## Supporting Information

# Censoring trace-level environmental data: statistical analysis considerations to limit bias

Barbara Jane George[1*], Leslie Gains-Germain[2], Kristin Broms[2], Kelly Black[2], Marschall Furman[3], Michael D. Hays[4], Kent W. Thomas[1], Jane Ellen Simmons[1]

[1] Center for Public Health and Environmental Assessment, Office of Research and Development, U.S. EPA, Research Triangle Park, North Carolina 27711, United States

[2] Neptune and Company, Inc., Lakewood, Colorado 80215, United States

[3] Oak Ridge Institute for Science and Education (ORISE) Research Participant at U.S. EPA, Office of Research and Development, Center for Public Health and Environmental Assessment, Research Triangle Park, North Carolina 27711, United States

[4] Center for Environmental Measurement and Modeling, Office of Research and Development, U.S. EPA, Research Triangle Park, North Carolina 27711, United States

*Address correspondence to B.J. George, CPHEA/ORD/U.S. EPA, 109 T.W. Alexander Dr., Research Triangle Park, NC 27711 USA. Telephone: (919) 541-4551. E-mail: george.bj@epa.gov

## Table of Contents

```
### Statistical approaches for non-detects
### Authors: Neptune and Company, Inc.
### June 2020
### Simulation study
### This code does the following
### 1) simulates censored datasets and
### 2) for each simulated dataset, estimates statistical parameters using 11 methods
### 3) saves results to the 'all_results' list of arrays
### 4) organizes the results into long format in the 'out' dataframe
### The number of simulations is specified on line 216
### Version dependencies:
###    R version 3.6.3
###    fitdistrplus v1.1-1
###    EnvStats 2.3.1
###    dplyr 1.0.1
###    foreach 1.5.0
###    doParallel 1.0.15
###    abind 1.4-5
###    tidyr 1.1.1
###    openxlsx 4.1.5

### Set up libraries ---------------------------------------

## set working directory
# setwd("")
getwd()


library(fitdistrplus)
library(EnvStats)
library(dplyr)

## Try in parallel:
library(foreach)
library(doParallel)
library(abind)

#setup parallel backend to use many processors
cores <- detectCores()


### Functions used ------------------------------

# Lognormal Distribution: mean/SD to meanlog/sdlog (mu/sigma)
#
m2p_lnorm <- function(lmean, lsd){
  sdlog <- sqrt(log(lsd ^ 2 / lmean ^ 2 + 1))
  meanlog <- log(lmean) - sdlog ^ 2 / 2

  list(meanlog = meanlog, sdlog = sdlog, gm = exp(meanlog), gsd = exp(sdlog))
}

# Lognormal distribution: meanlog/sdlog to geometric mean/SD
#
p2m_lnorm <- function(meanlog, sdlog){
  mean <- exp(meanlog + 0.5 * sdlog ^ 2)
  sigma2 <- mean ^ 2 * (exp(sdlog ^ 2) - 1)

  #   # geometric mean and sd
  gm <- exp(meanlog)
  gsd <- exp(sdlog)

  list(mean = mean, sd = sqrt(sigma2), gm = gm, gsd = gsd)
}

# Gamma distribution:  shape, rate to mean, SD
p2m_gamma <- function (shape, scale = 1/rate, rate = 1/scale) {
  list(mean = shape * scale, sd = sqrt(shape * scale ^ 2))
}

## function to calculate GOF stats (via AIC) and
```

```
## to return the resulting distributional parameters for the
## best-fitting distribution
gof <- function (concentrations, fit_cens=FALSE) {
  ## For complete data and sub methods, fit_cens = FALSE
  if (fit_cens == FALSE){
    fitn <- fitdist(concentrations, "norm")

    if ( all(concentrations > 0) ){
      fitln <- fitdist(concentrations, "lnorm")
      fitg <- fitdist(concentrations, "gamma")
      ## Compare fits:
      aic_fits <- c(fitn$aic, fitln$aic, fitg$aic)
