eAppendix

This document contains R programs that perform the simulation based on the North Carolina infant mortality data. There are a total 10 files (9 programs and 1 data file) presented in this document:

1. Code_AIDD.q
2. Code_Hybrid.q
3. Code_TPS.q
4. Code_UsefulFunctions.q
5. logLikeH_C.c
6. RunSimulation-ADD.q
7. RunSimulation-Hybrid.q
8. RunSimulation-TPS-Strata.q
9. RunSimulation-TPS.q
10. Data_NCinfants.dat

Please note the following:

- Each program with a filename beginning 'Code_' provides underlying functions need to generate data and run each of the analysis approaches.
- Each program with a filename beginning 'RunSimulation_' runs a separate simulation for the specific analysis approach. The exception is 'RunSimulation-TPS-Strata.q' which sets up the phase I stratification schemes considered in the simulation presented in the paper. This program must be run prior to 'RunSimulation-TPS.q'
- Prior to running 'RunSimulation-Hybrid.q', the 'logLikeH_C.c' script must be compiled. The latter contains C code which, when complied, is called by R in the main function of 'Code_Hybrid.q'.
- Each of the 'RunSimulation_' scripts current has the number of replications set to 10. It is suggested that this be used initially as a means to gauge how long running a larger simulation (e.g., as with the 10,000 reported in the manuscript) would take.
- The datafile can be imported into R using the source() command.

Note the code is also available at http://www.hsph.harvard.edu/faculty/sebastien-haneuse/. If you have any questions, comments, or discover an error, please contact Sebastien Haneuse at shaneuse@hsph.harvard.edu.

Code_AIDD.q

```
## Evaluate the ADD score function
##
evalADDU <- function(beta, Ymat, Xmat, subset, weighted, returnComps=F)
{
  ##
  p       <- length(beta)
  designX <- as.matrix(cbind(1, Xmat[,2:p]))
  Kstar   <- nrow(designX)
  K       <- nrow(Ymat)

  ## Core quantities for each of the epsilon functions
  ##
  pi_ki     <- matrix(exp(designX %*% beta), nrow=Kstar, ncol=p)
  eMat      <- array(0, dim=c(Kstar, p, 2))
  eMat[,,1] <- designX * pi_ki
  eMat[,,2] <- designX * pi_ki * pi_ki

  ## Area/covariate pattern-specific totals; depends on if complete vs
survey information is available.
  ##
  counts <- ifelse(rep(subset, nrow(Xmat)), Xmat$mx, Xmat$Mx)
  totals <- as.vector(tapply(counts, list(Xmat$Area), sum))

  ## Basics
  ##
  Yk  <- Ymat$N1 / Ymat$N
  pik <- as.vector(tapply(counts*eMat[,1,1], list(Xmat$Area), FUN=sum)
/ totals)
  Vk  <- rep(1, K)
  if(weighted == TRUE) Vk  <- as.vector(tapply(counts*eMat[,1,1]*(1-
eMat[,1,1]), list(Xmat$Area), sum) / (totals^2))
  Dk  <- matrix(NA, nrow=K, ncol=p)
  for(i in 1:p) Dk[,i] <- as.vector(tapply(counts*eMat[,i,1],
list(Xmat$Area), FUN=sum) / totals)

  ##
  value <- as.vector(t(Dk) %*% ((Yk - pik) / Vk))

  ##
  if(returnComps == TRUE) value <- list(Yk=Yk, pik=pik, Vk=Vk, Dk=Dk)
  ##
  return(value)
}


## Evaluate the individual-level data score function
##
evalINDU <- function(beta, Imat, weighted, returnComps=F)
{
  ##
  p       <- length(beta)
  designX <- as.matrix(cbind(1, Imat[,2:p]))
  Kstar   <- nrow(designX)

  ## Core quantities
```

```
  ##
  pikx <- exp(designX %*% beta)
  Ykx  <- Imat$n1x / Imat$mx
  Vkx  <- rep(1, Kstar)
  if(weighted == TRUE) Vkx  <- as.vector((pikx * (1-pikx)) / Imat$mx)
  Dkx  <- designX * matrix(pikx, nrow=Kstar, ncol=p)

  ##
  value <- as.vector(t(Dkx) %*% ((Ykx - pikx) / Vkx))
  if(returnComps == T) value <- list(Area=Imat$Area, Ykx=Ykx,
pikx=pikx, Dkx=Dkx, Vkx=Vkx)
  return(value)
}


##
evalU <- function(beta, Ymat, Xmat, Imat, design, subset, weighted)
{
  value <- rep(0, length(beta))
  if(design[1] == T) value <- value + evalADDU(beta, Ymat, Xmat,
subset=subset, weighted=weighted)
  if(design[2] == T) value <- value + evalINDU(beta, Imat,
weighted=weighted)
  return(value)
}


## Numerical evalution of the derivative of the score function
##
derivU <- function(beta, Ymat, Xmat, Imat, design, subset, weighted,
h=0.0000001)
{
  ##
  p     <- length(beta)
  value <- matrix(NA, nrow=p, ncol=p)
  for(i in 1:p)
  {
    h0        <- rep(0, p)
    h0[i]     <- h
    value[,i] <- (evalU(beta+h0, Ymat, Xmat, Imat, design=design,
subset=subset, weighted=weighted) - evalU(beta-h0, Ymat, Xmat, Imat,
design=design, subset=subset, weighted=weighted)) / (2*h)
  }
  return(value)
}


## Solve the AIDD estimating equation
##
AIDD <- function(betaStart, data, design=c(T, T), subset=TRUE,
weighted=TRUE, beta.tol=1e-10, maxiter=100, verbose=FALSE, scaling=1)
{
  ## Initialize coefficients
  beta <- betaStart

  ## Begin iteration
  conv <- FALSE
```

```
  iter <- 1
  p    <- length(betaStart)
  betaOld <- rep(0, p)
  if(verbose == T)
  {
    cat("\n\nITERATION # 0", fill=T)
    cat("Beta :", beta, fill=T)
    cat("\n")
  }

  repeat
  {
    ## Newton-Raphson update
    ##
    Uvec <- evalU(beta, data$Ymat, data$Xmat, data$Imat, design=design,
subset=subset, weighted=weighted)
    Hmat <- derivU(beta, data$Ymat, data$Xmat, data$Imat,
design=design, subset=subset, weighted=weighted)
    beta <- beta - c((solve(Hmat) %*% Uvec) / scaling)

    ##
    if(verbose==T)
    {
      cat("\n\nITERATION #", iter, fill=T)
      cat("Beta :", beta, fill=T)
      cat("Score :", Uvec, fill=T)
      cat("\n")
    }

    ##
    problem <- (sum( is.element(beta, NA) ) > 0) | (sum(
is.element(beta, NaN) ) > 0)
    if(problem) break

    ## Stop if parameters are all within specified tolerances.
    if(iter > 1 && all(abs(beta - betaOld) < beta.tol))
    {
      conv <- T
      break
    }

    ## Stop if maximum number of iterations has been reached
    if(iter >= maxiter) break

    ## Otherwise, update parameter estimates and run another iteration
    betaOld <- beta
    iter <- iter + 1
    NULL
  }

  ## Final evaluation for the standard error estimates
  ##
  Areas <- NULL
  if(design[1] == TRUE)
  {
    UcompsADD <- evalADDU(beta, data$Ymat, data$Xmat, subset=subset,
weighted=weighted, returnComps=T)
```

```
    deltaA    <- as.vector((UcompsADD$Yk - UcompsADD$pik))
    Areas     <- c(Areas, c(1:length(UcompsADD$Yk)))
  }
  if(design[2] == TRUE)
  {
    UcompsIND <- evalINDU(beta, data$Imat, weighted=weighted,
returnComps=T)
    deltaI    <- as.vector((UcompsIND$Ykx - UcompsIND$pikx))
    Areas     <- c(Areas, UcompsIND$Area)
  }

  ##
  Areas <- sort(unique(Areas))
  ##
  Fm <- matrix(0, nrow=p, ncol=p)
  Ir <- matrix(0, nrow=p, ncol=p)
  ##
  for(k in Areas)
  {
    ##
    Dk     <- NULL
    Vk     <- NULL
    deltak <- NULL
    ##
    if(design[1] == TRUE)
    {
      Dk     <- rbind(Dk, matrix(UcompsADD$Dk[k,], nrow=1, ncol=p))
      Vk     <- c(Vk, UcompsADD$Vk[k])
      deltak <- c(deltak, deltaA[k])
    }
    if(design[2] == TRUE)
    {
      if(is.element(k, UcompsIND$Area))
      {
        subk   <- c(1:length(UcompsIND$Area))[UcompsIND$Area == k]
        Dk     <- rbind(Dk, UcompsIND$Dkx[subk,])
        Vk     <- c(Vk, UcompsIND$Vkx[subk])
        deltak <- c(deltak, deltaI[subk])
      }
    }
    Vk  <- diag(Vk, length(Vk))
    ddT <- deltak %o% deltak
    ##
    Fm <- Fm + (t(Dk) %*% solve(Vk) %*% Dk)
    Ir <- Ir + (t(Dk) %*% solve(Vk) %*% ddT %*% solve(Vk) %*% Dk)
  }
  ##
  modse <- as.vector(sqrt(diag(solve(Fm))))
  robse <- as.vector(sqrt(diag(solve(Fm) %*% Ir %*% solve(Fm))))

  ##
  coef <- data.frame(cbind(beta, modse, robse))
  return(list(coef=coef, conv=conv, iter=iter))
}
```

