Functional Models for Reaction Time Distributions: Hyperactive versus Normal Children

Annotated Analyses in Matlab

1. The data

Our data consist of around 70 reaction times (RT's) for each child for (i) a sample of 17 children diagnosed as attention-deficit (hyperactive) disorder, or ADHD for short, and (ii) a sample of 16 normal children who act as controls. The data have already been screened to eliminate really long reaction times that seem not to reflect the normal reaction processes, and as a result of this screening, all reaction times are no larger than 3000 milliseconds.

Your first task is to inform Matlab about where the files that you are going to use are located using the `addpath` command. These files are:

- Raw data files to be input
- Functional data analysis functions that you will need to use
- Special m-files containing other information, probably in your working directory.

Here are the commands that are appropriate to my Matlab installation on a personal computer:

```
%  Location of the M-files for the analysis of the data
addpath('c:\matlab\ADHD')
%  Location of the raw data to be input
addpath('c:\matlab\ADHD\data')
%  Location of the FDA functions
addpath('c:\matlab\fdaM')
```

The data are in two files, `Hdat.txt` for the hyperactives, and `Cdat.txt` for the controls. In each file, the first column contains the number of the child, and the second a reaction time. These are loaded into Matlab and some variables set up with these commands:

```
load Cdat.txt      %  Control group data
load Hdat.txt      %  ADHD    group data

nfilesC = 16;      %  Number of control cases
nfilesH = 17;      %  Number of ADHD    cases

RTC = Cdat(:,2);   %  Control reaction times
RTH = Hdat(:,2);   %  ADHD    reaction times

CN = length(RTC);  %  Control total sample size
HN = length(RTH);  %  ADHD    total sample size
```

2. Some histograms

The first thing we want to do is to take a peak at how the RT's are distributed. These commands set up a two-panel display of histograms for the RT's for the first subject in each sample. Note that we use the `histc` and `bar` functions in order to force the width of the bars to be the same for each plot.

```
RTC1 = Cdat(Cdat(:,1)==1,2);
RTH1 = Hdat(Hdat(:,1)==1,2);

edges = 0:150:3000;

subplot(1,2,1)
nh1 = histc(RTH1,edges);
bar(edges, nh1, 'histc')
axis([0, 2500, 0, 45])
xlabel('\fontsize{12} Milliseconds')
title('\fontsize{16} First ADHD')
subplot(1,2,2)
nc1 = histc(RTC1,edges);
bar(edges, nc1, 'histc')
axis([0, 2500, 0, 45])
xlabel('\fontsize{12} Milliseconds')
title('\fontsize{16} First Control')
```

As an exercise, you might like to try different numbers of bars, achieved by modifying the `edges` variable.

3. Fitting group densities

Now we'd like to estimate the probability density functions for the two groups, ignoring individual differences. That is, we treat each set of data as being drawn from a homogeneous sample. This is, of course, not correct, but it will give us a picture of the gross differences between the two groups that will also be a bit more refined than the histograms that we did above.

The density function is defined as the normalized exponential of a functional data object. Consequently, the first task is define the basis for representing the object. The natural choice in this case would be a B-spline basis. The lower limit of the range is set slightly below the smallest observed reaction time. These commands specify the number of basis functions, their order, and the range.

```
nbasis = 34;        %  use 34 basis functions
norder =  5;        %  use order 5 splines
rangex = [0,3000];  %  set range
```

Now we set up equally spaced knots, except for a knot at 100 msec, which we drop because there are no observations between 0 and 200 msec.

```
breaks = [0,200:100:3000];
```

Now we can create the basis. We also set up an initial coefficient vector containing all, and then load the coefficient vector and basis into an functional data object, `wfd0`.

```
basis = create_bspline_basis(rangex, nbasis, norder, breaks);
cvec0 = zeros(nbasis,1);
Wfd0  = fd(cvec0, basis);
```

We will exercise a light amount of smoothing on the estimated functional data objects by penalizing their third derivative. Why the third derivative, you ask? Well, the first derivative tends to be linear for densities resembling the normal distribution in the sense of having a central single mode. By penalizing the third derivative, therefore, we smooth the first derivative towards linearity (i. e. no curvature). We chose to make the order of splines 5 so that the third derivative would itself be reasonably smooth or differentiable. But the penalty parameter in this case is small, and the result is simply to remove some minor wiggles but still allow the data to speak for themselves. Here are the set ups for the linear differential operator `Lfd` and the smoothing parameter:

```
Lfd    = 3;
lambda = 1e-6;
```

