# Appendix: Codes

*Chi Po Choi, Amy Kim*

*10 June, 2015*

## HyperCube.R

```
#' Hypercube Estimator Fit
#'
#' The package "HyperCube" provides functions and methods for fitting linear model
#' with hypercube estimator. This package is an implementation of the hypercube
#' estimators introduced in [Beran, Rudolf. "Hypercube estimators: Penalized least
#' squares, submodel selection, and numerical stability." Computational Statistics &
#' Data Analysis 71 (2014): 654-666."]
#'
#' @author
#' Chi Po Choi \email{cpchoi@@ucdavis.edu},
#' Amy T. Kim \email{atykimi@@ucdavis.edu}
#'
#' Maintainer: Chi Po Choi \email{cpchoi@@ucdavis.edu},
#'             Amy T. Kim \email{atykimi@@ucdavis.edu}
#'
#' @aliases HyperCube HyperCube-package
#' @importFrom MASS ginv
#' @importFrom Matrix rankMatrix
#' @details
#' \tabular{ll}{
#' Package: \tab HyperCube\cr
#' Type: \tab Package\cr
#' Version: \tab 0.1-1\cr
#' Date: \tab 2015-05-08\cr
#' License: \tab GPL\cr
#' }
#'
#' @docType package
#' @name HyperCube
NULL
#> NULL
```

## hypercubeEst.R

```
#' Estimation on linear Mode for Hypercube
#'
#' This function calculates the hypercube estimates.
#' Users should use \code{\link{hypercube}} which is a better user interface.
#'
#' @param X design matrix, or data-incidence matrix (as described Beran (2014))
```

```
#' @param y response, or observation (as described Beran (2014))
#' @param V sysmmetric matrix whose eigenvalues all lie in [0,1]
#'            (as in equation (1.3) in Beran (2014))
#' @param ... other optional arguments
#' @return The function \code{hypercubeEst} returns a list containing the following:
#' \describe{
#'   \item{coefficients}{Estimated coefficents \eqn{\beta} under hypercube estimator}
#'   \item{fitted.values}{Fitted values (\eqn{\eta}) under hypercube estimator}
#'   \item{residuals}{residuals, \eqn{y - \eta}}
#'   \item{V}{Symmtric matrix with all eigenvalues lie in [0,1].
#'   The matrix V used the hypercube estimator.
#'   (as in equation (1.3) in Beran (2014))}
#' }
#'
#' @references Beran, Rudolf. "Hypercube estimators: Penalized least squares,
#'              submodel selection, and numerical stability."
#'              Computational Statistics & Data Analysis 71 (2014): 654-666.
#' @seealso \code{\link{hypercube}} is the user interfaces of hypercube estimators.
#' The matrix V can be constructed using the function \code{\link{projectCreate}}.
#'
#' @examples
#' ## The use of the function \code{hypercubeEst}.
#' mf <- model.frame(weight ~ mother:infant -1, data = litter)
#' X <- model.matrix(attr(mf, "terms"), mf)
#' y <- model.response(mf)
#' litter.proj <- projectCreate( ~ mother:infant -1, data = litter)
#' V <- projectWeight(litter.proj, weights = c(1,1,1,0))
#' hcmod <- hypercubeEst(X, y, V)
#' hcmod$coefficients
#'
#' @export
hypercubeEst <-
function(X, y, V, ...) {
  args <- list(...)

  X <- as.matrix(X)
  y <- as.numeric(y)
  #rX <- Matrix::rankMatrix(X)
  A <- hypercubeOp(X, V)
  #rA <- Matrix::rankMatrix(A)
  etahat <- A %*% y
  coef <- MASS::ginv(X) %*% etahat
  rownames(coef) <- colnames(X)
  residuals <- y - etahat

  list(coefficients = coef, fitted.values = etahat,
       residuals = residuals,
       V = V
       )
}

#' Hypercube Estimator Fits
#'
```

```
#' @description
#' The user interface for fitting linear model with Hypercbue Estimator.
#'
#' @note This package is still under development. For data with degree of freedom equal
#' to zero, it may not be handled well. It is due the fact that it is not obvious
#' to estimate the variance of the random error.
#' Users need to provide estimated variance of random error in such cases.
#'
#' @examples
#' ## Example 1 in Beran (2014)
#' ## Fitting Canadian earning data with Hypercube Estimator
#'
#' # The number of age, p, in Example 1 in Beran (2014).