Code_Hybrid.q

```
##
dyn.load("logLikeH_C.so")
source("Code_UsefulFunctions.q")


## Evaluate the log-likelihood for the Hybrid design
##  - restricted to the case with two binary exposures; X and Z
##  - beta: the coefficient vector: intercept, main effect for X, main
effect for Z, interaction term
##
logLikeH <- function(beta, data, approx=NULL)
{
  ##
  ids <- unique(data$county)
  if(length(approx) == 0) approx <- rep(F, length(ids))
  if(length(approx) < length(ids)) approx <- rep(F, length(ids))

  ##
  logLike <- 0
  for(k in 1:length(ids))
  {
    ##
    areak <- data[data$county == ids[k],]

    ##
    Ny   <- c(sum(areak$N0xz), sum(areak$N1xz))
    Mxz  <- areak$Mxz

    ##
    ny   <- c(sum(areak$n0xz), sum(areak$n1xz))     ## c(n0, n1)
    mxz  <- areak$n0xz + areak$n1xz                 ## c(m00, m10, m01,
m11)
    n0xz <- areak$n0xz
    n1xz <- areak$n1xz

    ## Make sure beta is of length 4
    ##
    designX <- cbind(1, areak[,2:3], inter=areak[,2]*areak[,3])
    if(length(beta) == 3) eta <- as.vector(c(beta,0) %*% t(designX))
    if(length(beta) == 4) eta <- as.vector(beta %*% t(designX))
    pixz    <- expit(eta)

    ##
    if(approx[k] == FALSE)
    {
      ##
      logLikek <- .C("dHybrid",
                     pixz   = as.double(pixz),
                     Ny    = as.integer(Ny),
                     Mxz   = as.integer(Mxz),
                     n0xz  = as.integer(n0xz),
                      n1xz  = as.integer(n1xz),
                      value = as.double(0))$value
    }
```

```
    ##
    if(approx[k] == TRUE)
    {
      ## Adjusted group-level data
      ##
      Ny  <- Ny - ny
      Mxz <- Mxz - mxz

      ##
      mu    <- sum(Mxz * pixz)
      compE <- dbinom(Ny[2], sum(Ny), mu/sum(Ny), log=T)

      ##
      compN  <- dmhyper(Ny, ny, log=T)
      compM  <- dmhyper(Mxz, mxz, log=T)
      compCC <- sum(dbinom(n1xz, mxz, pixz, log=T))

      ##
      logLikek <- compE + compM - compN + compCC
    }

    ##
    logLike <- logLike + logLikek

  }
  return(-logLike)
}
```

