```r
#' Run simulation for independent mean differences
#'
#' This function calls the primary design function \code{ind.smd.mc()} for
#' running the Monte Carlo simulation into independent standardized mean
#' differences
#'
#' @param stder      standardizer to be used (either \code{"d1"} or \code{"d2"}
#'                   or \code{"d3"})
#' @param d.type     distribution type for generated data (\code{"symmetric"}
#'                   for symmetric skewness around normal or \code{"asymmetric"}
#'                   for one negatively-skewed and three positively-skewed
#'                   distributions around normal)
#' @param d.range    range of scores for delta (\code{"pos"} for non-negative
#'                   range, \code{"neg"}) for non-positive, and \code{"full"}
#'                   for full range of positive and negative values
#' @param scores     type of scores (\code{"fixed"} or \code{"random"})
#' @param level      level of confidence for interval (default = 0.95)
#' @param N.reps     number of replications in each cell of simulation
#' @param r.seed     seed number for random number generator
#' @param inter.n    interim cell numbers to report (e.g., every "inter-n" cells)
#' @param parallel   run simulation in parallel (default = FALSE)
#' @param cl         number of clusters if running in parallel (default = 5)
#'
#' @returns
#' List object containing \code{design} matrix summarising each cell of the
#' simulation design,
#'
#' @import doParallel
#' @import doRNG
#' @import foreach
#'
#' @importFrom stats pnorm
#' @importFrom stats pt
#' @importFrom stats qnorm
#' @importFrom stats rbinom
#' @importFrom stats uniroot
#'

ind.smd.run <- function(stder,
                        d.range  = "full",
                        d.type   = "asymmetric",
                        scores   = "fixed",
                        level    = 0.95,
                        N.reps   = 99,
                        r.seed   = 123456789,
                        inter.n  = 225,
                        parallel = FALSE,
                        cl       = 9) {


  source("ind.smd.mc.R")
  source("min.sec.R")

  # The primary r.seed number that is an argument entered in
  # ind.smd.run() function above
  set.seed(r.seed)

  # This generates the f.seed numbers that are subsequently used as
  # different seeds in each foreach() loop within the ind.smd.mc() function
  # called below
  f.seed <- sample(101:999999, cl)

  if (parallel == TRUE) {
    cl <- parallel::makeCluster(cl)
    doParallel::registerDoParallel(cl)
```

```
  }

  start <- Sys.time()

  cat("\n Started", format(Sys.time(), "%b %d %X %Y"), "\n\n")

  mcs <- ind.smd.mc(stder   = stder,
                    d.type  = d.type,
                    d.range = d.range,
                    scores  = "fixed",
                    level   = level,
                    N.reps  = N.reps,
                    inter.n = inter.n,
                    f.seed  = f.seed)

  end <- Sys.time()

  cat("\n Finish", format(Sys.time(), "%b %d %X %Y"), "\n\n")
  cat("\n Total time to complete...", min.sec(end, start), "\n\n")

  if (parallel == TRUE) {
    parallel::stopCluster(cl)
  }

  return(mcs)

}

reduce.data <- function(d) {
  for (i in (1:length(d))) {
    d[[i]]$data$data.1 <- d[[i]]$data$data.1[, 1:10]
    d[[i]]$data$data.2 <- d[[i]]$data$data.2[, 1:10]

    d[[i]]$data$mu.1  <- d[[i]]$data$mu.1[1:10]
    d[[i]]$data$mu.2  <- d[[i]]$data$mu.2[1:10]

    d[[i]]$data$sig.1 <- d[[i]]$data$sig.1[1:10]
    d[[i]]$data$sig.2 <- d[[i]]$data$sig.2[1:10]

    d[[i]]$data$gamma.1 <- d[[i]]$data$gamma.1
    d[[i]]$data$gamma.2 <- d[[i]]$data$gamma.2

    d[[i]]$results$lsn.ests <- d[[i]]$results$lsn.ests[1:10]
  }
  return(d)
}


#' Main function defining the design factors in a simulation
#'
#' @param stder      standardizer to be used (either \code{"hedges"} or
#'                   \code{"bonett"})
#' @param d.type     distribution type for generated data (\code{"symmetric"}
#'                   for symmetric skewness around normal or \code{"asymmetric"}
#'                   for one negatively-skewed and three positively-skewed
#'                   distributions around normal)
#' @param d.range    range of scores for delta (\code{"pos"} for non-negative
#'                   range, \code{"neg"}) for non-positive, and \code{"full"}
#'                   for full range of positive and negative values
#' @param scores     type of scores (\code{"fixed"} or \code{"random"})
#' @param level      level of confidence for interval (default = 0.95)
#' @param N.reps     number of replications in each cell of simulation
#' @param inter.n    interim cell numbers to report (e.g., every "inter-n" cells)
#' @param f.seed     seed for random number generator in one of the foreach() loops
#'
```

```r
#' @returns
#' List object containing \code{design} matrix summarising each cell of the
#' simulation design
#'
#' @import doParallel
#' @import doRNG
#' @import foreach
#'
#' @importFrom stats pnorm
#' @importFrom stats pt
#' @importFrom stats qnorm
#' @importFrom stats rbinom
#' @importFrom stats uniroot
#'

ind.smd.mc <- function(stder,
                       d.type,
                       d.range,
                       scores  = "fixed",
                       level   = 0.95,
                       N.reps,
                       inter.n = 225,
                       f.seed) {

  if (d.range == "single") {
    delta.i <- c(-1.5, -0.8, -0.5, -0.2, 0,
                 0.2,  0.5,  0.8,  1.5)

  } else if (d.range == "pos") {
    delta.i <- c(0,  0.2,  0.5,  0.8,  1.5)

  } else if (d.range == "neg") {
    delta.i <- c(0, -0.2, -0.5, -0.8, -1.5)

  } else if (d.range == "full") {
    delta.i <- c(-1.5, -0.8, -0.5, -0.2, 0,
                 0.2,  0.5,  0.8,  1.5)
  }

  require(doParallel)
  require(doRNG)
  require(foreach)

  # Start foreach loop on delta ------------------------------------------------

  fe.delta <- foreach(i = icount(length(delta.i))) %dopar% { delta.i[i]

    fe.seed <- f.seed[i]

    doRNG::registerDoRNG(seed = fe.seed)

    start0 <- Sys.time()

    source("ind.smd.funcs.R")

    cat("\n Started", format(Sys.time(), "%b %d %X %Y"), "\n\n")


    # Initialise the full factorial design -------------------------------------
    design.struct  <- ind.initialise.design(stder,
                                            d.type,
                                            d.range)

    n.cells <- design.struct$n.cells
```

```r
if (d.range == "single") {
  design.struct$lens$len1 <- 1
  design.struct$lens$len2 <- 1
  design.struct$lens$len3 <- 1
  design.struct$lens$len4 <- 1
}

  # Initialise the results structures ------------------------------------
  results.struct <- ind.initialise.results(stder,
                                           n.cells,
                                           N.reps,
                                           level)

  # Make record of input arguments ---------------------------------------
  args <- list(delta   = delta.i[i],
               stder   = stder,
               d.type  = d.type,
               d.range = d.range,
               scores  = scores,
               level   = level,
               N.reps  = N.reps,
               seed    = fe.seed,
               RNG     = RNGkind())

  iter.j <- seq(0, n.cells, 225)

  # Start iteration count for each cell in the design --------------------
  j <- 1