#' p <- length(unique(canadian.earnings[,1]))
#' # D_5 as in equation (3.10) in Beran (2014)
#' D <- diffMatrix(p, 5)
#' # The parametor nu in equation (3.11) in Beran (2014)
#' nu <- 100
#' # The matrix W in equation (3.11) in Beran (2014)
#' W <- nu * t(D) %*% D
#' # Convert W to V, as described in (1.6) in Beran (2014)
#' V <- plsW2V(W)
#' # The variable age should be considered as a factor
#' canadian.earnings[,"age"] <- factor(canadian.earnings[,"age"])
#' # Hyperpercube Estimator Fit
#' hcmod <- hypercube( log.income ~ age -1, data=canadian.earnings, V)
#'
#' # Plot of data
#' plot(as.numeric(as.character(canadian.earnings$age)),
#'      canadian.earnings$log.income,
#'      xlab = "age", ylab = "log(income)")
#' # Plot of fitted line
#' lines(levels(canadian.earnings$age), hcmod$coefficients)
#'
#'
#' ## Example 2 in Beran (2014)
#' ## Fitting rat litter data
#'
#' # Projection matrices as decribed in equation (5.4) in Beran (2014)
#' litter.proj <- projectCreate( ~ mother:infant -1, data = litter)
#' # If only additive effect is consider,
#' # take V = P1 + P2 + P3 (notation in equation (5.4) in Beran (2014))
#' component <- cbind(c(0,0), c(1,0), c(0,1))
#' V <- projectWeight(litter.proj, component = component)
#' # Hypercube Estimator Fit
#' hcmod <- hypercube( weight ~ mother:infant -1, data = litter, V)
#'
#' # Estimated Risk
#' summary(hcmod)
#'
#' ## Hypercube Estimator with optimal risk
#' ##
#' hcmodopt <- hypercubeOptimization( weight ~ mother:infant -1,
```

```r
#'                                     data = litter
#'                                     )
#' # The optimal projection coefficient which minimizes the risk.
#' hcmodopt$projcoef
#'
#' # The minimum risk
#' hcmodopt$estrisk
#'
#' # The Hypercube Estimator fit with the V of the optimal projection.
#' summary(hcmodopt$$est)
#'
#' @export
hypercube <- function(...) UseMethod("hypercube")

#' @describeIn hypercube Default method for \code{hypercube}
#' @param X design matrix
#' @param y observation
#' @param V sysmmetric matrix whose eigenvalues all lie in [0,1]
#' @param ...  other optional arguments
#'
#'
#' @references Beran, Rudolf. "Hypercube estimators: Penalized least squares,
#'              submodel selection, and numerical stability."
#'              Computational Statistics & Data Analysis 71 (2014): 654-666.
#' @seealso \code{\link{hypercubeOptimization}}
#' @export
#' @method hypercube default
hypercube.default <- function(X, y, V, ...)
{
  X <- as.matrix(X)
  y <- as.numeric(y)
  est <- hypercubeEst(X, y, V)
  est$call <- match.call(expand.dots = TRUE)
  class(est) <- "hypercube"
  est
}

#' @describeIn hypercube Formula method for \code{hypercube}
#'
#' @param formula formula to get estimate
#' @param data data you want to analysis
#' @return A object of the class "\code{hypercube}" containing the following:
#' \describe{
#'   \item{coefficients}{Estimated coefficents \eqn{\hat{\beta}} under hypercube estimator}
#'   \item{fitted.values}{Fitted values (\eqn{\hat{\eta}})  under hypercube estimator}
#'   \item{residuals}{residuals, \eqn{y - \hat{\eta}}}
#'   \item{estsigma2}{Estimated variance of \eqn{\epsilon}.
#'    If the rank of the design matrix X is greater than the number of observation,
#'    the variance is estimated using least square regression.
#'    If the rank is equal to the number of observation,
#'    and if the model has more than one factor,
#'    the variance is estimated using additive submodel fit.
#'    If neither of the above cases,
```

```
#'    users need to provide their own estimated variance.}
#'   \item{estrisk}{Esitmated risk, which involved the estimated variance.}
#'   \item{V}{Symmtric matrix with all eigenvalues lie in [0,1].
#'   The matrix V used the hypercube estimator. }
#'   \item{modelframe}{The modelframe constructed from the argument formula and data.}
#' }
#' @export
#' @method hypercube formula
hypercube.formula <- function(formula, data, V, ...)
