PRINCIPAL DIFFERENTIAL ANALYSIS 11

Principal differential analysis (Ramsay 1996) estimates linear systems of ordinary differential equations approximately satisfied by functional data. This is of interest in physical processes, where, for example, the one-dimensional motion of an object is a function of time that solves an ordinary differential equation. The Maxwell equations are another well known physical example. Biological, chemical and other phenomena also often satisfy ordinary differential equations, and discovering the form of these equations can help to understand the nature of the underlying process.

More formally, for a sample of functions $f_j(t), j = 1, \ldots, n$, principal differential analysis determines a linear differential operator L of degree m and/or a function $\alpha(t)$ for which $Lf_j(t) \approx \alpha(t)$ for all j. Here L is defined as:

$$L = \sum_{i=0}^{m} \beta_i(t) D^i$$

where the operator notation has the following interpretation:

$D^0 f_j(t) = f_j(t)$ and $D^i f_j(t) = \dfrac{d^i f_j(t)}{dt^i}$, the i^{th} derivative of $f_j(t)$.

In principal, both the forcing function $\alpha(t)$ and the linear differential operator weights (coefficients) $\beta_i(t)$ may be functional data objects. However, the current implementation may not be reliable unless the linear differential operator is known to have constant coefficients. Methods that better handle more general linear differential operators are under active research, and will be added to S+FDA in the future.

Given the form of $\alpha(t)$ and the $\beta_i(t)$, the linear differential equation is estimated by least squares (or penalized least squares). Either $\alpha(t)$ or one of the $\beta_i(t)$ must be known. Computational procedures are discussed in Chapter 14 of Ramsay and Silverman (1997). Penalized least squares estimation minimizes the criterion:

which reduces to a least squares criterion if the penalty term is omitted.

$$\sum_{j=1}^{n} \int [Lf_j(s) - \alpha(s)]^2 ds + penalty$$

The function $\alpha(t)$ is called the *forcing* function since, when $m = 2$, it corresponds to the external force applied to a physical system. If $\alpha(t)$ is identically zero, the resulting differential equation is said to be *homogeneous.* Otherwise the system is *nonhomogeneous.*

When the forcing function $\alpha(t)$ and/or weight functions $\beta_i(t)$ are unknown, principal differential analysis can be used to estimate them and elucidate the process underlying the functions $f_j(t)$.

Like principal components, principal differential analysis allows re-expressing the functional data in terms of a set of basis functions that may be considerably more compact than the current representation. This follows from the fact that all solutions to a linear differential equation can be expressed as the sum of:

- a particular solution

and

- a linear combination of basis functions for the null space or *kernel* of the linear differential operator.

Although the current implementation of S+FDA cannot currently handle arbitrary bases, such a representation may nevertheless be useful in an analysis.

S+FDA FUNCTIONS FOR PRINCIPAL DIFFERENTIAL ANALYSIS

The S+FDA function `fPDA` estimates the weight functions $\beta_i(t)$ for the linear differential operator L, and/or the forcing function, $\alpha(t)$. The `fPDA` object is a list with two components:

- an object of class `fLinDop` which gives the coefficients of the estimated linear differential operator.
- an object of class `fFunction` which gives the estimated forcing function.

The `fPDA` object also has `fitted.values` and `residuals` from predictions of the original functional data as attributes.

There is a `predict` method for `fPDA` objects that calls a function `fLinDopSolve` to solve the linear differential equation. The fitting for prediction is done by linear regression involving the kernel basis functions of the linear differential operator and a particular solution to the differential equation if there is a nonzero forcing function. See Ramsay and Silverman (1997) for more details.

RADIOACTIVE DECAY EXAMPLE

Consider radioactive decay defined by

$$y'(t) = -ky(t)$$

where $y(t)$ is the amount of a chemical element present at time t, k is the rate constant intrinsic to the element, and $y'(t)$ is the rate of decay. The linear differential operator is:

$$L = D^{(1)} + kD^{(0)}$$

The goal is to estimate k.

To illustrate the S+FDA principal differential analysis function, `fPDA`, we construct an example of functional data described by the above radioactive decay equation. We simulate data for iodine 131, for which $k = 0.0864$ when the unit of time is days.

```
> rateConstantI131 <- 0.0864
> Time <- 0:50
> Y <- matrix(0, 51, 10)
> set.seed(0) # seed for reproducing random numbers
> for(j in 1:10)
      Y[,j] <- (100 + rnorm(1, sd=10))*
                             exp(rateConstantI131*Time)
```

Since the differential equation is of order one, we use a B-spline basis of order four so that the first derivative will be a smooth cubic spline.

```
> basis <- bsplineBasis(c(0,50), norder=4, nbasis=10)
```

The functional data object created from the basis is:

```
> fY <- fVector(basis, Y)
```

Plot the functional data (see Figure 11.1:):

```
> par(mfrow=c(1, 1))
> plot(fY, main="Radioactive Decay of Iodine 131")
```