Code_TPS.q

```
"tps" <- function(formula = formula(data), data = sys.parent(), nn0,
nn1, group, contrasts = NULL, method = "PL", cohort = T, alpha = 1)
{
  call <- match.call()
  m    <- match.call(expand = F)
  m$method <- m$contrasts <- m$nn0 <- m$nn1 <- m$group <- m$cohort <-
m$
  alpha  <- NULL
  m[[1]] <- as.name("model.frame")
  m      <- eval(m, sys.parent())
  Terms  <- attr(m, "terms")
  a      <- attributes(m)
  Y      <- model.extract(m, response)
  X      <- model.matrix(Terms, m, contrasts)

  ## Potential Errors
  if(length(nn0) != length(nn1))            stop("nn0 and nn1 should be
of same length")
  if(length(nn0) != length(unique(group))) stop("Number of strata
defined by group should be same as length of nn0")
  if(length(group) != nrow(X))             stop("Group and x are not
compatible")
  if((any(nn1 == 0)) || (any(nn0 == 0)))   stop("Zero cell frequency at
phase I")

  ## method
  imeth <- charmatch(method, c("PL", "WL", "ML"), nomatch = 0)
  methodName <- switch(imeth + 1,
                       stop("Method doesn't exist"),
                       "PL",
                       "WL",
                       "ML")
  ## data
  if(is.matrix(Y) && (ncol(Y) > 1))
  {
    case <- as.vector(Y[, 1])
    N    <- as.vector(Y[, 1] + Y[, 2])
  }
  else
  {
    case <- as.vector(Y)
    N    <- rep(1, length(case))
  }

  ## evaluation
  z   <- call(methodName, nn0 = nn0, nn1 = nn1, x = X, N = N, case =
case, group = group, cohort = cohort, alpha = alpha)
  #out <- eval(z, local = sys.parent())
  out <- eval(z)
  names(out$coef) <- dimnames(X)[[2]]
  if(!is.null(out$cove)) dimnames(out$cove) <- list(dimnames(X)[[2]],
dimnames(X)[[2]])
  if(!is.null(out$covm)) dimnames(out$covm) <- list(dimnames(X)[[2]],
dimnames(X)[[2]])
  out$method <- method
```

```
  class(out) <- "tps"
  out
}


"WL" <- function(nn0, nn1, x, N, case, group, cohort, alpha)
{
  # S function for the weighted likelihood or Horvitz-Thompson type of
estimator.
  nntot0 <- sum(nn0)  # Phase I control Total
  nntot1 <- sum(nn1)  # Phase I case Total
  u <- sort(unique(group))
  strt.id <- outer(group,u,FUN="==")
  strt.id <-
matrix(as.numeric(strt.id),nrow=length(group),ncol=length(u))
  n1 <- apply(strt.id * case, 2, sum)        # n1= Phase II case sample
sizes
  n0 <- apply(strt.id * (N - case), 2, sum) # n0 = phase II control
sample sizes
  if((any(n0 == 0)) || (any(n1 == 0)))
    stop("Zero cell frequency at phase II")
  strt.id <- rbind(strt.id, strt.id)
  N <- c(case, N - case)
  case <- c(rep(1, length(case)), rep(0, length(case)))
  group <- c(group, group)
  x <- rbind(x, x)
  nstrta <- length(nn0)
  ofs <- 0
  if(!cohort)
    ofs <- log(sum(nn1)/sum(nn0)) - log(alpha)
  ofs <- rep(ofs, length(case))
  pw0 <- nn0[group]/n0[group]
  pw1 <- nn1[group]/n1[group]
  pw <- case * pw1 + (1 - case) * pw0
  Nw <- N*pw
  lp <- rep(0, length(case))
  z <- rep(0, length(case))
  z[case == 1] <- Nw[case == 1]
  m <- glm(cbind(z, Nw - z) ~ -1 + x + offset(ofs), family = binomial,
x=T,
           control = glm.control(epsilon = 9.9999999999999995e-07,
maxit = 20))
  #  weights=pw, start = lp,
  uj0 <-  - t(m$x) %*% (strt.id * (1 - case) * m$fitted.values * N)
  #   Within group sum of scores for controls
  uj1 <- t(m$x) %*% (strt.id * case * (1 - m$fitted.values) * N)
  #     Within group sum of scores for cases
  g0 <- t(m$x * as.vector((1 - case) * m$fitted.values^2 * N * pw0^2))
%*% m$x           ## CHANGED 29TH OCT 2007
  g1 <- t(m$x * as.vector(case * (1 - m$fitted.values)^2 * N * pw1^2))
%*% m$x
  identity <- diag(rep(1, nstrta))
  if((any(n1 == 0)) || (any(n0 == 0)))
    stop("No cell should be empty in phase II")
  ##
  b0temp <- as.vector((nn0 * (nn0 - n0))/n0^3)
## CHANGED 30TH OCT 2007
```

```
  b0 <- diag(b0temp, nrow=length(b0temp))
  ##
  b1temp <- as.vector((nn1 * (nn1 - n1))/n1^3)
## CHANGED 30TH OCT 2007
  b1 <- diag(b1temp, nrow=length(b1temp))
  ##
  bb0temp <- as.vector((n0/(nntot0 * nn0)))
## CHANGED 30TH OCT 2007
  bb0 <- diag(bb0temp, nrow=length(bb0temp))
  ##
  bb1temp <- as.vector((n1/(nntot1 * nn1)))
## CHANGED 30TH OCT 2007
  bb1 <- diag(bb1temp, nrow=length(bb1temp))
  ##
  cov <- summary(m)$cov.unscaled
  cove <- g1 + g0 - uj0 %*% b0 %*% t(uj0) - uj1 %*% b1 %*% t(uj1)
  if(!cohort)
    cove <- cove - uj0 %*% bb0 %*% t(uj0) - uj1 %*% bb1 %*% t(uj1)
  cove <- cov %*% cove %*% cov
  z <- list(coef = m$coef, cove = cove)
  z
}


"PL" <- function(nn0, nn1, x, N, case, group, cohort, alpha)
{
# S-function for "Pseudo Likelihood"  method  as developed by Breslow
and Cain (1988)
  nntot0 <- sum(nn0)  # Phase I control Total
  nntot1 <- sum(nn1)  # Phase I case Total
  nstrata <- length(nn0)
  u <- sort(unique(group))
  strt.id <- outer(group,u,FUN="==")
  strt.id <-
matrix(as.numeric(strt.id),nrow=length(group),ncol=length(u))
  n1 <- apply(strt.id * case, 2, sum)       # n1 = Phase II case sample
sizes
  n0 <- apply(strt.id * (N - case), 2, sum) # n0 = phase II control
sample sizes
  if((any(n0 == 0)) || (any(n1 == 0)))
    stop("Zero cell frequency  at phase II")
  ofs <- log(n1/n0) - log(nn1/nn0)
  if(!cohort)
    ofs <- ofs + log(nntot1/nntot0) - log(alpha)
  ofs <- ofs[group]
  lp <-  - ofs
  pw <- rep(1, length(case))
  #  Fitting model using standard GLM procedure
  m <- glm(cbind(case, N - case) ~ -1 + x + offset(ofs), family =
binomial, weights = pw, x = T, control = glm.control(epsilon =
9.9999999999999995e-07, maxit = 20))
  fv <- m$fitted.values
  nhat0 <- apply(strt.id * (1 - fv) * N, 2, sum)
  #  Within group sum of fitted values for controls
  nhat1 <- apply(strt.id * fv * N, 2, sum)
  # Within group sum of fited values
  # Adjusted covariance Matrix
```

```
  uj0 <-  - t(m$x) %*% (strt.id * (N - case) * m$fitted.values)
  #   Within group sum of scores for controls
  uj1 <-  - t(m$x) %*% (strt.id * case * (1 - m$fitted.values))
  #     Within group sum of scores for cases
  g0 <- t(m$x * (N - case) * fv^2) %*% m$x
  g1 <- t(m$x * case * (1 - fv)^2) %*% m$x
  a <- t(m$x) %*% (strt.id * (1 - fv) * fv * N)
  zz <- matrix(1, nrow = nstrata, ncol = nstrata)
  identity <- diag(rep(1, nstrata))
  b0 <- identity/as.vector(n0)
  b1 <- identity/as.vector(n1)
  bb0 <- (identity/as.vector(nn0))
  bb1 <- (identity/as.vector(nn1))
  if(!cohort)
  {
    bb0 <- bb0 - (zz/nntot0)
    bb1 <- bb1 - (zz/nntot1)
  }
  aba <- a %*% bb0 %*% t(a) + a %*% bb1 %*% t(a)
  info <- t(m$x) %*% diag(m$weight) %*% m$x
  info2 <- solve(summary(m)$cov.unscaled)
  ghat <- info - a %*% b0 %*% t(a) - a %*% b1 %*% t(a)
  ghate <- g1 + g0 - uj0 %*% b0 %*% t(uj0) - uj1 %*% b1 %*% t(uj1)
  cov <- summary(m)$cov.unscaled
  cove <- cov %*% (ghate + aba) %*% cov
  # Empirical variance-covariance matrix
  cove <- cov %*% (ghate + aba) %*% cov
  # Model based variance-covariance matrix
  covm <- cov %*% (ghat + aba) %*% cov
  return(list(coef = m$coef, covm = covm, cove = cove))
}


"ML"<-
function(nn0, nn1, x, N, case, group, cohort, alpha, maxiter = 100)
{
# S function to solve the concentrated lagrangian equations developed
by
# Breslow and Holubkov (1997). This function implements a modified
# Newton-Rhapson  algorithm to solve the equations. Gradients as
computed by
# the authors were used to implement the NR algorithm.
  iter <- 0
  converge <- F
  rerror <- 100
  nntot0 <- sum(nn0)  # Phase I control Total
  nntot1 <- sum(nn1)  # Phase I case Total
  nn <- nn0 + nn1
  nntot <- nntot0 + nntot1
  nstrata <- length(nn0)
  nobs <- length(case)
  ncovs <- ncol(x)
  stpmax <- 1 * (ncovs + nstrata)
  u <- sort(unique(group))
  strt.id <- outer(group,u,FUN="==")
  strt.id <-
matrix(as.numeric(strt.id),nrow=length(group),ncol=length(u))
```

```
  ee1 <- cbind(matrix(1, nstrata, 1), matrix(0, nstrata, nstrata))
  ee2 <- cbind(matrix(0, nobs, 1), strt.id)
  ee <- rbind(ee1, ee2)
  n1 <- apply(strt.id * case, 2, sum)
  # n1= Phase II case sample sizes
  n0 <- apply(strt.id * (N - case), 2, sum)
  #  n0 = phase II control sample sizes
  if((any(n0 == 0)) || (any(n1 == 0)))
    stop("Zero cell frequency at phase II")
  n <- n0 + n1
  n1a <- c(nntot1, n1)
  n0a <- c(nntot0, n0)
  grpa <- c(rep(1, nstrata), (group + 1))  # Augmented group indicator