  # Group variances (Factor 1) .............................................

  for (i1 in 1:design.struct$lens$len1) {

    mom.2.1 <- design.struct$sigma[[i1]][1]
    mom.2.2 <- design.struct$sigma[[i1]][2]


    # Group sample sizes (Factor 2) ........................................

    for (i2 in 1:design.struct$lens$len2) {

      n.total <- design.struct$n.total[i2]


      # Group size ratios (Factor 3) .....................................

      for (i3 in 1:design.struct$lens$len3) {

        nr.1 <- design.struct$nr.1[i3]
        nr.2 <- design.struct$nr.2[i3]

        # Calculate group sizes
        n.grps <- calc.group.size(n.total  = n.total,
                                  n1.ratio = nr.1,
                                  n2.ratio = nr.2)

        n.1 <- n.grps[1]
        n.2 <- n.grps[2]


        # Initialise data list structure

        data.struct <- ind.initialise.data(n.1     = n.1,
                                           n.2     = n.2,
                                           N.reps  = N.reps,
```

```
                                        n.cells = n.cells,
                                        stder   = stder,
                                        f.seed  = fe.seed)

    # Distribution of groups (Factor 4) ..............................

    for (i4 in 1:design.struct$lens$len4) {

      mom.3.1 <- design.struct$d.sk[i4]
      mom.4.1 <- design.struct$d.kt[i4]

      mom.3.2 <- design.struct$d.sk[i4]
      mom.4.2 <- design.struct$d.kt[i4]


      # Now...create data structure

      if (RNGkind()[1] != "Mersenne-Twister") {
        results.struct$rng.seed[j, 1:3]  <- RNGkind()
        results.struct$rng.seed[j, 4:10] <- .Random.seed
      }

      data.struct <- ind.gen.data.HC(data.struct,
                                     delta.i[i],
                                     mom.2.1, mom.2.2,
                                     mom.3.1, mom.3.2,
                                     mom.4.1, mom.4.2,
                                     j)

      # Calculate summary statistics of replicated data

      results.struct <- ind.calc.sum.data(data.struct, results.struct, j,
                                          i1, i2, i3, i4,
                                          delta.i[i],
                                          n.1, n.2, (n.1 + n.2))



      # Calculate point and interval effect size estimates

      if (stder == "d1") {                  # Cohen's d

        results.struct <- ind.calc.smd.d1.cov(stder,
                                              data.struct,
                                              results.struct,
                                              delta.i[i], j)

      } else if (stder == "d2") {           # Glass' Delta

        results.struct <- ind.calc.smd.d2.cov(stder,
                                              data.struct,
                                              results.struct,
                                              delta.i[i], j)

      } else if (stder == "d3") {           # Bonett's delta

        results.struct <- ind.calc.smd.d3.cov(stder,
                                              data.struct,
                                              results.struct,
                                              delta.i[i], j)
      }

      # Calculate HC3 interval estimates

      results.struct <- ind.calc.smd.H3.cov(stder,
```

```
                                                    data.struct,
                                                    results.struct,
                                                    delta.i[i], j)

            # Display interim time
            if (j-1 == 0) {
              s.0 <- Sys.time()
            } else if (any(j == iter.j)) {
              e.0 <- min.sec(Sys.time(), s.0)
              cat("Cell number",
                  sprintf('%4d', j), "of", n.cells,
                  "cells", "for delta", delta.i[i],
                  "....interim time =", e.0, "\n")
              s.0 <- Sys.time()
            }

            # Iterators for cells of fully-crossed design
            j <- j + 1

          } # Group distribution
        } # Group size ratios
      }  # Group sample sizes
    } # Group variances

    end0  <- Sys.time()
    total <- min.sec(end0, start0)

    return(results = list(design  = design.struct,
                          results = results.struct,
                          data    = data.struct,
                          args    = args,
                          r.time  = list(start = start0,
                                         end   = end0,
                                         total = total)))
  }

  # End foreach loop ------------------------------------------------------------

  names(fe.delta) <- fe.delta[[1]]$design$labs$lab7

  return(fe.delta)

}  # main function name



# Initialise design list used to specify each cell of the simulation-----------
#' @noRd
#'
# - Revision of functions to correction calculation differences with XECI
# Version 0.0.9017
# - Also removes all references to GLS estimator

ind.initialise.design <- function(stder, d.type, d.range) {

  # Factor 1: Group variances

  sigma <- list(c(1, 4), c(1, 1), c(4, 1))

  lab1 <- c("LT", "EQ", "GT")
  len1 <- length(lab1)
  fac1 <- factor(1:len1, labels = lab1)
```

```r
# Factor 2: Total sample size

n.total <- c(40, 100, 200, 500, 1000)

lab2 <- as.character(n.total)
len2 <- length(lab2)
fac2 <- factor(1:len2, labels = lab2)


# Factor 3: Group sample size ratios

nr.1 <- c(1, 1, 3)
nr.2 <- c(3, 1, 1)

lab3 <- c("LT", "Bal", "GT")
len3 <- length(lab3)
fac3 <- factor(1:len3, labels = lab3)

n.grps <- calc.group.size(n.total, nr.1, nr.2)

# Factor 4: Group distributions

if (d.type == "symmetric") {

  d.sk <- c(-2.5, -2, -1.4, -0.7, 0, 0,  0,  0.7, 1.4,  2,  2.5)