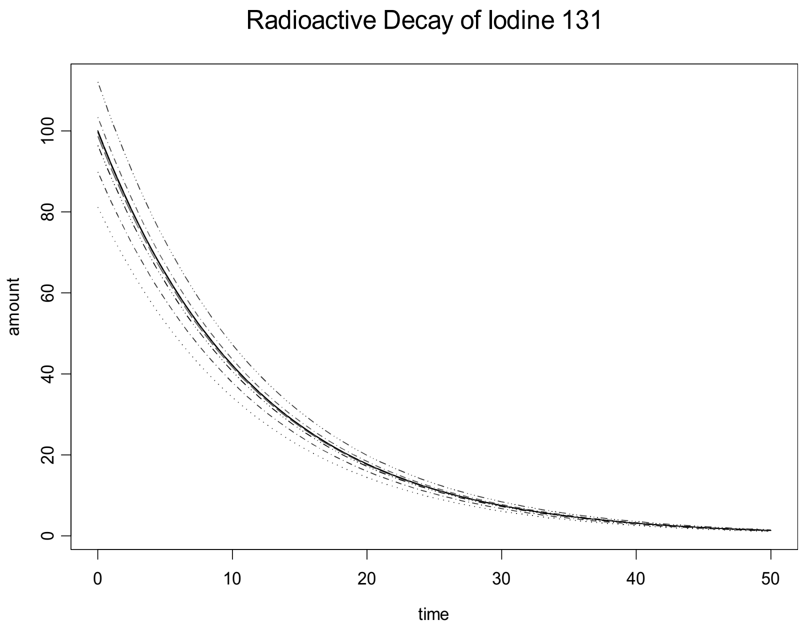

Figure 11.1: *Functional data for radioactive decay of Iodine 131.*

To estimate the rate constant, we first call `fPDA`.

```
> decayPDAconst <-
        fPDA(fY, weights=list(constantBasis(fDomain=
                             c(0, 50)), 1), forcing=0)
```

Here we have set `weights=list(constantBasis(fDomain= c(0,50)), 1)` to indicate that the first order coefficient is known to be equal to 1, and the zeroth order coefficient needs to be estimated. We use a `constantBasis` to ensure that the estimated coefficient is a constant.

The value of the rate constant estimate is then given as follows:

```
> rateConstantEstimate <-
                    fEval(decayPDAconst$linDop[[1]], 25)
> rateConstantEstimate
           [,1]
[1,] 0.08638197
```

(since the coefficient is constant it suffices to evaluate the weight function at any point in the domain).

The following code plots the original functional data object, the predictions produced by `predict.fPDA`, the residuals from the predictions, and the operator residuals (Lf):

```
> predictions <- predict(decayPDAconst)
> par(mfrow=c(2, 2))
> plot(fY, main="Original Functional Data")
> plot(predictions$fitted,
      main="Predicted Functional Data")
> plot(predictions$residuals,
      main="Residuals of Predicted Values")
> Lx <- fEval(fY, fArg=Time, linDop=decayPDAconst$linDop)
> plot(fVector(getBasis(fY), y=Lx),
      main="Differential Operator Residuals")
```

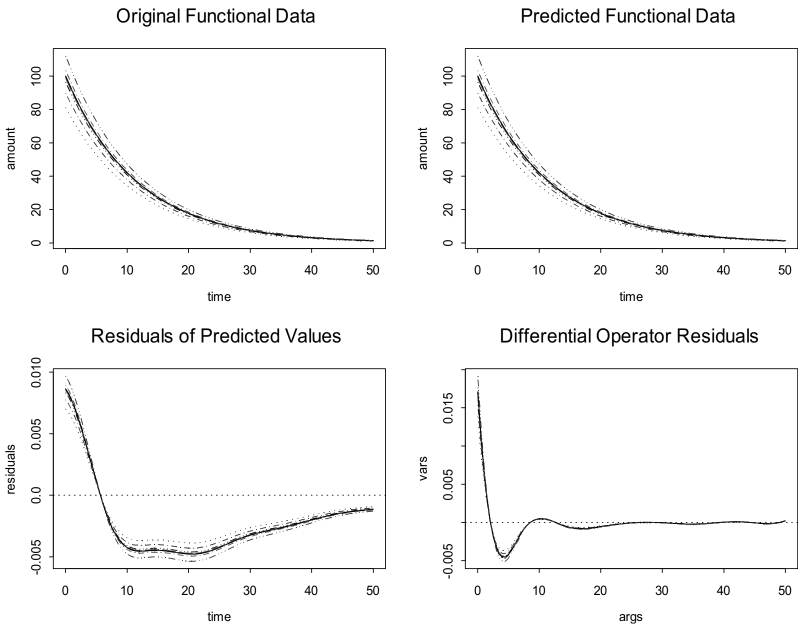

Figure 11.2: *Functional data, predicted fitted values, residuals of predicted values, and operator residuals, when constant coefficients are assumed.*

In the example just given, we chose a constant basis because of the theoretical equation of decay. But it may be of interest to know what `fPDA` would estimate if we did not make this assumption.

```
> decayPDAvar <- fPDA(fY, weights=list(NULL, 1), forcing=0)
```

Setting the first element of weights to `NULL` causes the basis of the functional data `fY` to be used in estimating the forcing function. More generally, a basis can be specified for each unknown function in the linear differential equation.

