adjustcolor("pink", .5))
```

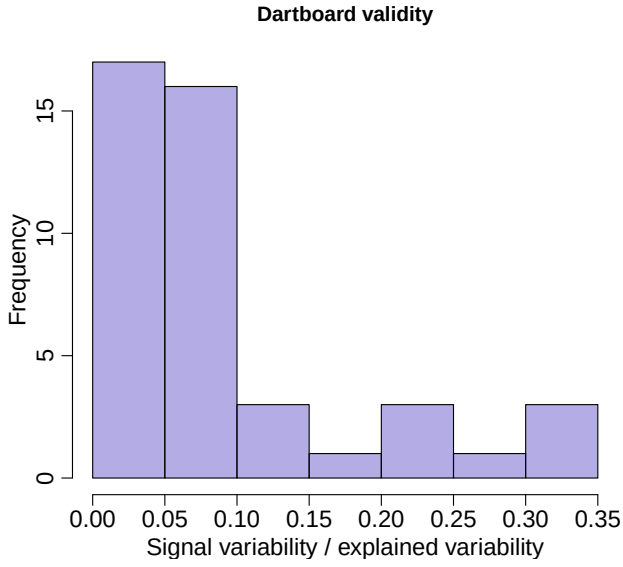


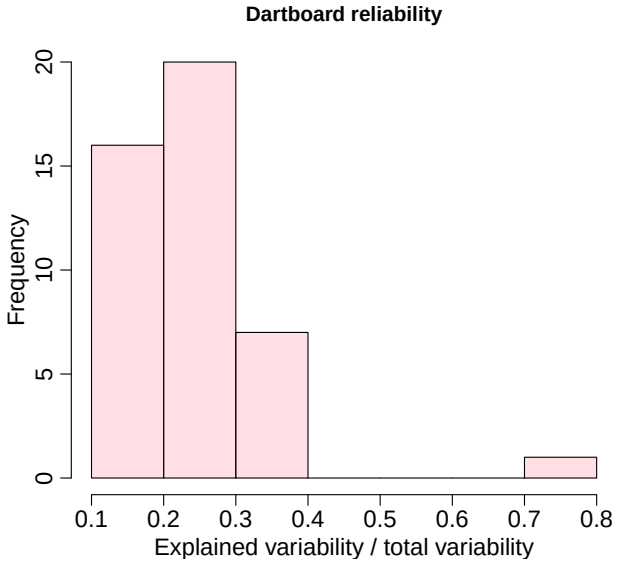

```r
ids.sel <- 1:44

alt.val <- matrix(ncol = 2, nrow = 44)
for(i in ids.sel){
  tmp <- readRDS(paste0("results_hierarchical_modeling/dataset", i, "_normal.RDS"))
  I <- ncol(tmp$alpha)
  J <- 2
  K <- 1000
  cond <- rep(c(0, 1), each = K, I)
  sub <- rep(1:I, each = J*K)
  dat.total <- rnorm(I*J*K
                     , mean = colMeans(tmp$alpha)[sub] + cond * colMeans(tmp$theta)[sub]
                     , mean(sqrt(tmp$s2)))
  dat.rel <- rnorm(I*J*K
                   , mean = colMeans(tmp$alpha)[sub] + cond * colMeans(tmp$theta)[sub]
                   , 0)
  dat.eff <- rnorm(I*J*K
                   , mean = mean(colMeans(tmp$alpha)) + cond * colMeans(tmp$theta)[sub]
                   , 0)
  alt.val[i, 1] <- var(dat.eff) / (var(dat.rel))
  alt.val[i, 2] <- (var(dat.rel)) / var(dat.total)
}

layout(matrix(1:2, ncol = 2))
par(mgp = c(2, .7, 0), mar = c(3,3.5,3,1), cex.lab = 1.3, cex.axis = 1.3)

hist(sqrt(alt.val[, 1])
     , main = "Dartboard validity"
     , xlab = "Signal variability / explained variability"
     , col = adjustcolor("slateblue", .5))
hist(sqrt(alt.val[, 2])
     , main = "Dartboard reliability"
     , xlab = "Explained variability / total variability"
     , col = adjustcolor("pink", .5))
```





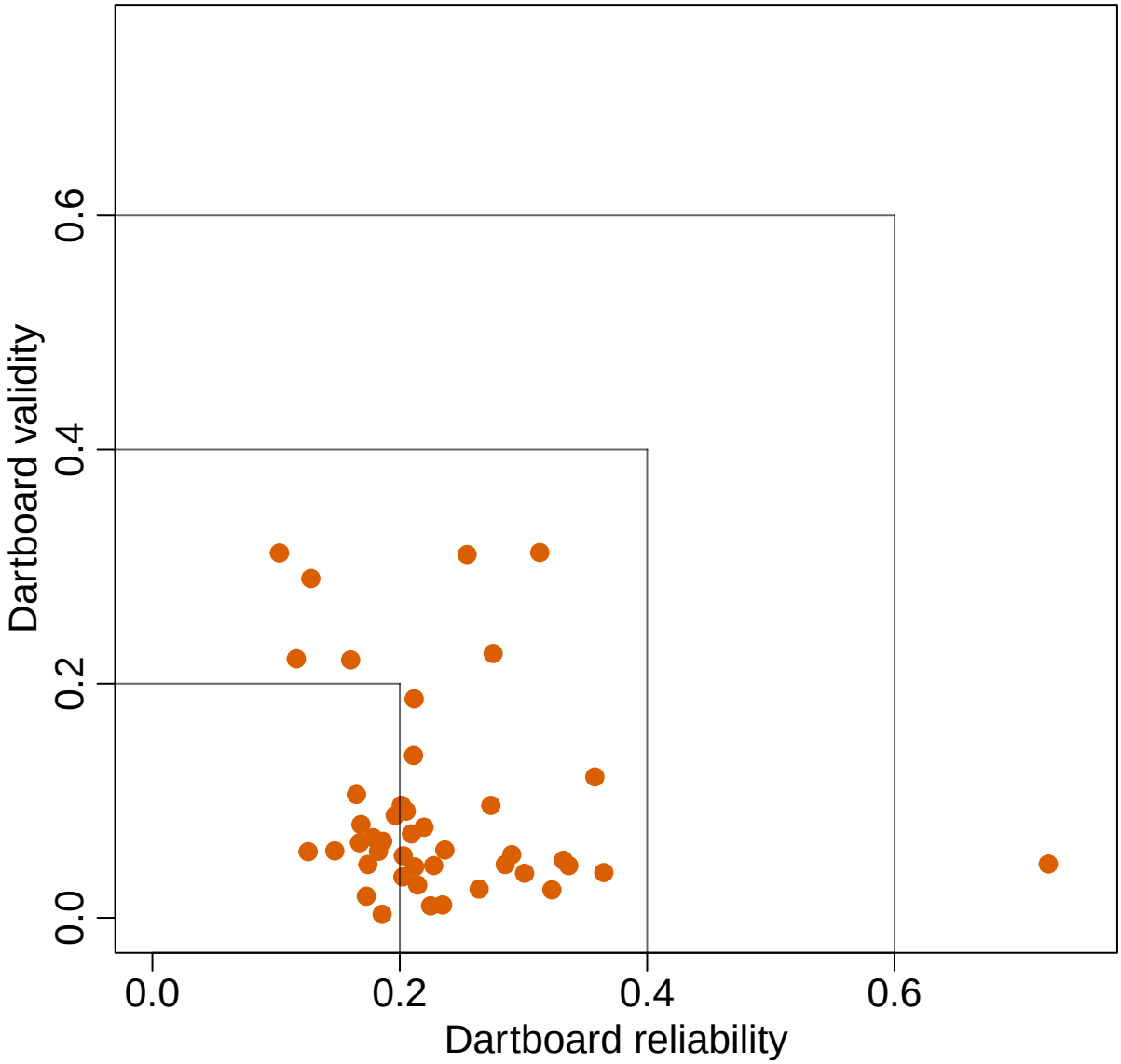