      names(aic_fits) <- c("X3.mle.norm", "X2.mle.lnorm", "X1.mle.gamma")
    } else{
      ## Compare fits:
      aic_fits <- fitn$aic
      names(aic_fits) <- c("X3.mle.norm")
    }

    best <- which(aic_fits == min(aic_fits))

  }

  ## For MLE, rROS, and KM, fit_cens = TRUE
  ## For these, make sure that concentrations = a two-column data frame
  if (fit_cens == TRUE) {
    fitn <- fitdistcens(concentrations, distr = "norm")
    fitln <- fitdistcens(concentrations, distr = "lnorm")
    fitg <- fitdistcens(concentrations, distr = "gamma")

    aic_fits <- c(fitn$aic, fitln$aic, fitg$aic)
    names(aic_fits) <- c("X3.mle.norm", "X2.mle.lnorm", "X1.mle.gamma")
    best <- which (aic_fits == min(aic_fits))
  }


  ## also return estimates for best-fitting distribution:
  if (best == 1)
    dist_out <- data.frame(Dist = "Normal",
                           lnorm_mu = NA, lnorm_sigma = NA,
                           gamma_shape = NA, gamma_rate = NA,
                           norm_mu = fitn$estimate[1], norm_sigma = fitn$estimate[2],
                           stringsAsFactors=FALSE)
  if (best == 2)
    dist_out <- data.frame(Dist = "Lognormal",
                           lnorm_mu = fitln$estimate[1], lnorm_sigma = fitln$estimate[2],
                           gamma_shape = NA, gamma_rate = NA,
                           norm_mu = NA, norm_sigma = NA, stringsAsFactors=FALSE)
  if (best == 3)
    dist_out <- data.frame(Dist = "Gamma", lnorm_mu = NA, lnorm_sigma = NA,
                           gamma_shape = fitg$estimate[1],
                           gamma_rate = fitg$estimate[2],
                           norm_mu = NA, norm_sigma = NA, stringsAsFactors=FALSE)

  return(list(aic = aic_fits, dist_out = dist_out))
}

## function to calculate GOF stats (via AIC) BUT
## excluding the gamma function because it can throw an error
gof_no_gamma <- function (concentrations, fit_cens=FALSE) {
  ## For complete data and sub methods, fit_cens = FALSE
  if (fit_cens == FALSE){
    fitn <- fitdist(concentrations, "norm")

    if ( all(concentrations > 0) ){
      fitln <- fitdist(concentrations, "lnorm")
      ## Compare fits:
      aic_fits <- c(fitn$aic, fitln$aic)
      names(aic_fits) <- c("X3.mle.norm", "X2.mle.lnorm")
    } else{
      ## Compare fits:
```

```
    aic_fits <- fitn$aic
    names(aic_fits) <- c("X3.mle.norm")
  }

  best <- which(aic_fits == min(aic_fits))

}

## For MLE, rROS, and KM, fit_cens = TRUE
## For these, make sure that concentrations = a two-column data frame
if (fit_cens == TRUE) {
  fitn <- fitdistcens(concentrations, distr = "norm")
  fitln <- fitdistcens(concentrations, distr = "lnorm")

  aic_fits <- c(fitn$aic, fitln$aic)
  names(aic_fits) <- c("X3.mle.norm", "X2.mle.lnorm")
  best <- which (aic_fits == min(aic_fits))
}


## also return estimates for best-fitting distribution:
if (best == 1)
  dist_out <- data.frame(Dist = "Normal",
                         lnorm_mu = NA, lnorm_sigma = NA,
                         gamma_shape = NA, gamma_rate = NA,
                         norm_mu = fitn$estimate[1], norm_sigma = fitn$estimate[2],
                         stringsAsFactors=FALSE)
if (best == 2)
  dist_out <- data.frame(Dist = "Lognormal",
                         lnorm_mu = fitln$estimate[1], lnorm_sigma = fitln$estimate[2],
                         gamma_shape = NA, gamma_rate = NA,
                         norm_mu = NA, norm_sigma = NA, stringsAsFactors=FALSE)

return(list(aic = aic_fits, dist_out = dist_out))
}