Plot the estimated rate of decay.

```
> plot(decayPDAvar$linDop[[1]],
       ylab="Decay Rate Estimate", xlab="Domain",
       main="Decay Rate Estimate, using inherited basis")
> abline(h=-0.0864)
```

Figure 11.3: shows that on average the estimated rate of decay is close to the theoretical rate. However, there are edge effects, hinting at the difficulties to be encountered in situations where less is known about the underlying process, and one or more coefficients are estimated using a nonconstant basis.

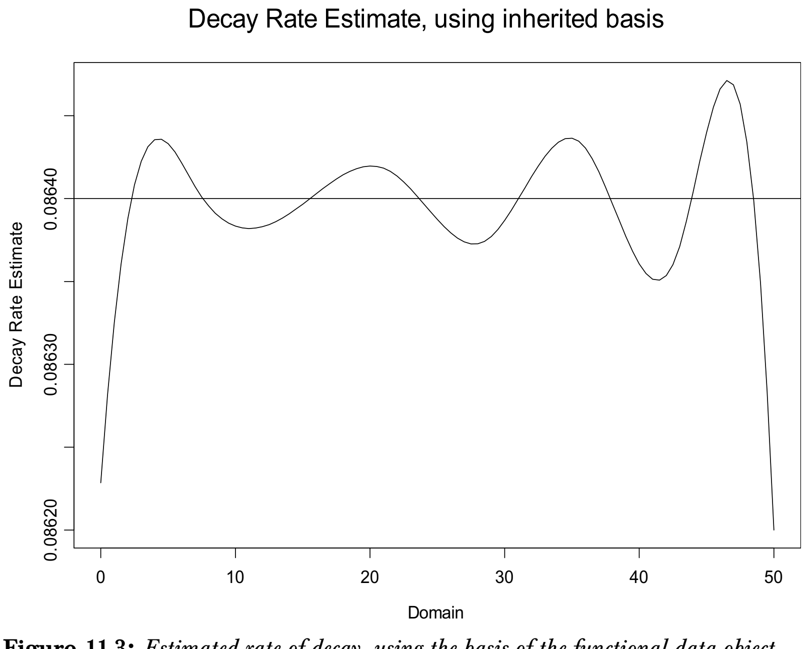

Figure 11.3: *Estimated rate of decay, using the basis of the functional data object. The horizontal line is drawn at the theoretical decay rate.*

HARMONIC OSCILLATOR EXAMPLE

A mechanical system is characterized by an external force applied to the system, together with internal or external frictional forces or viscosity. The classic example is a weight suspended from a spring. The spring will oscillate when the weight is attached to it provided the weight is not too heavy. This motion will fade over time depending on the viscosity of the air or other medium in which the system is situated.

The equation of motion for a harmonic oscillator with external force f is:

$$D^2 y + k_1 Dy + k_0 y = f$$

where k_1 is the damping constant and k_0 is the square of the natural oscillating frequency.

Underdamped + Resonance

The second-order equation of motion describes an *underdamped* system if $k_1 < 2\sqrt{k_0}$. In this case, oscillation will occur. If the forcing function exhibits periodicty, the oscillation is called *resonance.* An analytic solution is known when the forcing function is of the form $C\cos(2\pi\nu t)$, where $2\pi\nu$ is the resonance frequency. A particular solution in this case is

$$A\sin(2\pi\nu t) + B\cos(2\pi\nu t)$$

A general solution to the differential equation can be obtained by adding the particular solution to the homogeneous solution, which (ignoring the phase shift) is

$$C\exp\left(-k_1\frac{t}{2}\right)\sin(t\sqrt{k_0})$$

under the assumption that the system is underdamped. The following code simulates such a system and plots the resulting functional data:

```
> k0 <- 2
> k1 <- .5
> hconst <- fconst <- 10
> phase <- 0
> nu <- 1/3

> pi2nu <- 2*pi*nu
> a <- pi2nu*k1
> b <- k0 - pi2nu*pi2nu
> d <- a*a + b*b
> A <- a/d
> B <- b/d

> tt <- seq(from=0, to=5, length=101)
> Y <- matrix(0, 101, 10)
> set.seed(0) # seed for reproducing random numbers
> for(j in 1:10)
    Y[,j] <- hconst*exp(-k1*tt/2)*sin(sqrt(k0)*tt+phase) +
       fconst*(A*sin(pi2nu*tt) + B*cos(pi2nu*tt)) + rnorm(1)
```

