University of Central Florida

Department of Statistics

Report for STA 6908

Nonparametric Methods

Nonparametric Multivariate Change-Point Model Using Kernel Maximum Mean Discrepancy

Submitted by

Kanak Choudhury

Supervised by

Dr. Edgard Maboudou

Contents

1 The Maximum Mean Discrepancy (MMD)

The problem of comparing samples from two probability distributions, using statistical tests of the null hypothesis that these distributions are equal against the alternative hypothesis that these distributions are different (we call this the two-sample problem). Such tests have application in a variety of areas, as discussed in [1]. In bioinformatics, for example, we want to compare microarray data from identical tissue types as measured by different laboratories, to detect whether the data may be analyzed jointly, or whether differences in experimental procedures have caused systematic differences in the data distributions. As a second application example, in database attribute matching it is desirable to merge databases containing multiple fields where it is not known in advance which fields correspond to each other. In that case the fields are matched by maximizing the similarity in the distributions of their entries.

The ideas presented in [2] can be divided into three main categories:

1. Defining a distance metric for probability distributions based on distances between their Hilbert space mean embedding. This metric was called the maximum mean discrepancy (MMD).
2. Computing empirical estimates of the MMD.
3. Performing hypothesis tests using empirical MMD estimates as the test statistic, in order to determine whether two samples were drawn from the same distribution or not.

Problem statement: Let x and y be random variables defined on a topological space X, with distributions p and q. Define $X \coloneqq \{x_1, \dots, x_m\}$ and $Y \coloneqq \{y_1, \dots, y_n\}$ as independently and identically distributed samples from p and q respectively. The goal is to formulate a test statistic and do hypothesis testing to decide whether $p \neq q$.

It is important to test whether distributions p and q are different on the basis of samples drawn from each of them, by finding a well behaved (e.g., smooth) function which is large on the points drawn from p, and small (as negative as possible) on the points drawn from q. The test statistic was used the difference between the mean function values on the two samples; when this is large, the samples are likely from different distributions. This test statistic is called the Maximum Mean Discrepancy (MMD).

The quality of the MMD as a statistic depends on the class F of smooth functions that define it. On one hand, F must be "rich enough" so that the population MMD vanishes if and only if $p = q$. On the other hand, for the test to be consistent in power, F needs to be "restrictive" enough for the empirical estimate of the MMD to converge quickly to its expectation as the sample size increases.

Lemma 1: Let (X, d) be a metric space, and let p, q be two Borel probability measures defined on X. Then $p = q$ if and only if $E_x\big(f(x)\big) = E_y\big(f(y)\big)\ \forall\ f \in C(X)$, where $C(X)$ is the space of bounded continuous functions on X.

Note that while Lemma 1 defines a class of functions to uniquely identify whether $p = q$, $C(X)$ is a very rich function class and it is computationally impractical to work with such rich class in finite sample settings.

Definition 1: Let F be a class of functions $f: X \to \mathbb{R}$ and let p, q, x, y, X, Y be defined as above. Then, the maximum mean discrepancy (MMD) is defined as:

$$MMD[F, p, q] \coloneqq \sup_{f \in F} \Big(E_x\big(f(y)\big) - E_y\big(f(y)\big)\Big).$$

We want a smooth function $f \in F$ that maximizes the difference in the expectations of the function, where the first expectation is over p and the second expectation is taken over q. A biased

empirical estimate of the MMD can be obtained by replacing the population expectations with empirical expectations computed on the samples X and Y :

$$MMD_b[F, X, Y] \coloneqq \sup_{f \in F} \left(\frac{1}{m} \sum_{i=1}^{m} f(x_i) - \frac{1}{n} \sum_{i=1}^{n} f(y_i) \right)$$

Note that, $MMD_b[F, X, Y] \geq MMD[F, p, q]$ by Jensen's inequality.

The goal is then to identify a function class that is rich enough to uniquely identify whether $p = q$, yet restrictive enough to provide useful finite sample estimates.

Now we can write the MMD in terms of the kernel function and how we can estimate it using a finite set of samples from the two distributions of interest.

Lemma 2: Given x and x' independent random variables with distribution p, and y and y' independent random variables with distribution q, the squared population MMD can be written as:

$$MMD^2[F, p, q] = E_{x,x'}[k(x, x')] - 2E_{x,y}[k(x, y)] + E_{y,y'}[k(y, y')],$$

where x' is an independent copy of x with the same distribution, and y' is an independent copy of y. An unbiased empirical estimate is a sum of two U-statistics and a sample average:

$$\begin{aligned} MMD_u^2[F, X, Y] \\ &= \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} k(x_i, x_j) - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(x_i, y_j) \\ &+ \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k(y_i, y_j), \end{aligned}$$

The biased MMD estimate of equation 2 can also be written in a similar way, by replacing the U-statistics with V-statistics:

$$MMD_b[F,X,Y] = \left[\frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}k(x_i,x_j) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}k(x_i,y_j) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}k(y_i,y_j)\right]^{\frac{1}{2}}.$$

1.1 R Code 1

In the following R code, I have used KMMD function and calculate KMMD for any input matrix. Find the maximum change point position with KMMD statistics. It can be used to find the permutation test of KMMD from the sample. After getting the change point of a sample, if new observation comes, we need to find in which groups the new observations belongs to. To find the group for the new observation, we can use the following function as well. It will calculate KMMD statistics for each of the new observation with each of the groups and find the maximum KMMD for each pair of statistic. Then the maximum KMMD statistics will be returned for the new observations. For example, if there are three groups that is there are three change point locations, then for the new observations, it will calculate three KMMD statistics for the new observations with the each of the three groups and it finds the maximum value of the KMMD statistics.

```
require(kernlab)

## Loading required package: kernlab

require(plyr)
require(MASS)
x <- matrix(rnorm(100, 10, 5),50)
y <- matrix(rnorm(200, 20, 5),100)
z <- matrix(rnorm(100, 15, 10), ncol = ncol(x))

colSums(sweep(x, 2,colMeans(x)))

## [1]  1.687539e-14 -1.065814e-14
```

```
kk <- function(Kxx,Kyy, Kxy)
{

  m <- dim(Kxx)[1]
  n <- dim(Kyy)[1]

  sumKxx <- sum(Kxx)

  sumKyy <- sum(Kyy)

  sumKxy <- sum(Kxy)

  mmd1 <- sqrt(max(0,sumKxx/(m*m) + sumKyy/(n*n) - 2/m/n* sumKxy))

  return(list(mmd=mmd1))
}


#################################################################

# KMMD function
# return KMMD with 1st and 3rd order

kk1<- function(Kxx,Kyy, Kxy, alpha = 0.05)
{

  m <- dim(Kxx)[1]
  n <- dim(Kyy)[1]

  N <- max(m,n)
```

```
M <- min(m,n)

sumKxx <- sum(Kxx)

if(m!=n)
  sumKxxM <- sum(Kxx[1:M,1:M])
else
  sumKxxM <- sumKxx

dgxx <- diag(Kxx)

sumkd <- sumKxx - sum(dgxx)
R <- max(dgxx)
RM <- max(dgxx[1:M])
hu <- colSums(Kxx[1:M,1:M]) - dgxx[1:M]

sumkyy <- sum(Kyy)
if(m!=n)
  sumkyyM <- sum(Kyy[1:M,1:M])
else
  sumkyyM <- sumKyy

dgyy <- diag(Kyy)

sumkd <- sum(Kyy) - sum(dgyy)
R <- max(R,dgyy)
RM <- max(RM,dgyy[1:M]) # RM instead of R in original
hu <- hu + colSums(Kyy[1:M,1:M]) - dgyy[1:M]

sumKxy <- sum(Kxy)
if (m!=n)
```

```
    sumKxyM <- sum(Kxy[1:M,1:M])
  else
    sumKxyM <- sumKxy

  dg <- diag(Kxy) # up to M only
  hu <- hu - colSums(Kxy[1:M,1:M]) - colSums(t(Kxy[1:M,1:M])) + 2*dg #
one sided sum

  mmd1 <- sqrt(max(0,sumKxx/(m*m) + sumKyy/(n*n) - 2/m/n* sumKxy))
  mmd3 <- sum(hu)/M/(M-1)
  D1 <- 2*sqrt(RM/M)+sqrt(log(1/alpha)*4*RM/M)

  return(list(mmd1=mmd1,mmd3=mmd3,D1=D1))
}
```

Following function find the change point for a sample. It calculates change point by dividing the sample into two groups and it will start from a minimum sample needed to calculated the KMMD and then it will continue to calculate KMMD until the end of the sample. It will keep track of the KMMD and location group where it was divided. In this function it is possible to change the kernel function. The user define kernel function can be inserted and it will use that kernel function to calculate the KMMD.

```
## Find Change point location by calculating maximum KMMD for data

kk2<- function(x, keepobs = NULL){
  x = as.matrix(x)
  N = nrow(x)
  if (is.null(keepobs)){
    keepobs = ceiling(N*0.1)
```

```
  }
  aa <- function(z, n, rbf = rbf){
    nobs = nrow(z)
    ncoll = ncol(z)
    x = z[1:n,1:ncoll]
    y = z[(n+1):nobs,1:ncoll]
    stat <- kk(Kxx=kernelMatrix(rbf, x), Kyy =kernelMatrix(rbf, y), Kx
y=kernelMatrix(rbf, x,y))
    return(stat$mmd)
  }

  sig.opt <- sigest(x, scaled = FALSE)[2]
  rbf <- rbfdot(sigma = sig.opt)

  ntest = c((keepobs+1):(N-keepobs))
  stat = sapply(ntest, aa, z = x, rbf = rbf)
  mk = which.max(stat)
  return(list(stat = stat, change.Point = mk+keepobs))
}

bb = cpkmmd(rbind(x,y))
```

After separating the original sample into groups based on the KMMD statistics, if new observation comes, then the following function can be used to allocation the new observation into any one of the previously obtained groups. The algorithm has been discussed in the previous paragraph.

```
obskk2<- function(newobs, x, y){
  ncoll = ncol(x)
  newobs = matrix(newobs, ncol = ncoll)
  x = as.matrix(x)
```

```
  y = as.matrix(y)
  nnew = nrow(newobs)
  ns = c(1:nnew)
  sig.opt <- sigest(rbind(x,y), scaled = FALSE)[2]
  rbf <- rbfdot(sigma = sig.opt)

  aa <- function(i, z, x, y, rbf = rbf){
    a = z[i,]
    s1 = kk(Kxx=kernelMatrix(rbf, rbind(x,a)), Kyy =kernelMatrix(rbf,
y), Kxy=kernelMatrix(rbf, rbind(x,a),y))
    s2 = kk(Kxx=kernelMatrix(rbf, x), Kyy =kernelMatrix(rbf, rbind(y,a
)), Kxy=kernelMatrix(rbf, x,rbind(y,a)))
    if (s1$mmd > s2$mmd) c = 1
    else c = 2
    return(c)
  }
  co = sapply(ns, aa, z = newobs, x = x, y = y, rbf = rbf)
  return(co)
}

newobscpkmmd(z, x,y)
```

OUTPUT:

```
> bb
       T2Max change_Point
    268.1326      50.0000
```

```
##  [1] 2 1 2 1 2 1 2 1 1 1 2 2 1 2 1 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 2
2 2 2 2
## [36] 1 1 2 2 1 1 2 1 2 2 1 2 1 1 2
```

To illustrate this algorithm, I have generated two multivariate random samples from normal distribution with mean (10, 20) and variance (5 for both samples) respectively, and another sample from normal distribution with mean 15 and variance 10. This third sample has been considered as the new observations. Since the variance is higher and mean is 15, this sample should overlap in both samples. The sample size for the first two samples are 50 and 100 respectively. If we combine these two samples into one sample, we can say that there is a change point at observation 50. Because two samples are coming from different distribution. Now, when we have used the KMMD change point algorithm, it shows that there is a change point at observation 50 and the T2Max is 268.1326. that means that this algorithm correctly finds the change point location. For confirm it we have conducted a simulation and found that more than 95% times the algorithm correctly finds the change point location.