  na <- n0a + n1a
  yy <- c(nn1, case)
  identity <- diag(rep(1, nstrata))
  x0 <- cbind(identity, matrix(0, nrow = nstrata, ncol = ncovs))
  x <- cbind( - strt.id, x)  # Augmented covariate matrix
  xx <- rbind(x0, x)
  ofs <- log(n1a/n0a)
   ofs[1] <- ofs[1] - log(alpha)
  if(cohort)
    ofs[1] <- ofs[1] - log(nntot1/nntot0) + log(alpha)
  ofs <- ofs[grpa]
  repp <- c(nn1 + nn0, N)  # Augmented binomial denominator
  m <- glm(cbind(yy, repp - yy) ~ -1 + xx + offset(ofs), family =
binomial, x = T, control = glm.control(epsilon = 9.9999999999999995e-
07, maxit = 20))
  gamm.schill <- as.vector(m$coef)
  gamm0 <- gamm <- gamm.schill  # Initialize by Schill's estimates
  errcode <- 0
  while((iter < maxiter) && (rerror > 9.9999999999999995e-07))
  {
    #   cat("No of iterations=", iter, "\n")
    l <- lagrange(gamm0, nstrata, nn1, nn0, n1a, n0a, grpa, repp, xx,
ofs, yy)
    h <- lagrad(gamm0, nstrata, nobs, ncovs, nn1, nn0, n1a, n0a, ee,
repp, grpa, n0, n1, xx, ofs)
    p <-  - solve(h) %*% l  # Newton's direction
    sp2 <- sqrt(sum(p^2))
    if(sp2 > stpmax)
      p <- p * (stpmax/sp2)
    # Scale if the attempted stepis too big
    gamm <- lnsrch(gamm0, 0.5 * sum(l^2), t(h) %*% l, p, nstrata, nn1,
nn0, n1a, n0a, grpa, repp, xx, ofs, yy)
    # Search for new value along the line of Newton's direction
    rerror <- max(abs(gamm - gamm0)/pmax(abs(gamm0),
0.10000000000000001))
    gamm0 <- gamm
    iter <- iter + 1
  }
  if(iter < maxiter) converge <- T  #cat("No of iterations=", iter,
"\n")
  h <- lagrad(gamm0, nstrata, nobs, ncovs, nn1, nn0, n1a, n0a, ee,
repp, grpa, n0, n1, xx, ofs)
  fvv <- xx %*% gamm + ofs
  fvv <- as.vector(exp(fvv)/(1 + exp(fvv)))
```

```
  t1 <- nn1 - nn * fvv[1:nstrata]
  mu <- 1 - t1/n1
  t1 <- c(0, t1)
  mu <- c(1, mu)
  qn <- repp * n0a[grpa]
  qd1 <- n0a[grpa]
  qd2 <- (1 - mu[grpa]) * (n1a[grpa] - na[grpa] * fvv)
  q <- qn/(qd1 + qd2)
  gg <- t(xx) %*% ((1 - fvv) * fvv * q * ee)
  t00 <- (1 - fvv)^2 * q * ee
  d00 <- rep.int(1, nrow(t00)) %*% t00
  t01 <- (1 - fvv) * fvv * q * ee
  d01 <- rep.int(1, nrow(t01)) %*% t01
  t11 <- fvv * fvv * q * ee
  d11 <- rep.int(1, nrow(t11)) %*% t11
  d00 <- diag(as.vector(d00))
  d01 <- diag(as.vector(d01))
  d11 <- diag(as.vector(d11))
  tinfo <- (1 - fvv) * fvv * q * xx
  info <- t(xx) %*% tinfo
  cove <- solve(info)
  gig <- t(gg) %*% cove %*% gg
  zz1 <- cbind((gig + d00), ( - gig + d01))
  zz2 <- cbind(( - gig + d01), (gig + d11))
  zz <- rbind(zz1, zz2)
  zz <- solve(zz)
  gg <- cbind(gg,  - gg)
  gig <- cove %*% gg %*% zz %*% t(gg) %*% cove
  cov1 <- cove - gig
  # Model covariance matrix using Atchinson and Silvey formula
  # Computation of empirical variance-covariance matrix using within
group scores
  ta <- n0a * (n1a - t1)
  tb <- na * t1
  g <- (ta[grpa] * fvv)/(ta[grpa] + tb[grpa] * (1 - fvv))
  uj0 <-  - t(m$x) %*% (ee * (repp - yy) * g)
  uj1 <- t(m$x) %*% (ee * yy * (1 - g))
  gg <- t(m$x * yy * (1 - g)^2) %*% m$x + t(m$x * (repp - yy) * g^2)
%*% m$x
  #u1j0 <- -t(xx)%*%(ee*(repp-yy)*g)
  #u1j1 <- t(xx)%*%(ee*yy*(1-g))
  #print("uj0:")
  #print(uj0)
  #print("uj1:")
  #print(uj1)
  # gg <- t(xx*((1-g)^2*yy + g^2*(repp-yy)))%*%xx
  gig <- gg - t(t(uj0)/n0a) %*% t(uj0) - t(t(uj1)/n1a) %*% t(uj1)
  #gig <- gg - t( t(u1j0) / n0a)  %*% t(u1j0) -  t( t(u1j1) / n1a ) %*%
t(u1j1)
  #print("gig")
  #print(gig)
  h <- solve(h)
  cov2 <- t(h) %*% gig %*% h
  # Adjusting term for asymptotic variance-covariance matrix for
prospective first stage sampling
  adjust <- (1/nntot0) + (1/nntot1)
  adjust <- matrix(adjust, nrow = (nstrata + 1), ncol = (nstrata + 1))
```

```
  if(cohort)
  {
    cov1[1:(nstrata + 1), 1:(nstrata + 1)] <- cov1[1:(nstrata + 1),
1:(nstrata + 1)] + adjust
    cov2[1:(nstrata + 1), 1:(nstrata + 1)] <- cov2[1:(nstrata + 1),
1:(nstrata + 1)] + adjust
  }
  cov1 <- cov1[(nstrata + 1):(nstrata + ncovs), (nstrata + 1):(nstrata
+ ncovs)]
  # Exctract the covariance matrix for the regression coefficients
  cov2 <- cov2[(nstrata + 1):(nstrata + ncovs), (nstrata + 1):(nstrata
+ ncovs)]
  # Exctract the covariance matrix for the regression coefficients
  # jc <- jc[(nstrata + 1):(nstrata + ncovs), (nstrata + 1):(nstrata +
ncovs)]
  delta <- gamm[1:nstrata]
  b <- gamm[(nstrata + 1):(nstrata + ncovs)]
  q <- q/na[grpa]
  out <- list(coef = b, covm = cov1, cove = cov2, delta = delta, mu =
mu)
  out
}


"summary.tps"<- function(object)
{
  # produces summary from an object of the class "tps"
  coef <- object$coef
  method <- object$method
  if(method == "WL")
  {
    see <- sqrt(diag(object$cove))
    tee <- coef/see
    pee <- 2 * (1 - pnorm(abs(tee)))
    coefficients <- matrix(0, nrow = length(coef), ncol = 4)
    dimnames(coefficients) <- list(names(coef), c("Value", "Emp SE",
"Emp t", "Emp p"))
    coefficients[, 1] <- coef
    coefficients[, 2] <- see
    coefficients[, 3] <- tee
    coefficients[, 4] <- pee
  }
  else
  {
    se  <- sqrt(diag(object$covm))
    see <- sqrt(diag(object$cove))
    te  <- coef/se
    tee <- coef/see
    pe  <- 2 * (1 - pnorm(abs(te)))
    pee <- 2 * (1 - pnorm(abs(tee)))
    coefficients <- matrix(0, nrow = length(coef), ncol = 7)
    dimnames(coefficients) <- list(names(coef), c("Value", "Mod SE",
"Mod t", "Mod p", "Emp SE", "Emp t", "Emp p"))
    coefficients[, 1] <- coef
    coefficients[, 2] <- se
    coefficients[, 3] <- te
    coefficients[, 4] <- pe
```

```
    coefficients[, 5] <- see
    coefficients[, 6] <- tee
    coefficients[, 7] <- pee
  }
  structure(list(coefficients = coefficients), class = "summary.tps")
}

#function to compute the lagrangian equations
lagrange <- function(gamm0, nstrata , nn1 , nn0, n1a , n0a , grpa ,repp
, xx , ofs , yy)
{
  fv <- xx%*%gamm0 + ofs
  fv <- exp(fv)
  fv <- fv/(1+fv)
  r  <- fv[1:nstrata]
  r  <- nn1 - (nn1 + nn0)*r
  r  <- c(0,r)
  r1 <- (n1a-r)*n0a
  r2 <- (n0a+n1a)*r
  g  <- r1[grpa]*fv
  g  <- g/(r1[grpa] + r2[grpa]*(1-fv))
  g  <- t(xx)%*%((1-g)*yy - g*(repp-yy))
  g
}


#function to compute the gradients
lagrad <- function(gamm0, nstrata, nobs , ncovs ,nn1 , nn0 , n1a , n0a
,ee ,repp , grpa , n0 , n1 , xx , ofs)
{
  fv  <- xx%*%gamm0 + ofs
  fv  <- exp(fv)
  fv  <- fv/(1+fv)
  xxx <- xx[(nstrata+1):(nstrata+nobs),]
  grp <- grpa[(nstrata+1):(nstrata+nobs)]-1
  rep <- repp[(nstrata+1):(nstrata+nobs)]
  e   <- ee[(nstrata+1):(nstrata+nobs),2:(nstrata+1)]
  pp  <- fv[1:nstrata]
  g   <- fv[(nstrata+1):(nstrata+nobs)]
  r   <- nn1 - (nn1+nn0)*pp
  r0  <- r*(n0+n1)
  r1  <- n0*n1*(nn0+nn1)*(n0+n1)*pp*(1-pp)
  r2  <- n0*(n1-r)

  d <- r1[grp]*g*(1-g)*rep/(r2[grp]+r0[grp]*(1-g))^2
  d <- as.vector(d)                                    ## ADDITIONAL
29TH OCT 2007

  de <- d*e
  g  <- -t(xxx)%*%de
  g  <- cbind(g,matrix(0,(nstrata+ncovs),ncovs))
  r  <- c(0,r)
  r1 <- n0a*(n1a-r)
  r2 <- (n0a+n1a)*r
  r0 <- n0a*n1a*(n1a-r)*(n0a+r)
  fv <- r0[grpa]*fv*(1-fv)*repp/(r1[grpa]+r2[grpa]*(1-fv))^2
  fv <- t(xx)%*%(hdp(fv,xx))
```

```
  g <- g - fv
  g
}

#function to compute the norm of the lagrangian equations
s2 <- function(gamm,nstrata , nn1 , nn0 , n1a , n0a , grpa , repp , xx
, ofs , yy)
{
  l <- lagrange(gamm,nstrata , nn1 , nn0 , n1a , n0a , grpa , repp , xx
, ofs , yy)
  sum(l^2)
}

#function to compute horizontal direct product of two matrices
hdp <- function(x,y)
{
  x <- as.matrix(x)
  y <- as.matrix(y)
  if(nrow(x) != nrow(y))stop("row dimensions are not same")
  z <-
x[,rep(1:ncol(x),rep(ncol(y),ncol(x)))]*y[,rep(1:ncol(y),ncol(x))]
  z
}

#function to conduct line search for modified Newton method of ML
lnsrch <-
function(gamm0,f0,g0,p,nstrata,nn1,nn0,n1a,n0a,grpa,repp,xx,ofs,yy)
{
  scale <- 1.0
  rel.error <- max(abs(p)/pmax(gamm0,0.1))
  if(rel.error < 1e-07)
    gamm <- gamm0
  else
  {
    gamm <- gamm0 + p
    l <- lagrange(gamm,nstrata,nn1,nn0,n1a,n0a,grpa,repp,xx,ofs,yy)
    f <- 0.5*sum(l^2)
    slope <- sum(g0*p)
    # backtrack if the function does not decrease sufficiently
    if(f > f0+0.0001*slope)
    {
      scale <- -slope/(2*(f-f0-slope))
      scale <- min(scale,0.5)
      scale <- max(0.05,scale)
    }
    gamm <- gamm0 + scale*p
  }
  gamm
}
```

