Zooming in on Human Growth

Annotated Analyses in Matlab

In this chapter, we did two things. First, we used a monotone smoothing method to estimate growth curves and their derivatives. Then we considered the registration problem, and there again we used a nonparametric monotonic function estimation method to estimating the warping functions *h(t)* that aligned salient curve features. What is common to both problems, then, is the use of monotone function estimation. Before we get down to the data analysis, it might be worth thinking about the monotone smoothing process a bit.

Some tips on monotone smoothing

It is worth reviewing here how a monotone function is contructed using the functional data analysis software that we will use. Unlike other smoothing problems considered in the FDA book, in this case the functional data object does not approximate the data directly, but rather after passing through two transformations. The process is captured in this equation

$$h(t) = \beta_0 + \beta_1 \int_0^t e^{W(u)} du \qquad (1)$$

where *W(u)* is a functional data object with the usual basis expansion. The transformations are, first, exponentiation, which makes the function positive, and second, integration up to t, which then makes the function strictly increasing or monotonic. Function *h(t)* can be viewed as a monotone functional data (MDF) object. The two regression coefficients, β_0 and β_1, give *h(t)* the origin and range required to fit the data.

The fact that *W(t)* does not fit the data directly makes the computation more complex. The coefficients in its basis function expansion must now be estimated iteratively. That is, we have to supply a set of initial values for them, and we usually use 0's, which define a straight line. Then these initial values are updated at each iteration, each time improving the fitting criterion, until the algorithm decides that no further improvement is needed. This, naturally, takes time.

The algorithm itself is fairly complex, and like any complex algorithm, there will surely be situations where it will fail. The fact that it has worked successfully on any number of data sets is no guarantee that it will work on the next. Naturally the developer of the algorithm instinctively or by trial and error avoids applying it to data where failure happens, but once it gets in the hands of someone unaware of its structure, the chances that the data are inappropriate, or something else unforeseen by the developer happens, are greatly increased.

Here are some suggestions:

1. Start small. Use a small number of basis functions for W(u) at first, meaning a small number of knots for a B-spline expansion in most instances. For example, when I start out, I often use one interior knot. If I use order 6 B-spline basis functions, this translates into 7 basis functions. Eventually, I may use even more basis functions than there are data points, but I don't start out that way!
2. Position knots carefully. Make sure that there is at least one data value between each successive pair of knots, and if the data are noisy, aim for three to five. One more or less

fail-safe rule is to position knots at every k^{th} ordered data value, that is, at the quantiles of the data. When I start out, I often use one interior knot, positioned at the median. If that works, I move to three interior knots, and so on.

3. Use lots of roughness penalty. For monotone smoothing, I like to use a penalty on the third derivative of *W(u)*, since this ensures that the second derivative of *h(t)* is smooth. And I start off with large penalty parameter values λ, like 100. The worst thing that can happen is that the result will look like a straight line. But it is safer to decrease a smoothing parameter to an acceptable value than to start off too low, and especially at 0, and increase it. Roughness penalty values that are too small are likely to lead to local minima for least squares fitting, as well as large numbers of iterations.
4. Consider calibrating your smooth on simulated data where you know what the right answer is, and can actually calculate the roughness penalty that gives the best estimate. I calibrated the smoothing of growth data on a highly regarded parametric growth model, that of Jolicoeur (19??), and I added errors to that model's values that imitated in size and variation over t what I saw in the actual data.
5. When you've settled on the right number of knots, the right roughness penalty, and are ready to go into production, you might still consider fitting each set of data with a decreasing set of roughness penalties, usually equally spaced on the log scale, perhaps also with convergence criterion values that also decrease, with the final step using the roughness penalty and convergence criterion that you want. Of course, after the first in the sequence, in subsequent set of iterations will be started off with the final solution in the previous sequence, so this does not cost a great deal in terms of total number of iterations, but does ensure some extra stability, and tends to avoid local minima.

