```
#hw2
library(tidyverse)
library(R.matlab)

#problem 2
#program to compute and display geodesic path between any two points
#looking at equation 2.8 in Anuj's book
#working with S^infinty, points are functions on [0,1]

#function to build test functions
f_t <- function(t, a, b){
  part1 <- t^(a-1)
  part2 <- part1*(1-t)^(b-1)
  part3 <- part2/beta(a,b)
  part4 <- sqrt(part3)

  #return
  return(part4)
}

problem2 <- function(p, q, n_steps = 1000){
  #need to rescale the points
  p <- p / sqrt(sum(p^2))
  q <- q / sqrt(sum(q^2))

  #for unit vectors a,b in Hilbert space, \theta = arccos({x,y})
  angle <- acos(sum(p * q))

  #create matrix to hold the path
  #each row will be the path at t = n_step_i, and the length will
  #correspond to the length of the vector
  path <- matrix(NA, nrow = n_steps, ncol = length(p))

  #computing path, assuming that the points are not antipodal, since then any path will
do.
  #I think we only compute the angle once?
  for (i in 1:n_steps) {
    #compute time, starting at 0
    t <- (i-1) / (n_steps-1)

    #the first term
    term1 <- sin(angle*(1-t))*p
    term2 <- sin(t*angle)*q

    #path at time t
    combined <- (term1+term2)/sin(angle)
    path[i, ] <- combined
  }

  #return the path
  return(path)
}


num.points = 500
time.grid <- seq(0,1,length.out = num.points)

#building points
f1 <- f_t(time.grid, a = 2, b = 5)
f2 <- f_t(time.grid, a = 2, b = 2)
f1n <- f1 / sqrt(sum(f1^2))
f2n <- f2 / sqrt(sum(f2^2))

#rows are functions, columns are values of a function at a time point
```

```
#smooth function connecting f1 and f2?
path <- problem2(p=f1, q=f2, n_steps = 11)

#for the range.  Otherwise plot doesn't fit
yr <- range(c(path, f1n, f2n))

matplot(time.grid, t(path), type = "l", lty = 1, lwd = 1,
        xlab = "t", ylab = "f(t)",
        main = "Geodesic Path from f1 to f2",
        ylim = yr)

#endpoints
lines(time.grid, f1n, lwd = 2, col = "red")
lines(time.grid, f2n, lwd = 2, col = "blue")
legend("topright", legend = c("f1","f2"),
       col = c("red","blue"), lwd = 2)


#problem 3
#problem 3 asks for us to write function to compute exponential exp and inverse
#exponential maps on a unit Hilbert sphere

#function for inverse exponential mapping
#returns tangent vector
inv_exp_mapping <- function(p,q){
  #normalizing
  p <- p / sqrt(sum(p^2))
  q <- q / sqrt(sum(q^2))

  #computing angle
  angle <- acos(sum(p*q))

  #formula for inverse exponential (logarithmic???) mapping
  out <- (angle / sin(angle))*(q-cos(angle)*p)

  return(out)
}

#function for exponential mapping
#returns tangent vector back to the sphere
exp_mapping <- function(p, v){
  #scaling and normalizing
  p <- p / sqrt(sum(p^2))
  v_norm <- sqrt(sum(v^2))

  #formula for exponential mapping and normalize
  out <- (cos(v_norm)*p) + (sin(v_norm)*(v/v_norm))
  out / sqrt(sum(out^2))

  return(out)
}

#computing inverse exponent of f2 and then compute exp(v)
#normalize f1 and f2
f1n <- f1 / sqrt(sum(f1^2))
f2n <- f2 / sqrt(sum(f2^2))

v <- inv_exp_mapping(f1n, f2n)
exp_v <- exp_mapping(f1n,v)

#may need to compute the norm
l2_err <- sqrt(sum((exp_v - f2n)^2))
cat(l2_err)

#problem 4
```

```
#need function for the extrinsic mean
#just computing the euclidian mean and normalizing
extrinsic_mean_S2 <- function(Fdat){
  mean <- rowMeans(Fdat)
  norm_mean <- sqrt(sum(mean*mean))

  return(mean/norm_mean)
}


#function that minimizes the geodesdic distances of the data
#pg 238 in Anujs textbook
#need some number of iterations and a tolerance to stop early
