Online Appendix to "Longitudinal Penalized Functional Regression for Cognitive Outcomes on Neuronal Tract Measurements"

Jeff Goldsmith, Ciprian M. Crainiceanu,

Brian Caffo and Daniel Reich

September 4, 2011

A Computational Issues

The models described in the Section 2 are necessarily complex to account for the size and complexity of the data set. This raises reasonable concerns about computational feasibility, especially because many competing methods are slow even in the single-level case. To dispel these concerns we present key segments of code from two implementations, a likelihood-based approach in `R` and a Bayesian approach in `WinBUGS`, of our method for longitudinal functional regression. We also provide overview of the benefits and drawbacks of each. Complete code for each is available online, and the likelihood-based approach is implemented as the `lpfr` function in the `refund` package, available on `CRAN`.

For simplicity, the code given below assumes Gaussian outcomes and subject-specific random intercepts, with the I subjects observed J times each. The functional coefficient is modeled using a large spline basis with explicit penalty. Straightforward alterations to the basic code presented here allow for more complex models.

A.1 Likelihood-based R Implementation

Because LPFR takes advantage of standard mixed effects models to express longitudinal functional models, we can take advantage of standard mixed effects software to fit such models. Here, we use the `gam` function in the `mgcv` package (Wood, 2006) to fit functional models using the following code:

```
# evaluate the M matrix using numeric integration; assumes
# t is observed over an even grid
M.mat <- t(psi) %*% phi *(max(s) - min(s))/(length(s)-1)
```

```
# compute the CM matrix
CM <- C[,1:Kw] %*% M.mat

# the following code creates the design matrices so that
# the gam function performs the correct estimation
Z.int=matrix(rep(c(rep(1, J), rep(0, I*J)), I), nrow=length(Y), ncol=I)
X = cbind(1, CM,  Z.int)

# create the penalty matrices for the random effects
D1 = diag(c(rep(0, 1+2), rep(1, kb-2), rep(0, N_subj)))
D2 = diag(c(rep(0, 1+Kw), rep(1, N_subj)))

# fit the longitudinal functional regression model
fit = gam(Y~X-1, paraPen=list(X=list(D1, D2)), family=family, method="REML", ...)
```

The quantities `I`$= I$, `J`$= J$, `M.mat`$= \boldsymbol{M}_{\psi\phi}$, `C`$= \boldsymbol{C}$, `Y`$= Y$, `X`$= \boldsymbol{Z}$, `Z1`$= Z_1$, `Z2`$= Z_2$, and `Z`$= \boldsymbol{Z}$, are as defined in Section 2. We note that general scalar outcomes can be fit using the above code by altering the `family` argument to the `gam` function.

This approach has several advantages, especially as compared to the `WinBUGS` implementation. The first, and perhaps most important, is familiarity: `R` is commonly used among statisticians, and this implementation will be readily understood. Another advantage is that this code executes very quickly, so that computation time is significantly shorter. However, this approach lacks the ability to jointly model the exposure and the outcome, which can be significant in many situations.

A.2 Bayesian `WinBUGS` Implementation

`WinBUGS` is a powerful alternative when fitting mixed effects models, and as such can be used to implement the LPFR method. We note two minor changes that arise in this implementation: 1) we use a cubic b-spline basis for $\gamma(s)$ with a first order random walk prior to give better mixing properties; and 2) we take advantage of the ability to jointly model the exposure and outcome models (2 and 3). Further, to increase the computation efficiency of this implementation, we use a variation on the projection method

described in Di et al. (2009). In particular, let

$$A_{ijk} = \int_0^1 \{W_{ij}(s) - \mu(s)\}\, \psi_k(s) ds = c_{ijk} + \epsilon'_{ijk}$$

where ϵ'_{ijk} includes the variance from the excluded dimension in the truncated Karhunen-Loéve decomposition and the integrated measurement error. Note that A_{ijk} can be estimated using numerical integration, and that by letting $\boldsymbol{A}_{ij} = (A_{ij1}, \ldots, A_{ijK_w})^T$ we can replace the exposure model 2 with

$$\begin{aligned} \boldsymbol{A}_{ij} &= \boldsymbol{c}_{ij} + \epsilon'_{ij} \\ \boldsymbol{c}_{ij} &\sim \mathrm{N}(0, \Lambda),\ \epsilon'_{ij} \sim \mathrm{N}(0, \sigma^2_\epsilon \boldsymbol{I}), \end{aligned} \tag{5}$$

