

BASIS OBJECTS AND OPERATIONS

2

What is a Basis?	24
Univariate	24
Bivariate	24
Basis Objects	25
Univariate	25
Bivariate	25
Choosing a Univariate Basis	26
SelectBasis	26
Function Properties	26
Choosing a Bivariate Basis	28
Creating Univariate Bases	29
constantBasis	29
bsplineBasis	30
FourierBasis	31
polynomialBasis	33
polygonalBasis	34
exponentialBasis	35
compositeBasis	36
Creating Bivariate Bases	39
Separable Bases	39
Finite Element Bases	39
Operations on Univariate Bases	41
Derivatives	41
Integrals	42
Inner Products	43
Operations on Bivariate Bases	44
Derivatives	44

WHAT IS A BASIS?

Univariate

In the S+FDA library, univariate functions are represented as linear combinations of basis functions:

$$f(x) = \sum_{j=1}^{n_b} \beta_j b_j(x)$$

where the β_j are coefficients, and the b_j are known basis functions. For example, in an exponential basis, $b_j(t) = \exp(k_j t)$ for user-specified parameters k_j .

Bivariate

Similarly, bivariate functions may be represented as:

$$f(x, y) = \sum_{j=1}^{n_b} \beta_j b_j(x, y)$$

where $b_j(x, y)$ are the basis functions based on (triangle) finite elements.

Alternatively, by assuming that the basis functions are *separable*, the representation is:

$$f(x, y) = \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \beta_{ij} b_i^x(x) b_j^y(y)$$

BASIS OBJECTS

S+FDA supports both univariate and bivariate basis functions including:

- univariate bases: B-spline, Fourier series, polynomial, polygonal, exponential. Moreover, users can define their own bases, and composite bases of two or more bases are also possible.
- bivariate bases: finite element, or the product of two univariate bases.

Univariate

In S+FDA, each of the supported univariate bases is a class that inherits from a larger class called `fBasis`. Different subclasses of `fBasis` are defined by the number of basis functions and the domain. Once the user specifies the type of basis, the number of basis functions, and the domain, a basis-specific constructor function computes values for the coefficients from the data.

Bivariate

In S+FDA, the basis may be assumed to be separable, in which case it is the product of two univariate bases functions, and is of class `fProdBasis`.

If the finite element basis is used, the class is `fFinElemBasis`.

CHOOSING A UNIVARIATE BASIS

Selecting basis functions is perhaps the most important step in a functional data analysis: the basis functions need to have features as close as possible to the data they estimate so that an accurate representation of the function can be obtained with only a few basis terms.

fSelectBasis

Basis selection is so important that we provide the user with a function called `fSelectBasis` that is specifically designed for this task. Input to `fSelectBasis` includes information on whether the function is periodic, how many events are likely to occur in the basis, and whether or not derivatives are needed. Although `fSelectBasis` function can help in selecting a basis, there is no fully automatic way to select a good basis and knowledge of the problem and available data is of critical importance. Note also that `fSelectBasis` allows access to only some of the types of bases that are available in the S+FDA library.

Function Properties

The number of events or features that occur in a function is a measure of its complexity. Features or events can be viewed graphically and include peaks, valleys, zero crossings, plateaus, and linear slopes. In the S+FDA function `fSelectBasis`, if there is only one event, it is assumed that the basis is constant over all values of its domain, with value equal to the value of the single event (class `constantBasis`). On the other hand, if more than one event is specified and derivatives are required, then a polynomial basis (class `bsplineBasis`) is used with the number of basis functions equal to the number of events. Finally, if derivatives are not needed, then a piecewise linear spline (class `polygonalBasis`) can be used.

A function is *periodic* if its values are repeated in fixed intervals. For example, a function that varies in a regular pattern from day to day (e.g., temperature) or over the course of a year (e.g., mean daily temperature) can be thought of as periodic. A periodic function can often be expressed as a *Fourier* series, which is a sum of sine and cosine functions.

Choosing a Univariate Basis

Once a set of basis functions has been selected, coefficients must still be estimated. Values of the coefficients vary not only with the underlying data, but also with the fitting procedure. In particular, smoothing techniques may be needed to mitigate the influence of outliers and avoid overfitting. Smoothing is discussed in more detail in Chapter 4.