Smoothing the Berkeley growth data

Your first task in an analysis is usually to inform Matlab about the location of any files that you will use. These include M-files with the `.m` extension that contain the Matlab fucntions and code that you will use, and the data files.On my system, these two commands are required, the first giving the location of the FDA functions, and second the data files.

```
addpath ('c:\MATLABR11\fdaM.dir')
addpath ('c:\MATLABR11\fdaM.dir\examples\growth')
```

Now we can load the data. Since the data are not arranged in the usual observation by replication matrix that we need, we can't use the load command. These commands input the data and reshape them into the desired format.

```
% male data
fid = fopen('hgtm.dat','rt');
hgtmmat = reshape(fscanf(fid,'%f'),[31,39]);
% female data
fid = fopen('hgtf.dat','rt');
hgtfmat = reshape(fscanf(fid,'%f'),[31,54]);
```

The 31 age values t_j at which the observations were taken are the same for every child, and are set up by these commands, which also set up the number of cases for each gender.

```
age    = [ 1:0.25:2, 3:8, 8.5:0.5:18 ]';
nage   = length(age);
```

```
ncasem = size(hgtmmat,2);
ncasef = size(hgtfmat,2);
```

Now we're ready to set up the basis object that we will use for the expansion of function *W(u)*. We'll use B-splines of order 6, since we will be using a roughness penalty on the third derivative. We'll position a knot at every data point. This disregards the advice given above, but a lot of experimentation with smaller numbers of knots went on before I was able to work happily with this choice, which gives our monotone smoothing functions maximal flexibility. The number of basis functions that this knot choice implies is 31 + 6 – 2 = 35, a value that exceeds the number of data points, but the roughness penalty that we will use will enforce a reasonable amount of smoothness.

```
norder = 6;
nbasis = nage + norder - 2;
wbasis = create_bspline_basis([1,18], nbasis, norder, age');
```

Now we set up the initial estimate of *W(u)* by specifying a coefficient vector containing all zero's, corresponding to *h(t)* being a straight line. This is a good general choice, and the algorithm will usually produce a much better fit to the data after a single iteration.

```
cvec0 = zeros(nbasis,1);    %  initial coefficient estimates
Wfd0  = fd(cvec0, wbasis);  %  initial functional data object
```

We also need two vectors containing one's. Vector `zmat` is effectively a design matrix that could potentially permit the coefficient β_0 in (1) to vary over sampling points as a function of some covariates, but we don't need this flexibility here. Vector `wgt` could allow us to weight observations variably. This is actually a good idea for these data, since height measurements in early childhood are, understandably, more noisy than in later years. However, to keep this exposition simple, we also will just use ones as weights.

```
zmat = ones(nage,1);  %  design matrix for beta 0
wgt  = zmat;          %  weights for observations
```

Now wet set up the roughness penalty and the penalty parameter. In this case we are penalizing the size of the third derivative of *W(u)*. The penalty parameter `lambda` was found to give best results in calibration trials on a parametric model.

```
Lfd    = 3;           %  penalize curvature of acceleration
lambda = 10.^(-1.5);  %  smoothing parameter
```

The monotone smoothing function can only smooth one set of data at a time, unlike directing smoothing functions like `data2fd` or `smooth_basis`. Therefore we will have to loop through the children, and store the essential results in the arrays set up in the following commands. Matrix cvecstore will store the coefficients defining *W(u)*, matrix betastore will store the two regression coefficients in (1) per case, and vector `RMSEstore` will store the estimated standard errors of residuals computed for each case. The set up is for the 39 males.

```
cvecstore = zeros(nbasis, ncasem);
betastore = zeros(2, ncasem);
RMSEstore = zeros(1, ncasem);
```

Now we're ready to do the actual analyses. Inside the loop in the following code, the height observations are put in array hgt, the monotone smoothing function `smooth_monotone` is called, the standard error is estimated, the results stored in the appropriate arrays, and some

results are displayed for the case. These analyses took about ten minutes on my 750 mherz notebook computer.

```
for icase=1:ncasem
   hgt    = hgtmmat(:,icase);
   [Wfd, beta, Fstr, iternum, iterhist] = ...
       smooth_monotone(age, hgt, wgt, Wfd0, zmat, Lfd, lambda);
   hgthat = beta(1) + beta(2).*monfn(age, Wfd);
   RMSE   = sqrt(mean((hgt - hgthat).^2.*wgt)/mean(wgt));
   cvecstore(1:nbasis,icase) = getcoef(Wfd);
   betastore(1:nbeta, icase) = beta;
   RMSEstore(icase) = RMSE;
   fprintf('%5.f %5.f %12.5f %10.4f\n', [icase, iternum, Fstr.f,
RMSE])
end
```

When you run this code, you will also see results for each iteration, corresponding to the default for output during iterations. This output can either be eliminated or extended by supplying the appropriate argument to the function. You might want to review the options available in smooth_monotone in the code itself, or in the guide to the FDA functions.