{
  mf <- model.frame(formula=formula, data=data)
  X <- model.matrix(attr(mf, "terms"), data = mf)
  y <- model.response(mf)
  A <- hypercubeOp(X, V)
  est <- hypercubeEst(X, y, V)

  if("sigma2" %in% names(args)) {
    est$estsigma2 <- args[["sigma2"]]
  } else {
    est$estsigma2 <- estSigma(mf)$sigma
  }
  est$estrisk <- estRisk(X, y, A, est$estsigma2)
  est$modelframe <- mf
  est$formula <- formula
  est$call <- match.call(expand.dots = TRUE)
  class(est) <- "hypercube"
  est
}


#' Hypercube Operator
#'
#' @param X design matrix
#' @param V sysmmetric matrix whose eigenvalues all lie in [0,1]
#'
#' @references Beran, Rudolf. "Hypercube estimators: Penalized least squares,
#'              submodel selection, and numerical stability."
#'              Computational Statistics & Data Analysis 71 (2014): 654-666.
#' @export
hypercubeOp <-
function(X, V) {
  # Check V is symmetric
  VXt <- V %*% t(X)
  t(VXt) %*% MASS::ginv( VXt %*% t(VXt) + diag(dim(X)[2]) - V %*% V) %*% VXt
}

#' Estimate Risk
#'
#' @param X design matrix
#' @param y observation
#' @param A hypercuber operator
#' @param estsig estimated variance
#'
```

```r
#' @return The estimated risk
#'
#' @references Beran, Rudolf. "Hypercube estimators: Penalized least squares,
#'              submodel selection, and numerical stability."
#'              Computational Statistics & Data Analysis 71 (2014): 654-666.
#' @export
estRisk <-
function(X, y, A, estsig) {
  p <- dim(X)[2]
  n <- length(y)
  (l2sq(y - A %*% y) + ( 2 * tr(A) - n) * estsig)/p
}

#' Estimate Variance
#'
#' @param mf model frame
#'
#' @return The estimated variance
#'
#' @references Beran, Rudolf. "Hypercube estimators: Penalized least squares,
#'              submodel selection, and numerical stability."
#'              Computational Statistics & Data Analysis 71 (2014): 654-666.
#' @export
estSigma <-
function(mf) {
  mt <- attr(mf, "terms")
  X <- model.matrix(mt, data = mf)
  y <- model.response(mf)

  df <- nrow(X) - Matrix::rankMatrix(X)[1]

  if(df > 0) {
    # If df > 0, then use least square fit
    eta <- X %*% MASS::ginv(X) %*% y
    sigma2 <- l2sq(y - eta)/df
    return(list(sigma2 = sigma2, eta = eta))
  } else {
    # If df = 0, and more then one factor, then use submodel fit
    nvar <- length(all.vars(mt))-1
    if(nvar>1) {
      proj.formula <- reformulate(attr(mt,"term.labels"),
                                  response = NULL, intercept = FALSE)
      proj <- projectCreate(proj.formula, mf)
      component <- cbind(rep(0, nvar), diag(nvar))
      weights = rep(1, nvar+1)
      P <- projectWeight(proj, component, weights)
      XP <- X %*% P
      eta <- X %*% MASS::ginv(XP) %*% y
      rXP <- Matrix::rankMatrix(XP)[1]
      sigma2 <- l2sq(y - eta)/(nrow(X) - rXP)
      return(list(sigma2 = sigma2, eta = eta))
    } else {
      # At this moment, we cannot handle the case that df = 0 and only one factor.
```

```
      # Users need to provide estimated variance.
      stop("Please provide estimated sigma2.")
    }
  }
}


#' Hypercube Optimization
#'
#' @description Find the projection coefficients which minimizing the esitmated risk
#'
#' @param formula formula
#' @param data data
#' @param sigma estimated variance
#' @return A list containing the following:
#' \describe{
#'   \item{est}{a object of the classs "\code{hypercube}".}
#'   \item{projcoef}{optimal coefficients for the projections.}
#'   \item{estrisk}{the minimum estimated risk.}
#' }
#'
#' @references Beran, Rudolf. "Hypercube estimators: Penalized least squares,
#'              submodel selection, and numerical stability."
#'              Computational Statistics & Data Analysis 71 (2014): 654-666.