Now, to find the group for the new observation, I have used the third sample and output has given above. In the third sample, there were 50 observations taken from normal distribution with mean 25 and variance 10. When we apply this algorithm on this new sample, it is found that 27 observations which is about 50% (since it is normal distribution) belongs to the second group. I have also checked that is we take sample from any of those groups, it exactly defines the group for the new sample.

1.2 R Code 2

The following R code is the whole KMMD function based on S3 method. This function can be used to do permutation test with KMMD. It means that will create all possible permutation of the sample observations and calculated KMMD for all possible samples and finally create the distribution for the test statistics as well as find the best cut for the sample using KMMD.

```
##  calculates the kernel maximum mean discrepancy for samples from tw
o distributions

setGeneric("kmmd",function(x,...) standardGeneric("kmmd"))

## [1] "kmmd"

setMethod("kmmd", signature(x = "matrix"),
          function(x, y, kernel="rbfdot",kpar="automatic", alpha = 0.0
5, asymptotic = FALSE,  replace = TRUE, ntimes = 150, frac = 1,  ...)
          {
            x <- as.matrix(x)
            y <- as.matrix(y)

            res <- new("kmmd")


            if(is.character(kernel)){
              kernel <- match.arg(kernel,c("rbfdot","polydot","tanhdot
","vanilladot","laplacedot","besseldot","anovadot","splinedot","matrix
"))

              if(kernel == "matrix")
                if(dim(x)[1]==dim(x)[2])
                  return(kmmd(x= as.kernelMatrix(x), y = y, Kxy = as.k
ernelMatrix(x)%*%y, alpha = 0.05, asymptotic = FALSE,  replace = TRUE,
ntimes = 100, frac = 1,  ...))
              else
                stop(" kernel matrix not square!")

              if(is.character(kpar))
                if((kernel == "tanhdot" || kernel == "vanilladot" || k
```

```
ernel == "polydot"|| kernel == "besseldot" || kernel== "anovadot"|| ke
rnel=="splinedot") &&  kpar=="automatic" )
                {
                  cat (" Setting default kernel parameters ","\n")
                  kpar <- list()
                }
            }

            if (!is.function(kernel))
              if (!is.list(kpar)&&is.character(kpar)&&(kernel == "lapl
acedot"|| kernel=="rbfdot")){
                kp <- match.arg(kpar,"automatic")
                if(kp=="automatic")
                  kpar <- list(sigma=sigest(rbind(x,y),scaled=FALSE)[2
])
                cat("Using automatic sigma estimation (sigest) for RBF
or laplace kernel","\n")

              }
            if(!is(kernel,"kernel"))
            {
              if(is(kernel,"function")) kernel <- deparse(substitute(k
ernel))
              kernel <- do.call(kernel, kpar)
            }

            if(!is(kernel,"kernel")) stop("kernel must inherit from cl
ass `kernel'")

            m <- dim(x)[1]
            n <- dim(y)[1]
```

```
            N <- max(m,n)
            M <- min(m,n)

            Kxx <- kernelMatrix(kernel,x)
            Kyy <- kernelMatrix(kernel,y)
            Kxy <- kernelMatrix(kernel,x,y)

            resmmd <- .submmd(Kxx, Kyy, Kxy, alpha)

            H0(res) <- (resmmd$mmd1 > resmmd$D1)
            Radbound(res) <- resmmd$D1
            Asymbound(res) <- 0
            mmdstats(res)[1] <- resmmd$mmd1
            mmdstats(res)[2] <- resmmd$mmd3

            if(asymptotic){
              boundA <- .submmd3bound(Kxx, Kyy, Kxy, alpha, frac, ntim
es, replace)

              AsympH0(res) <- (resmmd$mmd3 > boundA)
              Asymbound(res) <- boundA
            }

            kernelf(res) <- kernel
            return(res)
          })

## [1] "kmmd"

.submmd <- function(Kxx,Kyy, Kxy, alpha)
{
```

```
m <- dim(Kxx)[1]
n <- dim(Kyy)[1]

N <- max(m,n)
M <- min(m,n)

sumKxx <- sum(Kxx)

if(m!=n)
  sumKxxM <- sum(Kxx[1:M,1:M])
else
  sumKxxM <- sumKxx

dgxx <- diag(Kxx)

sumKxxnd <- sumKxx - sum(dgxx)
R <- max(dgxx)
RM <- max(dgxx[1:M])
hu <- colSums(Kxx[1:M,1:M]) - dgxx[1:M]

sumKyy <- sum(Kyy)
if(m!=n)
  sumKyyM <- sum(Kyy[1:M,1:M])
else
  sumKyyM <- sumKyy

dgyy <- diag(Kyy)

sumKyynd <- sum(Kyy) - sum(dgyy)
R <- max(R,dgyy)
```

```
  RM <- max(RM,dgyy[1:M]) # RM instead of R in original
  hu <- hu + colSums(Kyy[1:M,1:M]) - dgyy[1:M]

  sumKxy <- sum(Kxy)
  if (m!=n)
    sumKxyM <- sum(Kxy[1:M,1:M])
  else
    sumKxyM <- sumKxy

  dg <- diag(Kxy) # up to M only
  hu <- hu - colSums(Kxy[1:M,1:M]) - colSums(t(Kxy[1:M,1:M])) + 2*dg #
one sided sum

  mmd1 <- sqrt(max(0,sumKxx/(m*m) + sumKyy/(n*n) - 2/m/n* sumKxy))
  mmd3 <- sum(hu)/M/(M-1)
  D1 <- 2*sqrt(RM/M)+sqrt(log(1/alpha)*4*RM/M)

  return(list(mmd1=mmd1,mmd3=mmd3,D1=D1))
}


setMethod("show","kmmd",
          function(object){

            cat("Kernel Maximum Mean Discrepancy object of class \"kmm
d\"","\n","\n")

            show(kernelf(object))

            if(is.logical(object@H0)){
              cat("\n")
```

```
              cat("\n","H0 Hypothesis rejected : ", paste(H0(object)))
              cat("\n","Rademacher bound : ", paste(Radbound(object)))
            }

            cat("\n")

            if(Asymbound(object)!=0){
              cat("\n","H0 Hypothesis rejected (based on Asymptotic bo
und): ", paste(AsympH0(object)))
              cat("\n","Asymptotic bound : ", paste(Asymbound(object))
)
            }

            cat("\n","1st and 3rd order MMD Statistics : ", paste( mmd
stats(object)))
            cat("\n")
          })

## [1] "show"
```

2 A Multivariate Change-Point Model for Statistical Process Control

The most widely used setting for multivariate SPC is the multivariate normal. This setting assumes that while in control, the readings follow independent common multivariate normal distributions with some mean vector μ and covariance structure Σ. Some important departures from control include the following:

- The mean vector could change from μ to μ_1 while the covariance structure remains unchanged.
- The covariance structure could be perturbed from Σ to Σ_1.
- Both mean and covariance could have a step change.
- One or both of these parameters could drift.
- The distribution could change from normal to some other form.

In this article it was concentrated on the first case; when there is a shift in the mean vector only, the covariance structure is assumed to remain unchanged. An out-of-control state can be transient or isolated, by which we mean that the system goes out of control but then returns to control even in the absence of any intervention. A departure from control can also be persistent or sustained, by which we mean that having left the state of control, the system will remain out of control, or even go further from control, until some corrective action is taken. It was concentrated on the second of these possibilities [9].

2.1 The Unknown–Parameter Change–Point Model

Let

$$\boldsymbol{X_i} \sim \begin{cases} N_p(\mu, \Sigma) & if \;\; i \leq \tau \\ N_p(\mu_1, \Sigma_1) & if \;\; i > \tau \end{cases}$$

defines the change-point formulation in a p-variate normal case. The parameter τ is called the change point (if one exists) and is unknown. The remaining parameters, $\mu, \mu_1, \Sigma \; and \; \Sigma_1$, are the in-control and out-of-control mean vectors and covariance matrices. Although there are change-point formulations with some of the parameters assumed known, it was focused on the unknown-parameter change-point problem in which none of the parameters is assumed known a priori. There are various specializations of the general model; the assumption that $\Sigma = \Sigma_1$ leads to a model in which the change is in the mean only.

2.2 Likelihood Ratio Test for a Change in Mean

It was assumed that $\Sigma_1 = \Sigma$, yielding a mean-only change-point model defined as

$$\boldsymbol{X_i} \sim \begin{cases} N_p(\mu, \Sigma) & if \;\; i \leq \tau \\ N_p(\mu_1, \Sigma) & if \;\; i > \tau \end{cases}$$

Define

$$\bar{\boldsymbol{X}}_{\boldsymbol{j,m}} = \frac{\mathbf{1}}{\boldsymbol{m-j}} \sum_{\boldsymbol{i=j+1}}^{\boldsymbol{m}} \boldsymbol{X_i}$$

$$\boldsymbol{W_k} = \frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} (\boldsymbol{X_i} - \bar{\boldsymbol{X}}_{\boldsymbol{0,k}})(\boldsymbol{X_i} - \bar{\boldsymbol{X}}_{\boldsymbol{0,k}})' + \sum_{\boldsymbol{i=k+1}}^{\boldsymbol{n}} (\boldsymbol{X_i} - \bar{\boldsymbol{X}}_{\boldsymbol{k,n}})(\boldsymbol{X_i} - \bar{\boldsymbol{X}}_{\boldsymbol{k,n}})'}{\boldsymbol{n-2}}$$

and assume that $n > p + 1$ to ensure full rank. $\boldsymbol{W_k}$ defines the pooled covariance structure at k. To investigate a possible shift occurring after observation k, define $\boldsymbol{Y}_k$, the standardized difference between the preshift and postshift observations, as

$$\boldsymbol{Y_k} = \left[\frac{\boldsymbol{k(n-k)}}{\boldsymbol{n}}\right]^{\frac{\mathbf{1}}{\mathbf{2}}} (\bar{\boldsymbol{X}}_{\boldsymbol{0,k}} - \bar{\boldsymbol{X}}_{\boldsymbol{k,n}})$$

and the Hotelling $\boldsymbol{T}^2$ statistic for testing a difference between preshift and postshift data at an assumed change point k as

$$\boldsymbol{T_k^2 = Y_k' W_k^{-1} Y_k}, \qquad \boldsymbol{k = 1, \ldots, n-1}$$

If the change point were known a priori to be at $\tau = k$, then $\boldsymbol{T_k^2}$ would be the generalized likelihood ratio test statistic for testing for a change between pre-k and post-k data. In the event that the changepoint is not knownahead of time, the maximum over all possible split points,

$$\boldsymbol{T_{\hat{\tau}}^2 = max\, T_k^2,} \qquad\qquad \boldsymbol{k = 1, \dots, n-1}$$

is the generalized likelihood ratio test statistic for change in the mean. The maximizing index $\hat{\boldsymbol{\tau}}$ is the maximum likelihood estimate (MLE) of the change. This statistic is a well-studied test for a single change point in a fixed sample of size n.

