#------------------------------

#

# Kernel 2D^2PCA

# JV

# 11/08/2023

#

#------------------------------

# x = array of matrices with size p1 x p2 x n

# s2 = a two-vector of tuning parameters for the Gaussian kernels

# r\_par = the number of singular values that are used

# k = a two-vector of latent dimensionalities (at least c(1, 1))

# even = TRUE/FALSE, should even or odd kernel be used

# eps = regularization parameter for inversion

# x0 = out-of-sample test observations

k2d2pca <- function(x, s2 = NULL, r\_par, k, even = TRUE, eps = 0.2, x0 = NULL){

 p1 <- dim(x)[1]

 p2 <- dim(x)[2]

 n <- dim(x)[3]

 # Compute singular value matrices

 # These have the first singular vectors of the observations as rows

 kerL <- matrix(0, n, p1)

 kerR <- matrix(0, n, p2)

 # Each submarix has the first r\_par singular vectors as columns

 svdLArray <- array(0, dim = c(p1, r\_par, n))

 svdRArray <- array(0, dim = c(p2, r\_par, n))

 # Each row contains the r\_par singular values

 svdValues <- matrix(0, n, r\_par)

 for(i in 1:n){

 temp <- svd(x[, , i])

 kerL[i, ] <- temp$u[, 1]

 kerR[i, ] <- temp$v[, 1]

 svdLArray[, , i] <- temp$u[, 1:r\_par]

 svdRArray[, , i] <- temp$v[, 1:r\_par]

 svdValues[i, ] <- temp$d[1:r\_par]

 }

 # Tuning parameters

 if(is.null(s2)){

 # s2[1] <- (1 - sum(colMeans(kerL)^2))

 # s2[2] <- (1 - sum(colMeans(kerR)^2))

 s2[1] <- sqrt(sum(tcrossprod(kerL)^2))/n

 s2[2] <- sqrt(sum(tcrossprod(kerR)^2))/n

 }

 # Compute K1 and K2

 evensign <- 2\*even - 1

 temp <- tcrossprod(kerL)

 K1 <- exp((temp - 1)/s2[1]) + evensign\*exp((-1\*temp - 1)/s2[1])

 # K1 <- exp((tcrossprod(kerL) - 1)/s2[1])

 temp <- tcrossprod(kerR)

 K2 <- exp((temp - 1)/s2[2]) + evensign\*exp((-1\*temp - 1)/s2[2])

 # K2 <- exp((tcrossprod(kerR) - 1)/s2[2])

 # Compute the F-matrices

 FArray <- array(0, dim = c(n, n, n))

 for(i in 1:n){

 temp <- kerL%\*%svdLArray[, , i]

 tempL <- exp((temp - 1)/s2[1]) + evensign\*exp((-1\*temp - 1)/s2[1])

 # tempL <- exp((kerL%\*%svdLArray[, , i] - 1)/s2[1])

 temp <- kerR%\*%svdRArray[, , i]

 tempR <- exp((temp - 1)/s2[2]) + evensign\*exp((-1\*temp - 1)/s2[2])

 # tempR <- exp((kerR%\*%svdRArray[, , i] - 1)/s2[2])

 FArray[, , i] <- tcrossprod(sweep(tempL, 2, svdValues[i, ], "\*"), tempR)

 }

 # Regularized inverses

 eigK1 <- eigen(K1, symmetric = TRUE)

 invK1 <- tcrossprod(sweep(eigK1$vectors, 2, (eigK1$values + eps\*eigK1$values[1])^(-1), "\*"), eigK1$vectors)

 halfinvK1 <- tcrossprod(sweep(eigK1$vectors, 2, (eigK1$values + eps\*eigK1$values[1])^(-1/2), "\*"), eigK1$vectors)

 eigK2 <- eigen(K2, symmetric = TRUE)

 invK2 <- tcrossprod(sweep(eigK2$vectors, 2, (eigK2$values + eps\*eigK2$values[1])^(-1), "\*"), eigK2$vectors)

 halfinvK2 <- tcrossprod(sweep(eigK2$vectors, 2, (eigK2$values + eps\*eigK2$values[1])^(-1/2), "\*"), eigK2$vectors)

 # Sample operators

 Fbar <- apply(FArray, 1:2, mean)

 opL <- matrix(0, n, n)

 opR <- matrix(0, n, n)

 for(i in 1:n){

 opL <- opL + FArray[, , i]%\*%invK2%\*%t(FArray[, , i])

 opR <- opR + t(FArray[, , i])%\*%invK1%\*%FArray[, , i]

 }

 opL <- opL/n - Fbar%\*%invK2%\*%t(Fbar)

 opR <- opR/n - t(Fbar)%\*%invK1%\*%Fbar

 opL <- halfinvK1%\*%opL%\*%halfinvK1

 opR <- halfinvK2%\*%opR%\*%halfinvK2

 eigL <- eigen(opL, symmetric = TRUE)

 eigR <- eigen(opR, symmetric = TRUE)

 vecL <- halfinvK1%\*%eigL$vectors[, 1:k[1], drop = FALSE]

 vecR <- halfinvK2%\*%eigR$vectors[, 1:k[2], drop = FALSE]

 ZArray <- array(0, dim = c(k[1], k[2], n))

 for(i in 1:n){

 ZArray[, , i] <- t(vecL)%\*%(FArray[, , i] - Fbar)%\*%vecR

 }

 # Test set

 if(!is.null(x0)){

 n0 <- dim(x0)[3]

 # Compute singular value matrices

 # kerL0 <- matrix(0, n0, p1)

 # kerR0 <- matrix(0, n0, p2)

 svdLArray0 <- array(0, dim = c(p1, r\_par, n0))

 svdRArray0 <- array(0, dim = c(p2, r\_par, n0))

 svdValues0 <- matrix(0, n0, r\_par)

 for(i in 1:n0){

 temp <- svd(x0[, , i])

 # kerL0[i, ] <- temp$u[, 1]

 # kerR0[i, ] <- temp$v[, 1]

 svdLArray0[, , i] <- temp$u[, 1:r\_par]

 svdRArray0[, , i] <- temp$v[, 1:r\_par]

 svdValues0[i, ] <- temp$d[1:r\_par]

 }

 # Compute the F-matrices

 FArray0 <- array(0, dim = c(n, n, n0))

 for(i in 1:n0){

 # temp <- kerL0%\*%svdLArray0[, , i]

 temp <- kerL%\*%svdLArray0[, , i]

 tempL <- exp((temp - 1)/s2[1]) + evensign\*exp((-1\*temp - 1)/s2[1])

 # tempL <- exp((kerL%\*%svdLArray[, , i] - 1)/s2[1])

 #temp <- kerR0%\*%svdRArray0[, , i]

 temp <- kerR%\*%svdRArray0[, , i]

 tempR <- exp((temp - 1)/s2[2]) + evensign\*exp((-1\*temp - 1)/s2[2])

 # tempR <- exp((kerR%\*%svdRArray[, , i] - 1)/s2[2])

 FArray0[, , i] <- tcrossprod(sweep(tempL, 2, svdValues0[i, ], "\*"), tempR)

 }

 ZArray0 <- array(0, dim = c(k[1], k[2], n0))

 for(i in 1:n0){

 ZArray0[, , i] <- t(vecL)%\*%(FArray0[, , i] - Fbar)%\*%vecR

 }

 }

 else{

 ZArray0 <- NULL

 }

 return(list(z = ZArray,

 evL = eigL$values,

 evR = eigR$values,

 s2 = s2,

 z0 = ZArray0))

}