#' @seealso \code{\link{hypercube}}
#' @examples
#' hcmodopt <- hypercubeOptimization( weight ~ mother:infant -1, data = litter)
#' hcmodopt$projcoef #projction coefficients
#' hcmodopt$estrisk #estimated risk
#'
#' @export
hypercubeOptimization <-
function(formula, data, sigma = NULL) {
  mf <- model.frame(formula=formula, data=data)
  X <- model.matrix(attr(mf, "terms"), data = mf)
  y <- model.response(mf)

  mt <- attr(mf, "terms")
  proj.formula <- reformulate(attr(mt,"term.labels"),
                              response = NULL, intercept = FALSE)

  proj <- projectCreate(proj.formula, data)
  proj.fun <- projectFun(proj)
  nproj <- length(proj)

  if(is.null(sigma)) sigma <- estSigma(mf)$sigma2

  projOptFun <- function(weights, estsigma) {
    proj <- proj.fun(weights)
    estRisk(X, y, hypercubeOp(X, proj), estsigma)
  }

  opt.ans <- optim(rep(0.5, nproj),
```

```
                    projOptFun,
                    estsigma = sigma,
                    method = "L-BFGS-B",
                    lower = rep(0, nproj),
                    upper = rep(1, nproj)
                    )
  V <- proj.fun(opt.ans$par)
  est <- hypercube(formula, data, V)

  ans <- list( est = est, projcoef = opt.ans$par, estrisk = opt.ans$value)
  ans
}




#' Print methods for hypercube
#'
#' @param x an object of class "\code{hypercube}"
#' @param ... other arguments
#'
#' @export
#' @method print hypercube
print.hypercube <- function(x, ...)
{
  cat("Call:\n")
  print(x$call)
  cat("\nCoefficients:\n")
  print(x$coefficients)
}


#' Predict method for Hypercube Estimator Fits
#'
#' @param object an object of the class "\code{hypercube}"
#' @param newdata new data
#' @param ... other arguments
#'
#' @export
#' @method predict hypercube
predict.hypercube <-
function(object, newdata=NULL, ...) {
  if(is.null(newdata))
    y <- fitted(object)
  else{
    if(!is.null(object$formula)){
      ## model has been fitted using formula interface
      x <- model.matrix(object$formula, newdata)
    }
    else{
      x <- newdata
    }
    y <- as.vector(x %*% coef(object))
```

```
  }
  y
}

#' Summary method for Hypercube Estimator Fits
#'
#' @param object an object of class "\code{hypercube}"
#' @param ... other arguments
#'
#' @export
#' @method summary hypercube
summary.hypercube <- function(object, ...) {
  X <- model.matrix(attr(object$modelframe, "terms"), object$modelframe)
  y <- model.response(object$modelframe)
  n <- nrow(X)
  r <- Matrix::rankMatrix(X)[1]
  df <- n - r

  if(df > 0) {
    estrisk <- object$estrisk
    fullestrisk <- estRisk(X, y, hypercubeOp(X, diag(ncol(X))), object$estsigma2)
  } else {
    estrisk <- object$estrisk
    fullestrisk <- NA
  }

  ans <- list(call = object$call,
              modelframe = object$modelframe,
              coefficients = object$coefficients,
              fitted.values = object$fitted.values,
              estrisk = estrisk,
              fullestrisk = fullestrisk
              )
  class(ans) <- "summary.hypercube"
  ans
}
#' Print methods for summary.hypercube
#'
#' @param x an object of class "\code{summary.hypercube}"
#' @param ... other arguments
#'
#' @export
#' @method print summary.hypercube
print.summary.hypercube <- function(x, ...)
{
  cat("Call:\n")
  print(x$call)
  cat("\n")
  cat(paste("The estimated risk of hypercube estimation:", x$estrisk))
  cat("\n")
  if(!is.na(x$fullestrisk)) {
    cat(paste("The estimated risk of least square estimation: ", x$fullestrisk))
    cat("\n")
```

```
  }
}
```

## projection.R

```
# As described in the example 2 in article Beran (2014),
# The ANOVA projections are constructed from
# the matrix "J" and "H" (same notation in the Beran (2014)).
# This function makes "J" and "H".
# The parameter "n" is the number of level of the factor
# of the variable in data set.
projectElement <- function(n){
  u <- rep(1,n)/sqrt(n)
  J <- outer(u,u)
  H <- diag(n) - J
  list(J=J,H=H)
}