  d.kt <- c(19.0, 10,  6.0,  3.4, 2, 3, 10,  3.4, 6.0, 10, 19.0)

  lab4 <- c("n01", "n05", "n20", "n50", "Plt", "Nrm",
            "Lep", "p50", "p20", "p05", "p01")

} else if (d.type == "asymmetric") {
  d.sk <- c(0, 0.5, 1,  2)
  d.kt <- c(3, 6,   8, 15)
  lab4 <- c("Nrm", "Min", "Mod", "Sev")
}

len4 <- length(lab4)
fac4 <- factor(1:len4, labels = lab4)


# Factor 5: Size of delta

if (d.range == "single") {
  delta <- 0.5
} else if (d.range == "pos") {
  delta <- c(0,  0.2,  0.5,  0.8,  1.5)
} else if (d.range == "neg") {
  delta <- c(0, -0.2, -0.5, -0.8, -1.5)
} else if (d.range == "full") {
  delta <- c(-1.5, -0.8, -0.5, -0.2, 0,
              0.2,  0.5,  0.8,  1.5)
}

len5  <- length(delta)
lab5  <- as.character(delta)
fac5  <- factor(1:len5, labels = lab5)

# Now...gather everything together

facs       <- list(fac1, fac2, fac3, fac4, fac5)
names(facs) <- c("fac1", "fac2", "fac3", "fac4", "fac5")

labs       <- list(lab1, lab2, lab3, lab4, lab5)
names(labs) <- c("lab1", "lab2", "lab3", "lab4", "lab5")
```

```r
  lens       <- list(len1, len2, len3, len4, len5)
  names(lens) <- c("len1", "len2", "len3", "len4", "len5")

  n.cells <- prod(as.numeric(lens)[1:4])

  # Exclude delta in fac.design below as it becomes the foreach() statement

  fac.design <- dae::fac.gen(generate = lens[c(1, 2, 3, 4)])

  names(fac.design) <- c("sigma", "n.total", "balance", "distrib")

  fac.design[,1] <- factor(fac.design[, 1], labels = labs$lab1)
  fac.design[,2] <- factor(fac.design[, 2], labels = labs$lab2)
  fac.design[,3] <- factor(fac.design[, 3], labels = labs$lab3)
  fac.design[,4] <- factor(fac.design[, 4], labels = labs$lab4)

  design.list = list(stder      = stder,
                     sigma      = sigma,
                     n.total    = n.total,
                     nr.1       = nr.1,
                     nr.2       = nr.2,
                     n.grps     = n.grps,
                     d.sk       = d.sk,
                     d.kt       = d.kt,
                     delta      = delta,
                     n.cells    = n.cells,
                     facs       = facs,
                     labs       = labs,
                     lens       = lens,
                     fac.design = fac.design)

  return(design.list)
}


# Calculate group sample size based on total.n & n.ratio
calc.group.size <- function(n.total, n1.ratio, n2.ratio) {

  nt.ratio <- n1.ratio + n2.ratio
  l.n      <- length(n.total)
  l.r      <- length(nt.ratio)
  n.grps   <- matrix(NA, (l.n * l.r), 3)

  i <- 1
  for (j in 1:l.n) {
    n <-  n.total[j]
    for (k in 1:l.r) {
      n1 <- n * n1.ratio[k] / nt.ratio[k]
      n2 <- n * n2.ratio[k] / nt.ratio[k]
      n.grps[i,] <- c(n1, n2, n)
      i <- i + 1
    }
  }

  return(as.vector(n.grps))
}


# Initialise results list used to summarise findings for each a cell -----------

ind.initialise.results <- function(stder, n.cells, N.reps, level) {

  # Initialise record of RNG seed numbers and type of RNG if FOREACH()

  if (RNGkind()[1] != "Mersenne-Twister") {
```

```
    type = matrix(NA, n.cells, 3)
    seed = matrix(NA, n.cells, 7)
    rng.seed <- data.frame(type, seed)
    colnames(rng.seed) <- c("name.1", "name.2", "name.3",
                            "seed.1", "seed.2", "seed.3", "seed.4",
                            "seed.5", "seed.6", "seed.7")
  } else {
    rng.seed <- NA
  }


  # Summary CI results data.frames

  if (stder == "d1") {

    lsn.d1 <- data.frame(matrix(0, n.cells, 11))
    hc3.d1 <- data.frame(matrix(0, n.cells, 11))

    mse.d1 <- data.frame(matrix(0, n.cells, 15))

    acov.d1 <- data.frame(matrix(0, n.cells, 7))

    sum.data <- data.frame(matrix(0, n.cells, 17))

    colnames(lsn.d1) <- c("cell.id",
                          "est", "se",
                          "pH0", "pH1", "cap",
                          "lbv", "ubv", "wdt",
                          "xlb", "xub")

    colnames(hc3.d1) <- colnames(lsn.d1)

    colnames(mse.d1) <- c("cell.id", "est",
                          "mse", "bias", "prec",
                          "d1", "cm",
                          "t.p", "df.p", "p.p", "pH0.p",
                          "t.s", "df.s", "p.s", "pH0.s")

    colnames(acov.d1) <- c("cell.id",
                           "avc11.1", "avc21.1", "avc22.1",
                           "avc11.2", "avc21.2", "avc22.2")

    colnames(sum.data) <- c("cell.id",
                            "mean.1", "var.1", "skew.1", "kurt.1",
                            "mean.2", "var.2", "skew.2", "kurt.2",
                            "var", "n.size", "bal", "dist", "delta",
                            "n.1", "n.2", "n.total")

    results.list <- list(lsn.d1   = lsn.d1,
                         hc3.d1   = hc3.d1,
                         mse.d1   = mse.d1,
                         acov.d1  = acov.d1,
                         sum.data = sum.data,
                         rng.seed = rng.seed,
                         lsn.ests = matrix(0, N.reps, 1),
                         level    = level,
                         hc.0.1   = as.list(rep(NA, 10)),
                         hc.0.2   = as.list(rep(NA, 10)),
                         hc.3.1   = as.list(rep(NA, 10)),
                         hc.3.2   = as.list(rep(NA, 10)))

  } else if (stder == "d2") {

    lsn.d2 <- data.frame(matrix(0, n.cells, 11))
    hc3.d2 <- data.frame(matrix(0, n.cells, 11))
```

```
    mse.d2 <- data.frame(matrix(0, n.cells, 15))

    acov.d2 <- data.frame(matrix(0, n.cells, 7))

    sum.data <- data.frame(matrix(0, n.cells, 17))

    colnames(lsn.d2) <- c("cell.id",
                          "est", "se",
                          "pH0", "pH1", "cap",
                          "lbv", "ubv", "wdt",
                          "xlb", "xub")

    colnames(hc3.d2) <- colnames(lsn.d2)

