Supplemental data to the article: The MDD concept for establishing trust in non-significant results - a critical review

Mair, MM, Kattwinkel, M, Jakoby, O and Hartig, F

last update: 2020-07-09

Contents

1 MDD power

1.1 R code - MDD power with equal variances

For a one-sided t test, the power of the MDD corresponds to the area below the probability density function of t for the MDD, ranging from the critical t value (derived from the t distribution for the null hypothesis of no effect) to infinity. As the t value corresponding to the MDD by definition equals the critical t value, the power of the MDD depends solely on the chosen α-level and the degrees of freedom, which define the critical value of t.

For equal variances, the degrees of freedom for the determination of the critical t value are calculated by

$$df = \frac{N_1 + N_2}{2}$$

with N_1 and N_2 representing the sample sizes in the two groups that are compared.

The non-central distribution of t for a real effect of the size of the MDD value is calculated by including a non-centrality parameter equalling t critical.

The following R code was used to calculate and plot the power of the MDD dependent on sample size in Figure 4 B.

```r
# Define sample sizes ranging from 2 to 10
sampleSize = c(seq(from = 2, to = 10, by = 1))

# Create empty data frame
out1 = data.frame(
  N = sampleSize,
  mddpower1 = rep(NA, length(sampleSize))
)

# Calculate MDD power for each sample size and store results in data frame 'out'
for (i in 1:length(sampleSize))
  {
  N = sampleSize[i]
  df1 = N + N - 2
  # Calculate MDD power:
  mdd.power1 <- pt(qt(p = 0.05/1, df = df1, lower.tail = F),
                   df = df1, ncp = qt(0.05/1, df = df1, lower.tail = F),
                   lower.tail = F)
  out1$mddpower1[i] <- mdd.power1
}

# Plot
op = par(mfrow = c(1, 1), mar = c(5, 5, 1, 1), las = 1, cex = 1.5, yaxt = "s", xaxt = "s")
plot(mddpower1 ~ N, data = out1, ylim = c(0.5, 0.6), pch = 19, bty = "l",
     ylab = "MDD power", xlab = "Sample size", cex.lab = 1.5, yaxt = "n", xaxt = "n")
axis(2, at = c(0.5, 0.55, 0.6), cex.lab = 1, labels =T)
axis(1, at = c(seq(2, 10, 1)), labels = T, cex.lab = 1)
text(10, 0.59, "alpha = 0.05", cex= 1, pos = 2, col ="grey50")
par(op)
```

1.2 MDD power with non-equal variances

If the variances of the two compared groups are non-equal, the Welch-Satterthwaite approximation is used to calculate the degrees of freedom:

$$df = \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})^2}{\frac{s_1^4}{N_1^2 * (N_1 - 1)} + \frac{s_2^4}{N_2^2 * (N_2 - 1)}}$$

with the measured variances in the two groups, s_1^2 and s_2^2, and sample sizes N_1 and N_2.

As t-critical depends on the degrees of freedom, t-critical (= t-MDD), varies not only with sample size here, but also with the estimated variances. In consequence, the power of the MDD also varies with the estimated variances of the two groups. More specifically, the degrees of freedom become less with larger differences between the estimated variances, and t-critical becomes larger. In consequence, MDD power also becomes larger, because the non-central t distribution for larger t values becomes more skewed. Note also that the effect of variance inequality becomes less pronounced with larger sample sizes.

This can be illustrated by calculating MDD power for different levels of variance inequality between the two groups:

```
# Define variances for group 1 and 2
variances <- data.frame(
  variance1 = seq(from = 0, to = 250, by = 1),
  variance2 = seq(from = 500, to = 250, by = -1)
)
# Define sample sizes ranging from 2 to 10
sampleSize = c(2, 3, 4, 5, 6, 8, 10)

# Create empty output dataframe
out2 = data.frame(
  N = NA,
  mdd.power2 = NA,
  vardiff = NA
)

# Calculate MDD power for each sample size and different variance combinations,
# and store results in output data frame
for (i in 1:length(sampleSize))
{
  N = sampleSize[i]
  for (j in 1:length(variances$variance1))
  {
    variance1 = variances$variance1[j]
    variance2 = variances$variance2[j]
     # Calculate degrees of freedom using the Welch-Satterthwaite approximation:
    df2 = (variance1/N + variance2/N)^2 / ((variance1/N)^2/(N-1) + (variance2/N)^2/(N-1))
    # Calculate MDD power:
    mdd.power2 <- pt(qt(p = 0.05/1, df = df2, lower.tail=F),
                     df = df2, ncp = qt(0.05/1, df = df2, lower.tail=F),
                     lower.tail=F)
    temp <- data.frame(
      N = N,
      mdd.power2 = mdd.power2,
      vardiff = variance2 - variance1
    )
    out2 <- rbind(out2, temp)
  }
}
out2 <- na.omit(out2)
```

```
# Plot results:
par(mar = c(4, 4, 1, 3))
plot(mdd.power2 ~ vardiff, data = out2, type="n", bty = "l",
     ylab = "MDD power", xlab = "Difference between groups' variances")
for (k in sampleSize)
{
  sub <- subset(out2, out2$N == k)
  lines(smooth.spline(sub$vardiff, sub$mdd.power2))
  mtext(paste("N =",as.character(k)), side = 4, line = 0, las =1,
        at = max(sub$mdd.power2))
}
```