These changes are implemented in the following code segment; note that we do not use the function `inprod` to substantially reduce computation time, and that the quantity `JxGam` $= \boldsymbol{M}_{\psi\phi}\boldsymbol{g}$ is updated once per model iteration, outside the main loop over subjects.

```
model
{ # Start model
for (i in 1:I) { # Begin loop over subjects
  for (j in 1:J) { # Begin loop over visits
    Y[((i-1)*J+j)]~dnorm(fitted.val[((i-1)*J+j)], tau_Y)
    fitted.val[((i-1)*J+j)]<-alpha + u[i] + eta[((i-1)*J+j)]
    eta[((i-1)*J+j)]<-C[((i-1)*J+j),1]*JxGam[1]+C[((i-1)*J+j),2]*JxGam[2]+
      C[((i-1)*J+j),3]*JxGam[3]+C[((i-1)*J+j),4]*JxGam[4]+
      C[((i-1)*J+j),5]*JxGam[5]+C[((i-1)*J+j),6]*JxGam[6]+
      C[((i-1)*J+j),7]*JxGam[7]+C[((i-1)*J+j),8]*JxGam[8]+
      C[((i-1)*J+j),9]*JxGam[9]+C[((i-1)*J+j),10]*JxGam[10]

      for (k in 1:Kw) {# Begin loop over observations within subject and visit
        A[((i-1)*J+j), k]~dnorm(C[((i-1)*J+j),k], tau)
        C[((i-1)*J+j),k]~dnorm(0,ll[k])
      } # End loop over observations within subject and visit

    } # End loop over visits
```

```
    u[i]~dnorm(0, tau_u)

  } # End loop over subjects

  for(l in 1:Kw){ # Begin loop over rows of M.mat matrix
    JxGam[l]<-M.mat[l,1]*g[1]+M.mat[l,2]*g[2]+M.mat[l,3]*g[3]+
      M.mat[l,4]*g[4]+M.mat[l,5]*g[5]+M.mat[l,6]*g[6]+
      M.mat[l,7]*g[7]+M.mat[l,8]*g[8]+M.mat[l,9]*g[9]+
      M.mat[l,10]*g[10]
  } # End loop over rows of M.mat

  for (i in 1:Kw){ # Begin loop over precisions of PC loadings
    ll[i]~dgamma(0.1,0.1)
    lambda[i]<-1/ll[i]
  } # End loop over precisions of PC loadings

  for (l in 2:Kg){ # Begin loop over g coefficients
    g[l]~dnorm(g[l-1],tau_g)
  }
  ... # Here are the prior distributions for non-functional covariates
}#End model
```

There are several important advantages that arise in this implementation. Most importantly, it allows for the easy construction of posterior credible intervals. As noted above, the functional regression literature has focused on estimation of functional coefficients, while very little attention has been paid to confidence intervals. Even then, among papers using a penalized approach to estimation, confidence intervals are generally empirical or bootstrap intervals. The direct computation of posterior credible intervals therefore addresses a major need in the functional regression literature. Another advantage is joint modeling, which accounts for the uncertainty in estimating the PC loadings; while in many cases this variability in negligible, it can be very important in the presence of large measurement error or when

the functional predictors are sparsely observed. That said, the WinBUGS implementation also has the disadvantage that hyperparamters sometimes must be chosen with care for the program to successfully execute.