Plotting the results

A special purpose menu-based function, `menusetup`, has been set up for displaying the results of a monotone smoothing of growth data. This functions allows you to choose the case to be plotted and four plots:

- the height observations along with the smoothing curve
- the velocity curve with the ages of observation indicated by +'s
- the acceleration curve with the ages of observation indicated by +'s
- the residuals, along with horizontal lines indicating plus and minus two standard errors.

To set up this program, we must first create struct objects for each set of data, as follows:

```
malestr.ncase = ncasem;
malestr.age   = age*ones(1,ncasem);
malestr.hgt   = hgtmmat;
malestr.knots = age*ones(1,ncasem);
malestr.cvec  = cvecm;
malestr.beta  = betam;
malestr.RMSE  = RMSEm;
malestr.nord  = norder;

femalestr.ncase = ncasef;
femalestr.age   = age*ones(1,ncasef);
femalestr.hgt   = hgtfmat;
femalestr.knots = age*ones(1,ncasef);
femalestr.cvec  = cvecf;
femalestr.beta  = betaf;
femalestr.RMSE  = RMSEf;
femalestr.nord  = norder;
```

Note that these struct objects allow ages of observation to vary, as well as knot placement, two features that we haven't used.

Now we can invoke the plotting function.

```
icase = 1;  %  load initial case to be displayed
[H_f1, H_f2, Hc_OK, ...
      Hc_Next, Hc_Last, Hc_This, Hc_Enter, Hc_Quit,  ...
      Hc_Height, Hc_Velocity, Hc_Accel, Hc_Residual, ...
      Hc_Male, Hc_Female, Hc_CaseNo] ...
       = menusetup(icase, malestr, femalestr);

close(H_f1);  close(H_f2);  %  close the figures
```

You will now see two windows, one for the plots on the left, and a smaller menu window on the right. You may have to adjust their size and location to match your screen size. You can control which plot you will see by clicking on the radio buttons in the menu, and when you have set up the plot that you want, click on the "OK" button.

It is essential to have all those handles returned by the function, incidentally, since Matlab must know them to change the state of the menu.

Note in the velocity and acceleration plots that there is strong evidence of a smaller growth spurt, called the mid-spurt by auxologists, before the main pubertal growth spurt for most cases.

Have a look at the residuals. Does it look like we might have over-smoothed the data for ages beyond 4 years or so? Should we consider a weighted smooth, changing the vector `wgt` in the call to `monotone` to something that varies over ages. How would we do this? How might this change what we see in the mid-spurt?

Registration of the Berkeley Data

The next step to consider is the register the growth curves for the Berkeley data. With these data, as with many others, the selection of what to register is an important decision. Curves to be registered should have oscillations, preferably about zero, and at the same time be fairly smooth. The acceleration curves oscillate about zero for the growth data, but tend to behave wildly for ages below 5 years. The velocity curves are better behaved, and do show at least one peak at the pubertal growth spurt (PGS). In these notes, we opt for registering the velocity curves.

Even the velocity curves are data to register for a number of reasons. First, the amount of phase variation is much larger than the other data sets that we consider; the timing of the PGS for girls can vary from 8 to 17 years. This means that a PGS event centered at 10 years is long over when an event centered at 15 begins, and the two therefore do not overlap. The whole-curve algorithm in function `registerfd` is going to have a tough time aligning events that far apart. A second problem is that the behavior of the curves before the PGS varies great in amplitude as well as phase, in the sense that a few curves show no midspurt peak, more show one, and some show two or more. Strictly speaking, we should not even be trying to register curves with varying numbers of peaks.