```
layout(matrix(1:2, ncol = 2))
par(mgp = c(2, .7, 0), mar = c(3,3.5,3,1), cex.lab = 1.3, cex.axis = 1.3)


hist((alt.val[, 1])
     , main = "Dartboard validity"
     , xlab = "Signal variability / explained variability"
     , col = adjustcolor("slateblue", .5))
hist((alt.val[, 2])
     , main = "Dartboard reliability"
     , xlab = "Explained variability / total variability"
     , col = adjustcolor("pink", .5))
```






```
myCols <- RColorBrewer::brewer.pal(8, "Dark2")

par(mgp = c(2, .7, 0), mar = c(3,3.5,3,1), cex.lab = 1.3, cex.axis = 1.3)

plot(alt.val[, 2], alt.val[, 1]
```

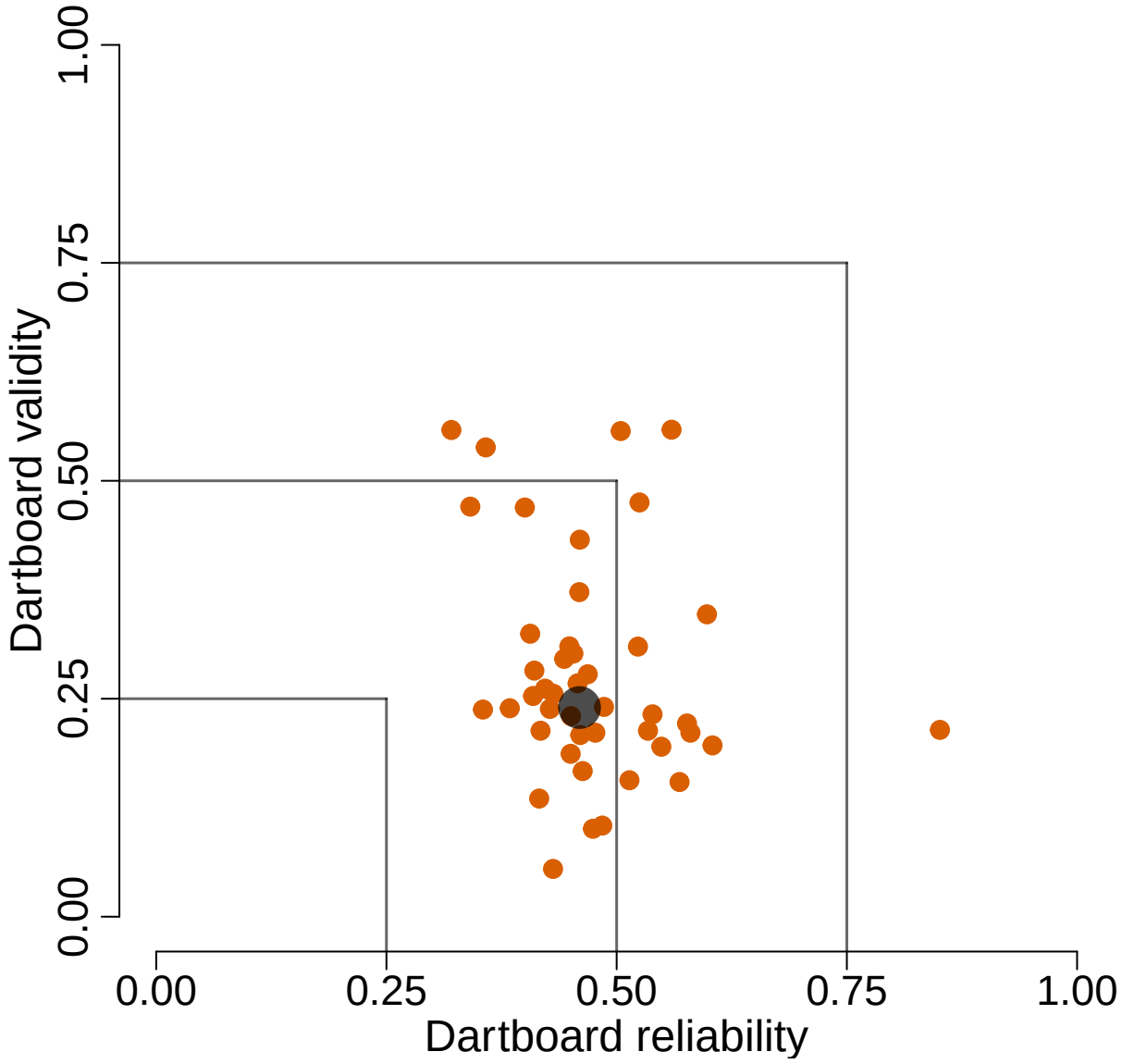
```
     , col = myCols[2]
     , pch = 19, cex = 1.3
     , xlab = "Dartboard reliability"
     , ylab = "Dartboard validity"
     , xlim = c(0, .75), ylim = c(0, .75))
lines(c(.2,.2), c(-.1, .2), col = adjustcolor(1, .6))
lines(c(-.1,.2), c(.2, .2), col = adjustcolor(1, .6))

lines(c(.4,.4), c(-.1, .4), col = adjustcolor(1, .6))
lines(c(-1,.4), c(.4, .4), col = adjustcolor(1, .6))

lines(c(.6,.6), c(-.1, .6), col = adjustcolor(1, .6))
lines(c(-1,.6), c(.6, .6), col = adjustcolor(1, .6))
```