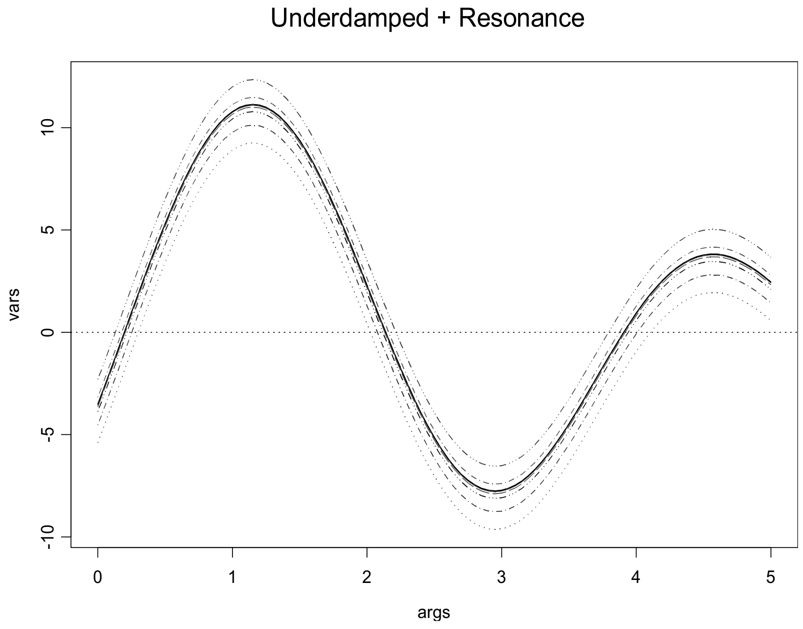

Figure 11.4: *Simulated functional data for an underdamped harmonic oscillator with resonance.*

Now we compute the constant coefficients of the linear differential operator assuming that the forcing function is known:

```
> par(mfrow=c(1, 1))
> basis <- bsplineBasis(c(0, 5), norder=5, nbasis=20)
> fY <- fVector(basis, Y)
> plot(fY, main="Underdamped + Resonance")

# compute constant coeffs using known forcing function
> forcing <- fFunction(basis, fconst * cos(pi2nu*tt))
> oscPDAconst <- fPDA(fY, weights=
                    list(constantBasis(c(0, 5)),
                         constantBasis(c(0, 5)), 1),
                    forcing=forcing)
> k0 <- fEval(oscPDAconst$linDop[[1]], 2.5)
> k0
         [,1]
[1,] 2.021411
> k1 <- fEval(oscPDAconst$linDop[[2]], 2.5)
> k1
          [,1]
[1,] 0.504323
```

The resulting coefficients are quite close to the true values underlying the simulated data.

In this case `fPDA` also gives a good estimate of the forcing function when the linear differential operator is known:

```
> oscPDAforc <- fPDA(fY, weights=list(2, .5, 1),
                   forcing=basis)
> plot(ans1$forcing, main="Estimated Forcing Function
                          (known LDO)")
> lines(tt, fEval(forcing, tt), lty=6)
```

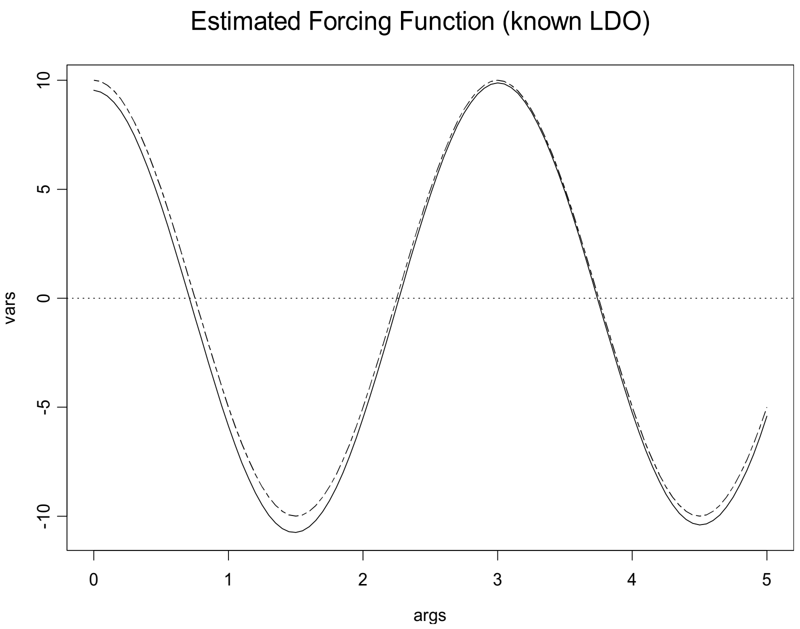

Figure 11.5: *Forcing function for underdamped harmonic oscillator estimated by fPDA when the linear differential operator is known. The dotted line is the true forcing function underlying the simulated data.*

However, if we attempt to estimate the linear differential operator coefficients as well as the forcing function, the resulting least squares problem is ill-conditioned.

```
> oscPDAall <- fPDA(fY, weights=list(constantBasis(c(0,5)),
                   constantBasis(c(0, 5)), 1),forcing=basis)
Warning in fPDA(fY, weights = list(constantBasis(c(..:
                 least-squares system is ill-conditioned
```