Now we're ready to estimate the densities. This is achieved by the `densityfd` function. These calls use several default argument values. The complete listing of the function is to be found at the end of these notes, and you may want to read the initial comment lines to learn more about the capabilities of this function. Here are the two calls for our two samples:

```
[WfdC, FCstr] = densityfd(RTC, Wfd0, Lfd, lambda);
[WfdH, FHstr] = densityfd(RTH, Wfd0, Lfd, lambda);
```

The function returns two functional data objects, `WfdH` and `WfdC`, for hyperactives and controls, respectively, along with two struct objects containing the final minimzed fitting criteria and the norms of the final gradients.

Now we want to plot the densities. The FD objects are not themselves densities, but rather log densities plus an arbitrary constant. The following code converts these FD objects, evaluated at a fine mesh of points, into actual density values for plotting.

```
nfine = 301;
xfine = linspace(0, 3000, nfine);

cvecC = getcoef(WfdC);
CvalC = normalize_phi(basis, cvecC);
WvecC = eval(WfdC,  xfine);
PvecC = exp(WvecC)./CvalC;

cvecH = getcoef(WfdH);
CvalH = normalize_phi(basis, cvecH);
WvecH = eval(WfdH,  xfine);
PvecH = exp(WvecH)./CvalH;
```

Now comes the plotting. We also plot the knots so that we can see where they are positioned with respect to the reaction time distributions.

```
subplot(1,1,1)
plot (xfine, PvecH, 'k-', xfine, PvecC, 'k-.')
hold on
for i=2:29
  plot([breaks(i),breaks(i)],[0,0.002], 'k:')
end
hold off
axis([0,3000,0,0.002])
xlabel('\fontsize{16} Reaction Time (msec)')
ylabel('\fontsize{16} Density')
legend('\fontsize{12} ADHD','\fontsize{12} Control')
```

4. Looking at individual densities

We should always be suspicious of results from aggregated data, and especially when there is enough information from individuals to give reasonable estimates of individual results. In this case there is; around 70 reaction times yields ample information about the shape of the density function. With this smaller sample, however, it makes sense to increase the smoothing parameter in order to secure a reasonably smooth density estimate.

Now we run through the cases, estimating the density separately for each by using individualized knot placements. These density functions are each defined by a function *W(t)* having an expansion in terms of the 34 B-splines in the basis that we have set up. The coefficients for this expansion are stored in the array `cvecstore`.

```
cvecstore = zeros(nbasis,17);
for isub=1:17
  fprintf(['\nSubject ',num2str(isub),'\n'])
  indexi = find(Hdat(:,1)==isub);
  xi  = RTH(indexi);
  Wfd = densityfd(xi, Wfd0, Lfd, lambda);
  cvecstore(:,isub) = getcoef(Wfd);
end
```

Next, these coefficients are used to construct functional data objects for the functions *W(t)*.

```
CmatH = cvecstore;
WfdH  = fd(CmatH,  basisfd);
```

We also store the values of these functions and their derivatives in arrays `WmatH` and `wmatH`.

```
WmatH = eval(WfdH, xfine);
wmatH = eval(WfdH, xfine, 1);
```

Next, we compute the values of the density functions defined by the functions *W(t)* and store them in array `PmatH`, and also store the logs of the densities.

```
CvalmatH = zeros(17,1);
for i=1:17
  CvalmatH(i) = normalize_phi(basisfd, CmatH(:,i));
end
PmatH = exp(WmatH)./(ones(nfine,1)*CvalmatH');
logdensmatH = WmatH - ones(nfine,1)*log(CvalmatH');
```

Now comes the plotting of the density functions

```
subplot(1,1,1)
plot (xfine, PmatH, '-')
axis([0,3000,0,.004])
xlabel('\fontsize{16} Reaction Time (msec)')
ylabel('\fontsize{16} Density')
```

and of the log density functions

```
plot(xfine,logPmatH)
axis([0,3000, -12, -5])
xlabel('\fontsize{16} Reaction Time (msec)')
ylabel('\fontsize{16} Log Density')
```

Later, we will also need the individual densities for the log-shifted data. These are computed here for the ADHD group only. These statements define these values.

```
LRTC = log10(RTC - 120);
LRTH = log10(RTH - 120);
```