karcher_mean_S2 <- function(Fdat, iterations, tolerance, step_size = 0.5){
  #need to initialize a guess.  Use the extrinsic mean
  mu <- extrinsic_mean_S2(Fdat)

  #iterating
  for (k in 1:iterations) {
    vbar <- c(0, 0, 0)

    #need the average log map at mu
    for (j in 1:ncol(Fdat)) {
      #cosine of the angle between mu and each data point
      cosine <- sum(mu*Fdat[, j])

      #running into NaN values.  This seemed to fix it
      if (cosine > 1){
        cosine <- 1
      }
      if (cosine < -1){
        cosine <- -1
      }

      #angle
      theta <- acos(cosine)

      #check for is the angle is too small
      #log map, projecting into the tangent plane
      #computing norm of v
      if (theta > 1e-14) {
        v <- Fdat[, j] - (cosine*mu)
        sv <- sqrt(sum(v*v))

        #tangent* distance
        vbar <- vbar + (theta/sv)*v
      }

    }

    #mean tangent direction
    vbar <- vbar / k

    #normalize
    v_norm <- sqrt(sum(vbar*vbar))

    #checking to see if the normalized length of mean tangent vector is below the
    #tolerance
    if (v_norm < tolerance) {
      return(list(mu = mu, iters = iterations - 1, converged = TRUE))
    }

    #if not below tolerance, update using the exponential mapping
    step <- step_size * v_norm
    mu   <- cos(step)*mu + sin(step)*(vbar / v_norm)
```

```
    mu <- mu / sqrt(sum(mu*mu))
  }

  return(list(mu = mu, iters = iterations, converged = FALSE))

  #whether you break or don't, you return a list of the mu, number of iterations,
  #and whether the algorithm converges or not
}


#euclidean function for S_infinity
euclidean_mean_S_infinity <- function(mat, t){
  dt <- diff(t)[2]

  mu <- rowMeans(mat)
  mu <- mu / sqrt(sum(mu^2) * dt)

  return(mu)
}

#Karcher function for S_infinity
karcher_mean_S_infinity <- function(mat, t, iterations = 1000, tolerance = 1e-12,
step_size = 0.5) {

  m  <- length(t)
  k  <- ncol(mat)

  #getting the width for the Reimann approx
  dt <- diff(t)[2]


  #getting the extrinsic mean as an initial guess and projecting onto unit sphere
  mu <- rowMeans(mat)
  mu <- mu / sqrt(sum(mu^2) * dt)

  for (iter in 1:iterations) {
    vbar <- rep(0, m)

    #compute average log map
    for (j in 1:k) {

      #inner product
      cosine <- sum(mu * mat[, j])*dt

      #kept getting NAs without bounding cosine
      if (cosine > 1){
        cosine <- 1
      }
      if (cosine < -1){
        cosine <- -1
      }

      #distance on the sphere
      theta <- acos(cosine)

      v  <- mat[, j] - cosine*mu
      sv <- sqrt(sum(v*v)*dt)
      vbar <- vbar + (theta / sv)*v
    }

    #average direction
    vbar  <- vbar / k
    vnorm <- sqrt(sum(vbar * vbar) * dt)

    #if the normalized v is below the tolerance, return list
```

```
    if (vnorm < tolerance) {
      return(list(mu = mu, iters = iter - 1, converged = TRUE))
    }

    #exp update
    step <- step_size * vnorm
    mu   <- cos(step) * mu + sin(step) * (vbar / vnorm)
    mu <- mu / sqrt(sum(mu^2) * dt)
  }

  list(mu = mu, iters = iterations, converged = FALSE)
}




#loading in data for S2
#first S2 dataset
S2.dat1 <- readMat("Datasets/HW2/Problem 4/S2DataFile1.mat")
str(S2.dat1)

#S2 mean euclidean mean
extrinsic_mean1 <- extrinsic_mean_S2(Fdat = S2.dat1$x)

#Karcher mean
karcher_mean1 <- karcher_mean_S2(Fdat = S2.dat1$x, iterations = 10000, tolerance = 1e-16)

extrinsic_mean1
karcher_mean1$mu



#second S2 dataset
S2.dat2 <- readMat("Datasets/HW2/Problem 4/S2DataFile2.mat")
str(S2.dat2)

#S2 mean euclidean mean