The ill-conditioning warning is usually means that the results will not be accurate, as is the case for this example. The constant weights are estimated to be 0 and 0.083, far from their true values of 2 and 0.5, respectively. The forcing function estimate is plotted below:

```
> plot(oscPDAall$forcing, main="Forcing Function Estimate")
> lines(tt, fEval(forcing, tt), lty=6)
```

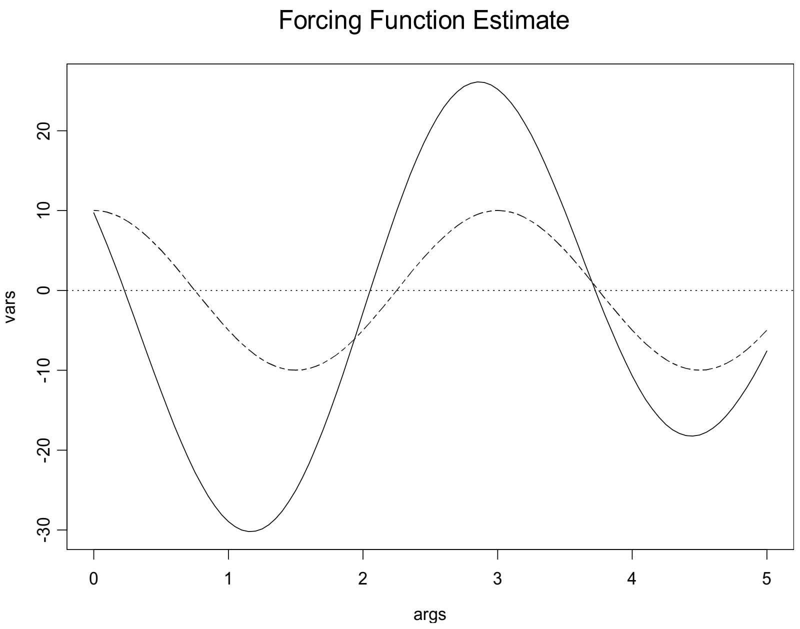

Figure 11.6: *Estimate forcing function when weights are assumed constant but unknown. The dotted line is the true forcing function underlying the simulated data.*

It may in some instances be possible to avoid ill-conditioning by increasing the arguments `k` or `nbasis` to `fPDA` (these affect the accuracy of the projections used in computing inner products for least squares), but in this case we weren't able to find a suitable set of inputs. It is also possible to include penalty terms on the weight functions and/or their derivatives, or on the derivatives of the forcing function, in `fPDA` to regularize principal differential analysis, but there are no systematic guidelines for doing so with the current implementation. New methods under development incorporate regularization mechanisms, and we plan to include them in future editions of this library.

LIP MOVEMENT EXAMPLE

The lip movement data, first used in the chapter on registration, consists of twenty replications measuring the vertical lip position as a single individual says the syllable “bob”. In order to perform principal differential analysis, first register and smooth the curves. The S+FDA code for creating, registering, and smoothing the lip data is as follows:

```
> lipBasis <- fBasis(type="bspline",fDomain=c(0, 1),
                     nbasis=31,params=(c(1:25)/26))
> fLip <- fVector(object=lipBasis, y=lipmat, fArgs=liptime,
                  fNames=list(NormalizedTime=liptime,
                  Replications=seq(20), Units="mm"))
> regLip1 <- fRegister(fLip, mean(fLip), nDeriv=1,
                       maxIter=120, lambda=0.1,
                       criterion=1, penalty=0.0005)
> regLip1 <- fRegister(fLip,mean(regLip1$fReg), nDeriv=1,
                       maxIter=120, lambda=0.1,
                       criterion=1, penalty=0.0005)
> yLip <- fVector(regLip1$fReg, penalty=
                  list(lambda=1.e-10, linDop=fDop(2)))
```

Note in this code that the registration is performed on the derivatives rather than the functions.

Because the lower lip is part of a mechanical system, with certain natural resonance frequencies and a stiffness or resistance to movement, it seems appropriate to explore to what extent this method can be expressed it terms of the second-order differential equation typically used to analyze such systems, in which the linear differential operator

$$\beta_0(t)f(t) + \beta_1(t)D^1f(t) + D^2f(t)$$

is a generalization of the which one used for the harmonic oscillator example in the previous section. Strictly speaking, the mechanical interpretation of the differential equations does not hold if the weight coefficients are allowed to be functions rather than constants, but higher-order effects can be ignored if they do not vary too rapidly with time. The principal differential analysis estimates, assuming nonconstant weights, are computed as follows:

```
> lipPDA <- fPDA(yLip, weights=list(NULL, NULL, 1),
                 forcing=0)
```

We set `forcing=0` to indicate that the differential equation is assumed to be homogeneous (no forcing function). A plot of the residuals for the homogeneous fit with the weight functions estimated by `fPDA`, as given in Figure 11.7:, is calculated as follows.

```
> lipPDA <- fPDA(yLip, weights=list(NULL, NULL, 1),
                 forcing=0)
# calculate and plot residual Lx
> lipResiduals <- fEval(yLip, fArg=liptime,
                        linDop=lipPDA$linDop)

> keep <- liptime >= 0.1 & liptime <= 0.9
> matplot(liptime[keep], lipResiduals[keep,], type="l")
```

Points at the ends of the plot have been removed in order to eliminate edge effects near 0 and 1.