    colnames(mse.d2) <- c("cell.id", "est",
                          "mse", "bias", "prec",
                          "d1", "cm",
                          "t.p", "df.p", "p.p", "pH0.p",
                          "t.s", "df.s", "p.s", "pH0.s")

    colnames(acov.d2) <- c("cell.id",
                           "avc11.1", "avc21.1", "avc22.1",
                           "avc11.2", "avc21.2", "avc22.2")

    colnames(sum.data) <- c("cell.id",
                            "mean.1", "var.1", "skew.1", "kurt.1",
                            "mean.2", "var.2", "skew.2", "kurt.2",
                            "var", "n.size", "bal", "dist", "delta",
                            "n.1", "n.2", "n.total")

    results.list <- list(lsn.d2   = lsn.d2,
                         hc3.d2   = hc3.d2,
                         mse.d2   = mse.d2,
                         acov.d2  = acov.d2,
                         sum.data = sum.data,
                         rng.seed = rng.seed,
                         lsn.ests = matrix(0, N.reps, 1),
                         level    = level,
                         hc.0.1   = as.list(rep(NA, 10)),
                         hc.0.2   = as.list(rep(NA, 10)),
                         hc.3.1   = as.list(rep(NA, 10)),
                         hc.3.2   = as.list(rep(NA, 10)))

} else if (stder == "d3") {

    lsn.d3 <- data.frame(matrix(0, n.cells, 11))
    hc3.d3 <- data.frame(matrix(0, n.cells, 11))

    mse.d3 <- data.frame(matrix(0, n.cells, 15))

    acov.d3 <- data.frame(matrix(0, n.cells, 7))

    sum.data <- data.frame(matrix(0, n.cells, 17))

    colnames(lsn.d3) <- c("cell.id",
                          "est", "se",
                          "pH0", "pH1", "cap",
                          "lbv", "ubv", "wdt",
                          "xlb", "xub")

    colnames(hc3.d3) <- colnames(lsn.d3)

    colnames(mse.d3)  <- c("cell.id", "est",
                           "mse", "bias", "prec",
```

```
                              "d1", "cm",
                              "t.p", "df.p", "p.p", "pH0.p",
                              "t.s", "df.s", "p.s", "pH0.s")

    colnames(acov.d3) <- c("cell.id",
                              "avc11.1", "avc21.1", "avc22.1",
                              "avc11.2", "avc21.2", "avc22.2")

    colnames(sum.data) <- c("cell.id",
                              "mean.1", "var.1", "skew.1", "kurt.1",
                              "mean.2", "var.2", "skew.2", "kurt.2",
                              "var", "n.size", "bal", "dist", "delta",
                              "n.1", "n.2", "n.total")

    results.list <- list(lsn.d3   = lsn.d3,
                         hc3.d3   = hc3.d3,
                         mse.d3   = mse.d3,
                         acov.d3  = acov.d3,
                         sum.data = sum.data,
                         rng.seed = rng.seed,
                         lsn.ests = matrix(0, N.reps, 1),
                         level    = level,
                         hc.0.1   = as.list(rep(NA, 10)),
                         hc.0.2   = as.list(rep(NA, 10)),
                         hc.3.1   = as.list(rep(NA, 10)),
                         hc.3.2   = as.list(rep(NA, 10)))
  }

  return(results.list)
}


# Initialise data list used in a cell of the simulation ------------------------
#' @noRd
#'
ind.initialise.data <- function(n.1, n.2,
                                N.reps,
                                n.cells,
                                stder,
                                scores = "fixed",
                                f.seed) {

  data.list <- list(data.1   = matrix(0, n.1, N.reps),
                    data.2   = matrix(0, n.2, N.reps),
                    mu.1     = matrix(0, N.reps, 1),
                    mu.2     = matrix(0, N.reps, 1),
                    sig.1    = matrix(0, N.reps, 1),
                    sig.2    = matrix(0, N.reps, 1),
                    sk.1     = 0,
                    sk.2     = 0,
                    kt.1     = 0,
                    kt.2     = 0,
                    n.1      = n.1,
                    n.2      = n.2,
                    N.reps   = N.reps,
                    stder    = stder,
                    scores   = scores,
                    n.cells  = n.cells,
                    RNG      = RNGkind(),
                    f.seed   = f.seed)

  return(data.list)
}
```

```r
# Calculate data for each group for an individual cell of design --------------

#' @noRd
ind.gen.data.HC <- function(data.list,
                            pop.smd,
                            pop.v1, pop.v2,
                            pop.s1, pop.s2,
                            pop.k1, pop.k2, j) {

  # Create group sample size lists

  Nreps <- data.list$N.reps

  N1   <- data.list$n.1
  N2   <- data.list$n.2

  Ns.1 <- Nreps * N1[1]
  Ns.2 <- Nreps * N2[1]

  # Calculate population mean for 1st group, depending on type of standardizer

  if (data.list$stder == "d1") {
    pop.m1 <- pop.smd * sqrt( (N1[1] * pop.v1 + N2[1] * pop.v2) /
                                (N1[1] + N2[1]))
    pop.m2 <- 0
  } else if (data.list$stder == "d2") {
    pop.m1 <- pop.smd * sqrt(pop.v1)
    pop.m2 <- 0
  } else if (data.list$stder == "d3") {
    pop.m1 <- pop.smd * sqrt(0.5 * (pop.v1 + pop.v2))
    pop.m2 <- 0
  }

  data.list$pop.diff <- pop.m1

  # Data for 1st group

  data.list$data.1 <- matrix(PearsonDS::rpearson(Ns.1,
                             moments = c(pop.m1, pop.v1, pop.s1, pop.k1)),
                             ncol    = Nreps)

  data.list$mu.1  <- apply(data.list$data.1, 2, FUN = "mean")
  data.list$sig.1 <- apply(data.list$data.1, 2, FUN = "var")
  data.list$sk.1  <- apply(data.list$data.1, 2,
                           FUN = e1071::skewness, type = 2)

  if (N1 <= 100) {