```
#abline(lm(alt.val[, 1] ~ alt.val[, 2]))
```

```
par(mgp = c(2, .7, 0), mar = c(3,3.5,3,1), cex.lab = 1.5, cex.axis = 1.4, lwd = 1.5)
```

```r
plot(sqrt(alt.val[, 2]), sqrt(alt.val[, 1])
     , col = myCols[2]
     , pch = 19, cex = 1.3
     , xlab = "Dartboard reliability"
     , ylab = "Dartboard validity"
     , xlim = c(0, 1), ylim = c(0, 1)
     , axes = F
     )

axis(1, seq(0, 1, .25))
axis(2, seq(0, 1, .25))

lines(c(.25,.25), c(-.1, .25), col = adjustcolor(1, .6))
lines(c(-.1,.25), c(.25, .25), col = adjustcolor(1, .6))

lines(c(.5,.5), c(-.1, .5), col = adjustcolor(1, .6))
lines(c(-1,.5), c(.5, .5), col = adjustcolor(1, .6))

lines(c(.75,.75), c(-.1, .75), col = adjustcolor(1, .6))
lines(c(-1,.75), c(.75, .75), col = adjustcolor(1, .6))

points(median(sqrt(alt.val[, 2])), median(sqrt(alt.val[, 1]))
       , cex = 3, pch = 19, col = adjustcolor(1, .7))
```



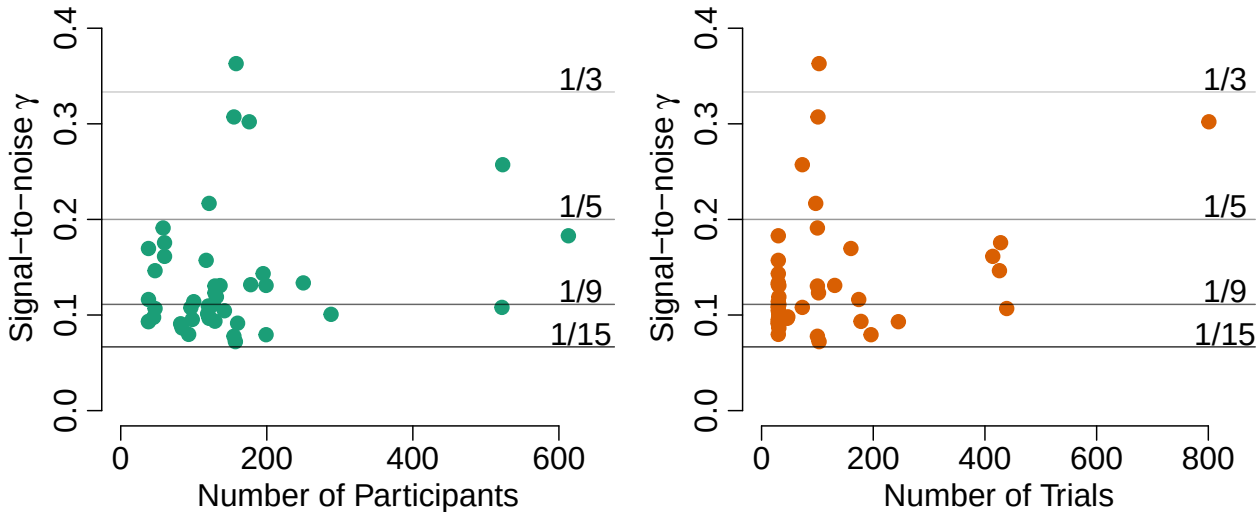

