The notch shape data

Reading in and preprocessing the data

The first step is to read in the data. Assume that the original data are held in the file `notchshape.dat` in the same directory as the Splus data files. In practice you will have the data in some other directory.

```
notchdat.in <- read.table(file="notchshape.dat", header=T)
```

The data in notchdat consist of one bone after another; we convert `notchdat.in` into a list, with one element corresponding to each bone shape.

```
notchdat.list <- split(notchdat.in, notchdat.in$FILE)
```

We now need to process each element of this list. The elements are themselves lists, so the first step is to convert these lists into matrices with two columns, corresponding to the X and Y coordinates around the notch. At the same time, we rescale the data so that the notch starts at (0,0) and ends at (0,1). This will be done by the function `notchproc1.fun`, defined as follows:

```
notchproc1.fun <- function(z)  {
    y <-  c( rev(z$Y), z$Y)
    x <-  c( rev(z$Xlat), z$Xmed)
    xscale <- diff(range(x))
    x <-  (x-min(x))/xscale
    y <-  (max(y)-y)/xscale
    return(cbind(x,y))
}
```

In order to yield a list whose elements are the individual two-column matrices, we use the function `lapply`:

```
notchdat1.list <- lapply(notchdat.list, notchproc1.fun)
```

To plot a typical notch, we can use the `plot` function; for example the first notch is displayed by

```
plot( notchdat1.list[[1]] )
```

Smoothing and parametrizing by arc length

Suppose that `z` is a two-column matrix representing a particular notch. In order to parametrize `z` by arc length, we use the following function

```
notchproc2.fun <- function(z) {
#
#  first of all replace all except the first and last rows
#    of z by the averages of neighboring rows
#
     nt <- dim(z)[1]
     z <- rbind(  z[1,], ( z[-nt,]+z[-1,])/2 , z[nt,])
#
#   now define the vector tl to be the cumulative arc length
#     along the curve defined by z by finding the distance
#     successive points and then cumulating and dividing by
#     the total length
#
     zd <- sqrt( diff( z[,1] )^2 + diff( z[,2] )^2 )
     tl <- c( 0, cumsum(zd) )/sum(zd)
#
#  finally, linearly interpolate the x and y coordinates
#     to yield the coordinates at points with evenly spaced
#     values of tl, and return a matrix with these coordinates
#
     x <- approx(tl, x, n = 50)$y
     y <- approx(tl, y, n = 50)$y
     return( cbind(x,y) )
     }
```

We now apply this function to every bone in the list `notchdat1.list`, and make the resulting list into a matrix with 96 rows corresponding to the individual bones and 100 columns corresponding to the x and then the y coordinates of the points around the notch. We can also, very similarly, define a 3-way array, by simply using different dimension attributes.

```
notchdat2.list <- lapply( notchdat1.list, notchproc2.fun)

notchdat.mat <- matrix(  unlist(notchdat2.list), byrow=T, nrow=96,
     dimnames=list( names(notchdat2.list), NULL ) )

notchdat.arr <- array( notchdat.mat, dim=c(96,50,2),
     dimnames=list(names(notchdat2.list), NULL, c("x", "y")))
```

In order to plot any particular bone, we can plot the corresponding sub-array of `notchdat.arr`; for example

```
plot( notchdat.arr[43, , ], type="l")
```

will give a line plot of the 43rd femur, which is the bone labelled "`234R`".

Finding and plotting mean curves

Part of our analysis involves the use of concomitant variables such as the presence or absence of eburnation, indicating arthritic changes. These are in three columns of the file `concom.dat` and can be read in as follows; the second instruction converts the variables to logical type. The "older" variable indicates whether the individual has been assessed to be over 45, and the "male" variable indicates the gender if this can be determined. The "eb" variable is true if eburnation is present; eburnation is the indicator of arthritic bone change. Because we shall use eburnation repeatedly in subsequent analysis, we define a separate variable `eb`

```
concom.dat <- read.table("concom.dat", header=T, row.names="FILE",
                      as.is=T)
concom.dat <- lapply(concom.dat, as.logical)
eb <- concom.dat$eb
```

We are now in a position to find the overall mean curve, and the mean curves for eburnated and noneburnated bones. It is easiest to work from the array notchdat.arr to obtain these means. To obtain the mean of all the curves

```
zmean <-  apply( notchdat.arr, c(2,3), mean)
```

and the means of the individual curves are given by

```
zebmean     <- apply(notchdat.arr[ eb,,], c(2,3), mean)
znonebmean  <- apply(notchdat.arr[!eb,,], c(2,3), mean)
```

Because each of these is a two-column matrix with x coordinates in the first column and corresponding y coordinates in the second, these can be plotted directly by using the `plot` or (to add to an existing plot) `lines` function.