Code_UsefulFunctions.q

```
##
expit <- function(x) exp(x) / (1 + exp(x))
logit <- function(p) log(p/(1-p))


## Probability mass function for a multivariate hypergeometric
distribution
##
dmhyper <- function(Mk, mk, log=F)
{
  M <- sum(Mk)
  m <- sum(mk)
  value <- sum(lchoose(Mk, mk)) - lchoose(M, m)
  if(log == F) value <- exp(value)
  return(value)
}


## Generate a random deviate from a multivariate hypergeometric
distribution
## - Mk is a vector of counts for the elements in the population
## - m is the size of the sample to be drawn
##
rmhyper <- function(Mk, m)
{
  k <- length(Mk)
  M <- sum(Mk)
  mk <- rep(0, k)

  ##
  if(m > M) mk <- Mk
  else
  {
    mk[1] <- rhyper(1, Mk[1], M - Mk[1], m)
    if(k > 2)
    {
      for(j in 2:(k-1))
      {
        mk[j] <- rhyper(1, Mk[j], M - sum(Mk[1:j]), m - sum(mk[1:(j-
1)]))
      }
    }
    mk[k] <- m - sum(mk[1:(k-1)])
  }
  return(mk)
}
```

logLikeH_C.c

```
/*
  TO COMPILE USE THE CODE:

  R CMD SHLIB logLikeH_C.c

*/

#include <stdio.h>
#include <math.h>

/*
#include "/usr/lib/R/include/R.h"
#include "/usr/lib/R/include/Rmath.h"
*/

/* For use on a mac */
#include
"/Library/Frameworks/R.framework/Versions/Current/Resources/include/R.h
"
#include
"/Library/Frameworks/R.framework/Versions/Current/Resources/include/Rma
th.h"


#define dbinom Rf_dbinom
#define dhyper Rf_dhyper
#define lchoose Rf_lchoose

/* */
int c_min(int value1,
          int value2)
{
  int min = (value1 < value2) ? value1 : value2;
  return min;
}


/* */
int c_max(int value1,
          int value2)
{
  int max = (value1 > value2) ? value1 : value2;
  return max;
}


/*
  HYBRID LIKELIHOOD
*/
void dHybrid(double pixz[],
             int Ny[],
             int Mxz[],
             int n0xz[],
             int n1xz[],
             double *value)
```

```
{
  int N, N0, N1, n0, n1;
  int M00, M01, M10, M11;
  int n000, n001, n010, n011;
  int n100, n101, n110, n111;

  int N000, N001, N010, N011;
  int N100, N101, N110, N111;
  int L11, U11;
  int L01, U01;
  int L10, U10;

  double compW1, compW2, compW3, compW4, logLikeI;

  /*
    Data
  */
  N    = Ny[0] + Ny[1];
  N0   = Ny[0];
  N1   = Ny[1];
  n0   = n0xz[0] + n0xz[1] + n0xz[2] + n0xz[3];
  n1   = n1xz[0] + n1xz[1] + n1xz[2] + n1xz[3];
  M00  = Mxz[0];
  M10  = Mxz[1];
  M01  = Mxz[2];
  M11  = Mxz[3];
  n000 = n0xz[0];
  n010 = n0xz[1];
  n001 = n0xz[2];
  n011 = n0xz[3];
  n100 = n1xz[0];
  n110 = n1xz[1];
  n101 = n1xz[2];
  n111 = n1xz[3];

  /*
    Hybrid likelihood calculation
  */
  compW3 = lchoose(N0, n0) + lchoose(N1, n1);

  /* */
  L11 = c_max(n111, N1 - (M00 + M10 + M01) + (n000 + n010 + n001));
  U11 = c_min(N1 - (n100 + n110 + n101), M11 - n011);
  for(N111 = L11; N111 <= U11; N111++)
  {
    /* */
    L01 = c_max(n101, N1 - N111 - (M00 + M10) + (n000 + n010));
    U01 = c_min(N1 - N111 - (n100 + n110), M01 - n001);
    for(N101 = L01; N101 <= U01; N101++)
    {
      /* */
      L10 = c_max(n110, N1 - (N111 + N101) - M00 + n000);
      U10 = c_min(N1 - (N111 + N101) - n100, M10 - n010);
      for(N110 = L10; N110 <= U10; N110++)
      {
        /* */
        N100 = N1 - (N111+N101+N110);
```

```
        N000 = M00 - N100;
        N010 = M10 - N110;
        N001 = M01 - N101;
        N011 = M11 - N111;

        /* */
        /* printf("\n %d  %d  %d", N111, N101, N110); */

        /* */
        compW1 = lchoose(N000, n000) + lchoose(N010, n010) +
lchoose(N001, n001) + lchoose(N011, n011);
        compW2 = lchoose(N100, n100) + lchoose(N110, n110) +
lchoose(N101, n101) + lchoose(N111, n111);

        /* */
        logLikeI = dbinom(N100, M00, pixz[0], 1) + dbinom(N110,
M10,pixz[1], 1) + dbinom(N101, M01, pixz[2], 1) + dbinom(N111, M11,
pixz[3], 1);

        /* */
        *value += exp(compW1 + compW2 - compW3 + logLikeI);
      }
    }
  }

  *value = log(*value);
  return;
}
```

RunSimulation-ADD.q

```
##
runCode <- FALSE
if(runCode == TRUE)
{
  ## Source in data and external functions, and get the 'true' model
and fitted values
  ##
  source("Code_AIDD.q")
  source("Code_UsefulFunctions.q")
  source("Data_NCinfants.dat")
  fit0      <- glm(cbind(Y, N-Y) ~ race + sex,
family=quasibinomial(link="log"), data=lbw)
  betaStart <- as.vector(fit0$coef)

  ## Run simulation
  ##
  R       <- 10
  results <- array(NA, dim=c(5,7,R))
  for(r in 1:R) source("RunSimulation-ADD.q")
}


## Generate new outcome information for each repitition of the
simulation
##
lbw$Y <- rbinom(nrow(lbw), lbw$N, fitted(fit0))


####
## Collate data structures for the aggregate data methods
####

## Ymat
##
Ymat <- aggregate(lbw$N, list(county=lbw$county), FUN=sum)
names(Ymat)[2] <- "N"
Ymat$N1 <- aggregate(lbw$Y, list(county=lbw$county), FUN=sum)$x
Ymat    <- Ymat[,-1]

## Xmat
##
Xmat <- lbw[,1:3]
names(Xmat)[1] <- "Area"
Xmat$Mx <- lbw$N
Xmat$mx <- rmhyper(Xmat$Mx, m=20000)

## Imat
##  * only sample complete individual-level data from the four largest
counties
##  * turns out to be those with populations of at least 10,000
##
countyN    <- aggregate(lbw$N, list(county=lbw$county), FUN=sum)$x
keepCounty <- c(1:100)[countyN > 10000]
Imat       <- lbw[is.element(lbw$county, keepCounty),]
names(Imat)[1] <- "Area"
```

```
##
n        <- 500                  ## Number of individual-level samples,
per area
Imat$mx  <- NA
Imat$n1x <- NA
for(i in 1:length(keepCounty))
{
  index <- c(1:nrow(Imat))[Imat$Area == keepCounty[i]]
  temp  <- matrix(rmhyper(c(Imat$N[index]-Imat$Y[index],
Imat$Y[index]), n), ncol=2)
  Imat$mx[index]  <- apply(temp, 1, sum)
  Imat$n1x[index] <- temp[,2]
}
Imat <- Imat[,-c(4,5,6)]


## Combine data and run AIDD analyses
##
data <- list(Ymat=Ymat, Xmat=Xmat, Imat=Imat)


##
fit1 <- try(AIDD(betaStart=betaStart, data=data, design=c(T,F),
subset=F, verbose=F, scaling=5, maxiter=1000))
fit2 <- try(AIDD(betaStart=betaStart, data=data, design=c(T,F),
subset=T, verbose=F, scaling=5, maxiter=1000))
fit3 <- try(AIDD(betaStart=betaStart, data=data, design=c(F,T),
verbose=F, scaling=5, maxiter=1000))
fit4 <- try(AIDD(betaStart=betaStart, data=data, design=c(T,T),
subset=F, verbose=F, scaling=5, maxiter=1000))
fit5 <- try(AIDD(betaStart=betaStart, data=data, design=c(T,T),
subset=T, verbose=F, scaling=5, maxiter=1000))


##
resTemp <- matrix(NA, nrow=5, ncol=7)
if(class(fit1) == "list") resTemp[1,] <- c(t(fit1$coef[,c(1,3)]),
fit1$conv)
if(class(fit2) == "list") resTemp[2,] <- c(t(fit2$coef[,c(1,3)]),
fit2$conv)
if(class(fit3) == "list") resTemp[3,] <- c(t(fit3$coef[,c(1,3)]),
fit3$conv)
if(class(fit4) == "list") resTemp[4,] <- c(t(fit4$coef[,c(1,3)]),
fit4$conv)
if(class(fit5) == "list") resTemp[5,] <- c(t(fit5$coef[,c(1,3)]),
fit5$conv)


##
cat("\nRepitition", r, "of", R, "complete")
results[,,r] <- resTemp
```