## function to fill-in censored values based on the various
## substitution methods
## returns a list.  Each element of the list is a vector of the
## "results" for the given sub method.
sub_results <- function (vector_NA_at_DL, DL) {
  NDmethods_results <- list()
  nND <- sum(is.na(vector_NA_at_DL))

  ## Calc results for substitution methods
  NDmethods_results$fullDL <- ifelse(is.na(vector_NA_at_DL),
                                     DL, vector_NA_at_DL)
  ## sub at DL /2
  NDmethods_results$halfDL <- ifelse(is.na(vector_NA_at_DL),
                                     0.5 * DL, vector_NA_at_DL)
  ## sub at 0
  NDmethods_results$sub0 <- ifelse(is.na(vector_NA_at_DL),
                                   0, vector_NA_at_DL)

  ## sub at equally spaced intervals:
  tmp_results <- vector_NA_at_DL
  int_values <- NULL
  for (k in 1:nND){
    tmp_value <- DL * k / (k + 1)
    int_values <- c(int_values, tmp_value)
  }
  tmp_results[is.na(vector_NA_at_DL)] <- int_values
  NDmethods_results$equal_int <- tmp_results

  ## sub at equal prob intervals:
  tmp_results <- vector_NA_at_DL
  int_values <- NULL
  for (k in 1:nND){
    tmp_value <- DL * sqrt(k / (k + 1))
    int_values <- c(int_values, tmp_value)
```

```
  }
  tmp_results[is.na(vector_NA_at_DL)] <- int_values
  NDmethods_results$equal_prob <- tmp_results

  ## omit NAs
  NDmethods_results$omit <- c(na.omit(vector_NA_at_DL))

  NDmethods_results
}

### Set up sim study -----------------------------------

nSims <- 1000

sampleSize <- c(20, 50)
censoringLevel <- c(0.3, 0.5, 0.8)
lnorm_mu <- c(1, -2.2)
lnorm_sigma <- c(0.5, 1.6)
dist_names <- c("Mildy skewed", "Highly skewed")

num_combos <- length(sampleSize) * length(censoringLevel) * length(dist_names)

resultNames <- c("Mean", "SD", "Dist",
                 "lnorm_mu", "lnorm_sigma", "gamma_shape", "gamma_rate",
                 "norm_mu", "norm_sigma", "nCensored")

methodNames <- c("Complete", "CompleteMLE", "fullDL", "halfDL", "sub0", "equal_int",
                 "equal_prob", "omit", "MLE", "rROS", "KM")
nMethods <- length(methodNames)
methodNamesPlot <- c("Full Sample", "Full Sample MLE", "CensoringLevel", "CensoringLevel/2",
"Substitution at 0", "Equal Interval Substitution",
                     "Equal Probability Substitution", "Complete Case", "Maximum Likelihood
Estimation (MLE)", "Robust Regression on Order Statistics", "Kaplan-Meier")

distnames <- c("norm", "lnorm", "gamma")


### Simulation study loop ---------------------------------

## Create objects to store output
full_results <- list()  ## stores all_results, all_aic, samples

comboNum <- 1
set.seed(2017)  ## so that results are exactly reproducible
for (dist_type in c(1:2)) {  # loop over mild vs highly skewed
  Dist <- dist_names[dist_type]

  for (n in sampleSize) {  # loop over sample sizes
    for (cens in censoringLevel){  # loop over censoring levels

      cat("\nCombo number: ", comboNum, "\n")
      ## Try in parallel:
      cl <- makeCluster(cores[1] - 2) #not to overload your computer
      registerDoParallel(cl)
      acomb <- function(...) abind(..., along=3)

      ## conduct the nSims for each combo
      full_results[[comboNum]] <-
        foreach(simnum = 1:nSims, #.combine='acomb',
                .multicombine=TRUE,
                .packages=c("bootBCa", "fitdistrplus", "EnvStats", "abind")) %dopar% {
                  set.seed(comboNum*1000 + simnum)
                  ## create tmp data frame to hold simulation results
                  tmp_results <- as.data.frame(matrix(nrow = length(methodNames),
                                                      ncol=length(resultNames)))