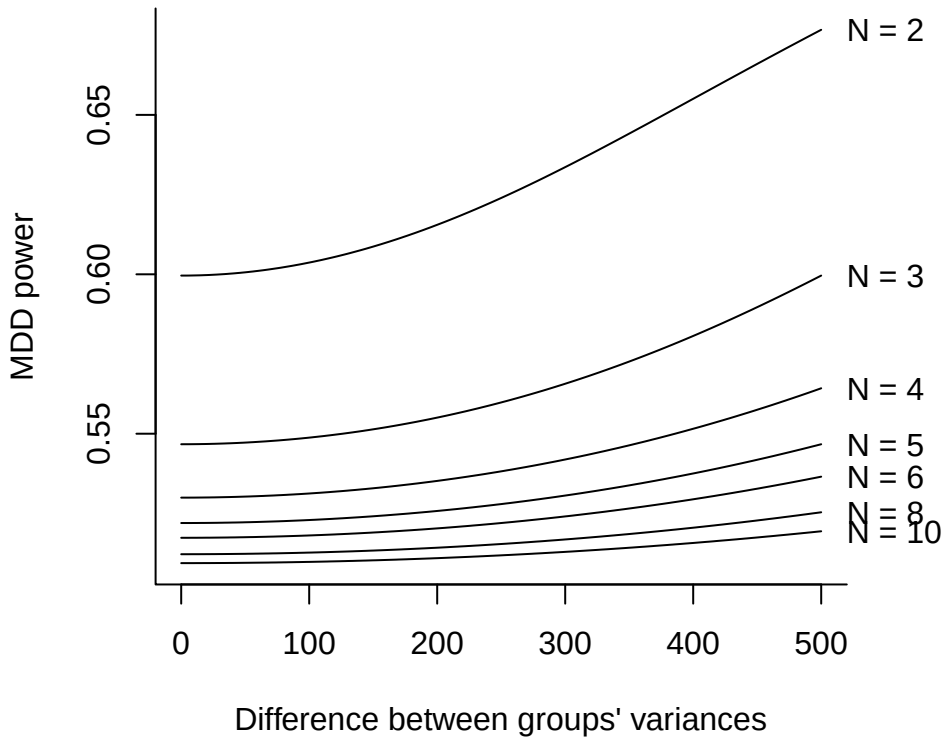


Figure S1: Dependency of MDD power on sample size and the difference between variances of the compared groups. Degrees of freedom calculated by the Welch-Satterthwaite approximation (one-sided t test). Sample sizes are indicated on the right and are always the same for both groups.

2 R function used to calculate the MDD

The following function was used to calculate the MDD in our analyses:

```
MDD=function (N1, N2, variance1, variance2 = NULL, alpha = 0.05, two.sided = TRUE,
              var.equal = TRUE)
{
  m <- NULL
  m$n1 <- N1
  m$n2 <- N2
  m$variance1 <- variance1
  m$variance2 <- variance2
  m$alpha <- alpha
```

```
  m$two.sided <- two.sided
  if (var.equal) {
    dfreedom <- N1 + N2 - 2
    nu <- sqrt(variance1) * sqrt(1/N1 + 1/N2)
    m$nu <- nu
  }
  if (!var.equal) {
    dfreedom <- (variance1/N1 + variance2/N2)^2 / ((variance1/N1)^2/(N1-1) +
                                                     (variance2/N2)^2/(N2-1))
    nu <- sqrt(variance1/N1 + variance2/N2)
    m$nu <- nu
  }
  m$df <- dfreedom
  if (two.sided) {
    m$mdd <- qt(alpha/2, df = dfreedom, lower.tail=F) * nu
  }
  if (!two.sided) {
    m$mdd <- qt(alpha, df = dfreedom, lower.tail=F) * nu
  }
  class(m) <- "MDD"
  return(m)
}
```

with *N1* and *N2* being the number of samples in group 1 (e.g., control) and group 2 (e.g., treatment) respectively; *variance1* being the variance of group 1 (if var.equal = F) or the residual variance (if var.equal = T); *variance2* being the variance of group 2 (only if var.equal = F); *alpha* being the chosen alpha-level (default = 0.05); *two.sided* being a logical telling whether the MDD should be calculated for a two-sided (*two.sided* = TRUE, this is the default) or one-sided (*two.sided* = FALSE) test; and *var.equal* being a logical telling whether equal variances (*var.equal* = TRUE, default) or non-equal variances (*var.equal* = FALSE) should be assumed.

3 Can the MDD discriminate between true and false negative tests?

3.1 R Code: Simulation of experiments with zero-effect vs. effects of size 0.5

The following R code was used to simulate and evaluate the ability of pCI, pMDE and pMDD to distinguish between false negatives (type II errors) and real zero-effects.

First, the experimental conditions and real world parameters were defined:

name	defined as	range/ calculation
effect	no = no real effect, yes = real effect exists	half of experiments no, half of experiments yes
effectsize	real effect size	half of experiments 0 and half of experiments 0.5 (or 0.2, or 0.8)
theta	parameter related to variance	1 to 50 (drawn randomly from uniform distribution)
sampleSize	sample size	3 to 10 (drawn randomly with replacement)
abund	real species abundance (= control mean)	10 to 100 (drawn randomly with replacement, only integers)

name	defined as	range/ calculation
variance	real population variance	= theta * abundance (the same for control and treatment)

```
# Set number of experiments to be simulated:
size = 2000

# Create dataframe and specify real world and experimental conditions:
experiments = data.frame(
  effect = c(rep("no",size/2),rep("yes",size/2)),
  effectsize = c(rep(0,size/2),rep(0.5,size/2)),
  theta = runif(size,max = 50,min=1),
  sampleSize = sample(3:10,size, replace = T),
  abund = sample(10:100, size, replace=T)
)
experiments$variance = experiments$theta * experiments$abund
```

Second, the parameters and values to be included in the output were defined:

name	defined as	purpose of calculation
p	p value	used as primary effect size filter
pMDD	proportional MDD	evaluation of false trust/ false mistrust rates
pMDE	proportional MDE with 80% power	evaluation of false trust/ false mistrust rates
pCI	proportional upper bound of CI with 95% confidence level	evaluation of false trust/ false mistrust rates

```
# Create empty output dataframe:
out <- data.frame(
  p = rep(NA,size),
  pMDD = rep(NA,size),
  pMDE = rep(NA,size),
  pCI = rep(NA,size)
)
```