For these reasons, we counsel registering in two steps. First, use landmark registration to register the PGS because it is easily identifiable in all velocity curves. Then follow this up with the more sophisticated and automatic procedure in `registerfd` to fine-tune the registration. This is a

safer process. Note, though, that a few of the figures in the chapter were generated by using only `registerfd`, and that the results that we will obtain in these notes will not be identical to those. No matter, these are probably better!

Landmark registration of the female velocity curves

First, let's set up a functional data object for the velocity curves. This code sets up an order 4 B-spline basis with a knot at each age of observation, and then creates the object, using as data the velocities already stored in array `velffit` for the finely-spaced ages in vector `agefine`.

```
nbasis   = nage + 2;
hgtbasis = create_bspline_basis([1,18], nbasis, 4, age');
velffd   = data2fd(velffit, agefine, hgtbasis);
```

We now plot each velocity curve in turn. Calling Matlab function `ginput` displays cross-hairs on the plot, and positioning these using the mouse at the top of the PGS peak and clicking stores the time of the peak in the vector `marksf.`

```
marksf = zeros(ncasef,1);
for i = 1:ncasef
  plot(velffd(i))
  title(['Female ',num2str(i)])
  [marksf(i), y] = ginput(1);
end
```

Next we calculate the mean time, and use this as the landmark time for the registered curves.

```
meanmarksf = mean(marksf);
```

We need a basis object for the function landmarkreg. We set up a B-spline basis of order 4 with a single interior knot at the mean PGS time.

```
breaks    = [1,meanmarksf,18];
warpbasis = create_bspline_basis([1,18], 5, 4, breaks);
```

Now we are ready to invoke the landmark registration function, and then set up the functional data object for the registered curves.

```
lmrkstr = landmarkreg(velffd, marksf, meanmarksf, warpbasis);
velfregfd  = lmrkstr.regfd;  %  registered curves
```

We plot the two sets of velocities to see how well we have registered them.

```
subplot(1,2,1)
plot(velffd);     title('Unregistered Female Velocities')
subplot(1,2,2)
plot(velfregfd);  title('Registered Female Velocities')
```

Continuous registration of the female velocity curves

Now we can tune the alignment of these curves, which are already well-registered. This second step will, for example, align the midspurt peaks for curves having only one, or at least only one large one.

First we set up a basis for the function *W(t)*. This a relatively low-dimensional basis since we don't want to give the warping functions too much flexibility due to the complex nature of the amplitude variation prior to the PGS.

```
nbasisw = 8;
basisw  = create_bspline_basis([1,18], nbasisw, 5);
```

Now set up the functional data object for function *W(t)*.

```
coef0 = zeros(nbasisw,ncasef);
Wfd0  = fd(coef0, basisw);
```

To further stabilize the warping functions, we also apply a roughness penalty.

```
Lfd = 2;  lambda = 2;
```

Now we carry out the registration. The first argument is the target function, and is the cross-sectional mean of the landmarg-registered curves. This function has to compute each registration iteratively, so it will take a while to complete the computation. The function will display results at each iteration though, so you can track its progress.
When it is through, the result of this registration replaces that of the landmark registration.

```
regstrf = registerfd(mean(velfregfd), velfregfd, ...
                     Wfd0, Lfd, lambda);
velfregfd = regstrf.regfd;
```

The data from the ten year old boy

Here is the set up for the data for this boy.

```
fid = fopen('onechild.dat','rt');
temp = fscanf(fid,'%f');
data = reshape(temp, [n, 2]);
day  = data(:,1);
hgt  = data(:,2);
n    = length(hgt);
```

We will set up the analysis with the same smoothing parameters that we used for the Berkeley growth data, but we don't need to put at a knot at each of the 83 days. We'll use about half that many knots, putting a knot at every other day. This implies 45 basis functions.

```
knots  = day(1:2:n)';
nknots = length(knots);
norder = 5;
```

```
nbasis = nknots + norder - 2;
basis  = create_bspline_basis([day(1), day(n)], nbasis, norder,
              knots);
cvec0  = zeros(nbasis,1);
Wfd0   = fd(cvec0, basis);
wgt    = ones(n,1);
zmat   = wgt;
Lfd    = 3;
lambda = 10^(-.5);
```