                  rownames(tmp_results) <- methodNames
                  colnames(tmp_results) <- resultNames

                  all_aic_tmp <- as.data.frame(matrix(nrow = length(methodNames), ncol =
length(distnames)))
                  rownames(all_aic_tmp) <- methodNames
```

```
                colnames(all_aic_tmp) <- distnames

                ## Create the "data" for the loop
                all_values <- rlnorm(n, lnorm_mu[dist_type], lnorm_sigma[dist_type])
                ## Dl based on cens% of n samples being censored
                DL <- median(sort(all_values)[(cens * n):(cens * n + 1)])

                # results censored at the DL
                results_NA_at_DL <- ifelse(all_values < DL, NA, all_values)
                results_DL_at_DL <- ifelse(all_values < DL, DL, all_values)
                censored <- ifelse(all_values < DL, TRUE, FALSE)

                ## Create a data frame for fitdistcens() functions:
                results_cens <- data.frame(left = ifelse(censored == TRUE, 0, all_values),
                                           right = ifelse(censored == TRUE, DL,
                                                          all_values))


                ## Calc results for "complete" data set:
                tmp_results["Complete", "Mean"] <- mean(all_values)
                tmp_results["Complete", "SD"] <- sd(all_values)

                tmp <- gof(all_values)
                tmp_results["Complete", c("Dist",
                                          "lnorm_mu", "lnorm_sigma", "gamma_shape",
"gamma_rate",
                                          "norm_mu", "norm_sigma")] <- tmp$dist_out
                all_aic_tmp["Complete", ] <- tmp$aic
                names(all_aic_tmp[ , ]) <- names(tmp$aic)

                # Calc MLE for mean and SD of complete data set:
                fullmle_tmp <- elnormAlt(all_values, method="mle")$parameters
                fullmle_mean <- fullmle_tmp[[1]]
                fullmle_sd <- fullmle_tmp[[1]]*fullmle_tmp[[2]]
                tmp_results["CompleteMLE", "Mean"] <- fullmle_mean
                tmp_results["CompleteMLE", "SD"] <- fullmle_sd

                tmp <- gof(all_values)
                tmp_results["CompleteMLE", c("Dist",
                                             "lnorm_mu", "lnorm_sigma", "gamma_shape",
"gamma_rate",
                                             "norm_mu", "norm_sigma")] <- tmp$dist_out
                all_aic_tmp["CompleteMLE", ] <- tmp$aic
                names(all_aic_tmp[ , ]) <- names(tmp$aic)


                ## Calc results for substitution methods
                sub_values <- sub_results(results_NA_at_DL, DL)
                tmp_results[c("fullDL", "halfDL", "sub0","equal_int", "equal_prob", "omit"),
"Mean"] <-
                  sapply(sub_values, mean)
                tmp_results[c("fullDL", "halfDL", "sub0","equal_int", "equal_prob", "omit"),
"SD"] <-
                  sapply(sub_values, sd)

                tmp <- sapply(sub_values, function(x) {
                  tmp2 <- try(gof(x)$dist_out)
                  if (class(tmp2) == "try-error"){
                    print(class(tmp2))
                    tmp2 <- gof_no_gamma(x)$dist_out
                  }
                  tmp2
                })

                ind <- which(names(tmp_results["fullDL",]) %in% names(tmp[,"fullDL"]))
                tmp_results["fullDL", ind] <- tmp[, "fullDL"]
                tmp_results["halfDL", ind] <- tmp[, "halfDL"]
                tmp_results["sub0", ind] <- tmp[, "sub0"]
                tmp_results["equal_int", ind] <- tmp[, "equal_int"]