Third, experiments were simulated by drawing random samples from a negative binomial distribution creating overdispersed count data based on experimental and real-world parameters as defined above. Each simulated experiment was subsequently analyzed and the results were stored in the output dataframe.

```
# Simulate experiments, analyze experimental data and store results into output dataframe:
for (i in 1:size)
{
  N <- experiments$sampleSize[i]
  theta <- experiments$theta[i]
  abund <- experiments$abund[i]
  effectsize <- experiments$effectsize[i]
  #simulate experiments:
  control <- rnbinom(n = N, mu = abund, size = abund/(theta-1))
  tmean <- abund * (1 - effectsize)
  treatment <- rnbinom(n = N, mu = tmean, size = tmean/(theta-1))
  #log-transform data prior to analysis:
```

```
  transc <- log((2*control)+1)
  transt <- log((2*treatment)+1)
  #tests and calculations on transformed data:
  test <- t.test(transc, transt, alternative = "greater", var.equal = T)
  mdd.ln <- MDD(N1 = N, N2 = N,
                variance1 = var(c(transc-mean(transc),transt-mean(transt))),
                alpha=0.05, two.sided = F, var.equal = T)
  postpower <- power.t.test(n = N, delta = NULL,
                            sd = sd(c(transc-mean(transc),transt-mean(transt))),
                             sig.level = 0.05, alternative="one.sided",
                            type = "two.sample", power = 0.8)
  upperCI <- mean(transc)-mean(transt) +
    qt(0.05/1, df = 2*N-2, lower.tail = FALSE) * sqrt(var(c(transc-mean(transc),
                                                transt-mean(transt)))) * sqrt(2/N)
  # Backtransformation:
  mdd.abu <- (exp(mean(transc))-exp(mean(transc) - mdd.ln$mdd))/2
  mde.abu <- (exp(mean(transc))-exp(mean(transc) - postpower$delta))/2
  upperCI.abu <- (exp(mean(transc))-exp(mean(transc) - upperCI))/2
  # Relation to control mean:
  pMDD <- 100 * mdd.abu / ((exp(mean(transc))-1)/2)
  pMDE <- 100 * mde.abu / ((exp(mean(transc))-1)/2)
  pCI <- 100 * upperCI.abu / ((exp(mean(transc))-1)/2)
  # Store in output data frame:
  out[i,1] <- test$p.value
  out[i,2] <- pMDD
  out[i,3] <- pMDE
  out[i,4] <- pCI
}
```

3.2 R Code: Figure 5 B - False trust and false mistrust rates of pCI, pMDE and pMDD

The following R code was subsequently used to calculate false trust and false mistrust rates dependent on different threshold levels applied to the secondary filters pCI, pMDE and pMDD:

```
# Combine the dataframe with experimental and real world parameters
# and the output dataframe:
all <- cbind(experiments, out)

# Filter experiments based on p-values from the t test:
nonsig <- all[all$p >= 0.05,]

# Define thresholds to be applied to secondary filters:
thresholds = seq(from = 30, to = 100, by = 10)

# Create empty dataframe to be filled with false trust and false mistrust rates:
ErrorRates <- data.frame(
  threshold = thresholds,
  mdd.ftrust = rep(NA,length(thresholds)),
  mdd.fmist = rep(NA,length(thresholds)),
  post.ftrust = rep(NA,length(thresholds)),
  post.fmist = rep(NA,length(thresholds)),
  CI.ftrust = rep(NA,length(thresholds)),
  CI.fmist = rep(NA,length(thresholds))
```

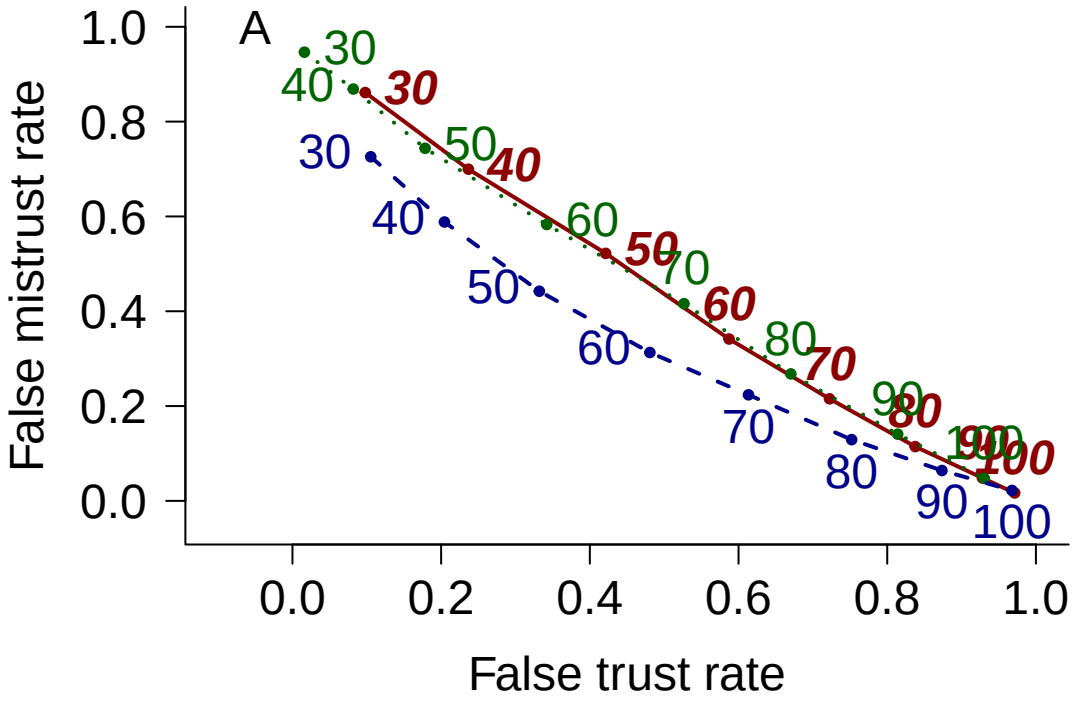
```
)

# Calculate false trust and false mistrust rates for
# pMDD, pMDE and pCI and write results into the ErrorRates dataframe:
for (i in 1:length(thresholds))
{
  mdd.ftrust <- length(which(nonsig$pMDD <= ErrorRates$threshold[i] &
                        nonsig$effect == "yes"))/length(which(nonsig$effect == "yes"))
  mdd.fmist <- length(which(nonsig$pMDD > ErrorRates$threshold[i] &
                        nonsig$effect == "no"))/length(which(nonsig$effect == "no"))
  post.ftrust <- length(which(nonsig$pMDE <= ErrorRates$threshold[i] &
                        nonsig$effect == "yes"))/length(which(nonsig$effect == "yes"))
  post.fmist <- length(which(nonsig$pMDE > ErrorRates$threshold[i] &
                        nonsig$effect == "no"))/length(which(nonsig$effect == "no"))
  CI.ftrust <- length(which(nonsig$pCI <= ErrorRates$threshold[i] &
                        nonsig$effect == "yes"))/length(which(nonsig$effect == "yes"))
  CI.fmist <- length(which(nonsig$pCI > ErrorRates$threshold[i] &
                        nonsig$effect == "no"))/length(which(nonsig$effect == "no"))
  ErrorRates[i,2] <- mdd.ftrust
  ErrorRates[i,3] <- mdd.fmist
  ErrorRates[i,4] <- post.ftrust
  ErrorRates[i,5] <- post.fmist
  ErrorRates[i,6] <- CI.ftrust
  ErrorRates[i,7] <- CI.fmist
}
```

