The Crime Life Course data

Because the original data are proprietary, it will not be possible for users to run these analyses on the same data as discussed in the book. Nevertheless, it should be possible to follow the steps of the analysis as described below. The first few lines of the real data set are as follows:

```
Number of reported offences for 413 men year by year from age 11 to age 35 inclusive
3 0 0 1 0 1 0 0 0 0 0 1 3 0 1 0 0 1 0 0 1 1 0 0 0
1 0 2 0 0 2 1 0 5 6 0 4 4 2 7 3 1 3 4 3 3 1 0 0 0
0 1 0 0 1 3 2 5 3 0 0 0 0 2 1 1 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 2 0 0 0 0
1 2 1 0 0 0 4 2 2 1 0 2 0 1 0 2 2 6 6 0 1 0 0 1 1
0 0 3 2 1 0 0 0 0 0 0 0 0 0 0 0 2 1 0 0 0 0 1 0 0
1 0 1 1 0 0 0 0 1 2 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 2 4 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

Each line refers to a single individual in the data set, and gives the 25 counts of numbers of arrests of that individual at each age from 11 to 35.

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 `crime.dat` in the same directory as the Splus data files. The original file `crime.dat` has some comments in the first line, which we need to skip. Also, because the data refer to years 11 to 35, we give the resulting table column names to that effect. Finally, we want to handle our data as a matrix rather than a data frame, so we turn it into one by using the function `as.matrix`.

```
crimdat.in <- as.matrix( read.table(file="crime.dat", skip=1,
    col.names=11:35))
```

If we construct a histogram of the total number of crimes committed by each individual in the sample, we get a very long-tailed distribution. Taking the square root of the number of crimes committed in each year reduces the skewness somewhat. To see this, plot

```
hist( apply(crimdat.in, 1, sum), nclass=20)
crimdat.sqrt <- sqrt(crimdat.in)
hist(apply (crimdat.sqrt, 1, sum), nclass=20)
```

The square root transformation is further suggested because a Poisson distribution might be a rough model for the number of crimes committed per year, and the square root transformation stabilizes the variance of the Poisson distribution.

Creating functional observations

The next step is to create functional observations from the annual data. To do this, we refer the data to a B-spline basis. We can either use a basis with essentially as many (or even more) basis functions as there are time points in the original observations, or we can use a basis with fewer basis functions. In the former case the fit to the original data will be exact and we may have to choose between basis representations by choosing the one with smallest roughness; in the latter case we fit the basis by least squares.

In any case, the following function will yield a list with the necessary components:

```
bsplinesetup.fun <- function(y, n=min(50, dim(y)[2]))
{
#
#  Given a matrix y of data, find a B-spline basis based on at most (n-2)  internal knots.
#  In this version, the number of knots is reduced if there are not at least n time points
#   observed data
#
# On input:  The rows of y represent observations and the columns time points.
#
# Output: ycoef      Matrix of spline coefficients, columns represent observations
#         bmat       Matrix giving values of bsplines at observation points
#         jmat       Matrix of integrals of crossproducts of B-splines
#         kmat       Matrix of integrals of crossproducts of second derivatives of B-splines
#         meancoef   Mean B-spline coefficients
#         funmean    Average function values
#
#   First find the knots
#
    if(!is.matrix(y))  y <- t(as.matrix(y))
    m <-  dim(y)[2]
    nd <- dim(y)[1]
    n <- min(n, m)
    intkn <- seq(from = 0, to = 1, length = n)
    kn <- c(0, 0, 0, intkn, 1, 1, 1)
#
#  Now find the crossintegral matrix jmat
#
    bmat1 <- spline.des(knots = kn, x = seq(from = 0, to = 1,
        length = 101))[[4]]
    jmat <- 0.01 * t(bmat1) %*% bmat1
#
# find the matrix kmat of integrals of products of second derivatives
#
    zs <- spline.des(kn, intkn, deriv = rep(2,n))[[4]]
    z1 <- zs[-1,  ]
    z2 <- zs[-n,  ]
    kmat <- 2*t(z1)%*%z1 + 2*t(z2)%*%z2 + t(z1)%*%z2 + t(z2)%*%z1
    kmat <- kmat/(6*(n-1))
```

```
#
#  Now find the matrix of values at the evaluation points, and the coefficients of the data.
#  If there are more bsplines than evaluation points, find the fit with the least roughness.
#
    bmat <- spline.des(knots=kn, x=seq(from=0, to=1, length=m))[[4]]
    if((n + 2) <= m) {
        ycoef <- solve(bmat, t(y))
    }
    else {
        bsvd <- svd(bmat, nv = n + 2)
        baug <- rbind(bmat, t(bsvd$v[, (m + 1):(n + 2)]) %*% kmat)
        zmat <- matrix(0, nrow = n + 2 - m, ncol = nd)
        yaug <- rbind(t(y), zmat)
        ycoef <- solve(baug, yaug)
    }
#
# Now find the mean coefficients and the values of the mean of the original data
#
    meancoef <- apply(ycoef, 1, mean)
    funmean <- bmat %*% meancoef
    return(ycoef, bmat, jmat, kmat, meancoef, funmean)
}
>
```

To plot individual functions at a scale smaller than the original data, we need to have a spline interpolation function. The following function takes a vector yc of B-spline coefficients and produces a vector of values y at an equally spaced vector x of length nvals:

```
bvals.fun <- function(yc, nvals = 101)
{
#
#  Given a vector of B-spline coefficients yc,
#
    kn <- c(0,0,0, seq(from=0, to=1, length=length(yc)-2), 1,1,1)
    xx <- seq(from = 0, to = 1, length = nx)
    bmat1 <- spline.des(knots = kn, x = xx)[[4]]
    yfun <- bmat1 %*% yc
    return(x = xx, y = yfun)
}
```

Smoothing the mean function

In some cases, for example the crime data, the overall mean function is not very smooth and it will be desirable to find a smoothed version. The output of the function bsplinesetup.fun contains all the information needed to perform a roughness penalty smooth of the mean, and indeed to calculate a functional cross-validation score to help with the choice of the smoothing parameter. The following functions enables this to be done.

