Predicting Lip Acceleration from EMG Activity

Annotated Analyses in Matlab

These analyses use some functions that are not part of the standard functional data analysis package. You can find them here in this web site.

To appreciate some of the details in the calculations, you will want to obtain the technical paper by Malfait, Ramsay and Froda (2001). This is also available at this web site.

Before you begin, don't forget to add the path to the functional data analysis functions. On my system, this is achieved by the command

```
addpath('../fdaM')
```

Inputting the Data

We have 32 records to input for each of three measurements taken over the 690 milliseconds taken to utter "Say bob again." The measurements are:

1. The position of the lower lip along its trajectory of motion in millimeters.
2. The acceleration of the lower lip in meters per second per second.
3. A measure of electromyographical (EMG) activity in the principal muscle that controls lip motion, the depressor labii inferior or DLI.

Each of these measures is stored in a file with a row for each of the 501 time points and a column for each record.

```
EMGmat    = load ('EMG.dat');
lipmat    = load ('LipAcc.dat');
lipposmat = load ('LipPos.dat');
```

We won't actually use lip position in our analyses, but we include the data in case someone wants to try some of alternative analyses.

We also want to define the number of records and the time values in seconds.

```
N = size(emgmat,2);
timevec = linspace(0,0.69,501)';
```

Converting the Lagged Data to Functional Data Objects

Now we want to set up the data with EMG lagged by 50 milliseconds. We first set up a fine mesh of time values, and then use linear interpolation to estimate lip acceleration

values between 50 and 690 milliseconds, and EMG activity values from 0 to 640 milliseconds.

```
tfine = (0:0.001:0.69)';
nfine = length(tfine);

lipmatlag = zeros(nfine-50,N);
EMGmatlag = zeros(nfine-50,N);

for i=1:N
    liptemp = interp1(timevec,lipmat(:,i),tfine);
    EMGtemp = interp1(timevec,EMGmat(:,i),tfine);
    lipmatlag(:,i) = liptemp(51:nfine);
    EMGmatlag(:,i) = EMGtemp(1:(nfine-50));
end

nfine = nfine - 50;
tfine = tfine(1:nfine);
T     = 0.64;
```

Now we convert these discrete data to functional data objects using a B-spline basis.

```
nbasis = 93;
norder =  6;
basis  = create_bspline_basis([0,T], nbasis, norder);

lipfd = data2fd(lipmatlag, tfine, basis);
emgfd = data2fd(EMGmatlag, tfine, basis);
```

We'll also need the mean function for each variable.

```
emgmeanfd = mean(emgfd);
lipmeanfd = mean(lipfd);
```

And, too, centered functions, which are the functions less their respective means.

```
lipfd0 = center(lipfd);
emgfd0 = center(emgfd);
```

Plotting the Bivariate Correlation Function

The plot of the correlation between lip acceleration and EMG activity in Figure 10.2 was a useful preliminary look at the relationship between the two measures. Here is a color version of that figure.

First we define a fine mesh of time values.

```
nfiney = 101;
tmesh = linspace(0,T,nfiney)';
```

Then we get the discrete data from the lagged functions.

```
lipmat = eval_fd(lipfd, tmesh);
emgmat = eval_fd(emgfd, tmesh);
```

Then compute the correlations between the measures across curves.
We are only interested in correlations between EMG activity at times less than or equal to times for lip acceleration.

```
lipemgcorr = corrcoef([lipmat',emgmat']);
lipemgcorr = lipemgcorr(102:202,1:101);
for i=2:101, lipemgcorr(i,1:i-1) = 0; end
```

Now display the correlation surface. Use the rotation and zoom features of Matlab's surface display function to examine the surface from various angles.

```
subplot(1,1,1)
colormap(hot)
surf(tmesh, tmesh, lipemgcorr')
xlabel('\fontsize{16} s')
ylabel('\fontsize{16} t')
axis([0,.64,0,.64,-1,1])
axis('square')
```

Defining the Finite Element Basis

Our basis system for expressing the bivariate regression function $b(s,t)$ is defined by dividing the area over which $b(s,t)$ is defined into a set of right triangles, as in Figure 10.3. Each of these triangles is called an *element* of the system.

The triangular elements that define our basis system are determined by splitting the range of time t into M equal intervals, each of width $l = T/M$, where T is 640 milliseconds, the time over which we have observed the data. Here we use $M = 13$.

```
M = 13;
lambda = T/M;
```

We also need to specify over how many intervals back we will integrate the EMG activity to predict lip acceleration. We designate the number of intervals by B, and here we use $B = 5$. The width of the interval of integration is $d = B\ l$.

```
B = 5;
```

Now we set up the *nodes*, which are the vertices of the triangles. See Figure 10.3 for an example. There will one basis function in our expansion of $b(s,t)$ per node. Our first step is to set up the relationship between elements (triangles) and nodes (vertices).