RunSimulation-Hybrid.q

```
##
runCode <- FALSE
if(runCode == TRUE)
{
  ## Source in data and external functions, and get the 'true' model
and fitted values
  ##
  source("Code_Hybrid.q")
  source("Code_UsefulFunctions.q")
  source("Data_NCinfants.dat")
  fit0      <- glm(cbind(Y, N-Y) ~ race + sex, data=lbw,
family=binomial)
  betaStart <- as.vector(fit0$coef)

  ## Run simulation
  ##
  R       <- 10
  results <- array(NA, dim=c(2,4,R))
  for(r in 1:R) source("RunSimulation-Hybrid.q")
}


## Generate new outcome information for each repitition of the
simulation
##
lbw$Y <- rbinom(nrow(lbw), lbw$N, fitted(fit0))


## Complete individual-level data
##
lbwI <- data.frame(county=rep(1:100, rep(4, 100)), race=rep(c(0,1),
200), sex=rep(c(0,0,1,1), 100), Mxz=0, N0xz=0, N1xz=0)
for(i in 1:400)
{
  lbwI$Mxz[i]  <- sum(lbw$N[lbw$county == lbwI$county[i] & lbw$race ==
lbwI$race[i] & lbw$sex == lbwI$sex[i]])
  lbwI$N1xz[i] <- sum(lbw$Y[lbw$county == lbwI$county[i] & lbw$race ==
lbwI$race[i] & lbw$sex == lbwI$sex[i]])
}
lbwI$N0xz <- lbwI$Mxz - lbwI$N1xz


## n=500 case-control samples in each of the four largest areas
##
lbwH      <- lbwI
lbwH$n0xz <- 0
lbwH$n1xz <- 0
#K <- length(unique(lbwH$county))
for(k in c(26, 41, 60, 92))
{
  lbwH$n0xz[(lbwH$county == k)] <- rmhyper(lbwH$N0xz[(lbwH$county ==
k)], 250)
  lbwH$n1xz[(lbwH$county == k)] <- rmhyper(lbwH$N1xz[(lbwH$county ==
k)], 250)
}
```

```
## Approximate areas that have N1 > 100
##
N1       <- tapply(lbwH$N1xz, lbwH$county, FUN=sum)
approxN1 <- (N1 > 100)


##
fitHYa <- nlm(f=logLikeH, p=betaStart, data=lbwH, approx=approxN1,
stepmax=5, print.level=2, hessian=F)
lbwH$n0xz <- 0
fitHYb <- nlm(f=logLikeH, p=fitHYa$estimate, data=lbwH,
approx=approxN1, stepmax=5, print.level=2, hessian=F)


##
cat("\nRepitition", r, "of", R, "complete")
results[1,,r] <- c(fitHYa$code, fitHYa$estimate)
results[2,,r] <- c(fitHYb$code, fitHYb$estimate)
```

RunSimulation-TPS-Strata.q

```
## Case-control study
##
lbw$strata0 <- 1


## Marginal, county-specific non-white rates
##
tabNonW <- aggregate(lbw$N, list(county=lbw$county), FUN=sum)
names(tabNonW)[2] <- "N"
tempAgg <- aggregate(lbw$N, list(county=lbw$county, race=lbw$race),
FUN=sum)
tabNonW$R <- tabNonW$N - tempAgg$x[tempAgg$race == 0]
tabNonW$rate <- round(tabNonW$R/tabNonW$N * 100, 1)
##
lbw$Qrace <- tabNonW$rate[lbw$county]
##
lvls <- c(15, 31, 46, 61)
lbw$strata1 <- 1
for(i in 1:length(lvls)) lbw$strata1[lbw$Qrace > lvls[i]] <- (i+1)


## Marginal, county-specific male rates
##
tabMale <- aggregate(lbw$N, list(county=lbw$county), FUN=sum)
names(tabMale)[2] <- "N"
tempAgg <- aggregate(lbw$N, list(county=lbw$county, race=lbw$sex),
FUN=sum)
tabMale$G <- tabMale$N - tempAgg$x[tempAgg$race == 0]
tabMale$rate <- round(tabMale$G/tabMale$N * 100, 1)
##
lbw$Qmale <- tabMale$rate[lbw$county]
##
lvls <- c(47, 50, 52, 54)
lbw$strata2 <- 1
for(i in 1:length(lvls)) lbw$strata2[lbw$Qmale > lvls[i]] <- (i+1)
```

RunSimulation-TPS.q

```
##
runCode <- FALSE
if(runCode == TRUE)
{
  ## Source in data and external functions, and get the 'true' model
and fitted values
  ##
  source("Code_TPS.q")
  source("Code_UsefulFunctions.q")
  source("Data_NCinfants.dat")
  fit0 <- glm(cbind(Y, N-Y) ~ race + sex, data=lbw, family=binomial)

  ##
  source("RunSimulation-TPS-setup-Strata.q")
  ##
  R       <- 10
  results <- array(NA, dim=c(9,6,R))
  for(r in 1:R)
  {
    lbw$Y <- rbinom(nrow(lbw), lbw$N, fitted(fit0))
    for(Sindex in 0:2)
    {
      source("RunSimulation-TPS.q")
      results[(((Sindex*3)+1):((Sindex*3)+3)),,r] <- tempRes
    }
    cat("\nRepitition", r, "of", R, "complete")
  }
}

##
if(Sindex == 0) lbw$strata <- lbw$strata0
if(Sindex == 1) lbw$strata <- lbw$strata1
if(Sindex == 2) lbw$strata <- lbw$strata2

##
phaseOne <- aggregate(lbw$N, list(strata=lbw$strata), FUN=sum)
names(phaseOne)[2] <- "N"
phaseOne$Y <- aggregate(lbw$Y, list(strata=lbw$strata), FUN=sum)$x
##
strata <- phaseOne$strata
nn0    <- phaseOne$N - phaseOne$Y
nn1    <- phaseOne$Y


## Phase II sample sizes, assuming balanced sampling
##
n  <- 2000
m0 <- n / (2 * length(strata))
m1 <- n / (2 * length(strata))


## Phase II data
##
phaseTwo <- aggregate(lbw$N, list(strata=lbw$strata, sex=lbw$sex,
race=lbw$race), FUN=sum)
```

```
names(phaseTwo)[names(phaseTwo) == "x"] <- "N"
phaseTwo$Y <- aggregate(lbw$Y, list(strata=lbw$strata, sex=lbw$sex,
race=lbw$race), FUN=sum)$x
##
phaseTwo$alive <- NA
phaseTwo$death <- NA
for(i in 1:length(strata))
{
  index <- c(1:nrow(phaseTwo))[phaseTwo$strata == strata[i]]
  phaseTwo$alive[index] <- rmhyper(phaseTwo$N[index] -
phaseTwo$Y[index], m0)
  phaseTwo$death[index] <- rmhyper(phaseTwo$Y[index], m1)
}


##
fitWL <- tps(cbind(death, alive) ~ race + sex, phaseTwo, nn0, nn1,
phaseTwo$strata, method="WL")
fitPL <- tps(cbind(death, alive) ~ race + sex, phaseTwo, nn0, nn1,
phaseTwo$strata, method="PL")
fitML <- tps(cbind(death, alive) ~ race + sex, phaseTwo, nn0, nn1,
phaseTwo$strata, method="ML")
##
tempRes <- cbind(summary(fitWL)$coef[,c(1,2)],
                 summary(fitPL)$coef[,c(1,5)],
                 summary(fitML)$coef[,c(1,5)])
```