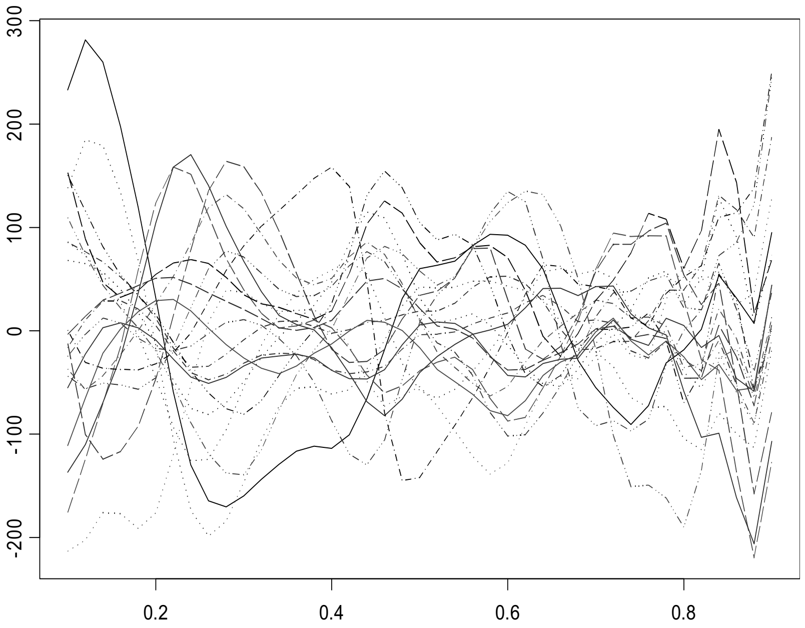

Figure 11.7: *Operator residuals from the second-order differential equation fit to the lip movement data.*

Although the residual results are not nearly as small as we would prefer (for comparison, the largest magnitude of the second derivatives is near 500), they still appear to be more or less random, indicating that the linear differential operator is capturing the functional behavior.

Kernel Basis Functions

Like principal components, principal differential analysis allows re-expressing the functional data in terms of a set of coefficients that may be much smaller than the current representation. Although the current implementation of S+FDA does not allow arbitrary bases, this property may still be useful in an analysis.

Specifically, if L is a linear differential operator of degree m, then there are m linearly independent functions $u_1, u_2, \ldots, u_m$ (the *kernel basis functions*) that span the null space or *kernel* of L, that is, for which $Lu_i = 0$. The kernel basis functions are determined by m constraints, which may include initial conditions and/or boundary conditions.

In the theory of linear ordinary differential equations, all solutions to the homogeneous equation are linear combinations of the kernel basis functions. If the weight functions defining L are determined by principal differential analysis, and residuals for the ordinary differential equation are small, then for homogeneous equations there should be a linear combination of the form

$$f_j(t) = \sum_{i=1}^{m} \gamma_{ij} u_i(t) + \varepsilon_j(t)$$

in which the residual terms $\varepsilon_j(t)$ are relatively small.

For nonhomogeneous equations, any solution can be expressed as the sum of a particular solution and a linear combination of the kernel basis functions. So if $x(t)$ is a particular solution, then the functional data have the following representation:

$$f_j(t) = x(t) + \sum_{i=1}^{m} \gamma_{ij} u_i(t) + \varepsilon_j(t)$$

Since each of the observed functions $f_j(t)$ is a solution, up to a random error, then the average function, $\overline{f(t)} = \frac{1}{n}\sum_{j=1}^{n} f_j(t)$ is also a solution with a random error, but the random error for $\overline{f(t)}$ is n times smaller than for each function $f_j(t)$. Thus, when the errors are small, $\overline{f(t)}$ approximates a particular solution to the differential equation, and consequently a good fit of the centered functions $f_j(t) - \overline{f(t)}$ can be obtained as a linear combination of the kernel basis functions. Notice the similarity with functional principal components analysis, which finds a set of (orthogonal) functions that can be used to re-express the functions $f_j(t)$ such that the integrated squared error $\varepsilon_j(t)$ is minimized.

Change of Basis

A set of kernel basis functions, $u_1, u_2, \ldots, u_m$ for a linear differential operator can be computed via the function `fLinDopSolve`:

```
> lipKernData <- fLinDopSolve(linDop=lipPDA$linDop,
                              x=liptime)
> lipKern <- fVector(lipKernData, basis=getBasis(yLip))
> par(mfrow=c(2, 1))
> plot(lipKern[1], main="First Kernel Basis Function")
> plot(lipKern[2], main="Second Kernel Basis Function")
```

The resulting basis is displayed in Figure 11.8. Notice that the first kernel basis function has a much larger range than the second one indicating, perhaps, that the first basis function is more important.