```
#abline(lm(alt.val[, 1] ~ alt.val[, 2]))
```

```
make.stn.plot <- function(x, y, cols){
  plot(x, y
       , col = cols, pch = 19
       , cex = 1.2)
  abline(h = 1/13, col = adjustcolor(1, .8))
  abline(h = 1/8, col = adjustcolor(1, .6))
  abline(h = 1/5, col = adjustcolor(1, .4))
  abline(h = 1/3, col = adjustcolor(1, .2))
}
```

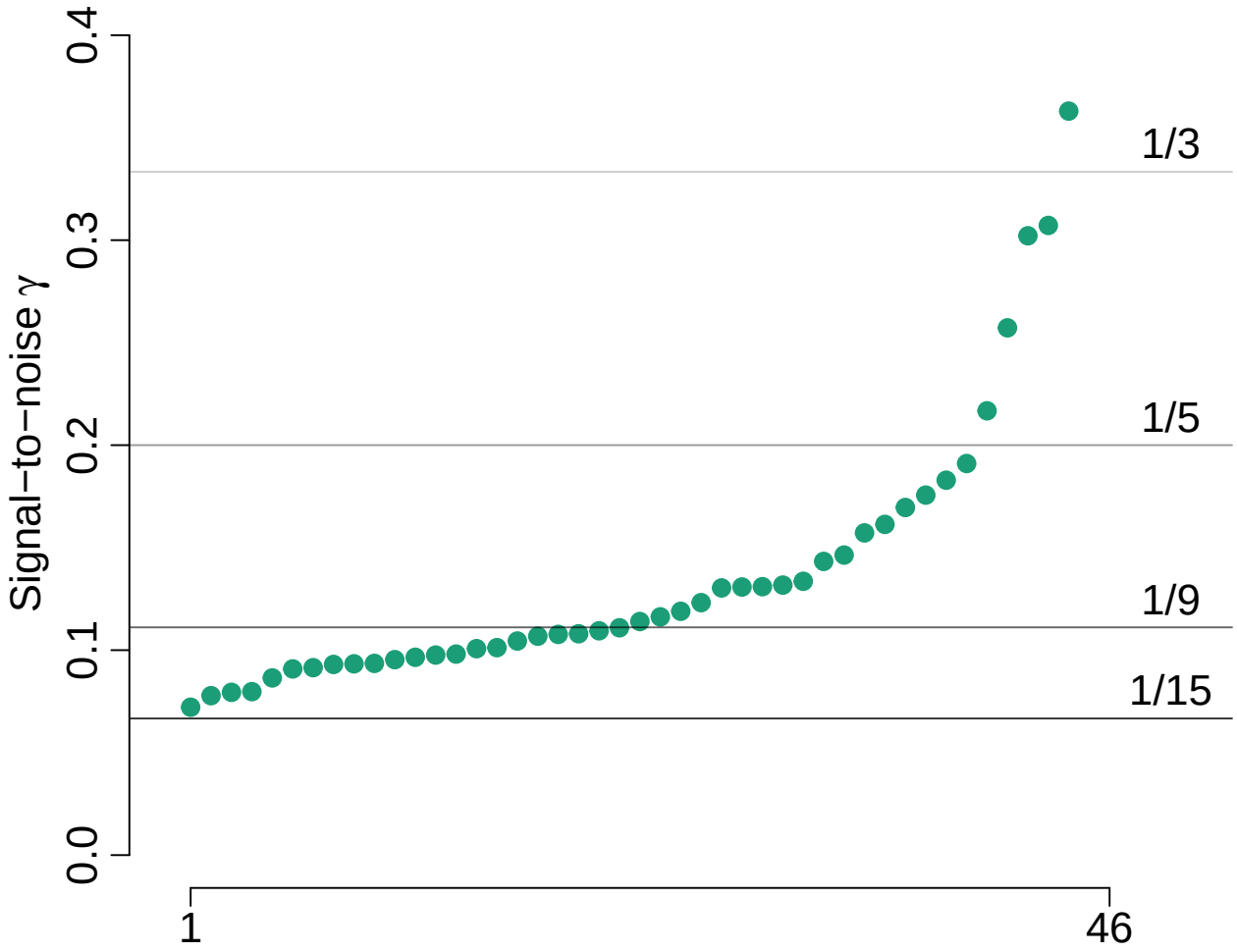
Signal-to-noise

```
myCols <- RColorBrewer::brewer.pal(8, "Dark2")
par(mgp = c(2, .7, 0), mar = c(3,3.5,.5,1), cex.lab = 1.3, cex.axis = 1.3)
layout(matrix(1:2, ncol = 2))
```

```r
plot(stn[, 2], sqrt(stn[, 1])
     , col = myCols[1]
     , pch = 19, cex = 1.3
     , ylab = expression("Signal-to-noise" ~ gamma)
     , xlab = "Number of Participants"
     , axes = F
     , xlim = c(0, 650)
     , ylim = c(0, .4))
axis(1, seq(0, 600, 200))
axis(2, seq(0, .4, .1))
abline(h = 1/15, col = adjustcolor(1, .8))
text(630, 1/15 + .014, "1/15", cex = 1.3)
abline(h = 1/9, col = adjustcolor(1, .6))
text(630, 1/9 + .014, "1/9", cex = 1.3)
abline(h = 1/5, col = adjustcolor(1, .4))
text(630, 1/5 + .014, "1/5", cex = 1.3)
abline(h = 1/3, col = adjustcolor(1, .2))
text(630, 1/3 + .014, "1/3", cex = 1.3)


plot(stn[, 3], sqrt(stn[, 1])
     , col = myCols[2]
     , pch = 19, cex = 1.3
     , ylab = expression("Signal-to-noise" ~ gamma)
     , xlab = "Number of Trials"
     , axes = F
     , xlim = c(0, 850)
     , ylim = c(0, .4))
axis(1, seq(0, 800, 200))
axis(2, seq(0, .4, .1))
abline(h = 1/15, col = adjustcolor(1, .8))
text(830, 1/15 + .014, "1/15", cex = 1.3)
abline(h = 1/9, col = adjustcolor(1, .6))
text(830, 1/9 + .014, "1/9", cex = 1.3)
abline(h = 1/5, col = adjustcolor(1, .4))
text(830, 1/5 + .014, "1/5", cex = 1.3)
abline(h = 1/3, col = adjustcolor(1, .2))
text(830, 1/3 + .014, "1/3", cex = 1.3)
```




```
myCols <- RColorBrewer::brewer.pal(8, "Dark2")
par(mgp = c(2, .7, 0), mar = c(3,3.5,.5,1), cex.lab = 1.4, cex.axis = 1.4)

plot(1:44, sort(sqrt(stn[, 1]))
     , col = myCols[1]
     , pch = 19, cex = 1.3
     , ylab = expression("Signal-to-noise" ~ gamma)
     , xlab = ""
     , axes = F
     , xlim = c(0, 50)
     , ylim = c(0, .4))
axis(1, c(1, 46))
axis(2, seq(0, .4, .1))
abline(h = 1/15, col = adjustcolor(1, .8))
text(49, 1/15 + .014, "1/15", cex = 1.4)
abline(h = 1/9, col = adjustcolor(1, .6))
text(49, 1/9 + .014, "1/9", cex = 1.4)
abline(h = 1/5, col = adjustcolor(1, .4))
text(49, 1/5 + .014, "1/5", cex = 1.4)
abline(h = 1/3, col = adjustcolor(1, .2))
text(49, 1/3 + .014, "1/3", cex = 1.4)
```



Comparing to Splithalf

```
get.splithalf <- function(dats){
  dats$congruency <- factor(dats$congruency, levels = 1:2, labels = c("congruent", "incongruent"))
  difference <- splithalf(data = dats,
                          outcome = "RT",
                          score = "difference",
                          halftype = "random",
                          permutations = 2000,
                          var.RT = "rt",
                          var.participant = "subject",
                          var.compare = "congruency",
                          compare1 = "congruent",
                          compare2 = "incongruent",
                          average = "mean",
                          plot = TRUE)
  return(c(sh = difference$final_estimates$splithalf, sb =  difference$final_estimates$spearmanbrown))
}
```