Principal components analysis

Because the standard routine `prcomp` works in terms of matrices rather than arrays of data, we carry out principal components analysis by using the matrix notchdat.mat, and converting vector results to matrices where appropriate. The principal component scores themselves are found directly from `prcomp()$x`, but we have to rearrange and redimension the matrix `prcomp()$rotation` to separate the x and y coordinates of the principal component weight functions. In the following paradigm, the array `notch.pca$z` contains in `notch.pca$z[jj, , ]` the two-column array yielding the `jj`th weight function.

```
notch.pca       <-  prcomp(notchdat.mat)
notch.pca$z     <-  array(t(notch.pca$rotation), dim=c(96,50,2),
    dimnames=list( paste("PC",1:96,sep=""), NULL, c("x","y") ) )
```

In order to plot the effects of individual principal components, we can now add or subtract multiples of individual components to the overall mean function `zmean`. For example to plot the first principal component, we can plot

```
plot    (zmean + 0.3*notch.pca$z[1,,], type="l", ylim=c(0,0.9))
lines   (zmean - 0.3*notch.pca$z[1,,], lty=3)
```

To find the percentage of variability explained by the various components, we can read off the standard deviations of the components and make the necessary transformation:

```
notch.pca$percentvar <- 100 * notch.pca$sdev^2/
         sum(notch.pca$sdev^2)
```

Discriminant Analysis

Within S-Plus there are functions `discr` and `discrim` that carry out linear discriminant analysis, but it is probably more convenient to build up simpler functions from scratch. For example the following function will find the simple linear discriminant between two classes.

```
discrim.fun <- function(z, gr)
{
# z is a data matrix and gr is a T/F matrix of class membership
# we find the linear discriminant and certain of its properties: on exit
#       ldisc is the vector of linear discriminant weights
#       dcrit is the critical value of the linear discriminant
#       xdisc is the vector of values of the linear discriminant for the data in the sample
#       alloc is the vector of allocations of the data in the sample
#-------------------------------------------------------------------
# find individual group means and pooled covariance matrix; add a very small constant to the
#       diagonal to ensure numerical stability
#
    m1 <- apply(z[gr,], 2, mean)
    m0 <- apply(z[!gr,], 2, mean)
    z1c <- sweep(z[gr,], 2, m1)
    z0c <- sweep(z[!gr,], 2, m0)
    vmat <- (t(z1c) %*% z1c + t(z0c) %*% z0c)/(dim(z)[1] - 2)
    diag(vmat) <- diag(vmat) + 0.0001 * mean(diag(vmat))
#
# find linear discriminant, normalized to be of norm 1, and classify the data in the sample
#
    ldisc <- solve(vmat, m1 - m0)
    ldisc <- ldisc/ vecnorm(ldisc)
    xdisc <- z %*% ldisc
    dcrit <- 0.5 * sum((m1 + m0) * ldisc)
    alloc <- (xdisc > dcrit)
    return(ldisc, dcrit, xdisc, alloc)
}
```

Cross-validation

One measure of the efficacy of the linear discriminant is the leave-one-out resubstitution error, calculated by leaving the observations out one at a time, and allocating each one on the basis of the remaining data. There are quick algorithms for doing this, but the following routine carries out the calculation by brute force; this is fine for the example we are considering. The output counts the number of false positives and false negatives; these can be combined in whatever way is appropriate.

```
discrimcv.fun <-  function(z, gr) {
    nobs <- length(gr)
    cvalloc <- vector(length = nobs)
    for(j in (1:nobs)) {
        ldj <- discrim.fun(z[-j,  ], gr[-j])
        discj <- sum(z[j,  ] * ldj$ldisc)
        cvalloc[j] <- (discj > ldj$dcrit)
    }
    falsepos <- sum(cvalloc & !gr)
    falseneg <- sum(gr & !cvalloc)
    return(c(falsepos, falseneg))
}
```