    data.list$kt.1  <- apply(data.list$data.1, 2,
                             FUN = e1071::kurtosis, type = 2) + 3
  } else if (N1 <= 200) {
    data.list$kt.1  <- apply(data.list$data.1, 2,
                             FUN = e1071::kurtosis, type = 1) + 3
  } else {
    data.list$kt.1  <- apply(data.list$data.1, 2,
                             FUN = e1071::kurtosis, type = 3) + 3
  }


  # Data for 2nd group

  data.list$data.2 <- matrix(PearsonDS::rpearson(Ns.2,
                             moments = c(pop.m2, pop.v2, pop.s2, pop.k2)),
                             ncol    = Nreps)

  data.list$mu.2  <- apply(data.list$data.2, 2, FUN = "mean")
```

```r
  data.list$sig.2 <- apply(data.list$data.2, 2, FUN = "var")
  data.list$sk.2  <- apply(data.list$data.2, 2,
                           FUN = e1071::skewness, type = 2)

  if (N2 <= 100) {
    data.list$kt.2  <- apply(data.list$data.2, 2,
                             FUN = e1071::kurtosis, type = 2) + 3
  } else if (N2 <= 200) {
    data.list$kt.2  <- apply(data.list$data.2, 2,
                             FUN = e1071::kurtosis, type = 1) + 3
  } else {
    data.list$kt.2  <- apply(data.list$data.2, 2,
                             FUN = e1071::kurtosis, type = 3) + 3
  }

  # Save summary data for each replication


  return(data.list)
}  # End of ind.gen.data.HC


# Calculate summary statistics of generated data ------------------------------

#' @noRd
ind.calc.sum.data <- function(data.list, results.list, j,
                              sigma, n.tot, bal, dist, delta,
                              n1, n2, nt) {

  results.list$sum.data[j, ] <- c(j,
                                  mean(data.list$mu.1),
                                  mean(data.list$sig.1),
                                  mean(data.list$sk.1),
                                  mean(data.list$kt.1),
                                  mean(data.list$mu.2),
                                  mean(data.list$sig.2),
                                  mean(data.list$sk.2),
                                  mean(data.list$kt.2),
                                  sigma, n.tot, bal, dist, delta,
                                  n1, n2, nt)

  return(results.list)

}  # End of ind.calc.sum.data


# Calculate Hedges-type smd estimates + intervals & coverage -------------------

#' @noRd
ind.calc.smd.d1.cov <- function(stder, data.list, results.list, delta, j) {

  n.1     <- data.list$n.1
  n.2     <- data.list$n.2
  n.tot   <- n.1 + n.2
  n.tilde <- (1 / n.1) + (1 / n.2)

  z.crit <- qnorm((1 + results.list$level)/2, 0, 1)

  df.1 <- n.1 - 1
  df.2 <- n.2 - 1

  df  <- calc.df(NA, NA, n.1, n.2, "d1")
  wgt <- df.1 / df

  cm  <- calc.cm(df)
```

```r
  # cm <- calc.ck(df, data.list$kt.1, data.list$kt.2, n.1, n.2)

  sds <- sqrt((df.1 * data.list$sig.1 + df.2 * data.list$sig.2) / df)

  d1 <- (data.list$mu.1 - data.list$mu.2) / sds
  g1 <- cm * d1

  # Calculate MSE of parameter estimates

  mse.g1 <- MSE(g1, delta)

  # Calculate t test on generated data

  tt.res <- calc.t.test(data.list, stder)

  # SE using Hedges' LSN approximation to noncentral t distribution
  se.g1  <- sqrt(n.tilde + (g1^2 / (2 * df)))

  cap.g1 <- capture.CI(g1, se.g1, delta, z.crit)
  cov.g1 <- vapply(cap.g1, FUN = "mean", FUN.VALUE = numeric(1))

  p0.g1 <- 2*pnorm(abs((g1 - delta) / se.g1), lower.tail = FALSE) < .05
  p1.g1 <- 2*pnorm(abs(g1 / se.g1),           lower.tail = FALSE) < .05


  results.list$lsn.d1[j, ] <- c(j,
                                mean(g1),    mean(se.g1),
                                mean(p0.g1), mean(p1.g1),
                                cov.g1)

  results.list$mse.d1[j, 1:7]  <- c(j, mean(g1), mse.g1, mean(d1), mean(cm))
  results.list$mse.d1[j, 8:15] <- tt.res

  # Save biased & unbiased smd estimates for use in HC3 interval estimation

  results.list$lsn.ests <- g1

  # Save summary data for replication

  if (data.list$N.reps < 10) {
    results.list$se.g1 <- se.g1
    results.list$df    <- df
    results.list$cm    <- cm
  }

  return(results.list)
}  # End of calc.smd.d1.cov


# Calculate Glass-type smd estimates + intervals & coverage --------------------

#' @noRd
ind.calc.smd.d2.cov <- function(stder, data.list, results.list, delta, j) {

  n.1    <- data.list$n.1
  n.2    <- data.list$n.2
  n.tilde <- (1 / n.1) + (1 / n.2)

  z.crit <- qnorm((1 + results.list$level)/2, 0, 1)

  df <- calc.df(NA, NA, n.1, n.2, "d2")
  wgt <- 1

  cm  <- calc.cm(df)
  # cm <- calc.ck(df, data.list$kt.1, data.list$kt.2, n.1, n.2)
```

```r
  sds <- sqrt(data.list$sig.1)
  v.1 <- data.list$sig.1
  v.2 <- data.list$sig.2

  d2 <- (data.list$mu.1 - data.list$mu.2) / sds
  g2 <- cm * d2

  # Calculate MSE of parameter estimates

  mse.g2 <- MSE(g2, delta)

  # Calculate t test on generated data

  tt.res <- calc.t.test(data.list, stder)

  # SE using Hedges' LSN approximation to noncentral t distribution
  se.g2  <- sqrt(n.tilde + (g2^2 / (2 * df)))    # From Hedges (1981) NO GOOD

  # tmp1 <- (1 / n.1) + v.2 / (n.2 * v.1)
  # tmp2 <- g2^2 / (2*df)
  # se.g2  <- sqrt(tmp1 + tmp2)    # From Delacre et al (2021)


  cap.g2 <- capture.CI(g2, se.g2, delta, z.crit)
  cov.g2 <- vapply(cap.g2, FUN = "mean", FUN.VALUE = numeric(1))

  p0.g2 <- 2*pnorm(abs((g2 - delta) / se.g2), lower.tail = FALSE) < .05
  p1.g2 <- 2*pnorm(abs(g2 / se.g2),           lower.tail = FALSE) < .05


  results.list$lsn.d2[j, ] <- c(j,
                                mean(g2),    mean(se.g2),
                                mean(p0.g2), mean(p1.g2),
                                cov.g2)

  results.list$mse.d2[j, 1:7]  <- c(j, mean(g2), mse.g2, mean(d2), mean(cm))
  results.list$mse.d2[j, 8:15] <- tt.res

  # Save biased & unbiased smd estimates for use in HC3 interval estimation

  results.list$lsn.ests <- g2