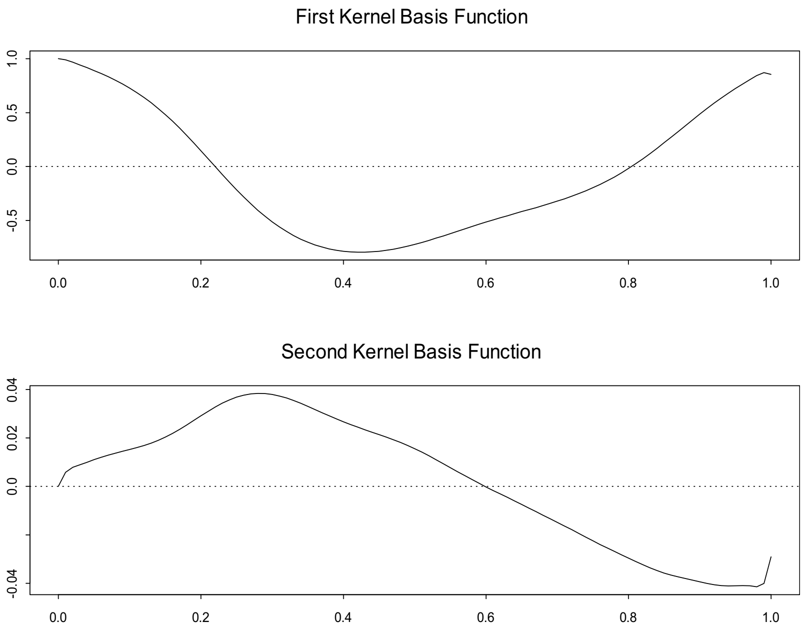

Figure 11.8: *Kernel basis functions for the linear differential operator fit by principal differential analysis to the lip force data.*

The S+FDA function `fLinDopSolve` is based on the adaptive ordinary differential equation solvers `DLSODA` and `DLSODI` (Hindmarsh 1983; Petzold 1983). Initial conditions may be specified through the `initialValues` argument. Different initial conditions can lead to different kernel basis functions, but all sets of kernel basis functions span the same function space and are thus equivalent for our purposes.

Given the kernel basis functions for a linear differential operator, the function `fLinDopFit` can be used to obtain a representation of a function in terms of the kernel basis. Below we compute this fit for the lip motion data, from which the linear differential operator was derived.

```
> lipFit <- fLinDopFit(yLip, linDop=lipPDA$linDop)
> par(mfrow=c(2, 1))
> plot(lipFit$fitted.values, main="Fitted Values")
> plot(lipFit$residuals, main="Residual Functions")
```

The fit is accomplished via least-squares projection of the observed functions onto the kernel basis. The fitted functions and the residuals for the registered lip movement functions are displayed in Figure 11.9. Note the difference between these residuals and those shown in Figure 11.7, which displays values of the linear differential operator applied to `yLip`, which are viewed as “residuals” when homogeneity is assumed.

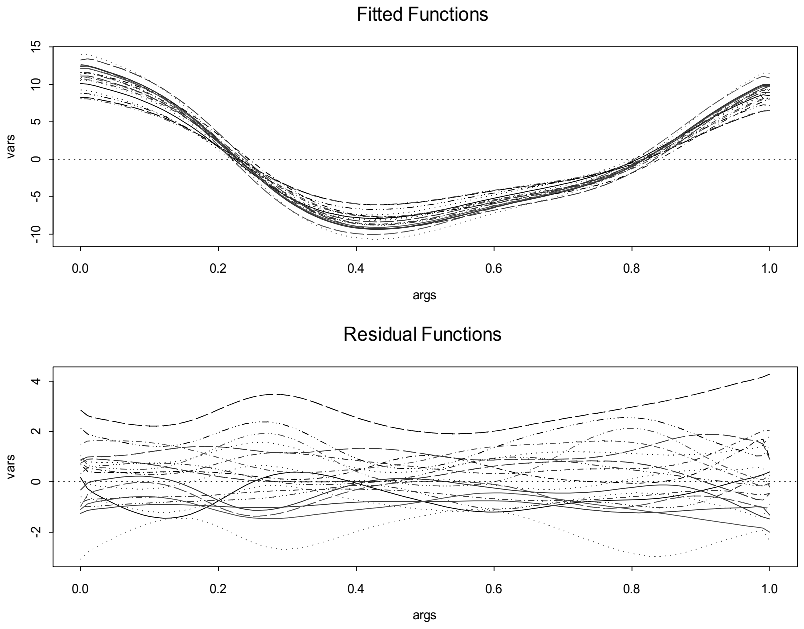

Figure 11.9: *Lip curves and residuals from the fit of the lip motion data to the kernel basis functions for the linear differential operator determined by fPDA.*

The fitted curves appear to be quite similar to the registered and smoothed lip curve functions, although the residual functions indicate that the fit is not perfect. Nevertheless, these residual functions are relatively small, with a range of about 25% of the range of the lip curves proper.