# Generate all the possible permutations for the projection matrices,
# as described in the equation (5.4) in Beran (2014).
# the parameter "variables": the names of all covariate in the data set.
# the parameter "JH.flag": If TRUE, represent the permutation by "J" and "H",
#                          If FALSE, J is 0, H is 1.
projectPerm <- function(variables, JH.flag = TRUE) {
  flist <- list()
  for(k in variables) flist[[k]] <- if(JH.flag) c("J", "H") else 0:1
  fperm <- t(as.matrix(expand.grid(flist, stringsAsFactors = FALSE)))
  fperm
}

# Contrust the names of the projection matrix, given the components
# as described in the equation (5.4) in Beran (2014).
# For example, in (5.4) in Beran (2014),
# the projection matrix "P_1" is corresponding to the component "c(0,0)";
# the projection matrix "P_2" is corresponding to the component "c(1,0)";
# the projection matrix "P_3" is corresponding to the component "c(0,1)";
# the projection matrix "P_4" is corresponding to the component "c(1,1)";
#
# We can get all the names of all components at once.
# projectName(variables, c(0,0,1,0,0,1,1,1))
# Or,
# component <- cbind(c(0,0), c(1,0), c(0,1), c(1,1))
# projectName(variables, component)
projectName <-
function(variables, component) {
  fname <- variables
  nname <- length(fname)
  if(length(component) %% nname != 0)
    stop("Incorrect component argument.")
  if(is.null(dim(component))) {
    component <- matrix(component, nrow = nname)
```

```
  } else if (dim(component)[1] != nname) {
    stop("Incorrect component argument.")
  }
  vname <- apply(component, 2, function(v) paste0(fname, v, collapse=":"))
  vname
}


#' Functions for object "\code{projection.hypercube}"
#'
#' The standard ANOVA projections as described in equation (5.4) in Beran (2014).
#'
#' @details
#' The standard ANOVA projections as described in equation (5.4) in Beran (2014).
#' The set of projections are sysmmetric, idempotent, mutually orthogonal matrix.
#' The linear combinations of the projection matrices are used as the matrix V (equation (1.3)
#' in Beran (2014)) of the hypercube estimators.
#' Please provide the formula in the form, for example in the data "litter", "~ mother:infant -1"
#' No responese and No intercept. Only the interaction of factors.
#'
#' @param formula input formula without response variable. For example, "~ mother:infant -1".
#' @param data input data
#' @return The function \code{projectCreate} creates a object of the "\code{projection.hyercube}",
#' which is a list containing all the projection matrices.
#' Also, the object "\code{projection.hyercube}" contains the following attributes:
#' \describe{
#'   \item{variables}{The names of the variables in the formula}
#'   \item{nlevels}{The numbers of levels of the factor in each variable}
#'   \item{dims}{The dimension of the projection matrices}
#'   \item{component}{The names of the projection matrices stored in the objects}
#' }
#'
#' @examples
#' proj <- projectCreate( ~ mother:infant -1, data = litter)
#'
#' ## proj contains the projection matrices :
#' ## proj[["mother0:infant0"]], proj[["mother1:infant0"]], ...
#' ## To see all the names of the projection matrices:
#' attr(proj, "component")
#'
#' ## If only the additive effects are needed,
#' ## i.e. "mother0:infant0", "mother1:infant0" and "mother0:infant1",
#' component <- cbind(c(0,0), c(1,0), c(0,1))
#' proj.sub <- projectSub(proj, component)
#' attr(proj.sub, "component")
#'
#' ## If we want
#' ## P = 1 * proj[["mother0:infant0"]] + 0.5 * proj[["mother1:infant0"]]
#' component <- cbind(c(0,0), c(1,0), c(0,1))
#' weights <- c(1, 0.5, 0)
#' proj.weights <- projectWeight(proj, component, weights)
#'
#' ## Create a function for more weighted projection matrices.
```

```
#' component <- cbind(c(0,0), c(1,0), c(0,1))
#' proj.fun <- projectFun(proj, component)
#'
#' ## Same as proj.weights in above example
#' weights <- c(1, 0.5, 0)
#' proj.fun(weights)
#'
#' ## Use the projection matrices for hypercube estimator
#' V <- proj.fun(weights)
#' hcmod <- hypercube(weight ~ mother:infant -1, data = litter, V = V)
#' summary(hcmod)
#'
#'
#' @references Beran, Rudolf. "Hypercube estimators:
#'              Penalized least squares, submodel selection, and numerical stability.
#'              " Computational Statistics & Data Analysis 71 (2014): 654-666.