  # Save summary data for replication

  # results.list <- ind.calc.sum.data(data.list, results.list, j)

  if (data.list$N.reps < 10) {
    results.list$se.g2 <- se.g2
    results.list$df    <- df
    results.list$cm    <- cm
  }

  return(results.list)
}  # End of calc.smd.d2.cov


# Calculate Bonett-type smd estimates, intervals & coverage ---------------------

#' @noRd
ind.calc.smd.d3.cov <- function(stder, data.list, results.list, delta, j) {

  n.1 <- data.list$n.1
  n.2 <- data.list$n.2
```

```r
  z.crit <- qnorm((1 + results.list$level)/2, 0, 1)

  df.1 <- n.1 - 1
  df.2 <- n.2 - 1
  df   <- calc.df(data.list$sig.1, data.list$sig.2, n.1, n.2, "d3")

  wgt <- 0.5

  cm   <- calc.cm(df)
  # cm <- calc.ck(df, data.list$kt.1, data.list$kt.2, n.1, n.2)

  sds <- sqrt(0.5 * (data.list$sig.1 + data.list$sig.2))

  d3 <- (data.list$mu.1 - data.list$mu.2) / sds
  g3 <- cm * d3

  # Calculate MSE of parameter estimates

  mse.g3 <- MSE(g3, delta)

  # Calculate t test on generated data

  tt.res <- calc.t.test(data.list, stder)

  # SE using Bonett's (2008) calculation on g3 (NOT d3)

  tmp2 <- g3^2

  tmp3 <- (data.list$sig.1^2 / df.1) + (data.list$sig.2^2 / df.2)
  tmp4 <-  8 * sds^4
  tmp5 <-  data.list$sig.1 / (sds^2 * df.1)
  tmp6 <-  data.list$sig.2 / (sds^2 * df.2)

  tmp    <- cm^2 * (tmp2 * (tmp3 / tmp4) + tmp5 + tmp6)
  se.g3  <- sqrt(tmp)
  cap.g3 <- capture.CI(g3, se.g3, delta, z.crit)
  cov.g3 <- vapply(cap.g3, FUN = "mean", FUN.VALUE = numeric(1))

  p0.g3 <- 2*pnorm(abs((g3 - delta) / se.g3), lower.tail = FALSE) < .05
  p1.g3 <- 2*pnorm(abs(g3 / se.g3),           lower.tail = FALSE) < .05


  results.list$lsn.d3[j, ] <- c(j,
                                mean(g3),    mean(se.g3),
                                mean(p0.g3), mean(p1.g3),
                                cov.g3)


  results.list$mse.d3[j, 1:7]  <- c(j, mean(g3), mse.g3, mean(d3), mean(cm))
  results.list$mse.d3[j, 8:15] <- tt.res

  # Save smd estimates for use in HC3 interval estimation

  results.list$lsn.ests <- g3

  # Save summary data for replication

  if (data.list$N.reps < 10) {
    results.list$se.g3 <- se.g3
    results.list$df    <- df
    results.list$cm    <- cm
  }

  return(results.list)
}  # End of calc.smd.d3.cov
```

```r
# Calculates HC3 confidence intervals & coverage rates ------------------------

#' @noRd
ind.calc.smd.H3.cov <- function(stder, data.list, results.list, delta, j) {

  # Internal function to calculate individual estimating function scores
  # for independent observations in two groups

  est.funs <- function(data, mu) {
    dev.dat    <- data - mu
    dev.sq     <- dev.dat^2
    dev.sq.mu  <- dev.sq - mean(dev.sq)
    est.dat    <- cbind(dev.dat, dev.sq.mu)
    return(est.dat)
  }
  # End of internal functions ..................................................

  z.crit <- qnorm((1 + results.list$level)/2, 0, 1)

  N.reps <- data.list$N.reps

  n.1 <- data.list$n.1
  n.2 <- data.list$n.2
  n   <- n.1 + n.2

  if (stder == "d1") {
    df <- calc.df(NA, NA, n.1, n.2, "d1")
  } else if (stder == "d2") {
    df <- calc.df(NA, NA, n.1, n.2, "d2")
  } else if (stder == "d3") {
    df <- calc.df(data.list$sig.1, data.list$sig.2, n.1, n.2, "d3")
  }

  cm <- calc.cm(df)

  # Retrieve "g" estimates for unbiased smd estimators, depending on "stder"
  # NB: "g" equals Hedges' / Bonett's / Glass' smd depending on "stder" value

  lsn.ests <- results.list$lsn.ests

  # Initialise SE vectors
  se.g4   <- matrix(NA, 1, N.reps)

  # Calculate partial derivatives for unbiased smd estimators
  derivs.g <- calc.deriv.delta(data.list, cm)

  # inflator for HC3 estimator
  h     <- c(1 / n.1, 1 / n.2)
  lev.3 <- (1 - h)^(-2)

  tmp.21 <- matrix(NA, N.reps, 3)
  tmp.22 <- matrix(NA, N.reps, 3)

  # Calculate HC3 SEs and coverage rates etc
  for (i in 1:N.reps) {

    est.fs.1 <- est.funs(data.list$data.1[, i], data.list$mu.1[i])
    est.fs.2 <- est.funs(data.list$data.2[, i], data.list$mu.2[i])

    hc.0.1 <- crossprod(est.fs.1) / n.1^2
    hc.0.2 <- crossprod(est.fs.2) / n.2^2

    hc.3.1 <- hc.0.1 * lev.3[1]
```

```r
    hc.3.2 <- hc.0.2 * lev.3[2]

    HC.0   <- magic::adiag(hc.0.1, hc.0.2)
    HC.3   <- magic::adiag(hc.3.1, hc.3.2)

    deriv.g <- as.matrix(derivs.g[i,])
    se.g4[i] <- sqrt(t(deriv.g) %*% HC.3 %*% deriv.g)

    if (i <= 10) {
      results.list$hc.0.1[[i]] <- hc.0.1
      results.list$hc.0.2[[i]] <- hc.0.2
      results.list$hc.3.1[[i]] <- hc.3.1
      results.list$hc.3.2[[i]] <- hc.3.2
    }

    tmp.21[i, ] <- c(hc.0.1[1,1], hc.0.1[2,1], hc.0.1[2,2])
    tmp.22[i, ] <- c(hc.0.2[1,1], hc.0.2[2,1], hc.0.2[2,2])

  }

  cap.g4 <- capture.CI(lsn.ests, se.g4, delta, z.crit)
  cov.g4 <- vapply(cap.g4, FUN = "mean", FUN.VALUE = numeric(1))

  p0.g4 <- 2*pnorm(abs((lsn.ests - delta) / se.g4), lower.tail = FALSE) < .05
  p1.g4 <- 2*pnorm(abs(lsn.ests / se.g4),           lower.tail = FALSE) < .05

  if (stder == "d1") {
    results.list$hc3.d1[j, ] <- c(j,
                                  mean(lsn.ests), mean(se.g4),
                                  mean(p0.g4),    mean(p1.g4),
                                  cov.g4)

    results.list$acov.d1[j, ] <- c(j, colMeans(tmp.21), colMeans(tmp.22))

    # print(c(lsn.ests, se.g4, cov.g4[1:2]), digits = 12)

  } else if (stder == "d2") {
    results.list$hc3.d2[j, ] <- c(j,
                                  mean(lsn.ests), mean(se.g4),
                                  mean(p0.g4),    mean(p1.g4),
                                  cov.g4)

    results.list$acov.d2[j, ] <- c(j, colMeans(tmp.21), colMeans(tmp.22))

  } else if (stder == "d3") {
    results.list$hc3.d3[j, ] <- c(j,
                                  mean(lsn.ests), mean(se.g4),
                                  mean(p0.g4),    mean(p1.g4),