2.3 Phase II Dynamic Application

Using the unknown-parameter change-point model in SPC leads to two statistical tasks: testing for heterogeneity and estimating the parameters. Most work on change-point problems has focused on its fixed-sample setting. This involves adapting the classic fixed-sample change-point formulation to a Phase II setting by defining a repeated-testing framework in which as each new observation accrues, the change-point test is reapplied to the data accumulated to date, but this is done in such a way that the probability of a false alarm remains constant.

Specifically, as each new process vector $\boldsymbol{X_n}$ accrues, the following steps should follow:

- Calculate $T_{max,n}^2$ the maximized split statistic, for the entire sequence of readings $\boldsymbol{X_1, X_2, \dots, X_n}$ to date.
- If $T_{max,n}^2$ exceeds some cutoff $h_{n,p,\alpha}$ then it is concluded that a shift has occurred. It is estimated the time of occurrence of the shift by the k value leading to the maximum T^2, and conduct a two-sample study of the two segments so defined to diagnose the change.
- If, however, $T_{max,n}^2 < h_{n,p,\alpha}$, then it can be concluded that there is no firm evidence of a shift, and let the process continue without interruption. This leads to the issue of choosing

the sequence of control limits, $h_{n,p,\alpha}$. If this probability is to be a constant α, say, then the sequence must satisfy the following equation:

$$P\left[T^2_{max,n} > h_{n,p,\alpha} | T^2_{max,j} \leq h_{j,p,\alpha}; j < n\right] = \alpha.$$

2.4 R Code

In this article, it is the multivariate change point model. The concept of change point model is exactly the same as we discussed in first section. However, here they have used the Hotelling T^2 as the test statistics instead of KMMD that we have discussed in first section. However, in this part, we have written the R code for the simulation study to find the threshold value for the change point so that when new observation comes, it does not need to create distribution for the test statistics instead we can just check the KMMD with the tabulated threshold value for the test statistics.

To do the simulation, we have generated million sample from the normal distribution with given parameter. For each of the sample we have calculated Hotelling T^2 by separating sample into two groups. To separate the sample into two groups we have started from the 5^{th} percentile and continue separation until 95^{th} percentile. For each the separation we have calculated Hotelling T^2 and store all those statistics so that can get the distribution at each separation. Using each distribution, we have calculated 95% quantile for every cut point. To ensure Type I error, we have considered the quantile of the previous cut point as described in the above equation.

First we have written a function to calculate the Hoteling T^2 as $\boldsymbol{T^2_k = Y'_k W^{-1}_k Y_k}$, $\boldsymbol{k = 1, \dots, n-1}$.

```
HotellingT2 <- function(y, w){
  y = matrix(y, nrow = 1)
  w = as.matrix(w)
  y %*% solve(w) %*% t(y)
```

```
}


## function - return two values (1) Maximum T2 and (2) the value of k

bb <- function(x, keepobs = NULL){
  x = as.matrix(x)
  #print(x)
  N = nrow(x)
  ncol = ncol(x)
  if (ncol > 1) {ccm = colMeans
  }else {ccm = mean}

  if (isnull(keepobs)){
    keepobs = ceiling(N*0.1)
    if (keepobs < ncol) keepobs = ncol + 1
  }

  bb <- function(z, n, nobs, N){
    x = z[1:n,]
    y = z[(n+1):nobs,]
    wk = (var(x)*n + var(y)*(nobs - N)) / (nobs - 2)
    yk = sqrt(n *(nobs - N) / nobs) * (ccm(x) - ccm(y))
    T2k = HotellingT2(yk, wk)
    return(T2k)
  }

  ntest = c((keepobs+1):(N-keepobs))
  T2 = apply(ntest, bb, z = x, nobs = N)
  max = which.max(T2)
  return(c(T2Max = T2[max], Change_Point = maxk +keepobs, T2 = T2))
```

```
}

## Repeat Max T^2 for N (SimNum) times and return as a Data Frame cont
aining two columns (T2Max and Change_Point (position))

set.seed(2345)
alpha = 0.05
p = 1
SimNum = 20
n =8
mu = rep(0, p)
sigma =  diag(x = 1, nrow = p, ncol = p)
a = rdply(SimNum, aa (mvrnorm(n = n, mu = mu, Sigma = sigma), keepobs
= p+1))
b = a[,4:ncol(a)]
print(a)

##    .n      T2Max Change_Point          T21        T22         T23
## 1   1  1.8829980            5 1.976256e-01 1.18928439 1.882998043
## 2   2  2.0967854            6 8.684432e-01 1.61662263 0.372369886
## 3   3  0.2575637            4 3.487597e-02 0.25756367 0.021267580
## 4   4  0.1594644            5 7.142481e-02 0.03850383 0.159464395
## 5   5 14.5026479            3 1.450265e+01 3.46340996 3.202057352
## 6   6  0.6875516            3 6.875516e-01 0.42903856 0.197372528
## 7   7  7.8049864            4 1.551667e+00 7.80498636 5.333896699
## 8   8  2.1744871            6 7.730171e-01 0.01824885 0.896953217
## 9   9  0.1587509            3 1.587509e-01 0.11077765 0.042786125
## 10 10  2.1665637            6 1.413523e-01 0.05266637 0.323061132
## 11 11  4.1975502            3 4.197550e+00 0.88762688 0.001032129
## 12 12  2.4023162            3 2.402316e+00 1.02088929 0.929425476
## 13 13  0.4402615            3 4.402615e-01 0.34262963 0.308200510
```

```
## 14 14  2.0852057           3 2.085206e+00 0.25522824 0.050946226
## 15 15  1.0769353           5 1.082131e-03 0.01762906 1.076935329
## 16 16  2.1001482           6 1.873204e-02 0.20743859 0.500004178
## 17 17  1.1985305           5 6.563868e-02 0.00330262 1.198530481
## 18 18  1.5399035           6 7.576673e-01 0.16630984 0.007474626
## 19 19  0.9173667           5 1.180747e-01 0.03966890 0.917366748
## 20 20  0.1913494           5 4.278521e-04 0.13630954 0.191349432
##           T24
## 1  0.60607440
## 2  2.09678543
## 3  0.02298761
## 4  0.01403302
## 5  0.74381224
## 6  0.60546422
## 7  6.10836265
## 8  2.17448712
## 9  0.06863711
## 10 2.16656367
## 11 0.03845976
## 12 0.50956755
## 13 0.18243332
## 14 0.22399582
## 15 0.09089440
## 16 2.10014819
## 17 0.19930501
## 18 1.53990350
## 19 0.36210154
## 20 0.04349802
```

```
quant = list(c(NULL, NULL))
for (i in 1:ncol(b)){
```

```
  quant[[i]] = c(as.integer(p+1+i), quantile(b[,i], probs = 0.95, name
s = FALSE, type = 1))
  b = b[-which(unlist(b[,i]) >= quant[[i]][2]),]
}
bb = t(as.data.frame(quant))
row.names(bb) = NULL
print(bb)

##      [,1]     [,2]
## [1,]    3 4.197550
## [2,]    4 7.804986
## [3,]    5 1.882998
## [4,]    6 2.174487
```

To illustrate this algorithm, we have generated multivariate normal distribution with given mean vector and covariance matrix of size 8. We have simulated it 20 times. For each of the sample Hotelling T2 has been calculated at every change point location and stored it. Table A shows the test statistics. We can consider these as a distribution of the test statistics for each change point location. From each of this distribution, we have calculated the 95% quantile controlling the effect of previous change point so that the total type I error rate is 0.05. Finally, we have stored threshold values for each change point location.

3 A Control Chart Based on A Nonparametric Multivariate Change-Point Model

In phase-II statistical process control (SPC), a potentially never-ending stream of observations is collected and each time a new observation is obtained a SPC check is performed. This procedure continues until the SPC check signals that the process has gone out-of-control. It is assumed that variability in process readings comes from two main sources: common cause random variability that cannot be removed without making fundamental changes to the process, and special cause variability that can be identified and removed. The process is in statistical control when only common cause variability exists. The process may go out-of-control in two distinct ways; an isolated special cause may affect one or a small number of process readings and then go away, while a sustained special cause continues until it is identified and fixed [3]. A special cause may be a shift in mean, a change in variance, or the process readings may change distribution.

Many challenges arise when conducting SPC on multivariate process readings that do not exist in the univariate case. Checking the assumption of multivariate normality is difficult, and real world data is unlikely to follow the multivariate normal (MVN) distribution. In addition, transforming data to normality is often possible in one dimension, but transformations to multivariate normality are limited [4]. For univariate data, shifts in mean can only occur in two directions: up or down. However, for multivariate data, shifts in the mean vector can occur in any arbitrary direction. This causes difficulties when defining distribution-free multivariate SPC procedures. Distribution-free multivariate SPC procedures are those based on a test statistic which has a null distribution which does not depend on the distribution of the original data vectors. Some nonparametric procedures are unable to detect shifts in all directions [5]. Others are able to detect shifts in all directions, but are designed to detect a specific type of shift, such as an upward or downward shift in a single component of the mean vector [6].

The problem of multivariate SPC has been studied to some extent, and the proposed procedures require a variety of assumptions about the parameters of the in-control or out-of-control process. [7] derived a multivariate generalization of the univariate CUSUM control chart that allows more rapid detection of sustained shifts in mean vector than the multivariate Shewart chart. [8] proposed a multivariate exponentially weighted moving average (MEWMA) control chart, which is an extension of the univariate exponentially weighted moving average procedure. Both procedures are designed for use when the data is distributed according to the multivariate normal distribution with fixed and known in-control mean and covariance matrix.

[9] defined a change-point model for multivariate normal data that uses the generalized likelihood ratio test in a similar fashion to the univariate procedure of [3]. As in the univariate case, the change-point model alleviates the need for a large phase-I sample to estimate in-control parameters of the multivariate normal distribution, and the chart performs well across a wide range of shifts in mean vector.

[5] and [6] proposed distribution-free CUSUM procedures that do not require the assumption of multivariate normality, but require knowledge of the in-control distribution of the antirank vectors. [4] proposed a distribution free CUSUM procedure that estimates the in-control distribution using log linear modeling. Because this procedure is distribution-free and a method for estimating the in-control distribution is specified, it is well suited for multivariate SPC use when little is known about the distribution of the process readings, but a large training sample is available.

In this paper, it was proposed a nonparametric multivariate phase-II SPC procedure designed to detect shifts in the location vector that does not require knowledge of the in-control or out-of-control distribution parameters and does not require a large historical sample of in-control data. This procedure is based on an approximately distribution free multivariate generalization of the Wilcoxon-Mann-Whitney test, and extends the nonparametric change-point model based on the univariate rank-sum test [10] to the multivariate setting. Although the underlying test statistic is approximately distribution free, this procedure may not be appropriate for some multivariate

distributions with unusual dependence structure between vector components. A diagnostic was proposed to aid in determining if the proposed procedure is appropriate that can be used if a historical sample of data is available.

3.1 Computationally Efficient Algorithm

The steps to compute the test statistic $r_{k,n+1}$ for all $k = 1, \dots, n$ when observation x_{n+1} is added to the data set using a method for which computation time scales linearly in n. For this method, it is stored and updated the centered rank vectors, $R_n(x_i)$ each time a new observation is acquired. This requires storage of a $p \times n$ matrix, which is not a restrictive requirement for realistic values of n and p.