#' @export
projectCreate <-
function(formula, data) {
  m <- model.frame(formula, data)
  fname <- names(m)
  fn <- length(fname)
  fnlevel <- sapply(fname, function(k) length(levels(m[,k])))
  fperm <- projectPerm(fname, JH.flag = TRUE)

  component <- array(,dim(fperm)[2])
  proj <- list()
  for(k in 1:dim(fperm)[2]) {
    v <- fperm[,k]
    vname <- paste0(fname,
                    sapply(v, function(k) switch(k, J = 0, H = 1)),
                    collapse=":")
    projtemp <- matrix(1,1,1)
    for(j in fname) {
      projtemp <- kronecker(projectElement(fnlevel[j])[[ v[j] ]], projtemp)
    }
    component[k] <- vname
    proj[[vname]] <- projtemp
  }
  attr(proj, "variables") <- fname
  attr(proj, "nlevels") <- fnlevel
  attr(proj, "dims") <- dim(projtemp)
  attr(proj, "component") <- component
  class(proj) <- c("projection.hypercube", class(proj))
  proj
}

#' @describeIn projectCreate
#' Subset of object "\code{projection.hypercube}"
#'
#' @param proj an object of class "\code{projection.hypercube}".
#' @param component a vector or a matrix to specify with components in the \code{proj} are used.
#'        See examples for how to use.
```

```r
#' @return The function \code{projectSub} returns a object of "\code{projection.hypercube}",
#'          which contains only the projection matrices according to the argument \code{component}.
#'
#' @export
projectSub <-
function(proj, component) {
  vname <- projectName(attr(proj, "variables"), component)
  subproj <- proj[vname]
  attr(subproj, "variables") <- attr(proj, "variables")
  attr(subproj, "nlevels") <- attr(proj, "nlevels")
  attr(subproj, "dims") <- attr(proj, "dims")
  attr(subproj, "component") <- vname
  class(subproj) <- c("projection.hypercube", class(subproj))
  subproj
}

#' @describeIn projectCreate
#' Weighted sum of matrix in object "\code{projection.hypercube}"
#'
#' @param weights The weights for the weighted sum of projection matrix.
#'        The order of the weights should be the same as the order of the component.
#' @return The function \code{projectWeight} returns a projection matrix
#'        which is the weigthed sum of the projection matrices according to
#'        \code{component} and \code{weights}.
#'
#' @export
projectWeight <-
function(proj, component = NULL, weights = NULL) {
  p <- length(attr(proj, "variables"))
  dims <- attr(proj, "dims")

  # If components is NULL, we want all the components in argument proj
  if(is.null(component)) {
    component <- projectPerm(attr(proj, "variables"), JH.flag=FALSE)
  }

  # Pojection matrices according to the argument component
  subproj <- projectSub(proj, component)
  vname <- attr(subproj, "component")

  # If weights is NULL, set all weights as 1
  if(is.null(weights)) weights <- rep(1, length(subproj))

  # The weighted sum of projection matrices
  ans <- matrix(0, nrow=dims[1], ncol=dims[2])
  for(k in seq_along(subproj)) {
    ans <- ans + weights[k] * subproj[[k]]
  }
  # Attr stores some information about the projection matrices
  attr(ans, "variable") <- attr(proj, "variables")
  attr(ans, "nlevels") <- attr(proj, "nlevels")
  attr(ans, "dims") <- attr(proj, "dims")
  attr(ans, "component") <- vname
```

```
  attr(ans, "weights") <- weights
  ans
}

#' @describeIn projectCreate
#' Functions of object "\code{projection.hypercube}"
#'
#' @return The function \code{projectFun} returns a function which computes and
#' returns a weighted sum of projection matrices according to \code{component}
#' in the argument of \code{projectFun}. The argument of the returned function
#' is the \code{weight}. See example for how to use.
#'
#' @export
projectFun <-
function(proj, component = NULL) {
  # If NULL, use all the component in argument proj
  if(is.null(component)) {
    subproj <- proj
  } else {
    subproj <- projectSub(proj, component)
  }

  # the function we want for the weighted projection matrices
  f <- function(weight) {
    Reduce(`+`, mapply(`*`, subproj, weight, SIMPLIFY = FALSE))
  }

  # Attr stores some information about the projection matrices
  attr(f, "variable") <- attr(proj, "variables")
  attr(f, "nlevels") <- attr(proj, "nlevels")
  attr(f, "dims") <- attr(proj, "dims")
  attr(f, "component") <- names(subproj)
  f
}
```

## ulits.R

```
# Trace of matrix
tr <- function(A) sum(diag(A))

# L2 norm of vector
l2sq <- function(v) sum(v^2)

#' Covert W matrix to V matrix
#'
#' @description Convert W to V, as described in (1.6) in Beran (2014)
#'
#' @param W a matrix, penalized least square
#' @return The function \code{plsW2V} returns a matrix V for the Hypercube Estimator.