Finally, the pareto-plot (Figure 5 B in the manuscript) was drawn using the following R code:

```
op <- par(mar = c(5,6,1,1), mfrow=c(1,1))

# Plot pMDD error rates:
plot(ErrorRates$mdd.ftrust, ErrorRates$mdd.fmist, xlim = c(-0.1, 1), ylim = c(-0.05,1),
     type = "l", xlab = "False trust rate", ylab = "", bty = "l", las = 1,
     cex.lab = 1.5, cex.axis = 1.5, col = "darkred", lwd = 2)
mtext("False mistrust rate", side = 2, line = 3.5, cex = 1.5)
text(ErrorRates$mdd.ftrust, ErrorRates$mdd.fmist, labels = ErrorRates$threshold,
     cex= 1.5, col = "darkred", font = 4, pos = c(4, 4, 4, rep(3, 5)))
points(ErrorRates$mdd.ftrust, ErrorRates$mdd.fmist, pch = 20, col = "darkred")

# Add pMDE error rates:
lines(ErrorRates$post.ftrust, ErrorRates$post.fmist, col = "darkgreen", lty = 3, lwd = 2)
text(ErrorRates$post.ftrust, ErrorRates$post.fmist, labels=ErrorRates$threshold,
     cex= 1.5, col = "darkgreen", pos = c(4, 2, 4, 4, 3, 3, 3, 3))
points(ErrorRates$post.ftrust, ErrorRates$post.fmist, pch = 20, col = "darkgreen")

# Add pCI error rates
lines(ErrorRates$CI.ftrust, ErrorRates$CI.fmist, col = "darkblue", lty = 2, lwd = 2)
text(ErrorRates$CI.ftrust, ErrorRates$CI.fmist, labels=ErrorRates$threshold,
     cex= 1.5, col = "darkblue", pos = c(rep(2, 4), rep(1, 4)))
points(ErrorRates$CI.ftrust, ErrorRates$CI.fmist, pch = 20, col = "darkblue")
par(op)
```

3.3 Results from simulations with zero-effect vs. effects of size 0.2 and 0.8

Using the same R code as above, we simulated experiments with real effects set to 0.2 vs. zero-effect and 0.8 vs. zero-effect, respectively, which led to the following patterns in false trust and false mistrust rates. Note that the general pattern is similar to simulations using an effect size of 0.5 (pMDEs and pMDDs are very similar and both are less beneficial than pCIs).