Data_NCinfants.dat

```
lbw <-
structure(list(county = c(1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L,
10L, 11L, 12L, 13L, 14L, 15L, 16L, 17L, 18L, 19L, 20L, 21L, 22L,
23L, 24L, 25L, 26L, 27L, 28L, 29L, 30L, 31L, 32L, 33L, 34L, 35L,
36L, 37L, 38L, 39L, 40L, 41L, 42L, 43L, 44L, 45L, 46L, 47L, 48L,
49L, 50L, 51L, 52L, 53L, 54L, 55L, 56L, 57L, 58L, 59L, 60L, 61L,
62L, 63L, 64L, 65L, 66L, 67L, 68L, 69L, 70L, 71L, 72L, 73L, 74L,
75L, 76L, 77L, 78L, 79L, 80L, 81L, 82L, 83L, 84L, 85L, 86L, 87L,
88L, 89L, 90L, 91L, 92L, 93L, 94L, 95L, 96L, 97L, 98L, 99L, 100L,
1L, 2L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L,
16L, 17L, 18L, 19L, 20L, 21L, 23L, 24L, 25L, 26L, 27L, 28L, 29L,
30L, 31L, 32L, 33L, 34L, 35L, 36L, 37L, 38L, 39L, 40L, 41L, 42L,
43L, 44L, 45L, 46L, 47L, 48L, 49L, 50L, 51L, 52L, 53L, 54L, 55L,
56L, 57L, 58L, 59L, 60L, 61L, 62L, 63L, 64L, 65L, 66L, 67L, 68L,
69L, 70L, 71L, 72L, 73L, 74L, 75L, 76L, 77L, 78L, 79L, 80L, 81L,
82L, 83L, 84L, 85L, 86L, 87L, 88L, 89L, 90L, 91L, 92L, 93L, 94L,
95L, 96L, 97L, 98L, 99L, 100L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L,
9L, 10L, 11L, 12L, 13L, 14L, 15L, 16L, 17L, 18L, 19L, 20L, 21L,
22L, 23L, 24L, 25L, 26L, 27L, 28L, 29L, 30L, 31L, 32L, 33L, 34L,
35L, 36L, 37L, 38L, 39L, 40L, 41L, 42L, 43L, 44L, 45L, 46L, 47L,
48L, 49L, 50L, 51L, 52L, 53L, 54L, 55L, 56L, 57L, 58L, 59L, 60L,
61L, 62L, 63L, 64L, 65L, 66L, 67L, 68L, 69L, 70L, 71L, 72L, 73L,
74L, 75L, 76L, 77L, 78L, 79L, 80L, 81L, 82L, 83L, 84L, 85L, 86L,
87L, 88L, 89L, 90L, 91L, 92L, 93L, 94L, 95L, 96L, 97L, 98L, 99L,
100L, 1L, 2L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L,
15L, 16L, 17L, 18L, 19L, 20L, 21L, 22L, 23L, 24L, 25L, 26L, 27L,
28L, 29L, 30L, 31L, 32L, 33L, 34L, 35L, 36L, 37L, 38L, 39L, 40L,
41L, 42L, 43L, 44L, 45L, 46L, 47L, 48L, 49L, 50L, 51L, 52L, 53L,
54L, 55L, 56L, 57L, 58L, 59L, 60L, 61L, 62L, 63L, 64L, 65L, 66L,
67L, 68L, 69L, 70L, 71L, 72L, 73L, 74L, 75L, 76L, 77L, 78L, 79L,
80L, 81L, 82L, 83L, 84L, 85L, 86L, 87L, 88L, 89L, 90L, 91L, 92L,
93L, 94L, 95L, 96L, 97L, 98L, 99L, 100L, 1L, 2L, 3L, 4L, 5L,
6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L, 16L, 17L, 18L,
19L, 20L, 21L, 22L, 23L, 24L, 25L, 26L, 27L, 28L, 29L, 30L, 31L,
32L, 33L, 34L, 35L, 36L, 37L, 38L, 39L, 40L, 41L, 42L, 43L, 44L,
45L, 46L, 47L, 48L, 49L, 50L, 51L, 52L, 53L, 54L, 55L, 56L, 57L,
58L, 59L, 60L, 61L, 62L, 63L, 64L, 65L, 66L, 67L, 68L, 69L, 70L,
71L, 72L, 73L, 74L, 75L, 76L, 77L, 78L, 79L, 80L, 81L, 82L, 83L,
84L, 85L, 86L, 87L, 88L, 89L, 90L, 91L, 92L, 93L, 94L, 95L, 96L,
97L, 98L, 99L, 100L, 1L, 2L, 4L, 7L, 8L, 9L, 10L, 11L, 12L, 13L,
14L, 15L, 16L, 17L, 18L, 19L, 20L, 21L, 23L, 24L, 25L, 26L, 27L,
28L, 29L, 30L, 31L, 32L, 33L, 34L, 35L, 36L, 37L, 38L, 39L, 40L,
41L, 42L, 43L, 44L, 45L, 46L, 47L, 48L, 49L, 50L, 51L, 52L, 53L,
54L, 55L, 56L, 57L, 59L, 60L, 62L, 63L, 64L, 65L, 66L, 67L, 68L,
69L, 70L, 71L, 72L, 73L, 74L, 75L, 76L, 77L, 78L, 79L, 80L, 81L,
82L, 83L, 84L, 85L, 86L, 87L, 89L, 90L, 91L, 92L, 93L, 94L, 96L,
97L, 98L, 99L, 1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L, 9L, 10L, 11L,
12L, 13L, 14L, 15L, 16L, 17L, 18L, 19L, 20L, 21L, 22L, 23L, 24L,
25L, 26L, 27L, 28L, 29L, 30L, 31L, 32L, 33L, 34L, 35L, 36L, 37L,
38L, 39L, 40L, 41L, 42L, 43L, 44L, 45L, 46L, 47L, 48L, 49L, 50L,
51L, 52L, 53L, 54L, 55L, 56L, 57L, 58L, 59L, 60L, 61L, 62L, 63L,
64L, 65L, 66L, 67L, 68L, 69L, 70L, 71L, 72L, 73L, 74L, 75L, 76L,
77L, 78L, 79L, 80L, 81L, 82L, 83L, 84L, 85L, 86L, 87L, 88L, 89L,
90L, 91L, 92L, 93L, 94L, 95L, 96L, 97L, 98L, 99L, 100L, 1L, 2L,
4L, 7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L, 16L, 17L, 18L,
```

```
19L, 21L, 23L, 24L, 25L, 26L, 27L, 28L, 29L, 30L, 31L, 32L, 33L,
34L, 35L, 36L, 37L, 39L, 40L, 41L, 42L, 43L, 44L, 45L, 46L, 47L,
48L, 49L, 50L, 51L, 52L, 53L, 54L, 55L, 56L, 57L, 59L, 60L, 62L,
63L, 64L, 65L, 66L, 67L, 68L, 70L, 71L, 72L, 73L, 74L, 75L, 76L,
77L, 78L, 79L, 80L, 81L, 82L, 83L, 84L, 85L, 86L, 87L, 89L, 90L,
91L, 92L, 93L, 94L, 95L, 96L, 97L, 98L, 99L), race = c(0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1), sex = c(0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
    teen = c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1), N = c(1283, 332, 73, 134, 234, 151, 352, 60, 272, 620,
2003, 713, 1697, 764, 92, 448, 132, 1445, 556, 199, 80, 93,
751, 378, 951, 2781, 207, 389, 1426, 337, 500, 2006, 243,
2963, 434, 1727, 52, 69, 359, 115, 3115, 193, 942, 428, 929,
61, 352, 38, 1318, 250, 1684, 53, 556, 308, 724, 380, 278,
203, 156, 7792, 131, 269, 596, 664, 1545, 76, 2169, 968,
74, 239, 309, 70, 256, 1060, 129, 1435, 307, 628, 766, 1145,
558, 574, 172, 578, 439, 720, 90, 201, 28, 1931, 235, 7806,
58, 80, 297, 971, 621, 522, 367, 139, 289, 20, 142, 1, 2,
144, 154, 124, 122, 192, 96, 299, 53, 12, 53, 54, 222, 71,
4, 74, 224, 228, 337, 1758, 13, 13, 175, 23, 122, 1369, 336,
1056, 124, 360, 30, 8, 147, 79, 2054, 387, 254, 9, 46, 175,
210, 11, 244, 60, 299, 16, 146, 271, 40, 20, 8, 1, 116, 3693,
2, 69, 169, 369, 391, 109, 603, 248, 19, 170, 87, 33, 110,
696, 7, 110, 180, 993, 150, 231, 71, 200, 194, 95, 12, 27,
54, 19, 9, 275, 267, 2859, 105, 72, 5, 466, 31, 366, 15,
2, 1308, 339, 86, 142, 225, 136, 343, 55, 242, 625, 2127,
765, 1750, 737, 79, 543, 151, 1576, 557, 224, 74, 71, 789,