Now we carry out the analysis.

```
[Wfd, beta, Fstr, iternum, iterhist] = ...
          monotone(day, hgt, wgt, Wfd0, zmat, Lfd, lambda);
yhat = beta(1) + beta(2).*monfngrad(day,Wfd);
RMSE = sqrt(mean((hgt - yhat).^2));
```

Then we plot the observations and the smoothing function.

```
dayfine  = linspace(day(1),day(n),151)';
yhatfine = beta(1) + beta(2).*monfn(dayfine,Wfd);
plot(day, hgt, 'o', dayfine, yhatfine, '-')
xlabel('\fontsize{16} Day')
ylabel('\fontsize{16} Centimeters')
title(['\fontsize{16} RMSE = ',num2str(RMSE)])
```

And also we plot the velocity function, which shows striking evidence of a series of mini growth spurts. Because of the gap in recording time during the Easter holidays, we can't be too sure about the velocity peak at 200 days, but at least it seems that there is a spurt every 100 days or so.

```
Dhgt = beta(2).*eval_mon(dayfine, Wfd, 1);
plot(dayfine, Dhgt)
xlabel('\fontsize{12} Days')
ylabel('\fontsize{12} Centimeters/day')
```

Principal components analysis of the Berkeley female data

Here is how to compute the principal components analysis results given in the chapter.

PCA of amplitude variation of acceleration.

Here we do a PCA of the registered acceleration curves. Because the acceleration curves vary a great deal more at ages less than four years, and since we are primarily interested in the components of variation above four, we first set up a functional data object for the acceleration from ages 4 to 18.

```
accfregfd  = deriv(velfregffd,1);
```

```
ageshort   = linspace(4,18,101)';
accfmat    = eval(accfregfd, ageshort);
kntshrt    = age(find(age >= 4 & age <= 18))';
nbasisshrt = length(kntshrt) + 2;
shrtbasis  = create_bspline_basis([4,18], ...
                  nbasisshrt, 4,  kntshrt);
accfregfd  = data2fd(accfmat, ageshort, shrtbasis);
```

Now we carry out the PCA, keeping three harmonics and using Varimax rotation.

```
nharm  = 3;
pcastr = pca(accfregfd, nharm);
pcastr = varmx_pca(pcastr);
```

You can now plot the harmonics using

```
plot_pca(pcastr);
```

PCA of phase variation of acceleration.

Let us assume here that we have used only the continuous registration function registerfd, and that we have stored the functions *W(t)* as the functional data object `velfWfd`. Now set up a matrix of values of the warping functions.

```
warpmatf = monfn(agefine, velfWfd);
warpmatf = 1 + 17.*warpmatf./(ones(101,1)*warpmatf(101,:));
```

With this in hand, just do a regular principal components analysis of this matrix. This can be done as follows.

```
warpcov = cov(warpmatf');
[eigvec, eigval] = eig(warpcov);
[eigval, eigind] = sort(diag(eigval));
eigvec = eigvec(:,eigind);
%  variance accounted for by each component
varval = 100*eigval./sum(eigval);
```

For plotting purposes we want the mean acceleration curve.

```
accfshort = eval(accfregmeanfd, ageshort);
```

The following code produces the last plot in the chapter.

```
subplot(1,3,1)
plot(ageshort+ 5.*eigvec(:,101), accfshort, 'k-',  ...
     ageshort,                   accfshort, 'k--', ...
     [4,18], [0,0], 'k:')
axis('square'); axis([4,18,-3,2])
title(['Harm. 1  ',num2str(round(varval(101))),'%'])
subplot(1,3,2)
```

```
plot(ageshort+ 5.*eigvec(:,100), accfshort, 'k-',  ...
     ageshort,                   accfshort, 'k--', ...
     [4,18], [0,0], 'k:')
axis('square'); axis([4,18,-3,2])
title(['Harm. 2  ',num2str(round(varval(100))),'%'])
subplot(1,3,3)
plot(ageshort+ 5.*eigvec(:, 99), accfshort, 'k-',  ...
     ageshort,                   accfshort, 'k--', ...
     [4,18], [0,0], 'k:')
axis('square'); axis([4,18,-3,2])
title(['Harm. 3  ',num2str(round(varval(99))),'%'])
```