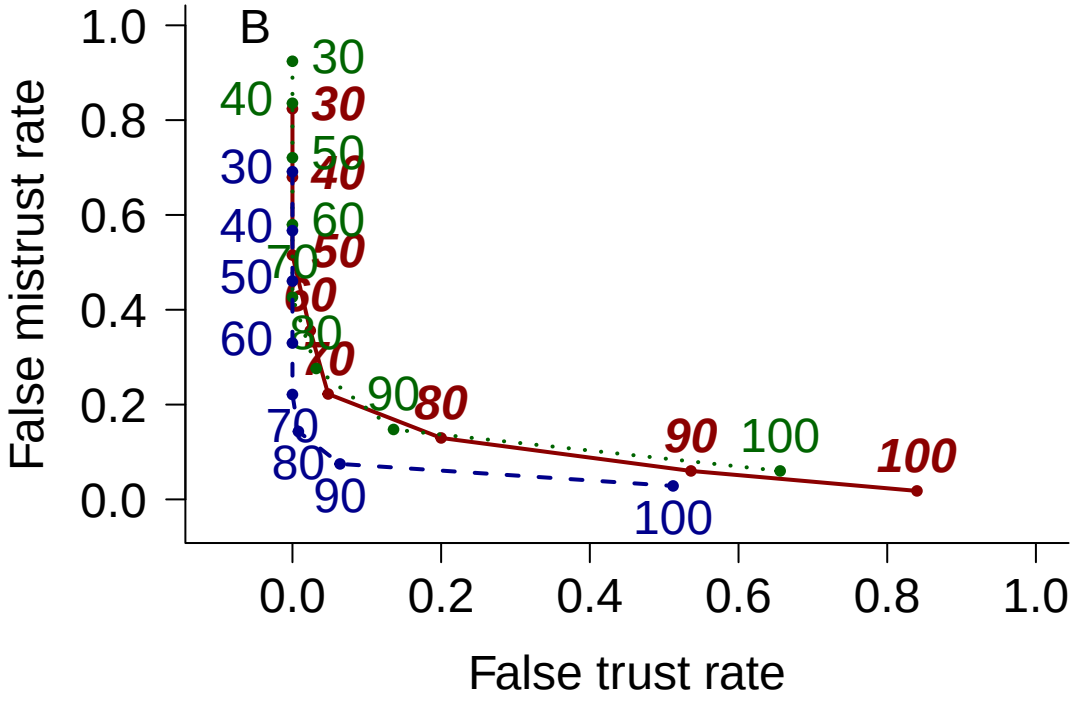


Figure S2: Rates at which pCI (blue, dashed), pMDE (green, dotted) and pMDD (red, solid, threshold levels in bold italics) erroneously suggest to trust (false trust rate) or mistrust (false mistrust rate) non-significant results dependent on varying threshold levels (numbers along lines). Rates derived from 1000 simulated experiments without effect (zero difference between means) and 1000 experiments with (A) small effect (20% loss in mean abundance) and (B) large effect (80% loss in mean abundance). Values towards the bottom left are more beneficial than values towards the top right.

3.4 Results from calculating 75% confidence intervals

Changing the confidence level for the calculation of the upper bound of the CI does not change the pattern in the pareto-plot, except that threshold levels for CIs with different confidence levels have to be re-scaled to achieve the same false trust/ false mistrust rates. This is similar to the shift of the curves for MDEs with different power levels (including the MDD).

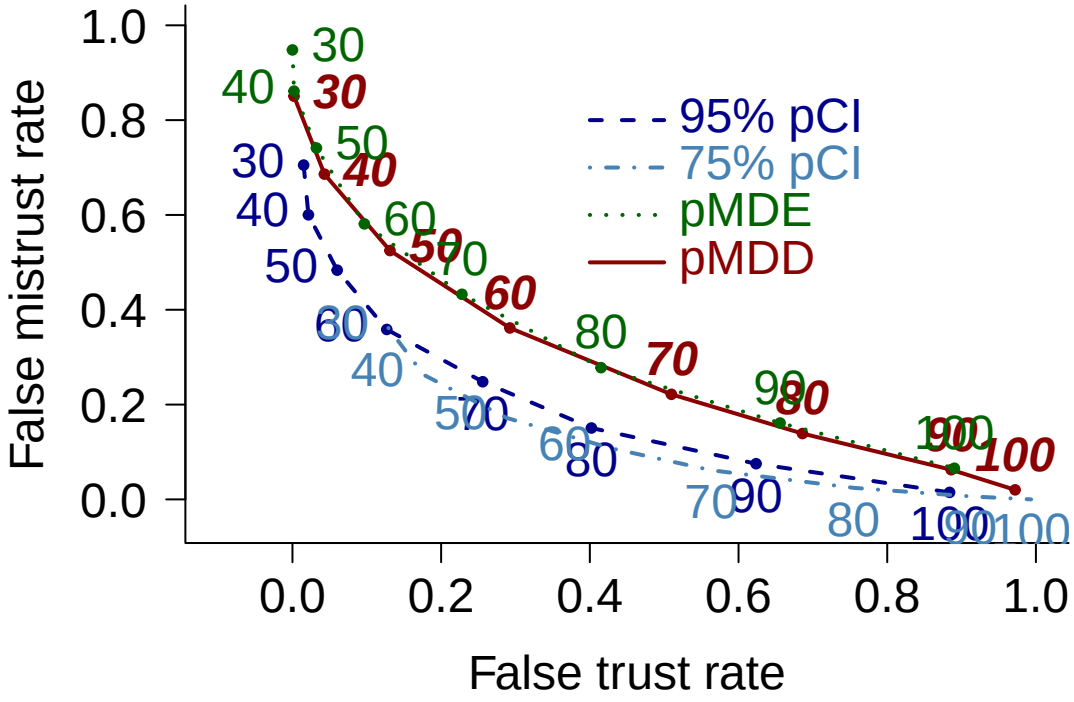


Figure S3: Rates at which pCI with 95% confidence (dark blue, dashed), pCI with 75% confidence (light blue, dash-dotted), pMDE (green, dotted) and pMDD (red, solid, threshold levels in bold italics) erroneously suggest to trust (false trust rate) or mistrust (false mistrust rate) non-significant results dependent on varying threshold levels (numbers along lines). Rates derived from 1000 simulated experiments without effect (zero difference between means) and 1000 experiments with effect (50% reduction in mean abundance). Values towards the bottom left are more beneficial than values towards the top right.

4 Sensitivity of the MDD and other secondary filters to real effect size

4.1 R Code: Simulation of experiments with variable effects ranging from zero to one

The following R code was used to simulate experiments to investigate the ability of the secondary filters pCI, pMDE and pMDD to filter large effects.

First, the experimental conditions and real world parameters were defined:

name	defined as	range/ calculation
effectsize	real effect size	0 to 1 (drawn randomly from uniform distribution)
theta	parameter related to variance	1 to 50 (drawn randomly from uniform distribution)
sampleSize	sample size	2 to 10 (drawn randomly with replacement)
abund	real species abundance (= control mean)	10 to 100 (drawn randomly with replacement)

name	defined as	range/ calculation
variance	real population variance	= theta * abundance (the same for control and treatment)

```
# Set number of experiments to be simulated:
size = 1000

# Create dataframe and specify real world and experimental conditions:
experiments = data.frame(
  effectsize = runif(n = size),
  theta = runif(size, max = 50,min = 1),
  sampleSize = sample(3:10, size = size, replace = T),
  abund = sample.int(91, size, replace=T) + 9
)
experiments$variance = experiments$theta * experiments$abund
```

Second, the parameters and values to be included in the output were defined:

name	defined as	purpose of calculation
p	p value	used as primary filter
pMDD	proportional MDD	secondary filter ability
pMDE	proportional MDE with 80% power	secondary filter ability
pCI	proportional upper bound of 95% CI	secondary filter ability

```
# Create empty output dataframe:
out <- data.frame(
  p = rep(NA,size),
  pMDD = rep(NA,size),
  pMDE = rep(NA,size),
  pCI = rep(NA,size)
)
```