First, we need to define a new basis.

```
rangex = [2.25, 3.5];
nbasis = 34;
norder =  5;
basis  = create_bspline_basis(rangex, nbasis, norder);
```

We'll also change the size of the smoothing parameter rather dramatically, but this is because the data now have a very different distribution.

```
lambda = 1e-8;
```

Now we are ready to calculate the density functions.

```
cvecstore = zeros(nbasis,nsub);
for isub=1:17
  fprintf(['\nSubject ',num2str(isub),'\n'])
  indexi = Hdat(:,1)==isub;
  xi   = LRTH(indexi);
  xi   = xi(xi >= rangex(1) & xi <= rangex(2));
  Wfd0 = fd(cvecstore(:,isub), basis);
  Wfd  = densityfd(xi, Wfd0, Lfd, lambda);
  cvecstore(:,isub) = getcoef(Wfd);
end
```

We set up a fine mesh of values for plotting and other computations.

```
nfine = 301;
xfine = linspace(2.25,3.5,nfine)';
delta = xfine(2) - xfine(1);
```

As we did above, we store the values of the functions W(t), their derivatives, the corresponding density functions and log density functions in arrays.

```
WfdLH     = fd(CmatLH,  basis);
WmatLH    = eval(WfdLH, xfine);
wmatLH    = eval(WfdLH, xfine, 1);
PmatLH    = exp(WmatLH)./(ones(nfine,1)*CvalmatLH');
logdensmatLH = WmatLH - ones(nfine,1)*log(CvalmatLH');
```

We also store the means across subjects, which we shall need later.

```
WvecLH = mean(WmatLH')';
wvecLH = mean(wmatLH')';
PvecLH = exp(WvecLH);
PvecLH = PvecLH./(delta*sum(PvecLH));
logdensmeanLH = mean(logdensmatLH')';
```

5. Estimating Individual Effects as well as Group Density of residuals

To save space, we will only give the commands for the ADHD group.

The first step is to set up a design matrix or matrix of dummy variable values coding the identify of the individual associated with each observation. We do this for both groups.

```
zmatH = zeros(HN,17);
for ifile=1:17
  index = (Hdat(:,1) == ifile);
  zmatH(index,ifile) = 1;
end
zmatC = zeros(CN,16);
for ifile=1:16
  index = (Cdat(:,1) == ifile);
  zmatC(index,ifile) = 1;
end
```

The density function that we now need to work with is no longer of either the reaction times themselves, or of their log-shifted transformed values. Instead, it is of the residuals from individual means for the log-shifted data. These residuals will have a mean of approximately 0, but we need to do some preliminary analyses to get a reasonable value for their standard deviation. In the following statements, we get estimates of individual effects by least squares fitting of the data, and then computing the standard deviation of the resulting residuals.

```
dvec0C = zmatC\LRTC;
resC   = LRTC - zmatC*dvec0C;
dvec0H = zmatH\LRTH;
resH   = LRTH - zmatH*dvec0H;
sigma0 = sqrt(sum([resC;resH].^2)/(CN+HN));
```

The resulting standard deviation `sigma0` is 0.155, corresponding to 121.4 msec.

We also calculate the normalized residuals and their ranges.

```
resstdC = resC./sigma0;
rngC    = [min(resstdC),max(resstdC)]
resstdH = resH./sigma0;
rngH    = [min(resstdH),max(resstdH)]
```

Looking at the ranges of these normalized residuals, it looks like we can work with a residual range of [-4,5]. We set up a basis accordingly.

```
rangex  = [-4, 5];
nbasis  = 34;
norder  =  5;
basis   = create_bspline_basis(rangex, nbasis, norder);
breaks  = -4:0.3:5;
```

We set the value of the smoothing parameter.

```
lambda  = 1e-6;
```

The following code gives us a reasonable set of starting values for the coefficients defining the density and for the individual effects, stored in arrays cvec0 and dvec0, respectively.

```
Wvec       = -(resstdH - mean(resstdH)).^2./var(resstdH)./2;
ind        = 3:nbasis-3;
inrng      = find(resstdH >= rangex(1) & resstdH <= rangex(2));
basismat   = getbasismatrix(resstdH(inrng), basis);
cvec0(ind) = basismat(:,ind)\Wvec(inrng);
cvec0      = cvec0 - mean(cvec0);
Wfd0       = fd(cvec0, basis);
dvec0      = zmatH\LRTH;
```

Now we compute the density and the individual effects using function LMfd.