Update procedure

1. For $i = 1, \dots, n$, update

$$R_{n+1}(x_i) = \sum_{j=1}^{n+1} h(x_i, x_j) = \sum_{j=1}^{n} h(x_i, x_j) + h(x_i, x_{n+1}) = R_n(x_i) + h(x_i, x_{n+1})$$

2. Compute

$$R_{n+1}(x_{n+1}) = \sum_{j=1}^{n+1} h(x_{n+1}, x_j)$$

3. Compute

$$\hat{\Sigma}_{n+1} = \frac{1}{n} \sum_{i=1}^{n+1} R_{n+1}(x_i) R_{n+1}(x_i)' \quad and \quad \hat{\Sigma}_{n+1}^{-1}$$

4. For $k = 1$, compute

$$\bar{r}_{n+1}^{(1)} = R_{n+1}(x_1)$$

For $2 \leq k \leq n$, compute

$$\bar{r}_{n+1}^{(k)} = \frac{1}{k}\sum_{i=1}^{k} R_{n+1}(x_i)$$

$$= \frac{1}{k}\left[\sum_{i=1}^{k-1} R_{n+1}(x_i) + R_{n+1}(x_k)\right]$$

$$= \frac{(k-1)}{k}\bar{r}_{n+1}^{(k-1)} + R_{n+1}(x_k)$$

5. Compute

$$\hat{\Sigma}_{k,n+1}^{-1} = \frac{(n+1)k}{n+1-k}\hat{\Sigma}_{n+1}^{-1}.$$

6. Compute

$$r_{k,n+1} = \bar{r}_{n+1}^{(k)'}\hat{\Sigma}_{k,n+1}^{-1}\bar{r}_{n+1}^{(k)}$$

Steps1–3 only need to be executed once each time a new observation is added to the data set. All of the operations in these steps scale linearly in n. It is used the unpooled covariance estimator because the inverse of the covariance matrix only needs to be computed once to obtain test statistics for each possible split point $k = 1, \ldots, n_1$, and within-group rank vectors do not need to be computed. Steps 4-6 need to be executed for each $k = 1, \ldots, n$ each time a new observation, x_{n+1}, is collected. For each k, the number of operations necessary to execute steps 4-6 once does not depend on n or k. Thus, the entire update procedure scales linearly in n. Because the computation time scales linearly in n, it is able to employ large scale Monte Carlo simulation studies to obtain control limits and evaluate the performance of the proposed procedure.

3.2 Future Work

This will be our future algorithm. We will implement this algorithm to find the threshold for the new observations. This algorithm is computationally efficient and will take less time to assign new observations into any one of the groups that can be found using KMMD or any other test statistics.

4 References

[1] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Scholkopf and A. Smola, "A kernal tow-sample test," *Journal of Machine Learning Research (JMLR),* vol. 13, pp. 723-773, 2012.

[2] S. Adhikari and A. Platanios, "Summary and Discussion of: "A kernel Two-Sample Test"," *Statistics Journal Club,* 2014.

[3] D. M. Hawkins, P. Qiu and C. W. Kang, "The changepoint model for statistical process control," *Journal of Quality Technology,* pp. 355-366, 2003.

[4] P. Qiu, "Distribution-Free Multivariate Process Control Based on Log-Linear Modeling," *IEEE Transactions,* pp. 664-677, 2008.

[5] P. Qiu and D. M. Hawkins, "A Rank-Based Multivariate CUSUM Procedure," *Technometrics,* vol. 43(2), pp. 120-132, 2001.

[6] P. Qiu and D. M. Hawkins, "A nonparametric Multivariate Cumulative Sum Procedure for Detecting Shifts in All Directions," *The Statistician,* vol. 52(2), pp. 151-164, 2003.

[7] R. B. Crosier, "Multivariate Generalizations of Cumulative Sum Quality Control Schemes," *Technometrics,* pp. 291-303, 1988.

[8] C. A. Lowry, W. H. Woodall, C. W. Champ and S. E. Rigdon, "A multivariate Exponentially Weighted Moving Average Control Chart," *Technometrics,* pp. 46-53, 1992.

[9] K. D. Zamba and D. M. Howkins, "A multivariate Change-point Model for Statistical Process Control," *Technometrics ,* pp. 539-549, 2006.

[10] D. M. Hawkins and Q. Deng, "A nonparametric Change-Point Control Chart," *Journal of Quality Technology,* pp. 165-173, 2010.

5 Appendix

R-Code

5.1 Import Library

```
require(plyr)
require(MASS)
require(methods)
require(kernlab)
require(methods, quietly =T)
require(mixOmics, quietly =T)
require("Hotelling", quietly =T)
require(plyr)
require(MASS)
library("GSAR")
```

5.2 Functions

```
## Hh- return value for given Y and W matrix

hh <- function(y, w){
  y = matrix(y, nrow = 1)
  w = as.matrix(w)
  y %*% solve(w) %*% t(y)
}


cp <- function(x, keepobs = NULL){
  x = as.matrix(x)
```

```
  #print(x)
  N = nrow(x)
  ncol = ncol(x)
  if (ncol > 1) {ccm = getFunction(colMeans)
  }else {ccm = get0("mean", as.environment("package:base"))}

  if (is.null(keepobs)){
    keepobs = ceiling(n*0.1)
    if (keepobs < ncol) keepobs = ncol + 1
  }

  aa <- function(z, n, nobs){
    x = z[1:n,]
    y = z[(n+1):nobs,]
    wk = (var(x)*N + var(y)*(nobs - N)) / (nobs - 2)
    yk = sqrt(n *(nobs - N) / nobs) * (ccm(x) - ccm(y))
    tk = Hh(yk, wk)
    return(Tk)
  }

  ntest = c((keepobs+1):(n-keepobs))
  T2 = apply(ntest, aa, z = x, nobs = N)
  k = which.max(T2)
  return(c(T = T2[k], Change_Point = maxk +keepobs, T2 = T2))
}


Hh <- function(y, w){
  y = matrix(y, nrow = 1)
  w = as.matrix(w)
  y %*% solve(w) %*% t(y)
```

```
}

KK@<- function(x, keepobs = NULL){
  x = as.matrix(x)
  N = nrow(x)
  if (is.null(keepobs)){
    keepobs = ceiling(N*0.1)
  }

  aa <- function(z, n){
    x = z[1:n,]
    y = z[(n+1):nobs,]
    wk = (var(x)*n + var(y)*(nobs - n)) / (nobs - 2)
    yk = (n *(nobs - n) / nobs) * (colmeans(x) - colmeans(y))
    Tk = hh(yk, wk)
    return(Tk)
  }

  ntest = c((keepobs+1):(N-keepobs))
  stat = sapply(ntest, aa, z = x, nobs = N)
  mk = which.max(stat)
  return(c(T2Max = stat[mk], change_Point = mk+keepobs))
}


KK@ <- function(Kxx,Kyy, Kxy)
{
  m <- dim(Kxx)
  n <- dim(Kyy)
  sumKxx <- sum(Kxx)
  sumKyy <- sum(Kyy)
```

```
  sumKxy <- sum(Kxy)
  mmd1 <- sqrt(max(0,sumKxx/(m*m) + sumKyy/(n*n) - 2/m/n* sumKxy))
  return(list(mmd=mmd1))
}

##############################################################

# KMMD function
# return KMMD with 1st and 3rd order

kmfunc<- function(Kxx,Kyy, Kxy, alpha = 0.05)
{
  m <- dim(Kxx)
  n <- dim(Kyy)
  N <- max(m,n)
  M <- min(m,n)
  sumKxx <- sum(Kxx)
  if(m!=n)
    sumKxxM <- sum(Kxx[1:M,1:M])
  else
    sumKxxM <- sumKxx
  dgxx <- diag(Kxx)
  sumKxxnd <- sumKxx - sum(dgxx)
  R <- max(dgxx)
  rm <- max(dgxx[1:M])
  hu <- colSums(Kxx[1:M,1:M]) - dgxx[1:M]
  sumKyy <- sum(Kyy)
  if(m!=n)
    sumKyyM <- sum(Kyy[1:M,1:M])
  else
    sumKyyM <- sumKyy
```

```
  dgy <- diag(Kyy)
  sumKyynd <- sum(Kyy) - sum(dgyy)
  R <- max(R,dgyy)
  RM <- max(RM,dgyy[1:M]) # RM instead of R in original
  hu <- hu + colSums(Kyy[1:M,1:M]) - dgyy[1:M]
  sumKxy <- sum(Kxy)
  if (m!=n)
    sumKxyM <- sum(Kxy[1:M,1:M])
  else
    sumKxyM <- sumKxy
  dg <- diag(Kxy) # up to M only
  hu <- hu - colSums(Kxy[1:M,1:M]) - colSums(t(Kxy[1:M,1:M])) + 2*dg
  mmd1 <- sqrt(max(0,sumKxx/(m*m) + sumKyy/(n*n) - 2/m/n* sumKxy))
  mmd3 <- sum(hu)/M/(M-1)
  D1 <- 2*sqrt(RM/M)+sqrt(log(1/alpha)*4*RM/M)
  return(list(mmd1=mmd1,mmd3=mmd3,D1=D1))
}


## Find Change point location by calculating maximum KMMD for data

ckpd <- function(x, keepobs = NULL){
  x = as.matrix(x)
  N = nrow(x)
  if (is.null(keepobs)){
    keepobs = ceiling(N*0.1)
  }
  aa <- function(z, n, rbf = rbf){
    nobs = nrow(z)
    ncoll = ncol(z)
    x = z[1:n,1:ncoll]
```

```
    y = z[(n+1):nobs,1:ncoll]
    stat <- KK@(Kxx=kernelMatrix(rbf, x), Kyy =kernelMatrix(rbf, y), K
xy=kernelMatrix(rbf, x,y))
    return(stat$mmd)
  }
  sig.opt <- sigest(x, scaled = FALSE)[2]
  rbf <- rbfdot(sigma = sig.opt)

  ntest = c((keepobs+1):(N-keepobs))
  stat = sapply(ntest, aa, z = x, rbf = rbf)
  mk = which.max(stat)
  return(list(stat = stat, change.Point = mk+keepobs))
}


nokm <- function(newobs, x, y){
  ncoll = ncol(x)
  newobs = matrix(newobs, ncol = ncoll)
  x = as.matrix(x)
  y = as.matrix(y)
  nnew = nrow(newobs)
  ns = c(1:nnew)
  sig.opt <- sigest(rbind(x,y), scaled = FALSE)[2]
  rbf <- rbfdot(sigma = sig.opt)

  aa <- function(i, z, x, y, rbf = rbf){
    a = z[i,]
    s1 = KK@(Kxx=kernelMatrix(rbf, rbind(x,a)), Kyy =kernelMatrix(rbf,
y), Kxy=kernelMatrix(rbf, rbind(x,a),y))
    s2 = KK2(Kxx=kernelMatrix(rbf, x), Kyy =kernelMatrix(rbf, rbind(y,
a)), Kxy=kernelMatrix(rbf, x,rbind(y,a)))
```

```
    if (s1$mmd > s2$mmd) c = 1
    else c = 2
    return(c)
  }
  co = sapply(ns, aa, z = newobs, x = x, y = y, rbf = rbf)

  return(co)

}


fkm<-function(k1,k2,k3){
  kmmd2<-0
  m<-nrow(k1)
  for(i in 1:m){
    t<-0
    for(j in 1:m){
      if( I != J){
        t <- t + k1[i,j]+k3[i,j]-k2[i,j]-k3[j,i]
      }# end if
    } #end loop j
    kmmd2<-kmmd2+t/(m*(m-1))
  } # end loop i
  return(kmmd2)
}

#Kernel function

kkrnl<- function(x1, y = NULL){
  if (!is.null(y)){  x1 = x1-y}
  if(all(x1 == 0)){return(0)}
  K<- sum(x1 / base::norm(as.matrix(x1),type = 'f'))
```

```
  return(K)
}

Knl1 <- function(x, y, R=499, testfun=fkm) {
  z <- rbind(x, y)  # pooled sample
  # Create kernel matrix
  sig.opt <- sigest(z, scaled = FALSE)[2]
  rbf <- rbfdot(sigma = sig.opt)
  myfun <- function(a, b,c) (unname(testfun(a, b,c)))
  #set.seed(123)
  DoIt <- function() {
    i <- sample(nrow(z), nrow(x))
    myfun( a=kernelMatrix(rbf, z[i,]), b =kernelMatrix(rbf, z[-i,]), c
=kernelMatrix(rbf, z[i,],z[-i,]))
  }
  pstats <- replicate(R, DoIt())
  stat <- myfun(a=kernelMatrix(rbf, x), b =kernelMatrix(rbf, y), c=ker
nelMatrix(rbf, x,y))
  hist(pstats, col="azure", main="Empirical distribution under H0")
  abline(v=stat, col="red", lty=2)
  p.v <-mean(c(stat, pstats) >= stat)
  res <- list(p.v=p.v, sig.opt=sig.opt, R=R, stat=stat, pstats=pstats,
sum=sum(c(stat, pstats) >= stat))
  return(res)
}


myheatmap <-function(z,my.xlab=NULL, my.ylab=NULL, my.main){
  rc <- cm.colors(nrow(z))
  cc <- cm.colors(ncol(z))
  hv <- heatmap(z, col = cm.colors(256), scale = "column",
```

```
                Rowv=NA, Colv=NA,
                RowSideColors = rc,
                ColSideColors = cc,
                margins = c(5,10),
                xlab = my.xlab, ylab =  my.ylab,
                main = my.main)
}


kkrnl<- function(x1, y = NULL){
  if (!is.null(y)){  x1 = x1-y}
  if(all(x1 == 0)){return(0)}
  K<- sum(x1 / base::(as.matrix(x1),type = 'f'))
  return(K)
}

kcalculator <- function(X, Y = NULL){
  if (is.null(Y)){
  N<-dim(X)
  K<-matrix(0,N,N)
    for(i in 1:N){
        K[i,]<-apply(sweep(X, 2, unlist(X[i,])), 1, kernel11)
    }
  return(K)
  }else{
    N1<-dim(X)
    N2 = dim(Y)
    K<-matrix(0,N1,N2)
    for(i in 1:N2){
      K[,i]<-sapply(sweep(X, 2, unlist(Y[i,])), 1, kernel11)
    }
```

```
    return(K)
  }
}
```