#'
#' @examples
```

```r
#' # D_5 as in equation (3.10) in Beran (2014)
#' D <- diffMatrix(p, 5)
#' # The matrix W in equation (3.11) in Beran (2014)
#' W <- t(D) %*% D
#' # Convert W to V, as described in (1.6) in Beran (2014)
#' V <- plsW2V(W)
#'
#' @export
plsW2V <-
function(W) {
  # check W is square matrix
  solve(expm::sqrtm(diag(dim(W)[1])+W))
}

#' Difference Matrix
#'
#' @description The difference matrix described in equation (3.10) in Beran (2014).
#'
#' @param p number of coefficients
#' @param dth order of difference matrix
#'
#' @examples
#' p <- 10
#' D <- diffMatrix(p, 5)
#' D
#'
#' @export
diffMatrix <-
function(p, dth) {
  D <- diag(p)
  dif <- function(p) cbind(diag(p-1),0) + cbind(0, -diag(p-1))
  for(k in seq(p, p-(dth-1),-1)){
    D <- dif(k) %*% D
  }
  D
}
```

## data.R

```r
#' Weigth gain of 61 infant rat litters
#'
#' A dataset containing the (average)  wight gain of an infant rat litter
#' when the infants in the litter are nursed by a rat foster-mother
#'
#' The rat litter data treated by Scheffe (1959) form an unbalnced two-way layout
#' Each response recorded is the average weight-gain of a rat litter when the infants
#' in the litter are nursed by a rat foster-mother. Factor1 with four levels, is the
#' genotype of the foster-moather. Factor2 with the same levels, in the genotype of the
#' infant litter.
#'
#' The response measured in the experiment is the (average) weight gain of
```

```
#' an infant rat litter when the infants in the litter are nursed by a rat foster-mother.
#' Factor 1 is the genotype of the foster-mother nursing the infants.
#' Factor 2 is the genotype of the infant litter.
#'
#' @format A data frame with 61 rows and 3 variables:
#' \describe{
#'   \item{weight}{the (averge) weight gain of an infant rat litter}
#'   \item{mother}{the genotype of the foster-mother nursing the infants}
#'   \item{infant}{the genotype of the infant litter}
#' }
#' @source D.W. Bailey, The Inheritance of Maternal Influences on the Growth of the Rat (1953).
"litter"

#' Vineyard
#'
#' A dataset containing records the grape yield harvested in each row of a vinyard
#' in three succes years
#'
#' @format A data frame with 52 rows and 4 variables:
#' \describe{
#'   \item{row}{the vineyard row number}
#'   \item{year1}{reporting the harvest yield in first year}
#'   \item{year2}{reporting the harvest yield in second year}
#'   \item{year3}{reporting the harvest yield in third year}
#' }
#' @source vineyard
"vineyard"

#' Response of 5 different monkey-pairs
#'
#' A dataset containing reports responses to a certain stimulus
#' that were measured for 5 different monkey-pairs (the subjects)
#' in 5 different periods under 5 different conditions
#'
#' @format A data frame with 25 rows and 4 variables:
#' \describe{
#'   \item{cond}{the condition}
#'   \item{monkeys}{the monkey pair}
#'   \item{period}{the monkey pair}
#'   \item{response}{responses to a certain stimulus}
#' }
#' @source Data from Query no. 113, edited by G.W. Sender, Biometrics, Vol. 11, 1955, p.112
"monkey"

#' The hardness of 120 dental fillings
#'
#' A dataset containing the response measures the hardness of dental filling
#' obtained by 5 Dentists using 8 Gold alloys and 3 Condensation methods.
#' The objective of the experiment was to find a dental gold filling with greater hardness.