                                  cov.g4)

    results.list$acov.d3[j, ] <- c(j, colMeans(tmp.21), colMeans(tmp.22))
  }


  # Save summary data for replication

  if (data.list$N.reps < 10) {
    results.list$se.g4 <- se.g4
    results.list$dx <- derivs.g
    results.list$HC0 <- HC.0
    results.list$HC3 <- HC.3
  }


  return(results.list)
```

```r
}  # End of calc.smd.H3.cov

# Calculates whether CI captures POPULATION DELTA ($cap) plus (a) interval -----
#  width ($wdt) and (b) exceedance of lower & upper bound ($xlb and $xub)

#' @noRd
capture.CI <- function(parm.est, se, pop.effect, z.crit) {

  # lbv <- parm.est - (matrix(z.crit) %*% se)
  # ubv <- parm.est + (matrix(z.crit) %*% se)
  lbv <- parm.est - (z.crit * se)
  ubv <- parm.est + (z.crit * se)

  cap <- ((pop.effect >= lbv) & (pop.effect <= ubv))
  wdt <- (ubv - lbv)

  xlb <- pop.effect < lbv
  xub <- pop.effect > ubv

  return( list(cap = cap, lbv = lbv, ubv = ubv,
               wdt = wdt, xlb = xlb, xub = xub) )
}  # End of capture.CI


# Partial derivatives of generalised standardized mean difference --------------
# NB... omega = sample weight for first group given by (n1 - 1) / N


#' @noRd
calc.deriv.delta <- function(data.list, cm) {

  m.1 <- data.list$mu.1
  v.1 <- data.list$sig.1

  m.2 <- data.list$mu.2
  v.2 <- data.list$sig.2

  N.reps <- data.list$N.reps

  if (data.list$stder == "d1") {
    omega <- (data.list$n.1 - 1) / (data.list$n.1 + data.list$n.2 - 2)

  } else if (data.list$stder == "d2") {
    omega <- 1

  } else if (data.list$stder == "d3") {
    omega <- 0.5
  }

  d <- matrix(NA, N.reps, 4)

  d[, 1] <-  1 / sqrt(omega*v.1 + (1 - omega)*v.2)
  d[, 2] <- -(omega * (m.1 - m.2)) /
    (2 * (omega*v.1 + (1 - omega)*v.2)^(3/2))
  d[, 3] <- -1 / sqrt(omega*v.1 + (1 - omega)*v.2)
  d[, 4] <- -((1 - omega) * (m.1 - m.2)) /
    (2 * (omega*v.1 + (1 - omega)*v.2)^(3/2))

  return(as.numeric(cm) * d)
}  # End of calc.deriv.delta


# Calculating noncentral intervals, dfs and cm values --------------------------

# Calculate the exact CI for the population delta value
```

```r
#' @noRd
ncp.t.ci <- function(t, n.1, n.2, level) {

  p.crit  <- c( (1 + level)/2, (1 - level)/2 )
  df      <- n.1 + n.2 - 2
  n.tilde <- (n.1 * n.2) / (n.1 + n.2)

  ncp.lb <- uniroot(function(ncp) pt(t, df, ncp) - p.crit[1],
                    lower    = t - 3,
                    upper    = t,
                    tol      = 1e-8,
                    extendInt = "yes")

  ncp.ub <- uniroot(function(ncp) pt(t, df, ncp) - p.crit[2],
                    lower    = t,
                    upper    = t + 3,
                    tol      = 1e-8,
                    extendInt = "yes")

  if (ncp.lb$iter >= 1000) {
    ncp.lb$root <- NA
  }

  if (ncp.ub$iter >= 1000) {
    ncp.ub$root <- NA
  }

  return(c(ncp.lb$root / sqrt(n.tilde),
           ncp.ub$root / sqrt(n.tilde)))
}  # End of ncp.t.ci


# Calculate degrees of freedom for equal and unequal variances

#' @noRd
calc.df <- function(v.1, v.2, n.1, n.2, stder) {

  if (stder == "d1") {
    df <- n.1 + n.2 - 2

  } else if (stder == "d2") {
    df <- n.1 - 1

  } else if (stder == "d3") {  # Huynh, 1989, Page 14, f_5
    df <- ((v.1 + v.2)^2) / (v.1^2 / (n.1 - 1) + v.2^2 / (n.2 - 1))

  } else if (stder == "t.test.sep") {

    tmp1 <- (v.1 / n.1 + v.2 / n.2)^2
    tmp2 <- (v.1 / n.1 )^2 / (n.1 - 1)
    tmp3 <- (v.2 / n.2 )^2 / (n.2 - 1)

    df <- tmp1 / (tmp2 + tmp3)

  }

  return(df)
}  # End of calc.df


# Calculates Hedges' correction

#' @noRd
calc.cm <- function(df) {
```

```r
  cm <- exp(lgamma(df/2) - log(sqrt(df/2)) - lgamma((df - 1)/2))

  return(cm)
}


# Calculate correction for single normal distributions

#' @noRd
calc.cn <- function(n) {

  cn <- exp(lgamma(n/2) - log(sqrt((n - 1)/2)) - lgamma((n - 1)/2))

  return(1/cn)
}

# Calculate correction for non-normal distributions based on Giles (2022)

#' @noRd
calc.ck <- function(df, kt) {

  # w1 <- n1/(n1+n2)
  # w2 <- n2/(n1+n2)
  #
  # wkt <- (kt1 + kt2) / 2

  ck <- (8 * df * (df - 1)) / (8 * df * (df - 1) - (df - 1) * (kt - 3) - 2*df)

  return(1 / ck)
}


# Serlin range null hypothesis for robustness inference ------------------------

#
# Calculates confidence interval for robustness of Monte Carlo result using
# a range null hypothesis test proposed by Serlin (1999).
#
#   INPUT:
#       lower.bnd = null hypothesised lower bound of alpha for robustness (e.g., 0.025)
#       upper.bnd = null hypothesised upper bound of alpha for robustness (e.g., 0.075)
#       n.size    = sample size in Monte Carlo study
#       omega     = alpha value for defining range null hypothesis (default = 0.05)
#
#   OUTPUT:
#       tau       = critical interval for Type 1 error rate
#       ci        = critical interval for level of confidence
#       power     = estimate of power of range null hypothesis test, given
#                   sample size and specified omega value
#       tau.list  = LIST object of output from NLEQSLV function
#
#   Based on Serlin (2000) Psychological Methods, 5, 230--240
#

#' @noRd
robust.serlin <- function(lower.bnd, upper.bnd,
                          n.size, omega  = .05) {

  # Normal approximation to binomial distribution
  bin2norm <- function(value, n.size, propPop) {

    x1    <- (value * n.size)
    mu    <- n.size * propPop
    sigma <- sqrt( n.size * propPop * (1-propPop) )
```

```r
    p       <- pnorm(x1, mu, sigma)

    return(p)
  }


  # Solve system of two equations in MYFUN to estimate critical tau values

  myfun <- function(x) {

    F    <-  numeric(2)
    F[1] <- (bin2norm(x[1], n.size, lower.bnd) -
             bin2norm(x[2], n.size, lower.bnd)) - omega
    F[2] <- (bin2norm(x[1], n.size, upper.bnd) -
             bin2norm(x[2], n.size, upper.bnd)) - omega

    return(F)
  }