Third, experiments were simulated using experimental and real-world parameters as defined above. Each simulated experiment was subsequently analyzed and the results were stored in the output dataframe.

```
# Simulate experiments, analyse experimental data and store results into output dataframe:
for (i in 1:size)
{
  N <- experiments$sampleSize[i]
  theta <- experiments$theta[i]
  abund <- experiments$abund[i]
  effectsize <- experiments$effectsize[i]
  # Simulate experiment:
  control <- rnbinom(n = N, mu = abund, size = abund/(theta-1))
  tmean <- abund * (1 - effectsize)
  treatment <- rnbinom(n = N, mu = tmean, size = tmean/(theta-1))
  # log-transform data prior to analysis:
  transc <- log((2*control)+1)
  transt <- log((2*treatment)+1)
  # t test, MDD, post-hoc power analysis and calculation of CI on transformed data:
  test <- t.test(transc, transt, alternative = "greater",var.equal = T)
  mdd.ln <- MDD(N1 = N, N2 = N, variance1 = (var(transc)+var(transt))/2,
```

```
                alpha=0.05, two.sided = F, var.equal = T)
  postpower <- power.t.test(n = N, delta = NULL,
                            sd = sd(c(transc-mean(transc),transt-mean(transt))),
                            sig.level = 0.05, alternative = "one.sided",
                            type = "two.sample", power = 0.8)
  upperCI <- mean(transc)-mean(transt) +
    qt(0.05/1, df = 2*N-2, lower.tail = FALSE) *
    sqrt(var(c(transc-mean(transc),transt-mean(transt)))) * sqrt(2/N)
  # Backtransformation of MDD, MDE and upper bound of the CI:
  mdd.abu <- (exp(mean(transc))-exp(mean(transc) - mdd.ln$mdd))/2
  mde.abu <- (exp(mean(transc))-exp(mean(transc) - postpower$delta))/2
  upperCI.abu <- (exp(mean(transc))-exp(mean(transc) - upperCI))/2
  # Relation to back-transformed control mean:
  pMDD <- 100 * mdd.abu / ((exp(mean(transc))-1)/2)
  pMDE <- 100 * mde.abu / ((exp(mean(transc))-1)/2)
  pCI <- 100 * upperCI.abu / ((exp(mean(transc))-1)/2)
  # Store results in output dataframe:
  out[i,1] <- test$p.value
  out[i,2] <- pMDD
  out[i,3] <- pMDE
  out[i,4] <- pCI
}
```

4.2 R Code: Figure 6 - Correlation of pCI, pMDE and pMDD with real effect sizes

The following R code was used to plot correlations of the pCI, pMDE and pMDD with real effect size.

```
# Combine experimental parameter dataset with output
# and remove experiments with significant results:
all <- cbind(experiments, out)
nonsig <- all[all$p >= 0.05,]

## set color palette, ylim and graphical parameters:
palette(c("grey60", "black", "red"))

ylim = max(max(all$pCI), max(all$pMDD), max(all$pMDE)) + 5

op <- par(mfrow=c(2, 3), oma = c(1, 1, 3, 1), mar = c(4, 5, 1, 1), pch = 1, bty = "l",
          cex.lab = 1.5, las = 1)

# Plot results
# (A-C) All experiments:
dotcol <- as.numeric(all$effectsize * 100 <= all$pCI)
plot(all$pCI ~ all$effectsize, ylim = c(0, ylim),
     xlab = "", col = dotcol + 1, ylab = "pCI", pch = dotcol,
     xaxt = "n", yaxt = "n", cex = 0.8)
axis(2, at = c(0, 50, 100, 150, 200))
axis(1, at = c(0, 0.5, 1))
abline(0, 100, col = "grey40", lwd = 2, lty = 5)
fit1 <- lm(all$pCI ~ all$effectsize)
abline(fit1, col = "darkred", lwd = 2, lty = "solid")
text(0.1, ylim, "A", cex = 1.5)
```

```
dotcol <- as.numeric(all$effectsize * 100 <= all$pMDE)
plot(all$pMDE ~ all$effectsize, ylim = c(0, ylim),
     xlab = "", col = dotcol + 1, ylab = "pMDE", pch = dotcol,
     xaxt = "n", yaxt = "n", cex = 0.8)
axis(2, at = c(0, 50, 100, 150, 200))
axis(1, at = c(0, 0.5, 1))
abline(0, 100, col = "grey40", lwd = 2, lty = 5)
fit2 <- lm(all$pMDE ~ all$effectsize)
abline(fit2, col = "darkred", lwd = 2, lty = "solid")
text(0.1, ylim, "B", cex = 1.5)

dotcol <- as.numeric(all$effectsize * 100 <= all$pMDD)
plot(all$pMDD ~ all$effectsize, ylim = c(0, ylim),
     xlab = "", col = dotcol + 1, ylab = "pMDD", pch = dotcol,
     xaxt = "n", yaxt = "n", cex = 0.8)
axis(2, at = c(0, 50, 100, 150, 200))
axis(1, at = c(0, 0.5, 1))
abline(0, 100, col = "grey40", lwd = 2, lty = 5)
fit3 <- lm(all$pMDD ~ all$effectsize)
abline(fit3, col = "darkred", lwd = 2, lty = "solid")
text(0.1, ylim, "C", cex = 1.5)

# (D-F) Only non-significant experiments:
dotcol <- as.numeric(nonsig$effectsize * 100 <= nonsig$pCI)
plot(nonsig$pCI ~ nonsig$effectsize, ylim = c(0, ylim), xlim = c(0, 1),
     xlab = "", col = dotcol + 1, ylab = "pCI", pch = dotcol,
     xaxt = "n", yaxt = "n", cex = 0.8)
axis(2, at = c(0, 50, 100, 150, 200))
axis(1, at = c(0, 0.5, 1))
abline(0, 100, col = "grey40", lwd = 2, lty = 2)
fit1 <- lm(nonsig$pCI ~ nonsig$effectsize)
abline(fit1, col = "darkred", lwd = 2, lty = "solid")
text(0.1, ylim, "D", cex = 1.5)

dotcol <- as.numeric(nonsig$effectsize * 100 <= nonsig$pMDE)
plot(nonsig$pMDE ~ nonsig$effectsize, ylim = c(0, ylim),  xlim = c(0, 1),
     xlab = "Real effect size", col = dotcol + 1, ylab = "pMDE", pch = dotcol,
     xaxt = "n", yaxt = "n", cex = 0.8)
axis(2, at = c(0, 50, 100, 150, 200))
axis(1, at = c(0, 0.5, 1))
abline(0, 100, col = "grey40", lwd = 2, lty = 2)
fit2 <- lm(nonsig$pMDE ~ nonsig$effectsize)
abline(fit2, col = "darkred", lwd = 2, lty = "solid")
text(0.1, ylim, "E", cex = 1.5)

dotcol <- as.numeric(nonsig$effectsize * 100 <= nonsig$pMDD)
plot(nonsig$pMDD ~ nonsig$effectsize, ylim = c(0, ylim),  xlim = c(0, 1),
     xlab = "", col = dotcol + 1, ylab = "pMDD", pch = dotcol,
     xaxt = "n", yaxt = "n", cex = 0.8)
axis(2, at = c(0, 50, 100, 150, 200))
axis(1, at = c(0, 0.5, 1))
abline(0, 100, col = "grey40", lwd = 2, lty = 2)
fit3 <- lm(nonsig$pMDD ~ nonsig$effectsize)
```

```r
abline(fit3, col = "darkred", lwd = 2, lty = "solid")
text(0.1, ylim, "F", cex = 1.5)

par(op)
```

