**## Supplemental File S11. Monte Carlo simulation of chi-squared power avoiding small-cell-size problems.**

**## Title of article:** Triplication: an important component of the modern scientific method.

**## Authors:** Jeremy S.C. Clark, Karina Szczypiór-Piasecka, Kamila Rydzewska, Konrad Podsiadło.

**## Analysis of this code snippet was assisted by ChatGPT (GPT-5; OpenAI, San Francisco, USA) [accessed 10 September 2025].** Unfortunately, R package functions designed to perform these simulations (avoiding problems from small-cell sizes) were not found for tables greater than 2x2. The code below produces explicit validation data (confidence intervals for the power) which validated the results for the data provided in the present article but, even so, if applied to other studies results should be treated with caution. For example, the size of the parameter "window" was chosen for our data and has not been optimized for general use. The method without simulation is given below. Unlike our previous work on power studies (Clark et al. "Empirical investigations into Kruskal-Wallis power studies utilizing Bernstein fits, simulations and medical study datasets." Scientific Reports 13.1 (2023): 2352) this has not been extensively empirically tested but relies on inherent theoretical advantages.

## Instructions: copy code into R and run.

## References are given in Supplemental\_File\_S7.

**## Power study for Multinomial (uses stats::rmultinom): alpha = 0.05, beta = 80%, null hypothesis = groups all equal, p1 = one cell increase.**

aaa <- Sys.time() ## takes approximately 2 minutes with present settings

**## One-cell increase under uniform null**

make\_p1\_onecell <- function(k, delta) {

p1 <- rep(1/k, k)

p1[1] <- p1[1] + delta

p1[-1] <- p1[-1] - delta/(k-1)

pmax(p1, 0) / sum(p1)

}

**## Calibration of chi-squared power at fixed n**

calibrated\_chisq\_power <- function(n, p0, p1, alpha = 0.05,

B0 = 1e5, B1 = 1e5,

conf = 0.95, seed = NULL) {

if (!is.null(seed)) set.seed(seed)

stat <- function(x, p0) { E <- sum(x) \* p0; sum((x - E)^2 / E) }

**## Calibration of critical value under H0**

x0 <- rmultinom(B0, n, p0)

crit <- unname(quantile(apply(x0, 2, stat, p0 = p0), 1 - alpha))

**## Estimation of power under H1**

x1 <- rmultinom(B1, n, p1)

st1 <- apply(x1, 2, stat, p0 = p0)

phat <- mean(st1 >= crit)

se <- sqrt(phat \* (1 - phat) / B1)

z <- qnorm((1 + conf) / 2)

ci <- c(lower = max(0, phat - z \* se),

upper = min(1, phat + z \* se))

list(power = phat, power\_CI = ci, crit = crit)

}

**## N for target power, with local grid refinement & clamped CI**

n\_for\_power\_calibrated\_with\_gridCI <- function(target\_power, p0, p1, alpha = 0.05, n\_lo = 10, n\_hi = 2000, B0\_search = 1e5, B1\_search = 1e5, B0\_grid = 1e5, B1\_grid = 1e5, conf = 0.95, seed = 1, window = 12, N\_policy = c("point","conservative")) {

N\_policy <- match.arg(N\_policy)

stat <- function(x, p0) { E <- sum(x) \* p0; sum((x - E)^2 / E) }

pwr <- function(n, B0, B1) {

set.seed(seed)

x0 <- rmultinom(B0, n, p0)

crit <- unname(quantile(apply(x0, 2, stat, p0 = p0), 1 - alpha))

x1 <- rmultinom(B1, n, p1)

st1 <- apply(x1, 2, stat, p0 = p0)

ph <- mean(st1 >= crit)

se <- sqrt(ph \* (1 - ph) / B1)

z <- qnorm((1 + conf) / 2)

c(power = ph,

low = max(0, ph - z \* se),

up = min(1, ph + z \* se))

}

**## Strict search (point power, no tolerance)**

lo <- n\_lo; hi <- n\_hi

**## ensure hi attains target**

while (pwr(hi, B0\_search, B1\_search)["power"] < target\_power && hi < 1e6) hi <- hi \* 2

if (pwr(hi, B0\_search, B1\_search)["power"] < target\_power) stop("Target unattainable within bounds.")

while (lo < hi) {

mid <- (lo + hi) %/% 2

if (pwr(mid, B0\_search, B1\_search)["power"] < target\_power) lo <- mid + 1 else hi <- mid

}

N\_point <- lo

**## Local grid**

ns <- max(n\_lo, N\_point - window) : min(n\_hi, N\_point + window)

grid\_mat <- t(sapply(ns, function(n) pwr(n, B0\_grid, B1\_grid)))

grid <- data.frame(n = ns,

power = grid\_mat[, "power"],

pwr\_low = grid\_mat[, "low"],

pwr\_up = grid\_mat[, "up"])

**## Choose final N by policy**

idx\_point <- which(grid$power >= target\_power)

if (!length(idx\_point)) stop("Grid window too narrow to cross target; increase `window`.")