6. Principal components analysis of the log densities

We now use principal components analysis to explore the variation in densities among the hyperactive kids. The strategy is not to use PCA directly on the densities functions, since we don't want variations in densities to come out negative in some regions. That is, PCA is best adapted to functions that are not constrained, for example in this case, to be nonnegative. Instead, we work with the log density functions.

But first we will need to get the density functions for the whole group to provide a reference density around which we will describe variation. This is achieved by a minor modification of the code above to compute individual densities.

```
breaksH = quantbrk(RTH, rangexH, nbasisH, norder);
basisH  = create_bspline_basis(rangexH, nbasisH, norder, breaksH);
WfdOH   = fd(cvecOH, basisH);
lambda = 1e-6;
[WfdH, FHstr] = densityfd(RTH, WfdOH, Lfd, lambda);
cvecH  = getcoef(WfdH);
WvecH  = eval(WfdH, xfineH);
PvecH  = exp(WvecH)./normalize_phi(basisH, cvecH);
```

As we noted in the chapter, log density exhibits great variation for extreme reaction times. But we are not particularly interested in what happens there since, for the density functions, since the log density is also greatly negative, and this translates into near zero density values. So we use a weighted version of PCA. We compute the inner product that defines PCA so that extreme values are considerably downweighted relative to the weight on central values of reaction time. The weight that seems reasonable is simply the density function itself, and we use the group density for this purpose, computed above. Now a weighted PCA can be achieved as follows:

- multiply the data to be analyzed to the data multiplied by the square root of the weights,
- carry out an unweighted PCA of these modified data
- multiply the eigenfunctions or harmonics of these modified data by the reciprocal of the square roots of the weights so as to back-transform them to the original metric. The eigenvalues, however, are correct as they are.

So here we set up the square root of the weights and the modified data:

```
wtvec   = (PvecH./sum(PvecH)).^(0.5);
logdens = log(PmatH).*(wtvec * ones(1,nfilesH));
```

Of course, these are the discrete modified data, and our PCA functional data function `pca` expects a functional data object. So now we use `data2fd` to set this up.

```
logdensfd = data2fd(logdens, xfineH, basisH);
```

Now comes the PCA, calling for two factors, followed by back-transformtion of the harmonic values.

```
pcastrH = pca(logdensfd, 2);
harmmat = eval(pcastrH.harmfd, xfineH);
harmmat = harmmat ./ (wtvec*ones(1,nfac));
```

These two harmonics of the variation in log density account for 92% of the variance, and dominate all remaining harmonics. This code plots them by add a judicious multiple (1.5) of the harmonics to the mean log density, and then exponentiating and normalizing the result to display it as a density.

```
deltaH = xfineH(2) - xfineH(1);
for ifac=1:nfac
  Wveci = WvecH + 1.5*harmmat(:,ifac);
  Pveci = exp(Wveci);
  Pveci = Pveci./sum(deltaH.*Pveci);
  subplot(1,2,ifac)
  plot(xfineH, Pveci, '-', xfineH, PvecH, '--')
  title(['\fontsize{12} PCV =
',num2str(floor(100*pcastrH.varprop(ifac)))])
  axis([0,3000,0,0.002])
  axis('square')
end
```

Just how dominant these first two eigenvalue are can be seen in the plot produced by this code. If you're puzzled about why there are only nine eigenvalues, remember that we only used nine basis functions in these analyses, and this, rather than sample size, is what limits the number of eigenvalues in this case.

```
eigvals = pcastrH.eigvals;
x = ones(7,2);
x(:,2) = reshape((3:9),[7,1]);
y = log10(eigvals(3:9));
c = x\y;

subplot(1,1,1)
plot(1:9,log10(eigvals(1:9)),'-o', 1:9, c(1)+ c(2).*(1:9), ':')
xlabel('\fontsize{16} Eigenvalue Number')
ylabel('\fontsize{16} Log10 Eigenvalue')
title('\fontsize{16} ADHD')
```

You can, if you wish, now go on to do an eigenanalysis for the normal kids for comparison purposes. It might be interesting, too, to have a look at a couple more harmonics, even if they don't account for much more of the variance.

Finally, a critical part of this analysis was a careful selection of the lower limit on the range, in this case 350 milliseconds. We didn't want the log density to be badly defined because there were no data points near the lower limit. If you analyze the control group data, a lower limit of 250 millisceonds is suggested.