CHOOSING A BIVARIATE BASIS

In bivariate analysis, the user must decide whether to assume that the basis functions are separable. If so, the product basis function is appropriate. Otherwise, the finite element basis is preferred. The relative advantages of each type are:

- Product: saves computation.
- Finite Element: theoretically more accurate, feasible to differentiate.

See Chapter for a comparison of computation times and accuracy of approximation...

CREATING UNIVARIATE BASES

In object-oriented programming, a constructor for an object usually has the same name as the class assigned to the object. This convention is followed for objects of class `fBasis`. For example, the function `FourierBasis` constructs an object of class `FourierBasis`, and the function `bsplineBasis` constructs an object of class `bsplineBasis`. The following section gives more detail on the basis functions available in `S+FDA`. The description is largely non-mathematical, but is hopefully sufficient to enable you to choose your own basis.

constantBasis

The simplest bases are those of class `constantBasis`. The basis functions in this class are constant (and equal to one) over their entire domain. There is only one argument to the constructor, the domain of the function. The basis may be constructed and plotted using the following commands:

```
> basis <- constantBasis(fDomain=c(0,10))  
> plot(basis)
```

The resulting plot is shown in Figure 2.1.

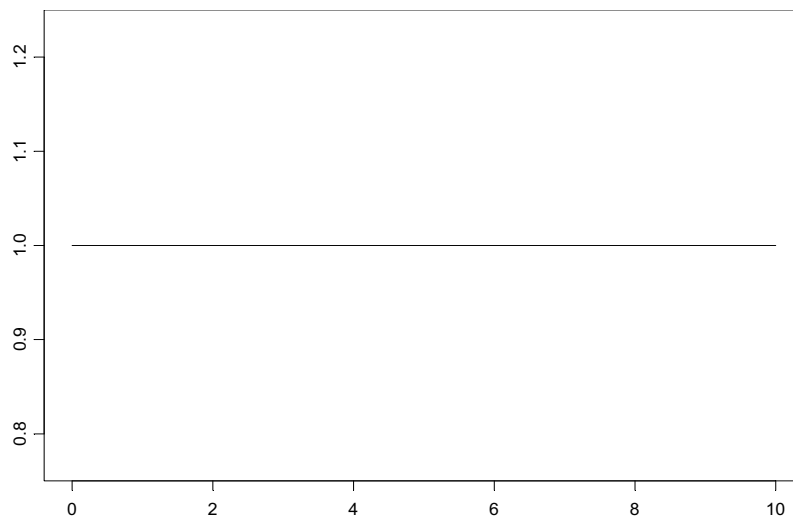


Figure 2.1: *Constant basis function over the domain (0,10).*

The `constantBasis` constructor is surprisingly useful. Measurements on subjects that do not change with time (e.g., sex) can be included in functional models as functions with a constant basis. Indeed, linear regression models can be viewed as functional linear models in which the basis functions are constant.

bsplineBasis

Piecewise polynomial splines consist of smoothly joined polynomials, where each polynomial is defined between two values called *knots*. In piecewise polynomials, function values of polynomials defined in adjacent intervals are constrained to be equal at the knots, and smoothness is obtained by constraining derivatives to be equal at the knots as well. In B-splines of order k (see, e.g., deBoor, 1978, or Green and Silverman, 1994), derivatives of order up to $k - 1$ are required to be equal at the knots in adjacent polynomials. The *order* of a B-spline is one more than the degree of the piecewise polynomials used in the fit. Thus, an order 4 B-spline is a piecewise cubic (degree 3) polynomial in which the values of the first and second derivatives, in addition to the function values, match at the knots. B-splines are usually preferred over piecewise polynomials because they give a smoother fit to the data.

The `bsplineBasis` constructor allows as input the domain of the function, the order of the spline, the number of B-spline basis functions, and the location of the knots (although not all of these are needed since, for example, the number of knots and the order of the spline determine the number of basis functions). Ramsay and Silverman (1997) recommend that the order of the spline to be at least as high as the highest-order derivative of interest plus three (or, equivalently, that the degree of the spline be equal to the number of desired derivatives plus two). Using this rule, the highest-order derivative of interest would be a smooth cubic spline.

Knots should be placed appropriately around features in the function such as maxima and minima, with fewer knots in locations where the function shows little variability. The exact knot location is not usually important, but using too few knots may lead to significant error in the functional representation, while using too many knots may lead to overfitting of the data. One common practice is to use a fixed grid of (many) knots, and then apply smoothing methods (see Chapter 4) to eliminate problems due to overfitting. An alternative is to use

regression methods to find “optimal” knot locations (see, e.g., Friedman and Silverman, 1989). Currently only the smoothing methods are supported in S+FDA.