```
dat_sub <- subset(dat, dataset_id == 4)
dat_sub <- subset(dat_sub, accuracy == 1)
dat_sub <- subset(dat_sub, rt > .2)
dat_sub <- subset(dat_sub, rt < 2.5)
```

```r
dat_sub <- subset(dat_sub, congruency %in% 1:2)
issue <- issue.sub(dat_sub)
dat_sub <- subset(dat_sub, !(subject %in% issue))
get.splithalf(dat_sub)
```

```
## Warning in splithalf(data = dats, outcome = "RT", score = "difference", : no
## condition variable specified, splithalf will treat all trials as one condition
```

```
##   |                                                                      |
```

```r
num <- c(seq(15, 50, 10), seq(50, 350, 50))
res.splithalf <- matrix(ncol = 2, nrow = length(num)+1)
res.splithalf[length(num) + 1, ] <- get.splithalf(dat_sub)
```

```
## Warning in splithalf(data = dats, outcome = "RT", score = "difference", : no
## condition variable specified, splithalf will treat all trials as one condition
```

```
##   |                                                                      |
```

```r
res.bayes <- list()


for(i in 1:length(num)){
  new_df <- dat_sub %>% group_by(subject, congruency) %>% slice_sample(n=num[i])
  res.splithalf[i, ] <- get.splithalf(new_df)
  res.bayes[[paste0("dataset_K", num[i])]] <- genModOneTask(new_df)
  saveRDS(res.bayes[[paste0("dataset_K", num[i])]], paste0("results_hierarchical_modeling/dataset_K", nu
  print(num[i])
}
```

```
## Warning in splithalf(data = dats, outcome = "RT", score = "difference", : no
## condition variable specified, splithalf will treat all trials as one condition
```

```
##   |                                                                      |
```

```
## [1] "Calculating split half estimates"
## [1] "split half estimates for 2000 random splits"
##   condition  n spearmanbrown SB_low SB_high
## 1       all 47          0.29  -0.04    0.56
## [1] "this could be reported as: using 2000 random splits, the spearman-brown corrected reliability  
## [1] 25
## Warning in splithalf(data = dats, outcome = "RT", score = "difference", : no
## condition variable specified, splithalf will treat all trials as one condition
##   |                                                                              |
```

```
## [1] "split half estimates for 2000 random splits"
##   condition  n spearmanbrown SB_low SB_high
## 1       all 47          0.39   0.08    0.63
## [1] "this could be reported as: using 2000 random splits, the spearman-brown corrected reliability
```

```
## [1] 150
```

```
## Warning in splithalf(data = dats, outcome = "RT", score = "difference", : no
## condition variable specified, splithalf will treat all trials as one condition
```

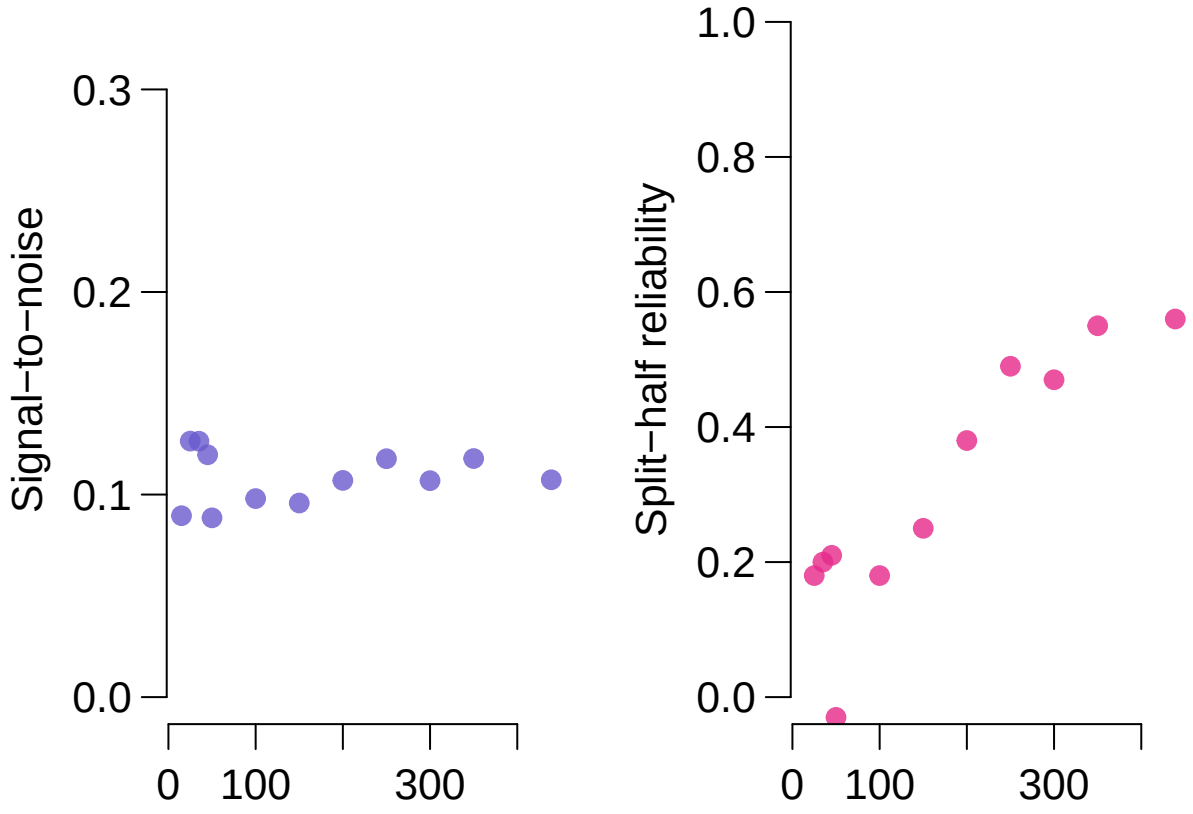
```
##   |                                                                              |
```

```
res.bayes[[paste0("dataset_K", 222)]] <- genModOneTask(dat_sub)
```

```
splithalf_check <- matrix(ncol = 4, nrow = length(num) + 1)
for(i in 1:(length(num))){
  new_df <- dat_sub %>% group_by(subject, congruency) %>% slice_sample(n=num[i])
  tmp <- res.bayes[[paste0("dataset_K", num[i])]]
```