```
funsmooth.fun <-  function(zs, spar)
{
#       carry out roughness penalty smoothing of the mean curve, following preprocessing
#       by the function bsplinesetup.fun.       On input
#       zs is the output of bsplinesetup.fun
#       spar is the smoothing parameter
#       On output
#       the values g  of the smoothed mean function at the evaluation points
#          and the coefficients gcoef of the smoothed mean function are returned
#
#
    jlk <- zs$jmat + spar * zs$kmat
    gcoef <- solve(jlk, zs$jmat %*% zs$meancoef)
    g <- zs$bmat %*% gcoef
    return(g, gcoef)
}
>


funsmoothcv.fun <- function(zs, spar)
{
#       calculates crossvalidation score for roughness penalty smoothing of the mean curve
#       following preprocessing by the function bsplinesetup.fun.  On input
#               zs is the output of bsplinesetup.fun
#               spar is a vector of smoothing parameters
#       The vector of cross-validation scores is returned.
#
    nsp <- length(spar)
    n <- dim(zs$ycoef)[2]
    cvscores <- rep(NA, nsp)
    for(j in (1:nsp)) {
        jlk <- zs$jmat + spar[j] * zs$kmat
        gcoef <- solve(jlk, zs$jmat %*% zs$meancoef)
        smat <- solve(jlk, zs$jmat %*% zs$ycoef)
        delresmat <- n * sweep(zs$ycoef, 1, gcoef) + smat - zs$ycoef
        delresmatj <- t(delresmat) %*% chol(zs$jmat)
        cvscores[j] <- sum(delresmatj^2)
    }
    return(cvscores)
}
```

Because of the way that the curves are scaled, in practice very small values of `spar` may be needed. For example, for the crime data, typical values of `spar` are `1e-6` or even smaller.

The minimum of the crossvalidation score is given by `spar=2e-7`. From a subjective point of view, the resulting curve requires a little more smoothing, and so the value `spar=1e-6` was actually used.

```
funpca.fun <-  function(zz, spar = 1, zmean = zz$meancoef)
{
#
#  Takes the output of bsplinesetup.fun and carries out a smoothed functional PCA
#   with smoothing parameter equal to spar. This is taken straight from page 118
#   of Ramsay and Silverman, 1997, but with a vital correction.
#     If centered=T then it is assumed that the
#   mean or estimated mean has been subtracted from the coefficients zz$ycoef
#
    zm <- zz$jmat + spar * zz$kmat
    zm <- 0.5 * (zm + t(zm))
    umat <- chol(zm)
    lmat <- t(umat)
    ycmat <- zz$jmat %*% sweep(zz$ycoef, 1, zmean)
    dmat <- solve(lmat, ycmat)
    cmat <- dmat %*% t(dmat)/dim(ycmat)[2]
    eig <- eigen(cmat)
    emat <- eig$vectors[, 1:10]
    pcasmat <- solve(umat, emat)
    pcabmat <- zz$bmat %*% pcasmat
    vn <- apply(pcabmat, 2, vecnorm)
    pcabmat <- sweep(pcabmat, 2, vn, "/")
    return(pcabmat)
}
```

Carrying out the analysis

We first of all refer the data to B-spline coefficients by using the function bsplinesetup.fun

```
crimdat.bs <- bsplinesetup.fun( crimdat.sqrt )
```

Having set up the functions, the analysis itself proceeds as follows. First we find the minimum of the crossvalidation score, by working out the score at a range of values. Plotting `spar` against `cvvals` gives the figure shown in the text.

```
spar <- c(1,2,4,7,10,20,40,70,100,200,400,700,1000,2000,4000,7000)/1e9
cvvals   <-   funsmoothcv.fun( crimdat.bs, sparvals)
```

The values and coefficients of the smoothed mean can be found by

```
crimdat.smcoefs <- funsmooth.fun( crimdat.bs, 1e-6)
```

To plot the smoothed mean, the easiest approach is

```
plot( spline (11:35, crimdat.smcoefs$g) , type="l")
```

The `spline` program will do exactly the same as finding the basis functions at a large number of intermediate points. If we simply plot the values `crimdat.smcoefs$g` we will get a piecewise linear plot between the integer time points.

Now we move on to the smoothed functional principal components analysis. Since the coefficients of the smoothed mean are held in crimdat.smcoefs$gcoef, we can use these to estimate the mean.

```
crimdat.spca <- funpca.fun(crimdat.bs, spar=1e-5,
    zmean=crimdat.smcoefs$gcoef, centered=F)
```

To plot the resulting components, we add and subtract multiples of the components to the mean. For instance, to display the first component,

```
matplot( 11:35,
    cbind( crimdat.smcoefs$g,
            crimdat.smcoefs$g + 0.05* crimdat.spca$wtfuns[,1],
            crimdat.smcoefs$g - 0.05* crimdat.spca$wtfuns[,1]),
    type="l",col=1, lty=c(1,4,2),ylim=c(0,0.85), cex=1.5, ylab="Square
    root of annual offences", xlab="Age")

zsmsm$pcscores _ crm.sqrt %*% zsmsm$spca
```