The follow constructs and plots the functions in a B-spline basis:

```
> basis <- bsplineBasis(fDomain=c(0,10), norder=6,
                        nbasis=20)
> plot(basis)
```

The resulting plot is given in Figure 2.2.

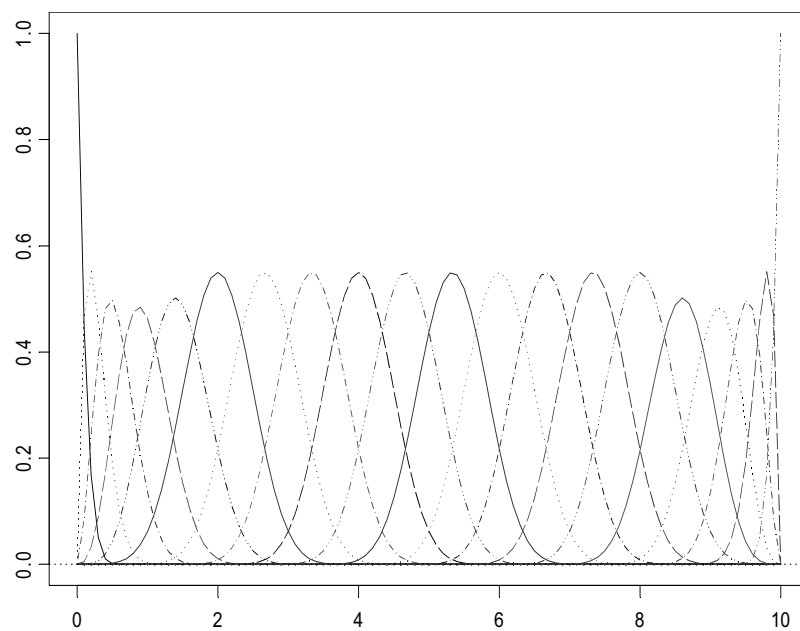


Figure 2.2: *B-spline basis functions over the domain (0,10).*

FourierBasis

Periodic functions such as time series are represented by objects of class `FourierBasis`. Functions represented in this basis have expansions of the following form:

$$f(t) = \beta_0 + \beta_1 \sin \omega t + \beta_2 \cos \omega t + \beta_3 \sin 2\omega t + \beta_4 \cos 2\omega t + \dots$$

Chapter 2 Basis Objects and Operations

The basis functions are $b_0(t) = 1$, $b_1(t) = \sin \omega t$, $b_2(t) = \cos \omega t$, $b_3(t) = \sin 2\omega t$, etc. The first basis function is a constant, the second a sine function, the third a cosine function with the same period, and so on.

In a Fourier basis there are always an odd number of basis functions, and the period is taken to be the same as the domain of the function. The basis functions are estimated using a fast Fourier transform. A plot of an object of class `FourierBasis` can be obtained as follows:

```
> basis <- FourierBasis(fDomain=c(0,10), nbasis=4)
> plot(basis)
```

The result is shown in Figure 2.3.

