```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
# Code to accompany Jones & Waller: The Normal-Theory  #
# and Asymptotic Distribution-Free (ADF) Covariance    #
# Matrix of Standardized Regression Coefficients:      #
# Theoretical Extensions and Finite Sample Behavior    #
#                                                      #
# This function uses the delta method to construct     #
# normal-theory and ADF confidence intervals for       #
# standardized regression coefficients.                #
#                                                      #
# Arguments:                                           #
# X            - matrix of predictor scores            #
# y            - vector of criterion scores            #
# cov.x        - covariance matrix for predictors      #
# cov.xy       - vector of covariances between         #
#                predictors and criterion              #
# var.y        - criterion variance                    #
# Nobs         - number of observations                #
# alpha        - desired Type I error rate             #
#                default = .05                         #
# ADF          - Logical (TRUE/FALSE) to select ADF    #
#                confidence intervals - requires raw X #
#                and y; default = TRUE                 #
# digits       - number of significant digits to       #
#                print; default = 3                    #
#                                                      #
# This function accepts either (1) raw data, or (2)    #
# second-order moments (covariances) and sample size.  #
#                                                      #
# Output                                               #
# cov.mat  - normal-theory or ADF covariance matrix of #
#            standardized regression coefficients      #
# SEs      - normal-theory or ADF standard errors for  #
#            standardized regression coefficients      #
# alpha    - desired Type I error rate                 #
# CIs      - normal-theory or ADF confidence intervals #
#            for standardized regression coefficients  #
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
seBeta <- function(X = NULL, y = NULL,
                   cov.x = NULL, cov.xy = NULL,
                   var.y = NULL, Nobs = NULL,
                   alpha = .05, ADF = TRUE,
                   digits = 3) {
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~Internal
Functions~~~~~~~~~~~~~~~~~~~~~~~~~~#
# Computes the ADF covariance matrix of covariances
adfCOV <- function(X, y) {

  dev <- scale(cbind(X,y),scale=FALSE)
  nvar <- ncol(dev)
  N <- nrow(dev)

# number of unique elements in a covariance matrix
  ue <- nvar*(nvar + 1)/2
```

```
# container for indices
  s <- vector(length=ue, mode="character")
  z <- 0
  for(i in 1:nvar){
    for(j in i:nvar){
      z <- z + 1
      s[z] <- paste(i, j, sep="")
    }
  }

# compute all combinations of elements in s
  v <- expand.grid(s, s)

# concatenate index pairs
  V <- paste(v[,1], v[,2], sep="")

# separate indices into columns
  id.mat <- matrix(0,nrow=ue^2,4)
  for(i in 1:4) id.mat[,i] <- as.numeric(sapply(V, substr, i, i))

# fill a matrix, M, with sequence 1:ue^2 by row;
# use M to locate positions of indices in id.mat
  M <- matrix(1:ue^2, ue, ue, byrow=TRUE)

# select rows of index pairs
  r <- M[lower.tri(M, diag=TRUE)]
  ids <- id.mat[r,]
  adfCovMat <- matrix(0,ue,ue)
  covs <- matrix(0,nrow(ids),1)

# compute ADF covariance matrix using Browne (1984) Eqn 3.8
  for(i in 1:nrow(ids)) {
    w_ij <- cov(dev[,ids[i,1]],dev[,ids[i,2]])*((N-1)/N)
    w_kl <- cov(dev[,ids[i,3]],dev[,ids[i,4]])*((N-1)/N)
    w_ik <- cov(dev[,ids[i,1]],dev[,ids[i,3]])*((N-1)/N)
    w_jl <- cov(dev[,ids[i,2]],dev[,ids[i,4]])*((N-1)/N)