5.3 Calling functions and get the results

```
## Repeat Max T^2 for N (SimNum) times and return as a Data Frame cont
aining two columns (T2Max and Change_Point (position))

set.seed(2345)
alpha = 0.05
p = 1
SimNum = 20
n =8
mu = rep(0, p)
sigma =  diag(x = 1, nrow = p, ncol = p)
a = rdply(SimNum, Cp  (mvrnorm(n = N, mu = mu, Sigma = sigma), keepobs
= p+1))
b = a[,4:ncol(a)]
print(a)
```

```
##    .n      T2Max Change_Point          T21        T22         T23
## 1   1  1.8829980            5 1.976256e-01 1.18928439 1.882998043
## 2   2  2.0967854            6 8.684432e-01 1.61662263 0.372369886
## 3   3  0.2575637            4 3.487597e-02 0.25756367 0.021267580
## 4   4  0.1594644            5 7.142481e-02 0.03850383 0.159464395
## 5   5 14.5026479            3 1.450265e+01 3.46340996 3.202057352
## 6   6  0.6875516            3 6.875516e-01 0.42903856 0.197372528
## 7   7  7.8049864            4 1.551667e+00 7.80498636 5.333896699
## 8   8  2.1744871            6 7.730171e-01 0.01824885 0.896953217
## 9   9  0.1587509            3 1.587509e-01 0.11077765 0.042786125
```

```
## 10 10  2.1665637           6 1.413523e-01 0.05266637 0.323061132
## 11 11  4.1975502           3 4.197550e+00 0.88762688 0.001032129
## 12 12  2.4023162           3 2.402316e+00 1.02088929 0.929425476
## 13 13  0.4402615           3 4.402615e-01 0.34262963 0.308200510
## 14 14  2.0852057           3 2.085206e+00 0.25522824 0.050946226
## 15 15  1.0769353           5 1.082131e-03 0.01762906 1.076935329
## 16 16  2.1001482           6 1.873204e-02 0.20743859 0.500004178
## 17 17  1.1985305           5 6.563868e-02 0.00330262 1.198530481
## 18 18  1.5399035           6 7.576673e-01 0.16630984 0.007474626
## 19 19  0.9173667           5 1.180747e-01 0.03966890 0.917366748
## 20 20  0.1913494           5 4.278521e-04 0.13630954 0.191349432
##           T24
## 1  0.60607440
## 2  2.09678543
## 3  0.02298761
## 4  0.01403302
## 5  0.74381224
## 6  0.60546422
## 7  6.10836265
## 8  2.17448712
## 9  0.06863711
## 10 2.16656367
## 11 0.03845976
## 12 0.50956755
## 13 0.18243332
## 14 0.22399582
## 15 0.09089440
## 16 2.10014819
## 17 0.19930501
## 18 1.53990350
```

```
## 19 0.36210154
## 20 0.04349802

quant = list(c(NULL, NULL))
for (i in 1:ncol(b)){
  quant[[i]] = c(as.integer(p+1+i), quantile(b[,i], probs = 0.95, name
s = FALSE, type = 1))
  b = b[-which((b[,i]) >= quant[[i]][2]),]
}
bb = t(as.data.frame(quant))
row.names(bb) = NULL
print(bb)

##      [,1]     [,2]
## [1,]    3 4.197550
## [2,]    4 7.804986
## [3,]    5 1.882998
## [4,]    6 2.174487
```

5.4 Change Point KMMD File

```
x <- matrix(rnorm(100, 10, 5),50)
y <- matrix(rnorm(200, 20, 5),100)
z <- matrix(rnorm(100, 15, 10), ncol = ncol(x))


a = rdply(20, ckpd(mvrnorm(n = 20, mu = c(11,15,32),
                           Sigma =  matrix(c(1, .6, .7, .6, 1, .75,
.7, .75,1),
                                           nrow = 3, byrow = TRUE))
))
print(a)
```

```
##    .n     T2Max change_Point
## 1   1 12.242666           16
## 2   2  9.213460           14
## 3   3  4.582759           18
## 4   4  4.565112           14
## 5   5  5.886297           12
## 6   6  5.975853            4
## 7   7  3.542667            4
## 8   8  6.244345            6
## 9   9 13.586066           16
## 10 10 11.481422            3
## 11 11  2.987013            3
## 12 12 29.403018            3
## 13 13 16.798784            7
## 14 14  4.441895           18
## 15 15 10.897145           17
## 16 16  8.596549           18
## 17 17 11.264867            3
## 18 18 12.146210           17
## 19 19  6.711628           17
## 20 20  5.656718            8
```

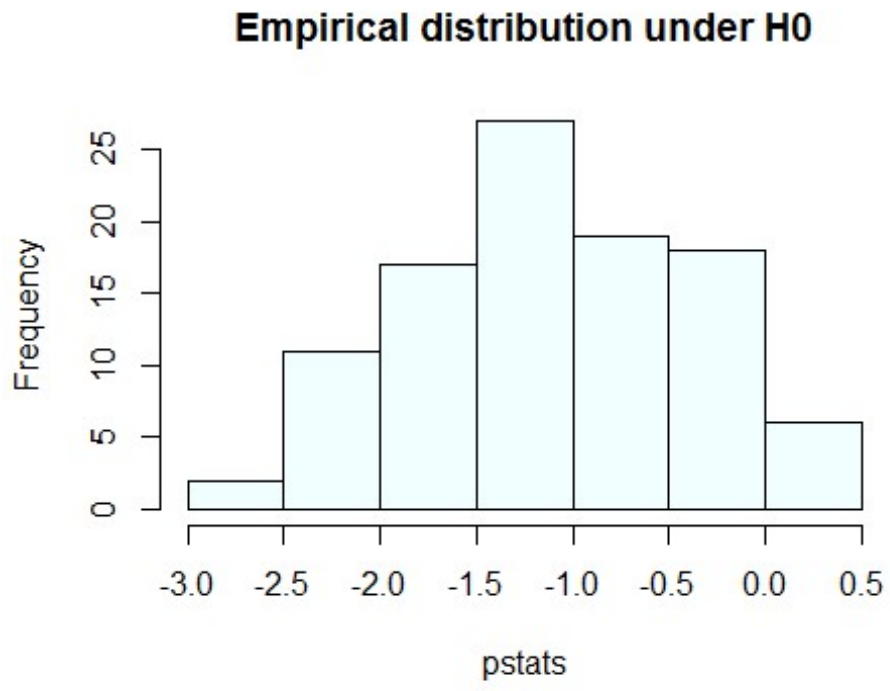
```
bb = ckpd(rbind(x,y))
```

```
nokm(z, x,y)
```

```
##  [1] 2 1 2 1 2 1 2 1 1 1 2 2 1 2 1 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 2
2 2 2 2
## [36] 1 1 2 2 1 1 2 1 2 2 1 2 1 1 2
```

5.5 Functions_kmmd_sent file

```
#
# Compute kernel MMD test

x <- matrix(rnorm(100, 10, 5),20)
y <- matrix(rnorm(100, 20, 5),20)

Knl1(x, y, R=100)
```




```
## $p.v
## [1] 0.00990099
##
## $sig.opt
##         50%
## 0.001912114
##
## $R
```

```
## [1] 100
##
## $stat
## [1] 3.522715
##
## $pstats
##   [1] -0.573244605 -1.544318055 -2.858193959 -1.175930731 -1.240145
971
##   [6] -1.593102055 -0.467843331  0.051493695 -0.955575516 -0.409234
591
##  [11] -2.429147879 -0.313282578 -2.083194558 -1.296442973 -0.767396
209
##  [16] -1.735550264 -2.490883554 -1.234734515 -0.991722215 -0.237391
620
##  [21] -1.335606365 -1.152808618 -0.370974485 -1.348666213 -1.961935
255
##  [26] -0.840291821  0.065800997 -2.046440293 -0.754747608 -2.291439
852
##  [31] -1.376248446 -0.499649547 -0.547827900 -1.336667246 -0.684022
673
##  [36] -2.206328016 -1.113366184 -1.310434025 -0.587861193 -0.169762
582
##  [41] -1.085419829 -0.034644588 -1.052660067 -0.803948860  0.357001
366
##  [46] -0.929056438 -0.088465891 -0.510154527 -0.671429605 -2.017312
873
##  [51] -0.931204346 -1.123687588 -0.298417606 -0.412319236 -1.446975
174
##  [56] -1.534812295 -1.687899543 -1.326247138 -0.721851354 -0.114043
001
##  [61] -0.268721828 -0.568919190 -1.094618045 -1.313777926 -0.723052
```