                tmp_results["equal_prob", ind] <- tmp[, "equal_prob"]
                tmp_results["omit", ind] <- tmp[, "omit"]
```

```
                  tmp_aic <- sapply(sub_values, function(x) try(gof(x)$aic))
                  all_aic_tmp["fullDL", ] <- as.numeric(tmp_aic$fullDL)
                  all_aic_tmp["halfDL", ] <- as.numeric(tmp_aic$halfDL)
                  all_aic_tmp["sub0", ] <- as.numeric(tmp_aic$sub0)
                  all_aic_tmp["equal_int", ] <- as.numeric(tmp_aic$equal_int)
                  all_aic_tmp["equal_prob", ] <- as.numeric(tmp_aic$equal_prob)
                  all_aic_tmp["omit", ] <- as.numeric(tmp_aic$omit)



                  ## Calc values for MLE method
                  # start with gof and distribution parameters:
                  gof_stuff <- gof(results_cens, fit_cens = TRUE)
                  all_aic_tmp["MLE", ] <- gof_stuff$aic
                  tmp_results["MLE", ind] <- gof_stuff$dist_out                    ## Mean and SD
are pop mean, SD for MLE method:
                  ## Will depend on chosen distribution:
                  if (tmp_results["MLE", "Dist"] == "Lognormal"){
                    tmp <- elnormAltCensored(results_DL_at_DL, censored, method =
"mle")$parameters
                    tmp_results["MLE", c("Mean", "SD")] <- c(tmp[[1]], tmp[[2]]*tmp[[1]])

                  }
                  if (tmp_results["MLE", "Dist"] == "Normal"){
                    tmp_results["MLE", c("Mean", "SD")] <-
                      c(tmp_results["MLE", "norm_mu"], tmp_results["MLE", "norm_sigma"])

                  }
                  if (tmp_results["MLE", "Dist"] == "Gamma"){
                    tmp <- egammaAltCensored(results_DL_at_DL, censored, method =
"mle")$parameters
                    tmp_results["MLE", c("Mean", "SD")] <- c(tmp[[1]], tmp[[2]]*tmp[[1]])

                  }

                  ## Calc values for rROS method
                  ## start with aic and best-fitting distribution:
                  all_aic_tmp["rROS", ] <- gof_stuff$aic
                  tmp_results["rROS", "Dist"] <- gof_stuff$dist_out[['Dist']]

                  ## mean, SD, dist params all dependent on chosen distribution:
                  if (tmp_results["rROS", "Dist"] == "Lognormal") {
                    tmp_results["rROS", c("lnorm_mu", "lnorm_sigma")] <-
                      elnormCensored(results_DL_at_DL, censored, method = "rROS")$parameters
                    ## Mean, SD
                    tmp <- elnormAltCensored(results_DL_at_DL, censored, method =
"rROS")$parameters
                    tmp_results["rROS", c("Mean", "SD")] <- c(tmp[[1]], tmp[[1]] * tmp[[2]])
                  }
                  if (tmp_results["rROS", "Dist"] == "Normal") {
                    tmp_results["rROS", c("norm_mu", "norm_sigma")] <-
                      enormCensored(results_DL_at_DL, censored, method = "rROS")$parameters
                    ## Mean, SD
                    tmp_results["rROS", c("Mean", "SD")] <-
                      tmp_results["rROS", c("norm_mu", "norm_sigma")]
                  }

                  ## Calc values for KM method
                  ## start with aic and best-fitting distribution:
                  all_aic_tmp["KM", ] <- gof_stuff$aic

                  ## KM method assumes no distributional form.
                  ## Calc mean, SD
                  tmp_results["KM", "Mean"] <- EnvStats::enparCensored(results_DL_at_DL,
censored)$parameters[[1]]
                  tmp_results["KM", "SD"] <- EnvStats::enparCensored(results_DL_at_DL,
censored)$parameters[[2]]