4.3 R Code: Figure 7 - Real effect sizes passing secondary effect size filters

The following R code was used to plot the distribution of real effects that passed both primary (p-value) and secondary (pCI, pMDE and pMDD) filters.

```r
# Combine experimental parameter dataset with output
# and remove experiments with significant results:
all <- cbind(experiments, out)
nonsig <- all[all$p >= 0.05,]

# Create empty dataframe for filtering output
outeffect <- data.frame(
  indicator = NA,
  threshold = NA,
  effectsize = NA
)

# Create list with indicator names to be used in the filtering process:
ind <- list("pCI", "pMDE", "pMDD")

# Apply different filter thresholds to the three indicators
# and store real effect sizes passing these thresholds in the output dataframe:
for (i in ind){
  for (j in 30:100){
    sub <- nonsig[nonsig[, i] <= j, ]
    temp <- data.frame(
      indicator = rep(i, nrow(sub)),
      threshold = rep(j, nrow(sub)),
      effectsize = sub$effectsize
    )
    outeffect <- rbind(outeffect, temp)
  }
}
# Remove 1st row with NA values:
outeffect <- na.omit(outeffect)

# Subset output dataframe for plotting
pCIpass <- outeffect[outeffect$indicator == "pCI", ]
pMDEpass <- outeffect[outeffect$indicator == "pMDE", ]
pMDDpass <- outeffect[outeffect$indicator == "pMDD", ]

# Calculate summary statistics for each indicator and threshold level:
library(dplyr)
summ.pCI <- pCIpass %>%
  group_by(threshold) %>%
  summarise(min = min(effectsize), q25 = quantile(effectsize, 0.25),
            median = median(effectsize), q75 = quantile(effectsize, 0.75),
            max = max(effectsize), N = n(),
            mean = mean(effectsize), sd = sd(effectsize))
```

```r
summ.pMDE <- pMDEpass %>%
  group_by(threshold) %>%
  summarise(min = min(effectsize), q25 = quantile(effectsize, 0.25),
            median = median(effectsize), q75 = quantile(effectsize, 0.75),
            max = max(effectsize), N = n(),
            mean = mean(effectsize), sd = sd(effectsize))
summ.pMDD <- pMDDpass %>%
  group_by(threshold) %>%
  summarise(min = min(effectsize), q25 = quantile(effectsize, 0.25),
            median = median(effectsize), q75 = quantile(effectsize, 0.75),
            max = max(effectsize), N = n(),
            mean = mean(effectsize), sd = sd(effectsize))

# Plot results:
layout(matrix(c(1,2,3,4), 1, 4, byrow = TRUE), width = c(0.5,1,1,1))

# plot1: effects sizes passing p-value filter:
op <- par(mar = c(3,0,1,0), oma = c(0.5,8,0,0), cex = 0.8, bty = "n")

boxplot(nonsig$effectsize, bty = "n", col = "grey90", yaxt = "n", ylim = c(0,1.03), pch = 20)
mtext(c("p-value filter","max", "75%", "median", "25%", "min"),
      side = 2, line = -1, las = 2, cex = 0.8,
      at = c(1.03, max(nonsig$effectsize), quantile(nonsig$effectsize, 0.75),
             median(nonsig$effectsize),
             quantile(nonsig$effectsize, 0.25), min(nonsig$effectsize)))
axis(side = 2, las = 2, labels = T, at = seq(0, 1, by =0.1), line = 4, outer = T)
mtext("Real effect size", side = 2, line = 7, outer = T)

# plot2: pCI
op <- par(mar = c(3,0,1,1), cex = 0.8, bty = "n")

plot(summ.pCI$max ~ summ.pCI$threshold, type ="n", ylim = c(0,1.03), xlim = rev(c(30,100)),
     las = 2, xlab = "", ylab = "", las = 1, bty ="l", yaxt = "n", xaxt = "n")
axis(side = 1)
xcor = c(summ.pCI$threshold, rev(summ.pCI$threshold))
ycor = c(smooth.spline(summ.pCI$max ~ summ.pCI$threshold)$y,
         rev(smooth.spline(summ.pCI$min ~ summ.pCI$threshold)$y))
polygon(xcor, ycor, border = NA, col = adjustcolor("slategray1", alpha.f = 0.2))
ycor = c(smooth.spline(summ.pCI$q75 ~ summ.pCI$threshold)$y,
         rev(smooth.spline(summ.pCI$q25 ~ summ.pCI$threshold)$y))
polygon(xcor, ycor, border =NA, col = "slategray1")
lines(smooth.spline(summ.pCI$max ~ summ.pCI$threshold), lty = 3, col = "darkblue")
lines(smooth.spline(summ.pCI$q75 ~ summ.pCI$threshold), lty = 2, col = "darkblue")