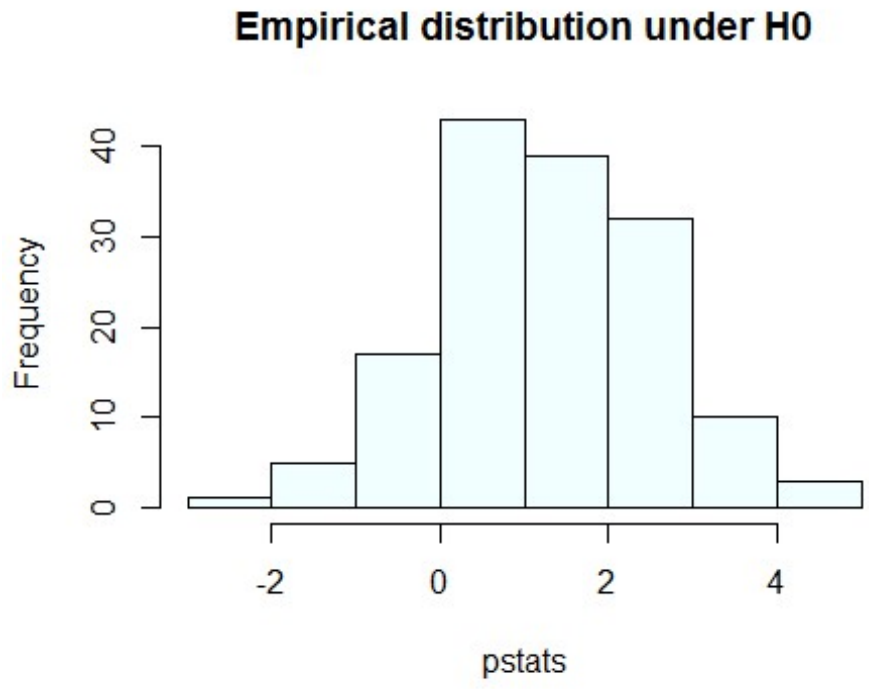
```
391
##  [66] -1.949709774 -1.844523372 -1.043960010 -1.874126441  0.151099
774
##  [71] -1.177840883 -1.043986457 -0.126754266 -1.424391797 -0.435615
483
##  [76] -1.403777453  0.006327668 -1.569261905 -0.566617363 -2.094851
301
##  [81] -0.361778722 -2.072303798 -1.714773126 -1.481449196 -2.552269
372
##  [86] -1.848687159 -1.961517241 -0.465407998 -1.185238776 -2.421271
535
##  [91] -1.420836524 -1.518451899 -0.619272939 -1.989355498 -1.507808
399
##  [96] -1.945341781 -1.484631733  0.006947217 -2.322984383 -0.489397
585
##
## $sum
## [1] 1


#########################################################################
#######
data(nutrimouse)

mydata.gene <- nutrimouse$gene

mydata.FA <- nutrimouse$lipid

gene.names <- names(mydata.gene)
FA.names <- names(mydata.FA)

ind.diet <- nutrimouse$diet
```

```
ind.genotype <- nutrimouse$genotype

ind.gen <- list()
for (i in levels(ind.genotype)) {
  ind.gen[[i]] <- which(ind.genotype==i)
}

ind.diet1 <- list()
for (i in levels(ind.diet)) {
  ind.diet1[[i]] <- which(ind.diet==i)
}

gene.names.sel.1 <- c("ACBP","AOX","BIEN","CPT2","CYP4A10",
                      "HPNCL","L.FABP","PECI","PMDCI", "THIOL",
                      "mHMGCoAS", "CACP","Tpalpha", "Tpbeta", "CYP4A14"
,"ACOTH")

pos.genes <- match(x=c(gene.names.sel.1), table=gene.names)

pos.FA <- match(x=FA.names[19:21], table=FA.names)

mydata.gene.sel <- mydata.gene[,pos.genes]
mydata.FA.sel <-mydata.FA[,pos.FA]

#########################################################################
#######

x <- as.matrix(mydata.gene.sel[ind.gen$wt,])
y <- as.matrix(mydata.gene.sel[ind.gen$ppar,])

#########################################################################
```

```
#######
################################################################################
#######

myheatmap(rbind(x,y), my.main="Gene expression")
```


```
################################################################################
#######

(hotelling.test(x, y))

## Test stat:  45.787
## Numerator df:  16
## Denominator df:  23
## P-value:  1.195e-13

source("http://bioconductor.org/biocLite.R")
```

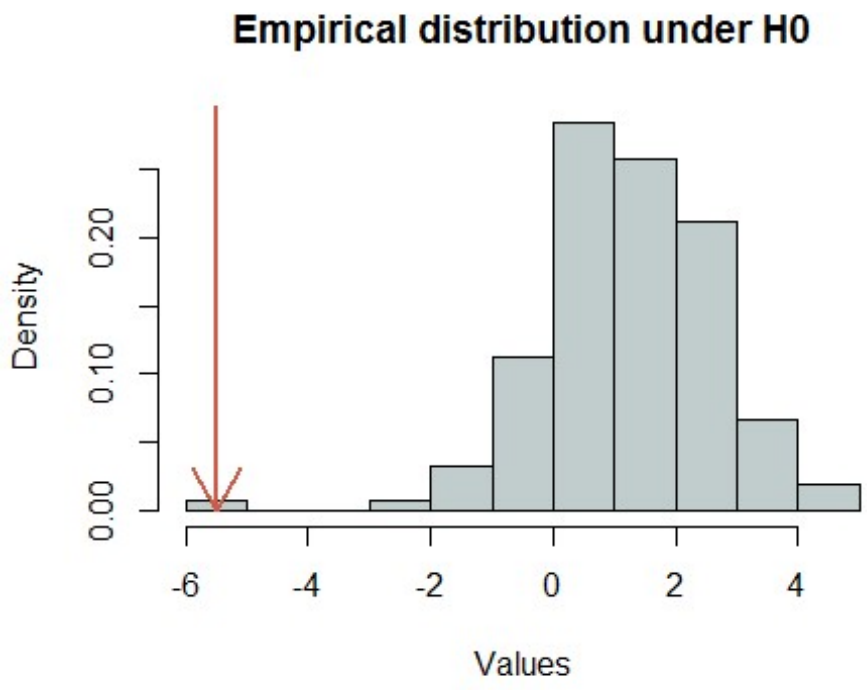
```
## Bioconductor version 3.4 (BiocInstaller 1.24.0), ?biocLite for help

## A new version of Bioconductor is available after installing the mos
t
##   recent version of R; see http://bioconductor.org/install


result <- KStest(object=cbind(t(x),t(y)), group=c(rep(1,20),rep(2,20))
)
result

## [1] 0.000999001

result <- WWtest(object=cbind(t(x),t(y)), group=c(rep(1,20),rep(2,20))
)
result

## [1] 0.000999001

#########################################################################
#######
# Kernel Maximum Mean Discrepancy: asymptotic distribution
#
#########################################################################
#######


source("C:/Users/ka746940/Desktop/UCF/STA 6908 - Edgard Maboudou/R cod
e/Functions_kmmd_sent.R")

time1 <- proc.time()
#results.kmmd.1 <- by (Tr_C_Y.filt_1[sel_row,],ffactor, kmmd.my1)
results.nl1 <- Knl1(x,y,R=150)
```



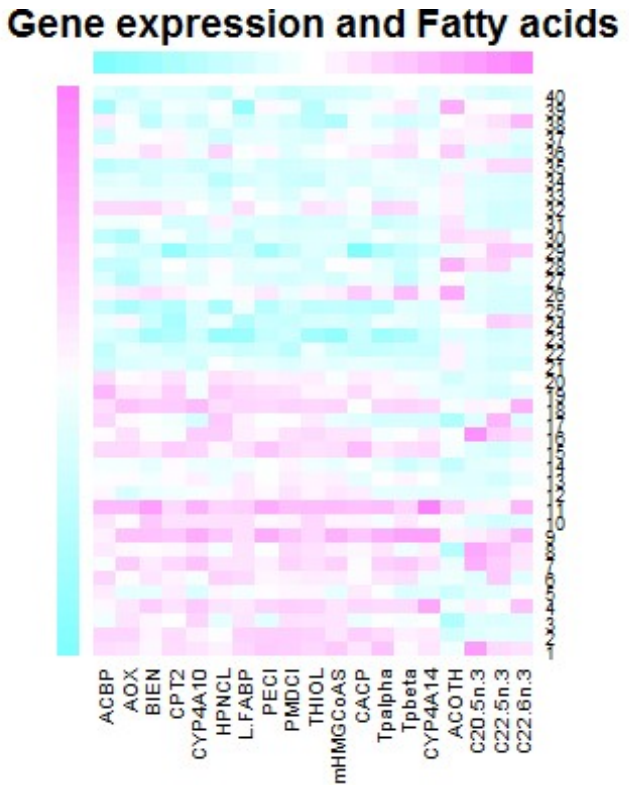

```
proc.time() - time1
```

```
##    user  system elapsed
##    4.43    0.00    4.46
```

```
hist(c(results.nl1[["pstats"]],results.nl1[["stat"]]), col="azure3",fr
eq=FALSE,
     main= "Empirical distribution under H0", xlab="Values")

x0=results.nl1[["stat"]]

arrows(x0=x0, y0=1.5, x1=x0, y1= 0, col="coral3", lty=1, lwd=2)
text(x=results.nl1[["stat"]], y=1.5, pos=3, labels="stat")
```



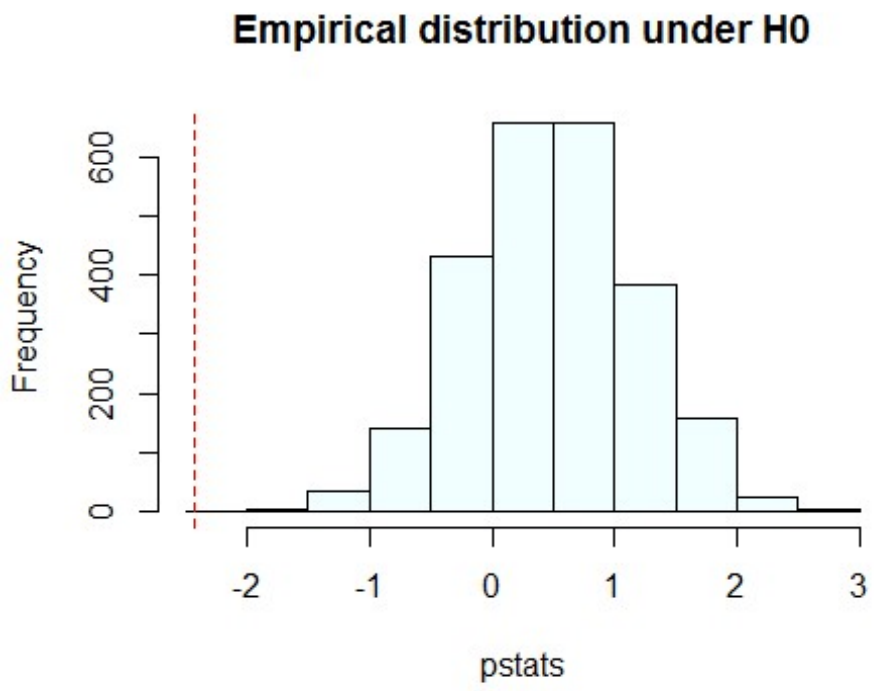

```
#########################################################################
#######

x1 <- as.matrix(mydata.gene.sel[ind.gen$wt,])
y1 <- as.matrix(mydata.gene.sel[ind.gen$ppar,])

x2 <- as.matrix(mydata.FA.sel[ind.gen$wt,])
y2 <- as.matrix(mydata.FA.sel[ind.gen$ppar,])

x <- cbind(x1,x2)
y <- cbind(y1,y2)

#########################################################################
#######

myheatmap(rbind(x,y), my.main="Gene expression and Fatty acids")
```




```
# Hotelling test

(hotelling.test(x, y))

## Test stat:  34.101
## Numerator df:  19
## Denominator df:  20
## P-value:  2.936e-11

result <- KStest(object=cbind(t(x),t(y)), group=c(rep(1,20),rep(2,20))
)
result

## [1] 0.001998002

result <- WWtest(object=cbind(t(x),t(y)), group=c(rep(1,20),rep(2,20))
)
result
```

```
## [1] 0.000999001
```

```
time1 <- proc.time()
results.nl1 <- Knl1(x,y,R=2499)
```




```
proc.time() - time1
```