extrinsic_mean2 <- extrinsic_mean_S2(Fdat = S2.dat2$x)

#Karcher mean
karcher_mean2 <- karcher_mean_S2(Fdat = S2.dat2$x, iterations = 10000, tolerance = 1e-12)

extrinsic_mean2
karcher_mean2$mu




#what about S^infinity
#first Sinf dataset
Sinf.dat1 <- readMat("Datasets/HW2/Problem 4/HilbertSphereDataFile1.mat")
str(Sinf.dat1)

euclid_s_inf1 <- euclidean_mean_S_infinity(mat = t(Sinf.dat1$h), t =
as.vector(Sinf.dat1$t))
kerchet_s_inf1 <- karcher_mean_S_infinity(mat = t(Sinf.dat1$h), t =
as.vector(Sinf.dat1$t))

#how to computel2_diff <- sqrt(sum((muE - muK)^2) * dt)
dt <- diff(as.vector(Sinf.dat1$t))[1]
diff1 <- sqrt(sum((euclid_s_inf1 - kerchet_s_inf1$mu)^2)*dt)
diff1

#second Sinf dataset
Sinf.dat2 <- readMat("Datasets/HW2/Problem 4/HilbertSphereDataFile2.mat")
```

```
str(Sinf.dat2)

euclid_s_inf2 <- euclidean_mean_S_infinity(mat = t(Sinf.dat2$h), t =
as.vector(Sinf.dat2$t))
kerchet_s_inf2 <- karcher_mean_S_infinity(mat = t(Sinf.dat2$h), t =
as.vector(Sinf.dat2$t))

diff2 <- sqrt(sum((euclid_s_inf2 - kerchet_s_inf2$mu)^2)*dt)
diff2

#Problem 5
#given a set of pdfs on [0,1].  Need to cluster them according to Fisher-Rao

p5.dat <- readMat("Datasets/HW2/Problem 5/PdfClusteringData.mat")
str(p5.dat)

#part 1.  compute the pairwise distance matrix
p5.t <- p5.dat$t
p5.f <- p5.dat$f

#need to compute an integral of a square root of products

dist.mat <- function(funs, t){
  #get dimensions
  n <- nrow(funs)
  m <- ncol(funs)
  dt <- t[2] - t[1]

  #normalize each function
  for (i in 1:n) {
    integral_i <- sum(funs[i, ] * dt)
    funs[i, ] <- funs[i, ]/integral_i
  }

  #take square root of densities
  sq.root <- matrix(0, n, m)
  for (i in 1:n) {
    for (j in 1:m) {
      sq.root[i, j] <- sqrt(funs[i, j])
    }
  }

  #need to get inner product and distances
  dist <- matrix(0, n,n)
  for (i in 1:n) {
    for (j in i:n) {
      inner.prod <- 0
      for (k in 1:m){
        inner.prod <- inner.prod + (sq.root[i,k]*sq.root[j,k])
      }
      inner.prod <- inner.prod*dt

      #running into NaN values.  This seemed to fix it
      if (inner.prod > 1){
        inner.prod <- 1
      }
      if (inner.prod < -1){
        inner.prod <- -1
      }

      #take arc cosine and fill upper and lower triangle matrices
      d <- acos(inner.prod)
      dist[i, j] <- d
      dist[j, i] <- d
    }
```

```
    dist[i, i] <- 0
  }

  return(dist)
}

pairwise.dist.mat <- dist.mat(funs = as.matrix(p5.f), t = as.vector(p5.t))

#plotting heatmap
heatmap(pairwise.dist.mat,
        Rowv = NA, Colv = NA,
        main = "Fisher-Rao Pairwise Distance Matrix",
        xlab = "PDF Number", ylab = "PDF Number")

#part 2, dendogram clustering.  List of linkage options
methods <- c("single", "complete", "average", "ward.D2", "median", "centroid")

par(mfrow = c(2, 3))
for (m in methods) {
  hc <- hclust(as.dist(pairwise.dist.mat), method = m)
  plot(hc, main = paste("Method:", m), xlab = "", ylab = "Distance", sub = "")
}
par(mfrow = c(1,1))


#part 3 and 4, generate MDS plot
hc <- hclust(as.dist(pairwise.dist.mat), method = "average")

#clustering by K clusters and getting labels, cuts clustering tree (produced by hclust)
#into a given number of clusters
k <- 3
grp <- cutree(hc, k = k)

xy2 <- cmdscale(as.dist(pairwise.dist.mat), k = 2)
plot(xy2, col = grp, pch = 19, xlab = "MDS-1", ylab = "MDS-2",
     main = "2D MDS colored by cluster")
text(xy2, labels = paste0("g", seq_along(grp)), pos = 3, cex = 0.8)
legend("bottomleft", legend = levels(factor(grp)), col = 1:length(unique(grp)), pch = 19)
```