lines(smooth.spline(summ.pCI$median ~ summ.pCI$threshold), lty = 1, col = "darkblue")
lines(smooth.spline(summ.pCI$q25 ~ summ.pCI$threshold), lty = 2, col = "darkblue")
lines(smooth.spline(summ.pCI$min ~ summ.pCI$threshold), lty = 3, col = "darkblue")
text(100, 1.03, "pCI filter", col = "darkblue", pos = 4)

# Add horizontal lines for primary filter (effects passing the p-value filter):
abline(h = max(nonsig$effectsize), lty = 3, col = "grey60")
abline(h = min(nonsig$effectsize), lty = 3, col = "grey60")
abline(h = median(nonsig$effectsize), lty = 1, col = "grey60")
abline(h = quantile(nonsig$effectsize, 0.25), lty = 2, col = "grey60")
```

```r
abline(h = quantile(nonsig$effectsize, 0.75), lty = 2, col = "grey60")

# plot3: pMDE
op <- par(mar = c(3,0,1,1))

plot(summ.pMDE$max ~ summ.pMDE$threshold, type ="n", ylim = c(0,1.03),
     xlim = rev(c(30,100)), las = 2, xlab = "", ylab = "",
     las = 1, bty ="l", yaxt = "n", xaxt = "n")
axis(side = 1)
mtext("Threshold level", side = 1, line = 2.5)
#axis(side = 4, las = 2, pos = c(30, 1), labels = F, at = seq(0, 1, by =0.1))
ycor = c(smooth.spline(summ.pMDE$max ~ summ.pMDE$threshold)$y,
         rev(smooth.spline(summ.pMDE$min ~ summ.pMDE$threshold)$y))
polygon(xcor, ycor, border =NA, col = adjustcolor("darkseagreen2", alpha.f = 0.2))
ycor = c(smooth.spline(summ.pMDE$q75 ~ summ.pMDE$threshold)$y,
         rev(smooth.spline(summ.pMDE$q25 ~ summ.pMDE$threshold)$y))
polygon(xcor, ycor, border =NA, col = "darkseagreen2")
lines(smooth.spline(summ.pMDE$max ~ summ.pMDE$threshold), lty = 3, col = "darkgreen")
lines(smooth.spline(summ.pMDE$q75 ~ summ.pMDE$threshold), lty = 2, col = "darkgreen")
lines(smooth.spline(summ.pMDE$median ~ summ.pMDE$threshold), lty = 1, col = "darkgreen")
lines(smooth.spline(summ.pMDE$q25 ~ summ.pMDE$threshold), lty = 2, col = "darkgreen")
lines(smooth.spline(summ.pMDE$min ~ summ.pMDE$threshold), lty = 3, col = "darkgreen")
text(100, 1.03, "pMDE filter", col = "darkgreen", pos = 4)

# Add horizontal lines for primary filter (effects passing the p-value filter):
abline(h = max(nonsig$effectsize), lty = 3, col = "grey60")
abline(h = min(nonsig$effectsize), lty = 3, col = "grey60")
abline(h = median(nonsig$effectsize), lty = 1, col = "grey60")
abline(h = quantile(nonsig$effectsize, 0.25), lty = 2, col = "grey60")
abline(h = quantile(nonsig$effectsize, 0.75), lty = 2, col = "grey60")

# plot4: pMDD
plot(summ.pMDD$max ~ summ.pMDD$threshold, type ="n", ylim = c(0,1.03),
     xlim = rev(c(30,100)), las=2, xlab = "", ylab = "",
     las = 1, bty ="l", yaxt = "n", xaxt = "n")
axis(side = 1)
ycor = c(smooth.spline(summ.pMDD$max ~ summ.pMDD$threshold)$y,
         rev(smooth.spline(summ.pMDD$min ~ summ.pMDD$threshold)$y))
polygon(xcor, ycor, border =NA, col = adjustcolor( "mistyrose2", alpha.f = 0.2))
ycor = c(smooth.spline(summ.pMDD$q75 ~ summ.pMDD$threshold)$y,
         rev(smooth.spline(summ.pMDD$q25 ~ summ.pMDD$threshold)$y))
polygon(xcor, ycor, border =NA, col = "mistyrose2")
lines(smooth.spline(summ.pMDD$max ~ summ.pMDD$threshold), lty = 3, col = "darkred")
lines(smooth.spline(summ.pMDD$q75 ~ summ.pMDD$threshold), lty = 2, col = "darkred")
lines(smooth.spline(summ.pMDD$median ~ summ.pMDD$threshold), lty = 1, col = "darkred")
lines(smooth.spline(summ.pMDD$q25 ~ summ.pMDD$threshold), lty = 2, col = "darkred")
lines(smooth.spline(summ.pMDD$min ~ summ.pMDD$threshold), lty = 3, col = "darkred")
text(100, 1.03, "pMDD filter", col = "darkred", pos = 4)


# Add horizontal lines for primary filter (effects passing the p-value filter):
abline(h = max(nonsig$effectsize), lty = 3, col = "grey60")
abline(h = min(nonsig$effectsize), lty = 3, col = "grey60")
```

```r
abline(h = median(nonsig$effectsize), lty = 1, col = "grey60")
abline(h = quantile(nonsig$effectsize, 0.25), lty = 2, col = "grey60")
abline(h = quantile(nonsig$effectsize, 0.75), lty = 2, col = "grey60")
par(op)
```