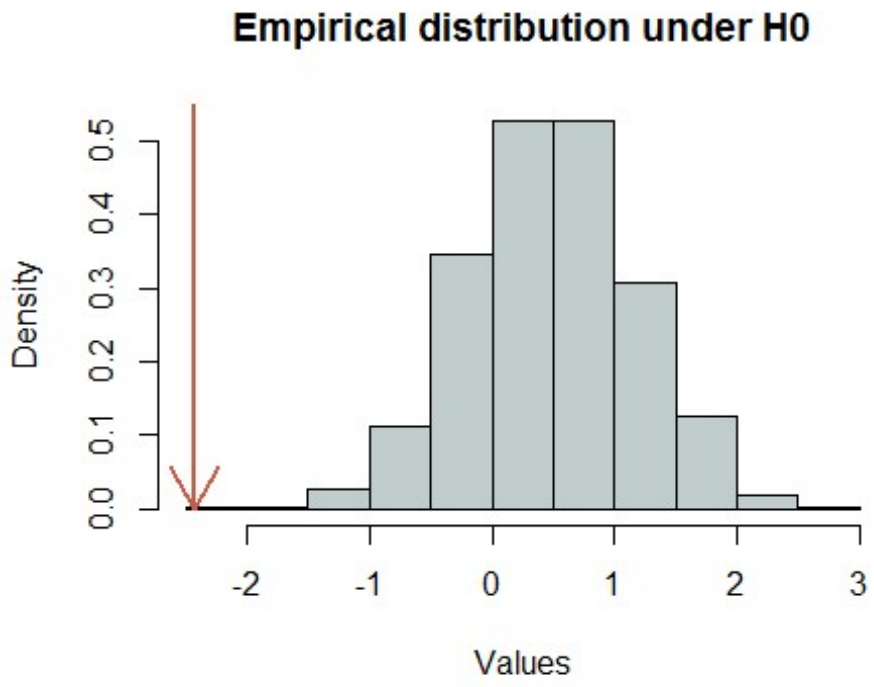
```
##    user  system elapsed
##   72.95    0.02   73.04
```

```
results.nl1$p.v  #p-value
```

```
## [1] 1
```

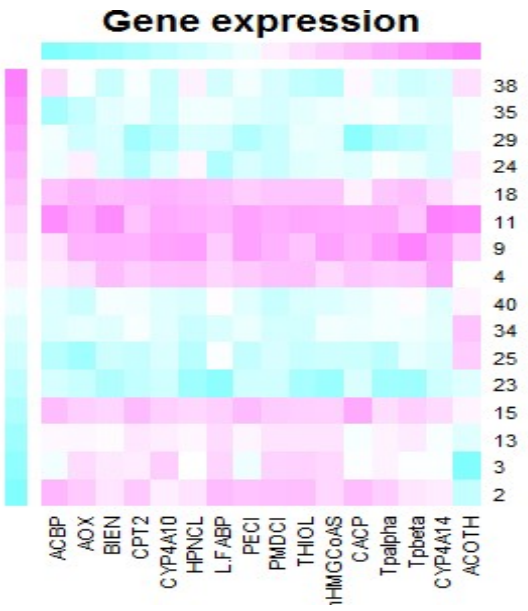
```
hist(c(results.nl1[["pstats"]],results.nl1[["stat"]]), col="azure3",fr
eq=FALSE,
     main= "Empirical distribution under H0", xlab="Values")
```

```
x0=results.nl1[["stat"]]

arrows(x0=x0, y0=1.5, x1=x0, y1= 0, col="coral3", lty=1, lwd=2)
text(x=results.nl1[["stat"]], y=1.5, pos=3, labels="stat")
```

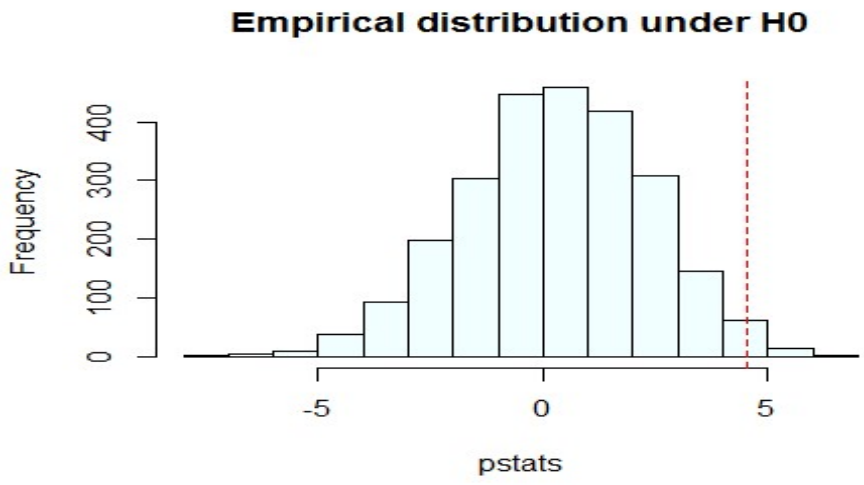



```
#############################################################################
#######

#############################################################################
#######
x <- as.matrix(mydata.gene.sel[ind.diet1$sun,])
y <- as.matrix(mydata.gene.sel[ind.diet1$fish,])

#############################################################################
#######
```

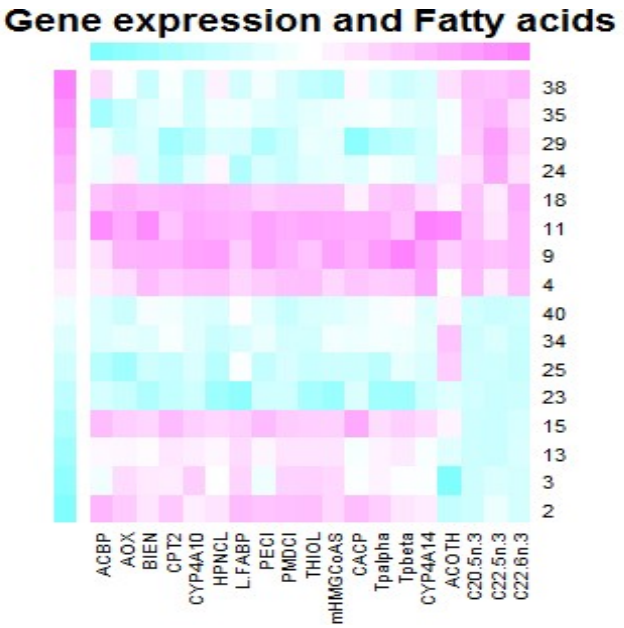
```
myheatmap(rbind(x,y), my.main="Gene expression")
```