416, 1058, 2892, 208, 432, 1532, 369, 512, 2167, 281, 3076,
467, 1926, 64, 79, 434, 127, 3351, 229, 984, 518, 1016, 65,
406, 38, 1345, 235, 1812, 40, 558, 369, 736, 458, 278, 186,
118, 8165, 149, 245, 691, 671, 1552, 78, 2395, 1030, 82,
286, 342, 72, 318, 1128, 155, 1452, 361, 641, 755, 1150,
583, 537, 215, 534, 410, 754, 91, 213, 31, 1981, 284, 8291,
58, 53, 322, 1030, 680, 511, 403, 163, 264, 17, 126, 3, 1,
139, 116, 137, 129, 202, 98, 320, 65, 15, 46, 59, 234, 80,
8, 56, 2, 238, 224, 376, 1819, 10, 10, 165, 22, 161, 1466,
346, 1054, 150, 378, 34, 11, 169, 89, 2069, 373, 308, 12,
34, 181, 253, 9, 242, 66, 295, 34, 160, 291, 48, 17, 8, 5,
119, 3960, 1, 53, 153, 362, 378, 128, 618, 246, 19, 198,
73, 31, 102, 742, 6, 106, 159, 1005, 158, 249, 81, 181, 217,
79, 19, 41, 56, 12, 9, 299, 277, 2787, 125, 69, 9, 468, 33,
356, 15, 3, 136, 45, 11, 14, 33, 21, 43, 4, 43, 100, 209,
115, 172, 112, 6, 64, 19, 174, 64, 45, 6, 2, 112, 61, 112,
301, 22, 36, 187, 37, 67, 163, 42, 328, 56, 245, 5, 14, 33,
22, 256, 33, 109, 86, 105, 9, 38, 4, 126, 30, 124, 10, 87,
42, 100, 57, 40, 32, 21, 430, 18, 37, 62, 59, 88, 10, 284,
48, 21, 28, 41, 9, 19, 66, 15, 219, 53, 111, 116, 152, 107,
93, 31, 63, 52, 98, 15, 37, 1, 142, 35, 373, 10, 11, 39,
122, 89, 68, 49, 12, 65, 4, 25, 39, 34, 23, 33, 49, 32, 55,
15, 4, 12, 12, 62, 14, 2, 12, 79, 69, 83, 325, 6, 1, 41,
3, 44, 168, 112, 208, 40, 80, 10, 3, 31, 22, 305, 102, 76,
4, 7, 54, 51, 4, 68, 13, 52, 8, 38, 62, 9, 1, 1, 38, 608,
12, 40, 94, 104, 32, 90, 20, 5, 43, 17, 11, 18, 158, 1, 19,
59, 248, 37, 84, 23, 45, 53, 22, 1, 5, 16, 3, 60, 84, 292,
24, 28, 128, 4, 100, 4, 191, 42, 8, 21, 38, 18, 45, 5, 36,
104, 246, 122, 157, 100, 5, 62, 21, 206, 49, 39, 9, 9, 129,
60, 120, 305, 26, 33, 241, 37, 82, 162, 26, 293, 37, 257,
11, 17, 41, 21, 256, 35, 122, 65, 109, 8, 38, 4, 150, 33,
177, 8, 102, 55, 98, 66, 46, 22, 18, 472, 21, 57, 87, 69,
```

```
113, 11, 290, 51, 13, 28, 33, 9, 27, 90, 17, 243, 56, 111,
130, 197, 103, 87, 27, 72, 64, 122, 19, 37, 4, 135, 44, 399,
7, 8, 27, 132, 106, 58, 55, 35, 70, 6, 42, 50, 41, 33, 39,
51, 30, 52, 12, 4, 11, 10, 57, 17, 16, 76, 61, 82, 314, 5,
4, 54, 2, 35, 200, 108, 240, 42, 83, 12, 32, 29, 333, 110,
69, 1, 8, 53, 67, 4, 71, 18, 74, 5, 28, 96, 14, 2, 3, 36,
597, 11, 34, 86, 85, 32, 97, 25, 35, 15, 8, 30, 158, 1, 19,
50, 264, 35, 68, 23, 52, 59, 28, 3, 12, 21, 4, 67, 99, 282,
27, 26, 2, 148, 8, 101, 2), Y = c(123, 31, 8, 8, 18, 11,
22, 5, 33, 60, 142, 59, 147, 66, 7, 32, 10, 103, 52, 32,
8, 6, 66, 36, 73, 220, 20, 26, 122, 27, 34, 137, 21, 240,
27, 167, 9, 5, 28, 6, 220, 10, 70, 49, 80, 6, 38, 1, 102,
21, 115, 5, 51, 30, 60, 35, 19, 15, 11, 549, 11, 28, 44,
48, 107, 9, 171, 79, 6, 21, 29, 2, 22, 94, 12, 120, 24, 50,
77, 113, 47, 48, 15, 44, 44, 54, 6, 12, 3, 125, 17, 521,
2, 3, 21, 83, 51, 43, 30, 9, 52, 2, 20, 0, 0, 26, 24, 17,
19, 34, 11, 51, 2, 1, 7, 12, 29, 7, 1, 8, 28, 40, 52, 263,
1, 1, 29, 5, 20, 205, 61, 171, 18, 48, 2, 0, 12, 14, 269,
77, 32, 1, 10, 33, 28, 2, 42, 13, 38, 2, 22, 42, 4, 2, 0,
0, 23, 491, 0, 11, 17, 48, 69, 19, 62, 36, 4, 23, 15, 5,
18, 128, 1, 17, 26, 148, 25, 48, 14, 25, 27, 15, 1, 8, 7,
1, 1, 36, 47, 332, 18, 8, 0, 63, 4, 64, 6, 0, 97, 32, 7,
14, 18, 12, 14, 6, 17, 70, 150, 60, 109, 55, 6, 39, 8, 104,
43, 15, 3, 6, 61, 33, 70, 191, 18, 29, 134, 23, 38, 117,
24, 211, 22, 134, 7, 4, 28, 5, 201, 21, 67, 48, 69, 5, 29,
2, 83, 19, 132, 5, 42, 23, 43, 41, 21, 12, 6, 455, 8, 22,
44, 42, 96, 7, 137, 67, 8, 16, 21, 7, 19, 76, 12, 103, 23,
46, 61, 86, 42, 42, 18, 34, 30, 52, 7, 19, 4, 127, 19, 494,
3, 2, 22, 52, 56, 36, 24, 19, 23, 2, 12, 1, 0, 22, 9, 26,
17, 23, 13, 41, 10, 0, 6, 7, 21, 13, 1, 6, 0, 28, 30, 41,
237, 0, 1, 17, 3, 17, 172, 45, 154, 21, 42, 9, 0, 26, 18,
265, 68, 42, 2, 4, 21, 33, 1, 24, 4, 33, 6, 25, 40, 4, 3,
0, 0, 19, 437, 0, 7, 12, 36, 51, 26, 63, 25, 1, 16, 10, 4,
14, 113, 2, 8, 25, 113, 17, 42, 10, 29, 24, 10, 1, 4, 3,
1, 1, 34, 42, 276, 18, 9, 0, 55, 4, 46, 2, 0, 16, 3, 1, 0,
4, 4, 1, 0, 6, 9, 25, 14, 9, 19, 1, 7, 3, 13, 5, 3, 1, 0,
16, 4, 9, 32, 6, 5, 13, 5, 8, 13, 10, 25, 2, 23, 0, 2, 4,
3, 25, 2, 11, 11, 11, 0, 3, 1, 11, 2, 13, 3, 3, 2, 9, 3,
2, 7, 5, 37, 2, 2, 6, 3, 7, 1, 20, 7, 1, 0, 2, 3, 2, 6, 1,
20, 4, 15, 13, 18, 14, 8, 2, 8, 10, 14, 1, 4, 0, 15, 5, 29,
0, 1, 4, 16, 8, 3, 6, 2, 10, 0, 1, 4, 7, 0, 3, 7, 5, 13,
2, 1, 3, 2, 7, 3, 0, 3, 13, 19, 17, 58, 1, 0, 8, 1, 6, 25,
16, 30, 9, 8, 1, 0, 1, 5, 51, 27, 12, 0, 1, 8, 9, 0, 12,
0, 14, 1, 3, 6, 1, 0, 0, 5, 95, 1, 7, 11, 17, 4, 13, 4, 1,
2, 3, 5, 4, 29, 0, 4, 12, 40, 4, 13, 4, 7, 6, 5, 0, 0, 0,
0, 5, 16, 47, 1, 2, 15, 0, 9, 1, 16, 6, 1, 0, 5, 0, 8, 0,
2, 3, 39, 6, 13, 7, 2, 6, 2, 20, 6, 4, 1, 0, 12, 5, 7, 24,
3, 3, 31, 6, 3, 13, 4, 23, 2, 18, 1, 1, 3, 3, 20, 5, 9, 11,
9, 1, 1, 2, 21, 3, 10, 1, 4, 2, 6, 5, 2, 3, 3, 30, 1, 4,
9, 9, 7, 0, 26, 1, 1, 0, 1, 1, 4, 6, 0, 25, 4, 13, 10, 22,
12, 5, 2, 9, 7, 12, 1, 1, 0, 10, 0, 26, 1, 0, 2, 8, 17, 2,
6, 5, 6, 0, 7, 3, 6, 4, 4, 12, 2, 8, 2, 0, 1, 2, 10, 1, 3,
9, 6, 8, 46, 0, 1, 4, 0, 5, 25, 16, 30, 5, 8, 3, 5, 4, 38,
22, 8, 0, 0, 8, 6, 0, 7, 1, 12, 0, 2, 10, 4, 0, 0, 7, 78,
4, 2, 11, 14, 4, 10, 7, 4, 4, 1, 3, 24, 1, 3, 8, 26, 5, 8,
2, 5, 11, 7, 0, 1, 0, 2, 9, 13, 34, 2, 2, 0, 23, 0, 13, 0
```

```
    )), .Names = c("county", "race", "sex", "teen", "N", "Y"),
row.names = c(NA,
-777L), class = "data.frame")
```