#'
#'
#' Seheult and Tukey (2001) analyzed a three-factor layout in which the response measures
#' the hardness of dental fillings obtained by 5 Dentists (D) using 8 Gold alloys (G) and
```

```
#' 3 Condensation methods (C). The objective of the experiment was to find a dental gold
#' lling with greater hardness. Condensation, properly carried out, was known to increase
#' the hardness of a filling. The three condensation techniques used in the experiment were:
#'   (1) electromalleting, in which blows are delivered mechanically at a steady frequency; (2)
#' hand malleting, in which a small mallet is used to deliver blows; and (3) hand condensation.
#' The reported hardness observations are each averages of ten measurements that are not
#' available. It was reported anecdotally that dentist 5 appeared to be physically tired before
#' the experiment.
#'
#' @format A data frame with 120 rows and 4 variables:
#' \describe{
#'   \item{y}{response mesures the hardness of dental fillings}
#'   \item{G}{the indice of 8 Gold alloys}
#'   \item{C}{the indice of 3 Condensation methods}
#'   \item{D}{the indice of 5 Dentists}
#' }
#' @source Seheult, A. H. and Tukey, J. W. (2001). Towards robust analysis of variance,
#' Data Analysis from Statistical Foundations (eds. A. K. Mohammed and E. Saleh),
#' 217-244, Nova Science Publishers, New York.
"dental"


#' Accelations over time
#'
#' A dataset containing 133 observation of motorsycle
#' acceleration against time in a simulated motorcycle accident.
#' The p = 277 possible observation times constitute the vector t = (1, 2, ... , 277)
#' Accelerations were observed at only q < p of these equally spaced time,
#' sometimes with replication.
#'
#' @format A data frame with 133 rows and 2 variables:
#' \describe{
#'   \item{t}{The time in milliseconds since impact}
#'   \item{accel}{The recorded head acceleration (in g)}
#' }
#' @source Silverman,  B.W. (1985) Some  aspects  of  the spline smoothing approach to
#' non-parametric curve fitting. Journal of the Royal Statistical Society, B, 47, 1-52.
"motor"

#' Canadian earnings
#'
#' The canadian.earnings data frame has 205 pairs observations on Canadian workers from a 1971 Canadian
#'
#' @format A data frame with 205 rows and 2 variables:
#' \describe{
#'   \item{age}{age in years.}
#'   \item{log.income}{logarithm of income.}
#' }
#' @source Ullah, A. (1985). Specification analysis of econometric models.
#' Journal of Quantitative Economics, 2, 187-209
"canadian.earnings"
```

## testthat

```
library(HyperCube)
context("Basic tests")

test_that("Check projectCreate", {
  proj <- projectCreate( ~ mother:infant -1, data = litter)
  expect_is(proj, "projection.hypercube")
})

test_that("Check hypercube, litter", {
  proj <- projectCreate( ~ mother:infant -1, data = litter)
  component <- matrix(c(0,0,1,0,0,1,1,1),2,4)
  V <- projectWeight(proj, component = component, weights = c(1,1,1,0))
  hcmod <- hypercube(weight ~  mother:infant -1, data = litter, V)
  expect_is(hcmod, "hypercube")
})

test_that("Check hypercube and lm, litter", {
  proj <- projectCreate( ~ mother:infant -1, data = litter)
  V <- projectWeight(proj, weights = c(1,1,1,1))
  hcmod <- hypercube(weight ~  mother:infant -1, data = litter, V)
  lmmod <- lm(weight ~  mother:infant -1, data = litter)
  expect_equal(object = as.numeric(hcmod$coefficients),
               expected = as.numeric(lmmod$coefficients)
               )
})

test_that("Check hypercube, canadian.earnings", {
  p <- length(unique(canadian.earnings[,1]))
  D <- diffMatrix(p, 5)
  nu <- 100
  W <- nu * t(D) %*% D
  V <- plsW2V(W)
  canadian.earnings[,"age"] <- factor(canadian.earnings[,"age"])
  hcmod <- hypercube( log.income ~ age -1, data=canadian.earnings, V)
  expect_is(hcmod, "hypercube")
})

test_that("Check hypercubeOptimization", {
  hcmodopt <- hypercubeOptimization(weight ~  mother:infant -1, data = litter)
  expect_is(hcmodopt$est, "hypercube")
  expect_equal(object = hcmodopt$projcoef,
               expected = c(0.997, 0.693, 0.000, 0.415),
               tolerance = .001)
})

test_that("Check diffMatrix", {
  D <- diffMatrix(5,3)
  Dtest <- rbind(c(1,-3,3,-1,0), c(0,1,-3,3,-1))
  expect_equal(object = D,
               expected = Dtest)
})
```