N\_final <- min(grid$n[idx\_point])

if (N\_policy == "conservative") {

idx\_cons <- which(grid$pwr\_low >= target\_power)

if (!length(idx\_cons)) stop("Grid window too narrow to meet conservative target; increase `window`.")

N\_final <- min(grid$n[idx\_cons])

}

**## Invert CI on grid to get an interval**

idx\_low <- which(grid$pwr\_up >= target\_power)

idx\_up <- which(grid$pwr\_low <= target\_power)

N\_low <- if (!length(idx\_low)) NA\_integer\_ else min(grid$n[idx\_low])

N\_up <- if (!length(idx\_up)) NA\_integer\_ else max(grid$n[idx\_up])

**## Safeguards:**

**## 1) If inversion produces reversed pair, collapse to point N**

if (!is.na(N\_low) && !is.na(N\_up) && N\_low > N\_up) {

N\_low <- N\_up <- N\_final

}

**## 2) Clamp both ends to include point estimate (policy: "size by point power")**

if (!is.na(N\_low) && !is.na(N\_final) && N\_low > N\_final) N\_low <- N\_final

if (!is.na(N\_up) && !is.na(N\_final) && N\_up < N\_final) N\_up <- N\_final

list(

N\_required = N\_final,

N\_CI = c(lower = N\_low, upper = N\_up),

power\_at\_N = subset(grid, n == N\_final)$power,

power\_CI\_at\_N= c(lower = subset(grid, n == N\_final)$pwr\_low,

upper = subset(grid, n == N\_final)$pwr\_up),

grid = grid

)

}

**## Run over deltas**

k <- 9

p0 <- rep(1/k, k)

alpha <- 0.05

target <- 0.80

deltas <- c(0.05, 0.10, 0.15, 0.20, 0.25, 0.30)

seed <- 123

set.seed(seed)

out <- do.call(rbind, lapply(deltas, function(d) {

p1 <- make\_p1\_onecell(k, d)

res <- n\_for\_power\_calibrated\_with\_gridCI(

target\_power = target, p0 = p0, p1 = p1,

alpha = alpha, n\_lo = 10, n\_hi = 2000,

B0\_search = 1e5, B1\_search = 1e5,

B0\_grid = 1e5, B1\_grid = 1e5, ## precise grid near N

conf = 0.95, seed = seed,

window = 12, N\_policy = "point"

)

data.frame(

delta = d,

N\_required = res$N\_required,

N\_low95 = res$N\_CI["lower"],

N\_up95 = res$N\_CI["upper"],

power\_at\_N = res$power\_at\_N,

pwr\_low95 = res$power\_CI\_at\_N["lower"],

pwr\_up95 = res$power\_CI\_at\_N["upper"],

row.names = NULL

)

}))

print(out)

bbb <- Sys.time()

print(bbb - aaa) ## approximately 2 minutes

## > print(out)

## delta N\_required N\_low95 N\_up95 power\_at\_N pwr\_low95 pwr\_up95

## 1 0.05 645 637 646 0.80004 0.7975610 0.8025190

## 2 0.10 173 172 173 0.80196 0.7994900 0.8044300

## 3 0.15 81 81 81 0.80318 0.8007157 0.8056443

## 4 0.20 47 47 47 0.80167 0.7991986 0.8041414

## 5 0.25 31 31 31 0.80025 0.7977720 0.8027280

## 6 0.30 22 22 22 0.80155 0.7990781 0.8040219

##

**## Power study for Multinomial using non-simulated chi-squared: alpha = 0.05, beta = 80%, null hypothesis = groups all equal, p1 = one cell bump. (For a better method with Monte Carlo simulation avoiding problems with small-cell sizes - see above.)**

install.packages("rcompanion")

install.packages("pwr")

library("rcompanion")

library("pwr")

k <- 9

df <- k - 1

p0 <- rep(1/9, 9)

p0

deltas <- c(0.05, 0.10, 0.15, 0.20, 0.25, 0.30) ## allowed increase for one-cell bump

p1 <- list(); ww <- list(); myN <- list();

for (i in 1:length(deltas)) {

p1[[i]] <- p0

p1[[i]][1] <- p1[[i]][1] + deltas[[i]]

p1[[i]][-1] <- p1[[i]][-1] - deltas[[i]]/8

p1[[i]] <- p1[[i]]/sum(p1[[i]]) ## gives sum to 1

ww[[i]] <- rcompanion::cohenW(x = p1[[i]], p = p0)

myN[[i]] <- ceiling(pwr::pwr.chisq.test(w = ww[[i]], df = df, sig.level = 0.05, power = 0.80)$N)

}

myN <- as.vector(unlist(myN))

Ndf <- rbind("One-cell increase" = deltas, "N" = myN, "CohenW" = ww)

Ndf

## [,1] [,2] [,3] [,4] [,5] [,6]

## One-cell increase 0.05 0.1 0.15 0.2 0.25 0.3

## N 594 149 66 38 24 17

## CohenW 0.1591 0.3182 0.4773 0.6364 0.7955 0.9546

##

##\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

645/594 = 1.09

173/149 = 1.16

81/66 = 1.23

47/38 = 1.24

31/24 = 1.29

22/17 = 1.29