```
  splithalf_check[i, 1] <- sqrt(mean(tmp$s2Theta / tmp$s2))
  splithalf_check[i, 2] <- get.I(new_df)
  splithalf_check[i, 3] <- get.K(new_df)
  splithalf_check[i, 4] <- res.splithalf[i, 1]
}

tmp <- res.bayes[[paste0("dataset_K", 222)]]
splithalf_check[length(num) + 1, 1] <- sqrt(mean(tmp$s2Theta / tmp$s2))
splithalf_check[length(num) + 1, 2] <- get.I(dat_sub)
splithalf_check[length(num) + 1, 3] <- get.K(dat_sub)
splithalf_check[length(num) + 1, 4] <- res.splithalf[length(num) + 1, 1]
```

```
layout(matrix(1:2, ncol = 2))
par(mgp = c(2.3, .7, 0), cex = 1.3, mar = c(2.2, 3.5, .5, 1))
plot(splithalf_check[, 3], splithalf_check[, 1], ylim = c(0, 1/3)
, pch = 19, col = adjustcolor("slateblue", .8)
, axes = F
, ylab = "Signal-to-noise"
, xlab = "Trials per condition")
axis(1, seq(0, 400, 100))
axis(2, seq(0, .3, .1), las = 2)
plot(splithalf_check[, 3], splithalf_check[, 4], ylim = c(0, 1)
, pch = 19, col = adjustcolor(myCols[4], .8)
, axes = F
, ylab = "Split-half reliability"
, xlab = "Trials per condition")
axis(1, seq(0, 400, 100))
axis(2, seq(0, 1, .2), las = 2)
```

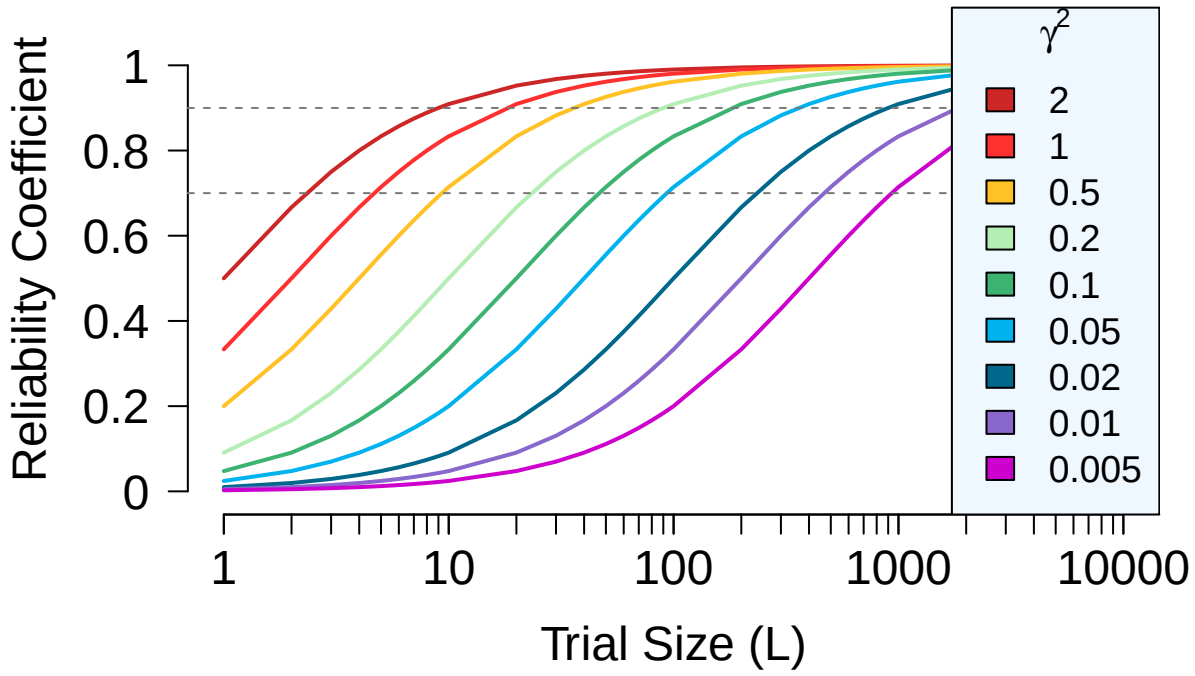


gamma-plots

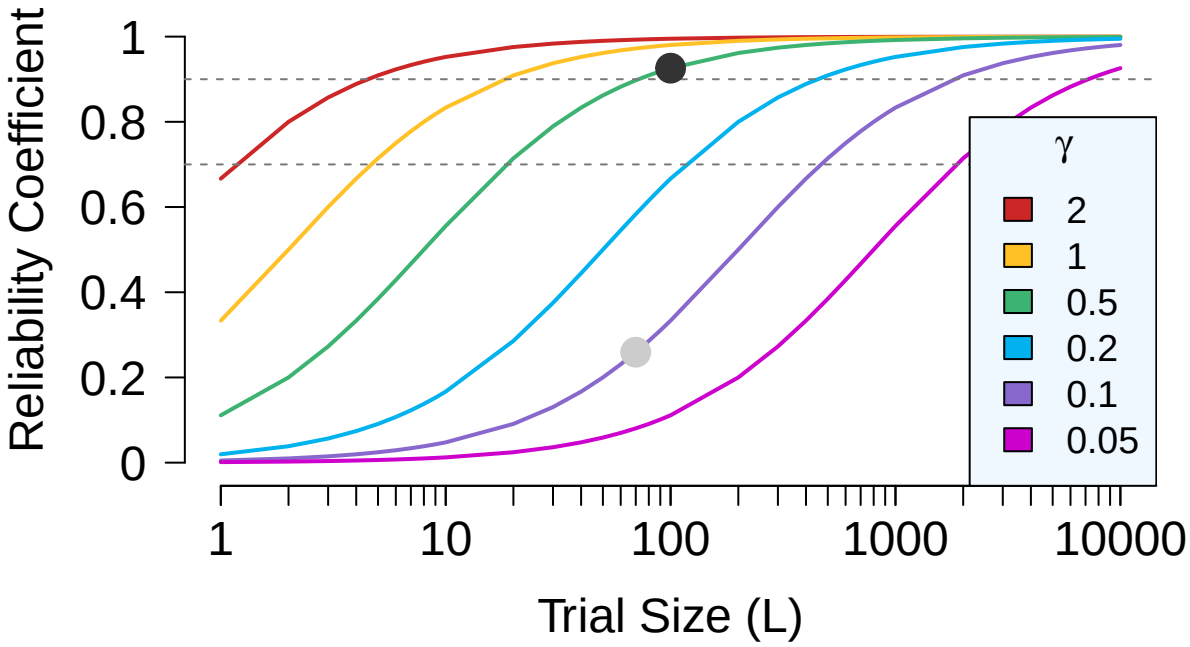
```
# This function generates a plot of the reliability coefficient against trial size.
# Parameters:
#   Var: A Boolean indicating the variant of the plot. Default is TRUE.
#   dots: An optional list of 'l' (trial sizes) and 'g' (gamma values) to plot as points.

