```
library(dplyr)
library(data.table)

# Main function applying the combat method without a reference batch
# Y: input data matrix (observations in rows, spots in columns)
# bx: batch indicator variable, coerced to factor
# mean.only: boolean, if TRUE the Location-only model is applied.
LS.NoRef = function(Y, bx, mean.only = FALSE){
  nObsPerBatch = summary(bx)
  nObs = sum(nObsPerBatch)

  B = model.matrix(~-1 + bx)
  lamb.hat = solve( t(B)%*% B, t(B) %*% Y)

  alpha_g = (nObsPerBatch/nObs) %*% lamb.hat
  siggsq  = t(t((Y - (B %*% lamb.hat))^2) %*% rep(1/nObs, nObs))

  Z = scale(Y, center = alpha_g, scale = sqrt(siggsq))

  lambda.hat = solve( t(B)%*% B, t(B) %*% Z) # unadjusted location
  sigma.hat = NULL
  for(i in 1:nlevels(bx)){
    sigma.hat = rbind(sigma.hat, apply(Z[bx == levels(bx)[i],,drop =
FALSE],2,var))
  } # unadjusted scale

  params = data.frame( bx = levels(bx),
                       bi = 1:nlevels(bx),
                       lambda.bar = rowMeans(lambda.hat),
                       t2 = apply(lambda.hat,1, var),
                       gamma = apply(sigma.hat, 1, aprior),
                       theta = apply(sigma.hat, 1, bprior),
                       n = summary(bx))

  # solving for batch effect
  post = params %>%do(getLSCorrection(., Z, lambda.hat, sigma.hat,
mean.only))
  Ystar = batchcorrect(Z, bx, post, alpha_g, siggsq)
}
# Main function applying the combat method with a reference batch
# Y: input data matrix (observations in rows, spots in columns)
# bx: batch indicator variable, coerced to factor
# refbatch: string identifying the reference batch
# mean.only: boolean, if TRUE the Location-only model is applied.
LS.Ref = function(Y, bx, refbatch, mean.only = FALSE){
  bx = relevel(factor(bx), ref = refbatch) # ref will be the first level of
bx
  lvbx = levels(bx)
  nObsPerBatch = summary(bx)
  nObs = sum(nObsPerBatch)

  # scaling based on ref.batch
  B = model.matrix(~bx)
  lamb.hat = solve( t(B)%*% B, t(B) %*% Y)
  alpha_g = lamb.hat[1,]
  siggsq  = t(t((Y[bx == lvbx[1], ] - (B[bx == lvbx[1],] %*% lamb.hat))^2)
%*% rep(1/nObsPerBatch[1], nObsPerBatch[1]))
  Z = scale(Y, center = alpha_g, scale = sqrt(siggsq))
```

```
  lambda.hat = solve( t(B)%*% B, t(B) %*% Z) # unadjusted location
  sigma.hat = NULL
  for(i in lvbx){
    sigma.hat = rbind(sigma.hat, apply(Z[bx == i,,drop = FALSE],2,var))
  } # unadjusted scale

  params = data.frame( bx = lvbx,
                       bi = 1:nlevels(bx),
                       lambda.bar = rowMeans(lambda.hat),
                       t2 = apply(lambda.hat,1, var),
                       gamma = apply(sigma.hat, 1, aprior),
                       theta = apply(sigma.hat, 1, bprior),
                       n = summary(bx)) %>%
  mutate(bx = relevel(bx, ref = lvbx[1])) # necessary to make sure ref
batch is level 1.

  #solving for barch effecy
   post = params %>%do(getLSCorrection(., Z, lambda.hat, sigma.hat,
mean.only))
  # make sure that location and scale for the ref batch is identical 0 and
1, respectively
  post$lambda.star[[1]] = matrix(nrow = 1, ncol = dim(Y)[2], data = 0)
  post$sigma.star[[1]]  = matrix(nrow = 1, ncol = dim(Y)[2], data = 1)
  Ystar = batchcorrect(Z, bx, post, alpha_g, siggsq)
  #Ystar[bx == lvbx[1],] = Y[bx == lvbx[1],] # Used in or4iginal ComBat
code, seems unnecessary
  return(Ystar)
}

# Helper functions
aprior = function(X) {
  m <- mean(X)
  s2 <- var(X)
  (2*s2 + m^2) / s2
}

bprior =  function(X){
  m <- mean(X)
  s2 <- var(X)
  (m*s2 + m^3) / s2
}

postmean =  function(g.hat,g.bar,n,d.star,t2){
  (t2*n*g.hat + d.star*g.bar) / (t2*n + d.star)
}

postvar = function(sum2,n,a,b){
  (.5*sum2 + b) / (n/2 + a - 1)
}

getLSCorrection = function(pardf, Z, lambda.hat, sigma.hat, mean.only =
FALSE){
  if(!mean.only){
    post = pardf %>% group_by(bx) %>% do(it.sol(., Z = Z[bx == .$bx,],
lambda.hat[.$bi,], sigma.hat[.$bi,]))
  } else {
    post = pardf %>% group_by(bx) %>% do({
```

```
      lambda.str= postmean(lambda.hat[.$bi,], .$lambda.bar, 1,1, .$t2)
      sigma.str = rep(1, length(lambda.str))
      result = data.table(lambda.star = list(lambda.str), sigma.star =
list(sigma.str))
    })
  }
  return(post)
}

it.sol = function(params, Z, lambda.hat, sigma.hat, conv = .0001)
{
  g.old = lambda.hat
  d.old = sigma.hat
  change <- 1
  count <- 0
  while(change>conv){
    g.new <- postmean(lambda.hat, params$lambda.bar, params$n, d.old,
params$t2)
    sum2 = colSums(scale(Z, center = g.new, scale = FALSE)^2)
    #sum2 <- rowSums((sdat - g.new %*% t(rep(1,ncol(sdat))))^2, na.rm=TRUE)
    d.new <- postvar(sum2, params$n, params$gamma, params$theta)
    change <- max(abs(g.new-g.old) / g.old, abs(d.new-d.old) / d.old)
    g.old <- g.new
    d.old <- d.new
    count <- count+1
  }
  cat("This batch took", count, "iterations until convergence\n")
  result = data.table(lambda.star = list(g.new), sigma.star = list(d.new))
}

batchcorrect = function(Z, bx,  post, lambda_g, sigmasq_g){
  Ystar = matrix(nrow = dim(Z)[1], ncol = dim(Z)[2])
  for (i in 1:dim(Z)[1]){
    bIdx = (1:nlevels(bx))[bx[i] == levels(bx)]
    zstar= (Z[i,] - post$lambda.star[[bIdx]])/sqrt(post$sigma.star[[bIdx]])
    Ystar[i,] = sqrt(sigmasq_g) * zstar + lambda_g
  }
  return(Ystar)
}
```