                  ## save "concentration values" to be able to double-check results and stats.
```

```
                  samples_tmp <- as.matrix(data.frame(all_values = all_values,
                                                      results_NA_at_DL = results_NA_at_DL,
                                                      results_cens = results_cens))
                  ## Also record num of censored values for each sim:
                  tmp_results$nCensored <- sum(is.na(results_NA_at_DL))

                  list(as.matrix(tmp_results), as.matrix(all_aic_tmp), samples_tmp)

                }

      stopCluster(cl)

      comboNum <- comboNum + 1

    }
  }
}

# save results for each method, for each combination of variables
all_aic <- rep(list(array(NA, dim=c(nSims, length(methodNames), length(distnames)),
                          dimnames = list(NULL, methodNames, distnames))), num_combos)
all_results <- rep(list(array(NA,
                              dim=c(nSims, length(methodNames), length(resultNames)),
                              dimnames = list(NULL, methodNames, resultNames))), num_combos)

samples <- rep(list(array(NA, dim=c(nSims, max(sampleSize), 4),
                          dimnames = list(NULL, NULL, c("all_vals", "R_NA_at_DL",
"results_cens_left",
                                                        "results_cens_rhgt")))), num_combos)

for(c in 1:num_combos){
  for(k in 1:nSims){
    all_results[[c]][k,,] <- full_results[[c]][[k]][[1]]
    all_aic[[c]][k,,] <- full_results[[c]][[k]][[2]]
  }
}

# results are in the 'all_results' array

## Make the simresults file for the mean bias and MSE ----------------------------------

gfun <- function(x){
  x %>%
    data.frame(.) %>%
    setNames(methodNames) %>%
    tidyr::gather(key=method, value=result)
}

out <- list()
for (comboNum in 1:num_combos) {
  dist <- ifelse(comboNum %in% c(1:6), 1, 2)
  n <- ifelse(comboNum %in% c(1, 2, 3, 7, 8, 9), 20, 50)
  cens <- ifelse(comboNum %in% c(1, 4, 7, 10), 0.3,
                 ifelse(comboNum %in% c(2, 5, 8, 11), 0.5, 0.8))
  true_mean <- p2m_lnorm(lnorm_mu[dist], lnorm_sigma[dist])$mean
  true_sd <- p2m_lnorm(lnorm_mu[dist], lnorm_sigma[dist])$sd
  true_out <- c(true_mean, true_sd, NA, NA, # true_lower, true_higher,
                lnorm_mu[dist], lnorm_sigma[dist])

  result_tmp <- array(NA, dim=c(nSims, nMethods, 2)) #2 params(mean and sd)
  bias_tmp <- array(NA, dim=c(nSims, nMethods, 2))
  relbias_tmp <- array(NA, dim=c(nSims, nMethods, 2))
  mse_tmp <- array(NA, dim=c(nSims, nMethods, 2))
  dist_tmp <- array(NA, dim=c(nSims, nMethods))

  for (i in 1:nSims) {
    for (j in 1:nMethods) {
      result_tmp[i, j, ] <- as.numeric(all_results[[comboNum]][i, j, 1:2])
      bias_tmp[i, j, ] <- as.numeric(result_tmp[i, j, ] - true_out[1:2])
      relbias_tmp[i, j, ] <- as.numeric((result_tmp[i, j, ] -
                                           true_out[1:2]) / abs(true_out[1:2]))
```

```
      mse_tmp[i, j, ] <- as.numeric((result_tmp[i, j, ] - true_out[1:2]) ^ 2)

      dist_tmp[i, j] <- ifelse(is.na(all_results[[comboNum]][i, j, 5]),
                               NA, ifelse(all_results[[comboNum]][i, j, 5]=="Lognormal",
                                          "Correct", "Incorrect"))
    }}

  loutmean <- suppressWarnings(lapply(list(result_tmp[,,1], bias_tmp[,,1],
                          relbias_tmp[,,1], mse_tmp[,,1],
                          dist_tmp), gfun))

  loutsd <- suppressWarnings(lapply(list(result_tmp[,,2], bias_tmp[,,2],
                        relbias_tmp[,,2], mse_tmp[,,2], dist_tmp), gfun))

  outmean <- cbind(loutmean[[1]][,1], do.call("cbind",
                                              lapply(loutmean, function(x) x[,2])))

  outsd <- cbind(loutsd[[1]][,1], do.call("cbind",
                                          lapply(loutsd, function(x) x[,2])))

  out[[comboNum]] <- rbind.data.frame(outmean, outsd)
  out[[comboNum]]$Parameter <- c(rep("Mean", nrow(outmean)), rep("SD", nrow(outsd)))
  names(out[[comboNum]]) <- c("Method", "Result", "Bias", "RelBias", "MSE", "CorrectDist",
"Parameter")

  out[[comboNum]]$Distribution <- ifelse(dist==1, "Mildly Skewed", "Highly Skewed")
  out[[comboNum]]$SampleSize <- n
  out[[comboNum]]$CensoringLevel <- cens

  out[[comboNum]]$CorrectDist <- out[[comboNum]]$CorrectDist
}

# save Mean and SD in long format
out_table <- do.call("rbind.data.frame", out) %>%
  dplyr::select(Parameter, Method, Distribution, SampleSize, CensoringLevel, CorrectDist,
                Result, Bias, RelBias, MSE)


openxlsx::write.xlsx(out_table, "simresultsv2_no_bootstrap.xlsx", row.names=FALSE)
```