makeRelCoefFig = function(Var = T, dots = NULL){
  # Define the reliability function
  rel = function(g2, L) g2 / (g2 + 2/L)

  # Set graphical parameters
  par(mgp=c(2,1,0), mar = c(5,5,4,2) + 0.1)

  # Define range for trial sizes
  n1 = 1:9
  n2 = 0:3

  # Define colors for the lines in the plot
  colours = c("firebrick3", "firebrick1", "goldenrod1",
              "darkseagreen2", "mediumseagreen", "deepskyblue2",
              "deepskyblue4", "mediumpurple3", "magenta3"
  )

  # Conditional plotting based on the Var parameter
  if (Var == F){
    colours = colours[c(1,3,5,6,8,9)]
    gamma_leg = c(2, 1, .5, .2, .1, .05)
    gamma = gamma_leg^2
    gamma_lab = expression(gamma)
  } else {
    gamma = c(2, 1, .5, .2, .1, .05, .02, .01, .005)
    gamma_leg = gamma
    gamma_lab = expression(gamma^2)
  }
  n = c(as.vector(outer(n1, n2, function(x, y) {x*10^y})), 10000)
  xlims = c(0,4)

  # Calculate reliability coefficients
  R = outer(gamma, n, rel)

  # Define major ticks for the x-axis
  majors = 0:5

  # Start plotting
  plot(NA, NA,
       type='l',
       lwd=2,
       axes=FALSE,
       xlab=" ",
       ylab=" ",
       ylim = c(-.01,1.1),
       xlim = xlims,
       xaxt = "n",
```

```
      yaxt = "n",
      frame.plot = F)

  # Add lines for each gamma value
  for(i in 1:length(gamma)) {
    lines(log10(n), R[i,], lwd=2, col=colours[i])
  }

  # Add labels and axes
  mtext("Trial Size (L)", side = 1, line = 3, cex=1.5)
  mtext("Reliability Coefficient", side = 2, line = 3.5, cex=1.5)
  axis(2, at = seq(0,1,length.out=6), labels =  seq(0,1,length.out=6),
       las=1, cex.axis=1.5)
  axis(1, at=majors, labels=10^majors, cex.axis=1.5)
  axis(1, at=log10(n), labels=NA, cex.axis=1.5)


  # Add horizontal lines for reference
  abline(h=c(.7, .9), lty=2, col = "gray47")

  # Add legend
  legend("bottomright",
         legend=gamma_leg,
         fill=colours[1:length(gamma)],
         title=gamma_lab,
         bg='aliceblue',
         cex = 1.2
  )

  # Function to generate grey shades (seems to be missing its first line, corrected here)
  enerate_grey_shades = function(n) {
    shades = grey(seq(0, 1, length.out = n))
    shades = paste0("gray", round(seq(20, 80, length.out = n)))
    return(shades)
  }
  ldots <- length(dots$l)
  # Add points to the plot if 'dots' is provided
  if (!is.null(dots) && length(dots) > 0){
    cols = enerate_grey_shades(ldots)
    for (i in 1:ldots){
      l = dots$l[i]
      g2 = dots$g[i]^2
      points(log10(l), rel(g2,l), pch = 19, col = cols[i], cex = 2)
    }
  }
}


# Examples of Using the makeRelCoefFig Function¡

# 1. Regular Plot
# Description: Generates the plot with default settings (Var = TRUE).
# Usage:
```

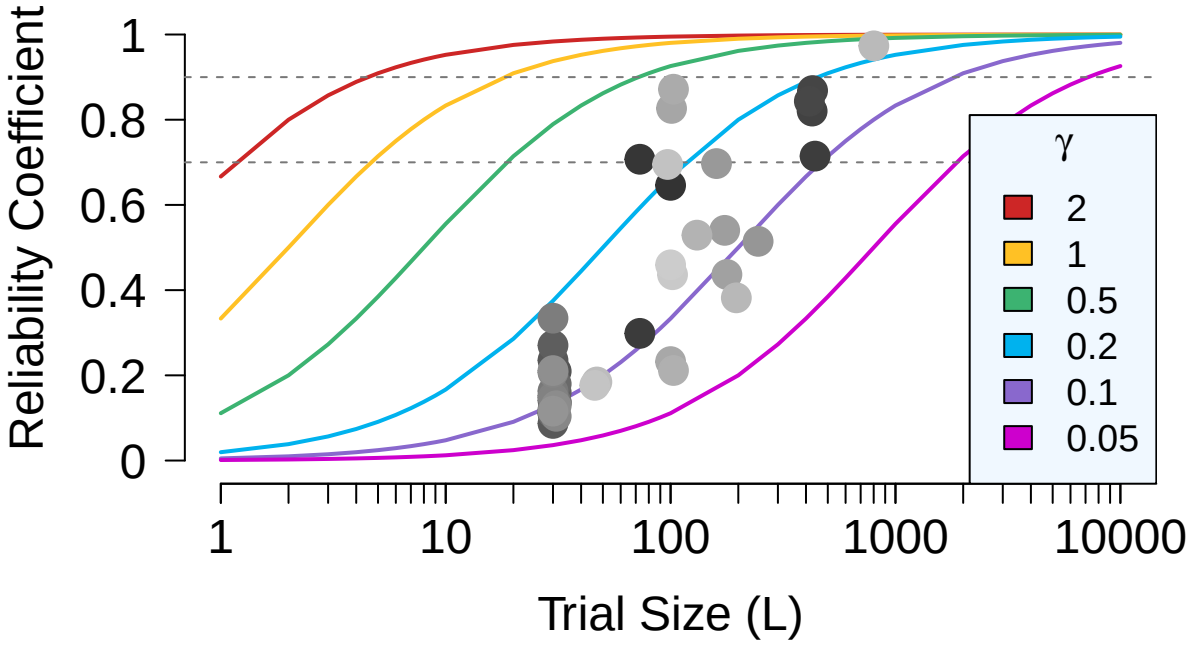
```
makeRelCoefFig()
```