  # end of internal functions ..................................................

  LOWER <- lower.bnd
  UPPER <- upper.bnd
  NSIZE <- n.size
  ALPHA <- omega


  tau.list <- nleqslv::nleqslv(c(upper.bnd, lower.bnd), myfun)

  # Construct output listing

  if (tau.list$termcd > 2) {
    tau <- c(NA, NA)
    ci  <- c(NA, NA)
    b   <- NA
  } else {
    tau <- tau.list$x[c(2,1)]
    ci  <- 1 - c(tau[2], tau[1])
    b   <- bin2norm(tau[2], n.size, omega) - bin2norm(tau[1], n.size, omega)
  }

  serlin <- list(bounds   = tau,
                 ci       = ci,
                 power    = b,
                 nlsq.out = tau.list)

  return(serlin)

}



# Random group sample sizes ------------------------------------------------------
#  wgt.1   = proportion of population for group 1
#  n.total = total sample size for both groups
#  Nreps   = number of replicated group sizes

#' @noRd
rand.n.sizes <- function(wgt.1, n.total, Nreps, n.seed = 240297) {

  # set.seed(n.seed)

  if (n.total <= 40) {
```

```r
    n.1 <- LaplacesDemon::rtrunc(Nreps, "binom",
                                 a = 2, b = (n.total-2),
                                 size = n.total, prob = wgt.1)
    n.2 <- n.total - n.1

  } else {

    n.1 <- rbinom(Nreps, n.total, wgt.1)
    n.2 <- n.total - n.1
  }

  return(list(n.1 = n.1, n.2 = n.2))
}


# MSE calculations ------------------------------------------------------------

#' @noRd
mse <- function(obs, exp) {
  e <- (obs - exp)^2
  return(mean(e))
}

#' @noRd
bias <- function(obs, exp) {
  b <- mean(obs) - exp
  return(b^2)
}

#' @noRd
prec <- function(obs, exp) {
  v <- (obs - mean(obs))^2
  return(mean(v))
}

#' @noRd
MSE <- function(obs, exp) {

  m <- mse( obs, exp)
  b <- bias(obs, exp)
  p <- prec(obs, exp)

  return(c(m, b, p))

}


# Calculating time intervals --------------------------------------------------

# Adapted from
# https://stackoverflow.com/questions/37506934/calculations-with-minutes-and-seconds-in-r

#' @noRd
min.sec <- function(fim, ini) {
  dif <- as.numeric(difftime(fim, ini, units='sec'))

  if (round(dif) <= 1) {
    dur <- paste0(sprintf('%2.0f', dif), " sec")
  } else if (round(dif) <= 60) {
    dur <- paste0(sprintf('%2.0f', dif), " secs")
  } else if (round(dif) > 3600) {
    dif <- as.numeric(difftime(fim, ini, units='hours'))
    if (round(dif < 2)) {
      dur <- paste0(sprintf('%2d', as.integer(dif)), " hour ",
                    sprintf('%2.0f', (dif - as.integer(dif))*60), " mins")
```

```
    } else {
      dur <- paste0(sprintf('%2d', as.integer(dif)), " hours ",
                    sprintf('%2.0f', (dif - as.integer(dif))*60), " mins")
    }
  } else {
    dif <- as.numeric(difftime(fim, ini, units='mins'))
    if (dif < 2) {
      dur <- paste0(sprintf('%2d', as.integer(dif)), " min ",
                    sprintf('%2.0f', (dif - as.integer(dif))*60), " secs")
    } else {
      dur <- paste0(sprintf('%2d', as.integer(dif)), " mins ",
                    sprintf('%2.0f', (dif - as.integer(dif))*60), " secs")
    }
  }
  return(dur)
}


# Clean up data.struct in each list element to remove excess data

reduce.data <- function(d) {

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

    d[[i]]$data$data.1 <- d[[i]]$data$data.1[, 1:10]
    d[[i]]$data$data.2 <- d[[i]]$data$data.2[, 1:10]

    d[[i]]$data$mu.1   <- d[[i]]$data$mu.1[1:10]
    d[[i]]$data$mu.2   <- d[[i]]$data$mu.2[1:10]

    d[[i]]$data$mu.1   <- d[[i]]$data$sig.1[1:10]
    d[[i]]$data$mu.2   <- d[[i]]$data$sig.2[1:10]

    d[[i]]$data$gamma.1 <- NA
    d[[i]]$data$gamma.2 <- NA

    d[[i]]$results$lsn.ests <- d[[i]]$results$lsn.ests[1:10]

  }

  return(d)
}

# For use in delta method using inverse of information matrix

calc.inv.exp.info <- function(data.list) {

  e <- matrix(NA, data.list$N.reps, 4)

  n.1 <- data.list$n.1
  n.2 <- data.list$n.2

  v.1 <- data.list$sig.1
  v.2 <- data.list$sig.2

  e[, 1] <- v.1 / n.1
  e[, 2] <- 2 * v.1^2 / n.1
  e[, 3] <- v.2 / n.2
  e[, 4] <- 2 * v.2^2 / n.2

  return(e)
}


calc.t.test <- function(data.list, stder) {
```

```r
  pop.diff <- data.list$pop.diff

  n.1 <- data.list$n.1
  n.2 <- data.list$n.2
  n.t <- (1/n.1 + 1/n.2)

  m.1 <- data.list$mu.1
  m.2 <- data.list$mu.2

  v.1 <- data.list$sig.1
  v.2 <- data.list$sig.2

  df.p <- n.1 + n.2 - 2
  df.s <- calc.df(v.1, v.2, n.1, n.2, "t.test.sep")

  vp <- ((n.1 - 1)*v.1 + (n.2 - 1)*v.2) / (df.p)
  vs <- (v.1 / n.1 + v.2 / n.2)

  t.p <- ((m.1 - m.2) - pop.diff) / sqrt(vp * n.t)
  t.s <- ((m.1 - m.2) - pop.diff) / sqrt(vs)

  p.p <- 2 * pt(abs(t.p), df.p, lower.tail = FALSE)
  p.s <- 2 * pt(abs(t.s), df.s, lower.tail = FALSE)

  if (stder == "d1") {
    pH0.p  <- mean(p.p < .05)
    pH0.s  <- mean(p.s < .05)
  } else if (stder == "d2") {
    pH0.p  <- mean(p.p < .01)
    pH0.s  <- mean(p.s < .01)
  } else if (stder == "d3") {
    pH0.p  <- mean(p.p < .001)
    pH0.s  <- mean(p.s < .001)
  }

  t.p  <- mean(t.p)
  df.p <- mean(df.p)
  p.p  <- mean(p.p)

  t.s  <- mean(t.s)
  df.s <- mean(df.s)
  p.s  <- mean(p.s)

  return(c(t.p, df.p, p.p, pH0.p,
           t.s, df.s, p.s, pH0.s))

}
```