The residuals and fitted values given above are precisely those that would be obtained from `predict` applied to `lipPDA`, because the linear differential operator and data input to `fLinDopFit` came from the principal differential analysis. Function `fLinDopFit` differs from the `predict` method for `fPDA` objects in that instead of an `fPDA` object it takes as input a linear differential operator and (optionally) a forcing function, and returns the predictors (kernel basis functions) and coefficients from the fit as well as the fitted values and residuals.

Comparison with PCA

Because of its relationship to functional principal components, it is useful to compare the fit obtained from the kernel basis functions obtained with the homogeneous functions with the fit obtained using a functional principal components analysis. Here we use the integrated residual variance as a measure of “goodness of fit”. This statistic has meaning for both the functional principal components solution and for the functional principal differential analysis solutions, but is minimized in the functional principal components models - we expect, apriori, that principal differential analysis will not do as well as the principal component analysis in predicting variation in our lip movement data (if the same number of “parameters” are estimated). However, if the principal differential analysis solution explains a good deal of the functions variance, then we would have some evidence that the estimated linear differential equation has the correct form and closely models the process that generated the data.

In the following code we compute the harmonics using function `fPCA` as well as the fitted values for `fPDA`, and computed the integrated variance using functions `fInt` and `fVar`.

```
> ansPCA <- fPCA(~yLip)
> phi <- double(3)
> phi[1] <- fInt(fVar(yLip, bivariate=F))
> phi[2] <- fInt(fVar(fVector(getCoef(yLip)
           -outer(c(getCoef(mean(yLip))), rep(1, 20))
           -getCoef(ansPCA$harmonics) %*% t(ansPCA$scores),
            getBasis(yLip)), bivariate=F))
> phi[3] <- fInt(fVar(predict(lipPDA)$residuals,
                       bivariate=F))

> par(mfrow=c(1, 1))
> plot(fVar(yLip, bivariate=F), xlab="t", ylab="Var(f(t))",
       main="Variance Functions", ylim=c(0,5))
> lines(fVar(fVector(getCoef(yLip)
         - outer(c(getCoef(mean(yLip))), rep(1,20))
         - getCoef(ansPCA$harmonics) %*% t(ansPCA$scores),
        getBasis(yLip)),bivariate=F), lty=2)
> lines(fVar(predict(lipPDA)$residuals, bivariate=F),
        lty=3)
> legend(0.6, 5, c("Mean", "PCA", "PDA"), lty=1:3)
```

The plot of the variance functions shown in Figure 11.10:.

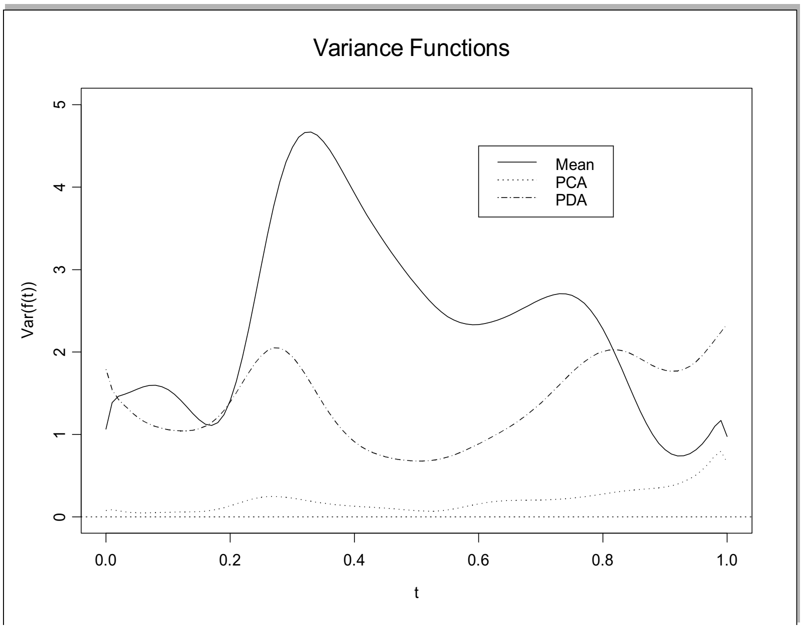

Figure 11.10: *Variance functions for PCA and PDA analyses of the lip motion data.*

Finally, we compute an "r-squared"-like measure for goodness of fit based upon these integrated variances:

```
> Rsq <- 1-phi[2:3]/phi[1]
> names(Rsq) <- c("PCA", "PDA")
> Rsq
      PCA       PDA
0.9147264 0.4107527
```

From both the R-squared statistics and the variance function plots we see that the two harmonic functional principal component solution provides the best fit, as expected.

Summary

An unique aspect of functional data analysis is its ability to provide insight into the processes underlying the functional data. The goal of principal differential analysis - to find an underlying differential equation describing the behavior a sample of observations - is an exciting and powerful idea. The simple least-squares approach currently implemented in S+FDA is limited in what it can handle. However, new iterative approaches under development for principal differential analysis show great promise for improvement, and in the next release of this library, we expect to significantly enhance the methods provided here.