```
# 2. Plot with Gamma Instead of Gamma^2
# Description: Generates the plot using gamma values directly, instead of their squares (sets Var to FA
# Usage:
makeRelCoefFig(Var = FALSE)

# 3. Adding Dots to Represent a Particular Study
# Description: Adds specific points to the plot, representing particular trial sizes (l) and gamma valu
# Useful for highlighting specific data points or studies.
# Usage:
dots = list("l" = c(100, 70), "g" = c(.5, .1))
makeRelCoefFig(dots = dots, Var = FALSE)
```




```
dots = list("l" = stn[, 3], "g" = sqrt(stn[, 1]))
makeRelCoefFig(dots = dots, Var = FALSE)
```



Old stuff

```
## equations from Rouder & Mehrvarz (2023), p. 7

var_d_est <- function(di) sum((di - mean(di, na.rm = T))^2, na.rm = T)/(length(di) - 1)

get.K <- function(dat){
  dat <- subset(dat, accuracy == 1)
```

```
  dat <- subset(dat, rt > .2)
  dat <- subset(dat, rt < 2.5)
  dat <- subset(dat, congruency %in% 1:2)
  issue <- issue.sub(dat)
  dat <- subset(dat, !(subject %in% issue))
  I <- length(unique(dat$subject))
  K <- table(dat$subject)/2
  Kall <- round(mean(K, na.rm = T))
  return(Kall)
}

get.I <- function(dat){
  dat <- subset(dat, accuracy == 1)
  dat <- subset(dat, rt > .2)
  dat <- subset(dat, rt < 2.5)
  dat <- subset(dat, congruency %in% 1:2)
  issue <- issue.sub(dat)
  dat <- subset(dat, !(subject %in% issue))
  return(length(unique(dat$subject)))
}

issue.sub <- function(dats){
  tmp <- table(dats$subject, dats$congruency)
  tmp <- cbind(tmp, as.numeric(rownames(tmp)))
  return(tmp[tmp[, 1] < 10 | tmp[, 2] < 10, 3])
}

sigma2_est <- function(dat){
  I <- length(unique(dat$subject))
  K <- table(dat$subject)/2
  Kall <- round(mean(K, na.rm = T))

  df <- I*2*(Kall - 1)
  dev <- sum((dat$rt - mean(dat$rt, na.rm = T))^2, na.rm = T)

  return(c(sigma2 = dev / df, K = Kall))
}

gamma2_est <- function(dat){
  # data exclusion
  dat <- subset(dat, accuracy == 1)
  dat <- subset(dat, rt > .2)
  dat <- subset(dat, rt < 2.5)
  dat <- subset(dat, congruency %in% 1:2)
  issue <- issue.sub(dat)
  dat <- subset(dat, !(subject %in% issue))

  m_ij <- tapply(dat$rt, list(dat$subject, dat$congruency), mean, na.rm = T)
  d_i <- m_ij[, 2] - m_ij[, 1]

  var_d <- var_d_est(d_i)
  sigma2 <- sigma2_est(dat)[1]
  K <- sigma2_est(dat)[2]
```

```
  return(c(gamma2 = var_d / sigma2 - 2/K, K = K, var_d = var_d, sigma2 = sigma2, d = mean(d_i, na.rm = T
}
```

```
ids <- unique(dat$dataset_id)
gamma_sum <- matrix(NA, nrow = length(ids), ncol = 6)
gamma_sum[, 6] <- ids

for(i in ids){
  dat_sub <- subset(dat, dataset_id == i)
  gamma_sum[i, 1:5] <- gamma2_est(dat_sub)
}
```

```
plot(gamma_sum[, 2], gamma_sum[, 1])
abline(h = 0)
plot(gamma_sum[, 4], gamma_sum[, 3])
```

```
id_sel <- ids[gamma_sum[, 1] < 0]

arguments <-  list()
arguments <- add_argument(
 list = arguments,
 conn = conn,
 variable = "dataset_id",
 operator = "equal",
 values = id_sel
)

query_results22 <- query_db(
 conn = conn,
 arguments = arguments,
 target_table = "dataset_table",
 target_vars = c("default", "publication_code")
)
```

```
layout(matrix(1:2, ncol = 2))
par(mgp = c(2, .7, 0), mar = c(3,3.3,.5,.8), cex = 2)
id_sel <- ids[gamma_sum[, 1] > 0]
hist(sqrt(gamma_sum[gamma_sum[, 6] %in% id_sel, 1])
     , xlab = expression("Signal-to-noise ratio" ~ gamma)
     , main = ""
     , col = "slategray1"
     , xaxt = "n")
axis(1, seq(0, .3, .1))
text(.25, 10, paste0("N = ", length(id_sel)), cex = 1.3)

plot(sqrt(gamma_sum[, 4])*1000, sqrt(gamma_sum[, 3])*1000
     , ylab = "SD of the effect (ms)", xlab = "SD of the observations (ms)"
     , xaxt = "n"
     , xlim = c(25, 375)
     , frame.plot = F)
axis(1, c(50, 150, 250, 350))
points(sqrt(gamma_sum[gamma_sum[, 6] %in% id_sel, 4])*1000, sqrt(gamma_sum[gamma_sum[, 6] %in% id_sel, 3
abline(a = 0, b = 1/4.5, col = "gray60", lwd = 4)
abline(a = 0, b = 1/3, col = "gray40", lwd = 4)
```

```r
abline(a = 0, b = 1/7, col = "gray80", lwd = 4)
legend("topleft", c(expression(gamma ~ "= 1/3"), expression(gamma ~ "= 1/4.5"), expression(gamma ~ "= 1/
       , lwd = 4, col = paste0("gray", seq(40, 80, 20)), cex = 1.2, bty = "n")
```