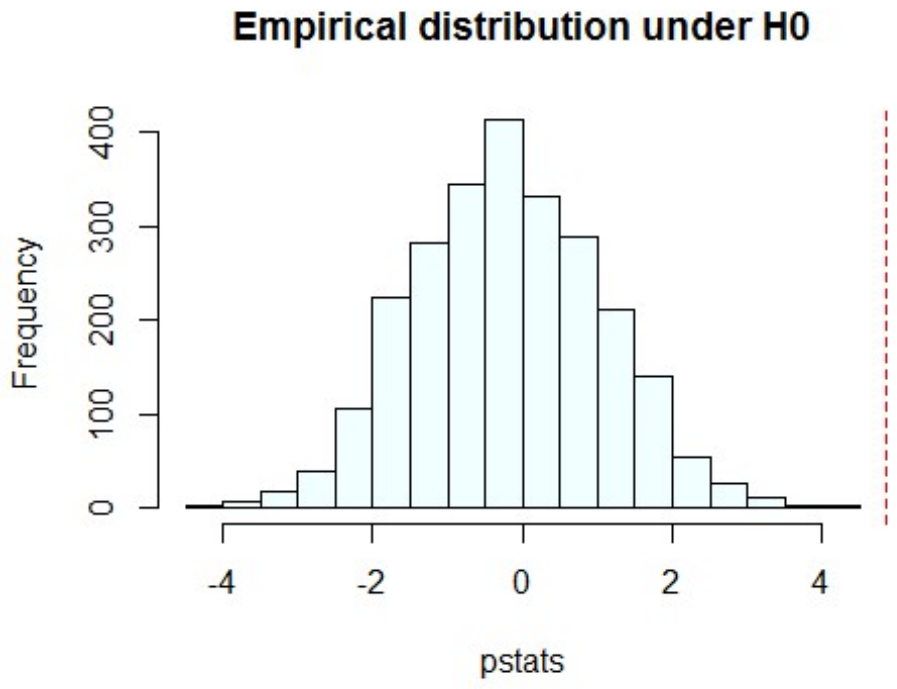

```
result <- KStest(object=cbind(t(x),t(y)), group=c(rep(1,8),rep(2,8)))
result

## [1] 0.08591409

result <- WWtest(object=cbind(t(x),t(y)), group=c(rep(1,8),rep(2,8)))
result

## [1] 0.3906094

time1 <- proc.time()
results.nl1 <- Knl1(x,y,R=2499)
```




```
proc.time() - time1

##    user  system elapsed
##   19.84    0.00   19.86

results.nl1$p.v  #p-value

## [1] 0.0124

#########################################################################
#######

x1 <- as.matrix(mydata.gene.sel[ind.diet1$sun,])
y1 <- as.matrix(mydata.gene.sel[ind.diet1$fish,])

x2 <- as.matrix(mydata.FA.sel[ind.diet1$sun,])
y2 <- as.matrix(mydata.FA.sel[ind.diet1$fish,])

x <- cbind(x1,x2)
y <- cbind(y1,y2)

myheatmap(rbind(x,y), my.main="Gene expression and Fatty acids")
```




```
# Hotelling test

#(hotelling.test(x, y))  # error number of samples < number of variabl
es

result <- KStest(object=cbind(t(x),t(y)), group=c(rep(1,8),rep(2,8)))
result

## [1] 0.000999001

result <- WWtest(object=cbind(t(x),t(y)), group=c(rep(1,8),rep(2,8)))
result

## [1] 0.000999001


time1 <- proc.time()
results.nl1 <- Knl1(x,y,R=2499)
```




```
proc.time() - time1

##    user  system elapsed
##   20.06    0.00   20.09

results.nl1$p.v  #p-value

## [1] 4e-04

datapath = 'C:/Data/'
df = read.table(paste(datapath, 'pb2.txt', sep = ''))
x = df[df$V1 == 1, 2:5]
y = df[df$V1 == 2, 2:5]


x <- matrix(runif(300),100)
y <- matrix(runif(300)+10,100)
a = kmmd(x, y)

## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
aa = kernelMatrix(kernel11, as.matrix(x))

k = kcalculator(x, y)

kmmd(x, y)

## Using automatic sigma estimation (sigest) for RBF or laplace kernel

## Kernel Maximum Mean Discrepancy object of class "kmmd"
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1.03664381559699
##
##
##  H0 Hypothesis rejected :  TRUE
##  Rademacher bound :  0.546163676520489
##
##  1st and 3rd order MMD Statistics :  1.12666890628934 1.26200285292
85

##  calculates the kernel maximum mean discrepancy for samples from tw
o distributions

setGeneric("kmmd",function(x,...) standardGeneric("kmmd"))

## [1] "kmmd"

setMethod("kmmd", signature(x = "matrix"),
          function(x, y, kernel="rbfdot",kpar="automatic", alpha = 0.0
5, asymptotic = FALSE,  replace = TRUE, ntimes = 150, frac = 1,  ...)
          {
            x <- as.matrix(x)
            y <- as.matrix(y)
```

```
            res <- new("kmmd")


            if(is.character(kernel)){
              kernel <- match.arg(kernel,c("rbfdot","polydot","tanhdot
","vanilladot","laplacedot","besseldot","anovadot","splinedot","matrix
"))

              if(kernel == "matrix")
                if(dim(x)[1]==dim(x)[2])
                  return(kmmd(x= as.kernelMatrix(x), y = y, Kxy = as.k
ernelMatrix(x)%*%y, alpha = 0.05, asymptotic = FALSE,  replace = TRUE,
ntimes = 100, frac = 1,  ...))
              else
                stop(" kernel matrix not square!")

              if(is.character(kpar))
                if((kernel == "tanhdot" || kernel == "vanilladot" || k
ernel == "polydot"|| kernel == "besseldot" || kernel== "anovadot"|| ke
rnel=="splinedot") &&  kpar=="automatic" )
                {
                  cat (" Setting default kernel parameters ","\n")
                  kpar <- list()
                }
            }

            if (!is.function(kernel))
              if (!is.list(kpar)&&is.character(kpar)&&(kernel == "lapl
acedot"|| kernel=="rbfdot")){
                kp <- match.arg(kpar,"automatic")
                if(kp=="automatic")
```

```
                  kpar <- list(sigma=sigest(rbind(x,y),scaled=FALSE)[2
])
                cat("Using automatic sigma estimation (sigest) for RBF
or laplace kernel","\n")

              }
            if(!is(kernel,"kernel"))
            {
              if(is(kernel,"function")) kernel <- deparse(substitute(k
ernel))
              kernel <- do.call(kernel, kpar)
            }

            if(!is(kernel,"kernel")) stop("kernel must inherit from cl
ass `kernel'")

            m <- dim(x)
            n <- dim(y)

            N <- max(m,n)
            M <- min(m,n)

            Kxx <- kernelMatrix(kernel,x)
            Kyy <- kernelMatrix(kernel,y)
            Kxy <- kernelMatrix(kernel,x,y)

            resmmd <- .submmd(Kxx, Kyy, Kxy, alpha)

            H0(res) <- (resmmd$mmd1 > resmmd$D1)
            Radbound(res) <- resmmd$D1
            Asymbound(res) <- 0
```

```
            mmdstats(res)[1] <- resmmd$mmd1
            mmdstats(res)[2] <- resmmd$mmd3

            if(asymptotic){
              boundA <- .submmd3bound(Kxx, Kyy, Kxy, alpha, frac, ntim
es, replace)

              AsympH0(res) <- (resmmd$mmd3 > boundA)
              Asymbound(res) <- boundA
            }

            kernelf(res) <- kernel
            return(res)
          })

## [1] "kmmd"

setMethod("kmmd",signature(x="list"),
          function(x, y, kernel="stringdot",kpar=list(type="spectrum",
length=4), alpha = 0.05, asymptotic = FALSE,  replace = TRUE, ntimes =
150, frac = 1,  ...)
          {

            if(!is(kernel,"kernel"))
            {
              if(is(kernel,"function")) kernel <- deparse(substitute(k
ernel))
              kernel <- do.call(kernel, kpar)
            }
            if(!is(kernel,"kernel")) stop("kernel must inherit from cl
ass `kernel'")
```

```
            Kxx <- kernelMatrix(kernel,x)
            Kyy <- kernelMatrix(kernel,y)
            Kxy <- kernelMatrix(kernel,x,y)

            ret <- kmmd(x=Kxx,y = Kyy,Kxy=Kxy, alpha=alpha, asymptotic
= asymptotic, replace = replace, ntimes = ntimes, frac= frac)

            kernelf(ret) <- kernel

            return(ret)

          })
## [1] "kmmd"
setMethod("kmmd",signature(x="kernelMatrix"), function (x, y, Kxy, alp
ha = 0.05, asymptotic = FALSE,  replace = TRUE, ntimes = 100, frac = 1
,  ...)
{
  res <- new("kmmd")
  resmmd <- .submmd(x, y, Kxy, alpha)
  H0(res) <- (resmmd$mmd1 > resmmd$D1)
  Radbound(res) <- resmmd$D1
  Asymbound(res) <- 0
  mmdstats(res)[1] <- resmmd$mmd1
  mmdstats(res)[2] <- resmmd$mmd3

  if(asymptotic){
    boundA <- .submmd3bound(x, y, Kxy, alpha, frac, ntimes, replace)

    AsympH0(res) <- (resmmd$mmd1 > boundA)
    Asymbound(res) <- boundA
```

```
  }
  kernelf(res) <- " Kernel matrix used as input."
  return(res)

})
```

```
## [1] "kmmd"
```

```
.submmd <- function(Kxx,Kyy, Kxy, alpha)
{

  m <- dim(Kxx)
  n <- dim(Kyy)

  N <- max(m,n)
  M <- min(m,n)

  sumKxx <- sum(Kxx)

  if(m!=n)
    sumKxxM <- sum(Kxx[1:M,1:M])
  else
    sumKxxM <- sumKxx

  dgxx <- diag(Kxx)

  sumKxxnd <- sumKxx - sum(dgxx)
  R <- max(dgxx)
  RM <- max(dgxx[1:M])
  hu <- colSums(Kxx[1:M,1:M]) - dgxx[1:M]

  sumKyy <- sum(Kyy)
```

```
  if(m!=n)
    sumKyyM <- sum(Kyy[1:M,1:M])
  else
    sumKyyM <- sumKyy

  dgyy <- diag(Kyy)

  sumKyynd <- sum(Kyy) - sum(dgyy)
  R <- max(R,dgyy)
  RM <- max(RM,dgyy[1:M]) # RM instead of R in original
  hu <- hu + colSums(Kyy[1:M,1:M]) - dgyy[1:M]

  sumKxy <- sum(Kxy)
  if (m!=n)
    sumKxyM <- sum(Kxy[1:M,1:M])
  else
    sumKxyM <- sumKxy

  dg <- diag(Kxy) # up to M only
  hu <- hu - colSums(Kxy[1:M,1:M]) - colSums(t(Kxy[1:M,1:M])) + 2*dg #
one sided sum

  mmd1 <- sqrt(max(0,sumKxx/(m*m) + sumKyy/(n*n) - 2/m/n* sumKxy))
  mmd3 <- sum(hu)/M/(M-1)
  D1 <- 2*sqrt(RM/M)+sqrt(log(1/alpha)*4*RM/M)

  return(list(mmd1=mmd1,mmd3=mmd3,D1=D1))
}


.submmd3bound <- function(Kxx,Kyy, Kxy, alpha, frac, ntimes, replace)
```

```
{
  ## implements the bootstrapping approach to the MMD3 bound by shuffl
ing
  ## the kernel matrix
  ##  frac   : fraction of data used for bootstrap
  ##  ntimes : how many times MMD is to be evaluated

  m <- dim(Kxx)[1]
  n <- dim(Kyy)[1]

  M <- min(m,n)
  N <- max(m,n)

  poslabels <- 1:m
  neglabels <- (m+1):(m+n)

  ## bootstrap
  bootmmd3 <- rep(0,ntimes)

  for (i in 1:ntimes)
  {
    nsamples <- ceiling(frac*min(m,n))
    xinds <- sample(1:m,nsamples,replace=replace)
    yinds <- sample(1:n,nsamples,replace=replace)
    newlab <- c(poslabels[xinds],neglabels[yinds])
    samplenew <- sample(newlab, length(newlab), replace=FALSE)
    xinds <- samplenew[1:nsamples]
    yinds <- samplenew[(nsamples+1):length(samplenew)]

    newm <- length(xinds)
    newn <- length(yinds)
```

```
    newM <- min(newm,newn)

    ##get new kernel matrices (without concat to big matrix to save me
mory)
    xind1 <- xinds[xinds<=m]
    xind2 <- xinds[xinds>m]- m
    yind1 <- yinds[yinds<=m]
    yind2 <- yinds[yinds>m]-m

    ##Kxx (this should be implemented with kernelMult for memory effic
iency)
    nKxx <- rbind(cbind(Kxx[xind1,xind1],Kxy[xind1,xind2]), cbind(t(Kx
y[xind1,xind2]),Kyy[xind2,xind2]))
    dgxx <- diag(nKxx)
    hu <- colSums(nKxx[1:newM,1:newM]) - dgxx[1:newM]

    rm(nKxx)

    nKyy <- rbind(cbind(Kxx[yind1,yind1],Kxy[yind1,yind2]), cbind(t(Kx
y[yind1,yind2]), Kyy[yind2,yind2]))
    dgyy <- diag(nKyy)
    hu <- hu + colSums(nKyy[1:newM,1:newM]) - dgyy[1:newM]
    rm(nKyy)

    nKxy <- rbind(cbind(Kxx[yind1,xind1],Kxy[yind1,xind2]), cbind(t(Kx
y[xind1,yind2]),Kyy[yind2,xind2]))
    dg <- diag(nKxy)
    hu <- hu - colSums(nKxy[1:newM,1:newM]) - colSums(t(nKxy[1:newM,1:
newM])) + 2*dg
    rm(nKxy)

    ## now calculate mmd3
```

```
    bootmmd3[i] <- sum(hu)/newM/(newM-1)
  }

  bootmmd3 <- sort(bootmmd3, decreasing=TRUE);
  aind <- floor(alpha*ntimes) ## better less than too much (-> floor);

  ## take threshold in between aind and the next smaller value:
  bound <- sum(bootmmd3[c(aind,aind+1)])/2;
  return(bound)

}
setMethod("show","kmmd",
          function(object){

            cat("Kernel Maximum Mean Discrepancy object of class \"kmm
d\"","\n","\n")

            show(kernelf(object))

            if(is.logical(object@H0)){
              cat("\n")
              cat("\n","H0 Hypothesis rejected : ", paste(H0(object)))
              cat("\n","Rademacher bound : ", paste(Radbound(object)))
            }

            cat("\n")

            if(Asymbound(object)!=0){
              cat("\n","H0 Hypothesis rejected (based on Asymptotic bo
und): ", paste(AsympH0(object)))
              cat("\n","Asymptotic bound : ", paste(Asymbound(object))
```

```
)
            }

            cat("\n","1st and 3rd order MMD Statistics : ", paste( mmd
stats(object)))
            cat("\n")
          })

## [1] "show"
```