Function `NodeIndexation` does this. It returns a matrix with a row for each triangle, and three columns. The columns contain the indices of the nodes that go with each triangle.

```
eleNodes = NodeIndexation(M, B);
```

Function `ParalleloGrid` sets up two arrays, one containing the *s* values for each node, and other containing the *t* values.

```
[Si, Ti] = ParalleloGrid(M, T, B);
```

Estimating the Regression Function

The actual computation requires a discretization of the continuous variable t. We define the spacing between these discrete values by specifying the number of discrete values within each of the M intervals. A value of two or four is usually sufficient to ensure a reasonably accurate approximation.

```
npts  = 4;
ntpts = M*npts;
delta = lambda/(2*npts);
tpts  = linspace(delta, T-delta, M*npts)';
```

Now we set up the design matrix that will be used in the discrete version of the regression analysis. This is a fairly length calculation.

```
psiMat  = DesignMatrixFD(emgfd0, npts, M, eleNodes, Si, Ti, B);
```

In any regression analysis we should check that our design matrix is not singular or near singular. This is especially important in approximating continuous integrals by discrete values since too many values will lead to poor conditioning of the design matrix. So we check this out by looking at its singular values and the condition number. Note that the Matlab function `full` is needed here because the design matrix is set up in sparse storage mode.

```
singvals  = svd(full(psiMat));
condition = max(singvals)/min(singvals);
disp(['Condition number = ',num2str(condition)])
```

Along with the design matrix, we need the corresponding vector of dependent variable values. Here's where we set this up.

```
yMat  = eval_fd(lipfd0,tpts)';
yVect = reshape(yMat, N*M*npts, 1);
```

Now we carry out the least squares approximation that gives us our vector of regression coefficients multiplying our basis functions.

```
bHat = psiMat\yVect;
```

Plotting the Regression Function

Once the coefficients defining the expansion of the regression function are available, we now want to have a look at the regression function itself. This is made easy by the built in Matlab function `trisurf`.

```
trisurf(eleNodes, Si, Ti, bHat)
xlabel('\fontsize{12} s');
ylabel('\fontsize{12} t');
title(['Lip/EMG Data ','M = ',num2str(M),', B = ',num2str(B)])
```

A more refined plot is achieved by the following code, which produces a color version of Figure 10.7 using the special function `BetaEvalFD`.

```
svec = linspace(0,T,13*6+1)';
tvec = linspace(0,T,13*6+1)';
betaMat = BetaEvalFD(svec, tvec, bHat, M, T, lambda, ...
                     eleNodes, Si, Ti, B);

subplot(1,1,1)
colormap(hot)
H = imagesc(tvec*1000, tvec*1000, betaMat);
xlabel('\fontsize{16} s (msec)')
ylabel('\fontsize{16} t (msec)')
axis([0,640,0,640])
axis('square')
Haxes = gca;
set(Haxes,'Ydir','normal')
```

Computing the Fit to the Data

We would also like to look at how well the model fits the data. To do this, we must compute the model approximation to the lip acceleration values. Our computations above were carried out using the centered data, so we must also compute an intercept function $a(t)$.

Next we set up a large super-matrix containing the approximated lip acceleration curves using the special function `XMatrix`. This is also a lengthy calculation.

```
psiArray = XMatrix(emgfd, tmesh, M, eleNodes, Si, Ti, B);
```

The matrix of approximation values for the lip acceleration curves is constructed.

```
yHat1 = zeros(nfiney,N);
```

```
for i=1:N
    Xmati = squeeze(psiArray(i,:,:))';
    yHat1(:,i) = Xmati*bHat;
end
```

This approximation, however, is based only on the estimated regression function *b(s,t).* To complete the approximation, we must get the intercept function *a(t).* This requires using the mean EMG curve as a model, and subtracting the fit that this gives from themean lip acceleration.

```
psimeanArray = XMatrix(emgmeanfd, tmesh, M, eleNodes, Si, Ti, B);
yHatMean     = squeeze(psimeanArray)'*bHat;
lipmeanvec   = eval_fd(lipmeanfd, tmesh);
alphaHat     = lipmeanvec - yHatMean;
```

Of course, we now want to plot the intercept function.

```
plot(tmesh,alphaHat)
title('Estimated intercept function');
```

Here is the final fit to the data.

```
yHat = alphaHat*ones(1,N) + yHat1;
```

We plot the data and the fit to the data.

```
subplot(2,1,1)
plot(tmesh, yHat)
axis([0,T,-4,4])
subplot(2,1,2)
plot(tmesh, lipmat)
axis([0,T,-4,4])
```

We also plot the residuals.

```
subplot(1,1,1)
resmat = lipmat – yHat;
plot(tmesh, resmat)
axis([0,T,-4,4])
```

Assessing the Fit

Here is the error sum of squares function.

```
SSE = sum(resmat.^2,2);
```

To provide a benchmark against which we can assess this fit, we need to get the corresponding error sum of squares function when the model is simply the mean lip acceleration curve.

```
lipmat0 = eval_fd(lipfd0, tmesh);
SSY = sum(lipmat0.^2,2);
```

Now we can compute a squared multiple correlation function and plot it. Don't be suprised, though, to see it go below zero; the fit from the mean is not embedded within the fit by the model.

```
RSQ = (SSY – SSE)./SSY;
subplot(1,1,1)
plot(tmesh,RSQ);
axis([0,0.64,-0.5,1])
```