Example calculations

We can now carry out the various calculations described in the text. For example, to find the discriminant function based on the entire data, we calculate

```
ld1         <- discrim.fun(notchdat.mat, eb)
ld1$zdisc   <- matrix(ld1$ldisc, ncol=2)
```

The second command gives the linear discriminant weights in the form corresponding to the x and y coordinates of the notch, so that `ld1$ldisc` gives the main discriminant direction between the two populations. In order to plot this as an arrow plot relative to the overall mean, the `arrows` command can be used, as follows:

```
plot(zmean)
arrows(zmean[,1],zmean[,2],(zmean+ld1$zdisc)[,1],(zmean+ld1$zdisc)[,2])
```

In order to carry out a discriminant analysis based on a subset of the principal components, we first find the discriminant as a function of these components. For example, to find the discriminant based on the first six principal components, we compute

```
ld6 <- discrim.fun(notch.pca$x[, 1:6], eb)
```

and if we wish to assess the leave-one-out crossvalidation score of this discriminant, we calculate

```
discrimcv.fun(notch.pca$x[, 1:6], eb)
```

which gives the estimated numbers of false positives and false negatives.

If we wish to find the discriminant function in terms of the coordinates of the notch rather than in terms of the principal components, we use the specification of the principal component weight functions to give the discriminant function as a 100-vector, and then reformat it to yield the appropriate matrix of coefficients:

```
zd6 <-   matrix( notch.pca$rotation[, 1:6] %*% ld6$ldisc, ncol=2)
```

The matrix `zd6` now contains the corresponding discriminant weight function in the form of x and y coordinates. Because `ldisc` is scaled to have sum of squares equal to 1, the same is true for the elements of the matrix `zd6`, because of the orthogonality property of the principal components rotation matrix.

To plot the discriminant direction as an arrow plot, we proceed as before:

```
plot(zmean)
arrows(zmean[,1],zmean[,2],(zmean+0.5*zd6)[,1],(zmean+0.2*zd6)[,2])
```

or to obtain two line plots showing the mean curve and a curve distorted in the main discriminant direction

```
plot(zmean, type="l")
lines(zmean + 0.2*zd6, lty=3)
```

To obtain the mean of the projection of the two types of curve onto the space spanned by `zmean` and `zd6`, we use the property that the sum of the squares of the matrix `zd6` is equal to 1. This means that the discriminant score `sum(zd6*zmean)` is the same as the overall mean discriminant score `mean(ld6$x)`. Also, for any `a`, adding `a*zd6` to `zmean` will increase the discriminant score by `a`. This means that the following sequence will plot curves which have the same discriminant score as the mean eburnated and mean noneburnated curves respectively:

```
m0 <- mean( ld6$x[!eb] ) - mean( ld6$x )
m1 <- mean( ld6$x[eb] )  - mean( ld6$x )
plot(zmean+ m0*zd6, type="l")
lines(zmean+ m1*zd6, lty=3)
```

One thing done in the text is to compare the linear discriminant based on the first 6 principal components with a discriminant based just on the difference in the group mean curves. The vector `ld6$x` contains the linear discriminant scores. We first rescale these so that the eburnated bones have mean score 1 and the noneburnated have mean score –1:

```
m6eb <- mean(ld6$x[eb])
m6noneb <- mean(ld6$x[!eb])
dscores <- list( d6eb= (ld6$x[eb] -  m6noneb)*2/(m6eb-m6noneb) -1,
             d6noneb =(ld6$x[!eb] - m6noneb)*2/(m6eb-m6noneb) -1)
```

Next we project the data onto the direction defined by the difference between the two mean curves, and perform the same calculation as above:

```
projdir <- as.vector(zebmean-znonebmean)
projsc  <- notchdat.mat %*% projdir
pseb <- mean(projsc[eb])
psnoneb <- mean(projsc[!eb])
dscores$meaneb <- (projsc[eb] - psnoneb)*2/(pseb-psnoneb)-1
dscores$meannoneb <- (projsc[!eb] - psnoneb)*2/(pseb-psnoneb) -1
```

And now we can produce a boxplot as in the text:

```
par(mex=1.5)
boxplot(dscores, boxcol=-1, medcol=1, outpch=NA, outline=F,
           cex=1.5, ylab="Score", xlab="Method/group")
```

To obtain various statistics of the different discrimination methods, we can, for example, find the mean and various quantiles, or the variances by

```
lapply(dscores, summary)
lapply(dscores, var)
```