    w_ijkl <- (t(dev[,ids[i,1]]*dev[,ids[i,2]])%*%
               (dev[,ids[i,3]]*dev[,ids[i,4]])/N)

   covs[i] <- (N*(N-1)*(1/((N-2)*(N-3)))*(w_ijkl - w_ij*w_kl) -
                N*(1/((N-2)*(N-3)))*(w_ik*w_jl - (2/(N-1))*w_ij*w_kl))
  }

# create ADF Covariance Matrix
  adfCovMat[lower.tri(adfCovMat,diag=T)] <- covs
  vars <- diag(adfCovMat)
  adfCovMat <- adfCovMat + t(adfCovMat) - diag(vars)

  adfCovMat
  } #end adfCOV

# vech function
```

```
  vech <- function(x) t(x[!upper.tri(x)])

# Transition or Duplicator Matrix
  Dn <- function(x){
    mat <- diag(x)
    index <- seq(x * (x+1) / 2)
    mat[lower.tri(mat, TRUE)] <- index
    mat[upper.tri(mat)] <- t(mat)[upper.tri(mat)]
    outer(c(mat), index, function(x, y ) ifelse(x == y, 1, 0))
  }
#~~~~~~~~~~~~~~~~~~~~~~~~~~~End Define Internal Functions~~~~~~~~~~~~~~~#
#~~~~~~~~~~~~~~~~~~~~~~~~~~~ Error Checking ~~~~~~~~~~~~~~~~~~~~~~~~~~~#
  if(is.null(X) & !is.null(y))
    stop("\n y is not defined\n Need to specify both X and y\n")
  if(!is.null(X) & is.null(y))
    stop("\n X is not defined\n Need to specify both X and y\n")
  if(is.null(X) & is.null(y)) {
    if(is.null(cov.x) | is.null(cov.xy) | is.null(var.y) | is.null(Nobs))
      stop("\nYou need to specify covariances and sample size\n")
    scov <- rbind(cbind(cov.x, cov.xy), c(cov.xy, var.y))
    N <- Nobs
    p <- nrow(cov.x)

  } else {
    scov <- cov(cbind(X,y))
    N <- length(y)
    p <- ncol(X)
  }

  if(ADF) {
    cov.cov <- adfCOV(X,y)
  } else {
# create normal-theory covariance matrix of covariances
# See Browne (1984) Eqn 4.6
    Kp.lft <- solve(t(Dn(p + 1)) %*% Dn(p + 1)) %*% t(Dn(p + 1))
    cov.cov <- 2 * Kp.lft %*% (scov %x% scov) %*% t(Kp.lft)
   }

  param <- c(vech(scov))
  ncovs <- length(param)

# find vector element numbers for variances of X
  v.x.pl <- c(1, rep(0, p - 1))
  for(i in 2:p) v.x.pl[i] <- v.x.pl[i - 1] + p - (i - 2)

# store covariances and variances
  cx  <- scov[1:p, 1:p]
  cxy <- scov[1:p, p+1]
  vy  <- scov[p+1, p+1]
  sx <- sqrt(diag(cx))
  sy <- sqrt(vy)
  bu <- solve(cx) %*% cxy
  ncx <- length(vech(cx))
```

```
# compute derivatives of standardized regression
# coefficients using Yuan and Chan (2011) Equation 13
  db <- matrix(0, p, ncovs)
  V <- matrix(0, p, ncx)
  V[as.matrix(cbind(1:p, v.x.pl))] <- 1

  db[, 1:ncx] <- (diag(c(solve(diag(2 * sx * sy)) %*% bu)) %*% V -
                  diag(sx / sy) %*% (t(bu) %x% solve(cx)) %*% Dn(p))

  db[, (ncx+1):(ncx+p)] <- diag(sx / sy) %*% solve(cx)
  db[,ncovs] <- -diag(sx / (2 * sy^3)) %*% bu

# re-order derivatives
  cx.nms <- matrix(0, p, p)
  cxy.nms <- c(rep(0, p), "var_y")

  for(i in 1:p) for(j in 1:p) cx.nms[i, j] <- paste("cov_x", i, "x", j,
sep='')
  for(i in 1:p) cxy.nms[i] <- paste("cov_x", i, "y", sep='')

  old.ord <- c(vech(cx.nms), cxy.nms)
  new.ord <- vech(rbind(cbind(cx.nms, cxy.nms[1:p]), c(cxy.nms)))

  db <- db[, match(new.ord, old.ord)]

# compute covariance matrix of standardized
# regression coefficients using the Delta Method
  DEL.cmat <- db %*% cov.cov %*% t(db) / N
  b.nms <- NULL

  for(i in 1:p) b.nms[i] <- paste("beta_", i, sep='')
  rownames(DEL.cmat) <- colnames(DEL.cmat) <- b.nms

# compute standard errors and confidence intervals
  DELse <- sqrt(diag(DEL.cmat))
  CIs <- as.data.frame(matrix(0, p, 3))
  colnames(CIs) <- c("lbound", "estimate", "ubound")
  for(i in 1:p) rownames(CIs)[i] <- paste("beta_", i, sep='')

  tc <- qt(alpha / 2, N - p - 1, lower = F)
  beta <- diag(sx) %*% bu * sy^-1
  for(i in 1:p) {
    CIs[i,] <- c(beta[i] - tc * DELse[i], beta[i], beta[i] + tc *
DELse[i])
  }
  cat("\n", 100 * (1 - alpha),
      "% CIs for Standardized Regression Coefficients:\n\n", sep='')
  print(round(CIs,digits))

  invisible(list(cov.mat=DEL.cmat,SEs=DELse,alpha=alpha,CIs=CIs))
}
```