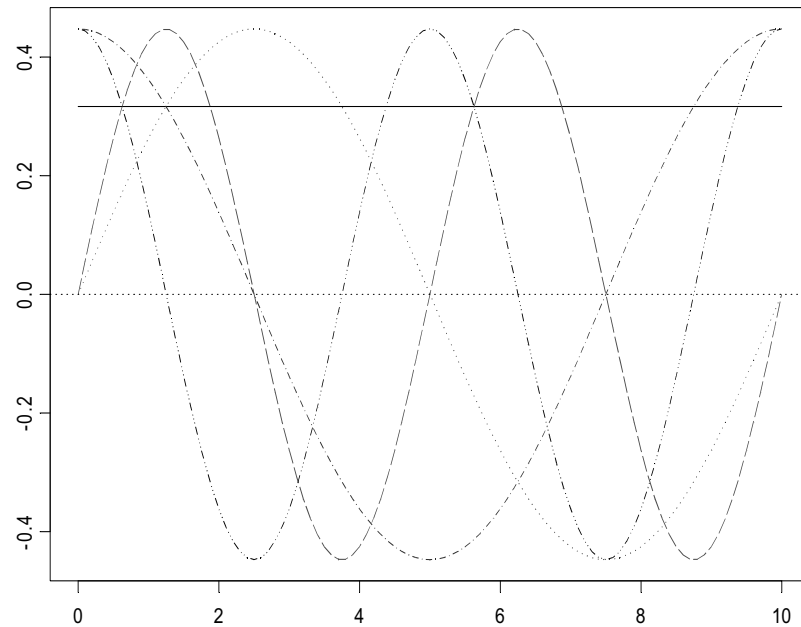


Figure 2.3: *Fourier basis functions over the domain (0,10).*

Notice that although four basis functions were specified, we obtained five basis functions. This is because a Fourier basis always has an odd number of basis functions: the constant basis function plus an equal number of sin and cos basis functions.

polynomialBasis The basis functions in an object of class `polynomialBasis` are the terms of a polynomial centered at a fixed scalar c , for example:

$$p(t) = \beta_0 + \beta_1(t - c) + \beta_2(t - c)^2 + \beta_3(t - c)^3 + \beta_4(t - c)^4$$

The number of basis functions is equal to the degree of the polynomial, plus one. In the polynomial shown, the number of basis functions is five, one for each term in the polynomial.

A plot of the first five functions in a polynomial basis over the domain $(0,10)$ and centered at 5 is obtained as follows (see Figure 2.4):

```
> basis <- polynomialBasis(fDomain=c(0,10),nbasis=5,ctr=5)
> plot(basis)
```

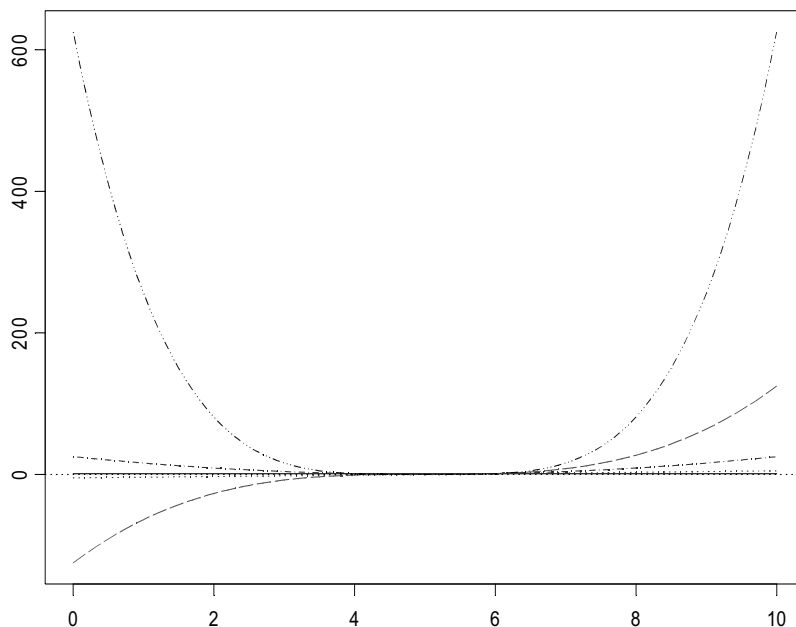


Figure 2.4: First five polynomial basis functions centered at 5 over domain $(0,10)$.

Note that any smooth function can be represented by a polynomial basis expansion, since the terms are those of its power series about a point. Unfortunately, the basis functions in polynomial bases tend to

be highly correlated and often exhibit numerical instability. Also, the fit may be poor away from the center c , and adaptation to local features removed from c may not be possible without a very large number of basis functions. Although polynomial bases play an important role in classical analysis, they have been superseded by the more flexible B-spline bases in applications.

polygonalBasis Objects of class `polygonalBasis` are piecewise linear, equivalent to an order 2 (linear) B-spline basis. This basis has the advantage of simplicity, and the disadvantage that the first derivatives are step functions. The `polygonalBasis` constructor has a single argument, `fArgs`, containing the points at which the function changes (the knots in a linear B-spline basis). Perhaps the most common way to use a polygonal basis is to set the knots in `fArgs` equal to the observation times of the function so that the unsmoothed function linearly interpolates the observed data. Smoothing is then used to prevent overfitting.

A plot of five polygonal basis functions is obtained as follows (see Figure 2.5):

```
> basis <- polygonalBasis(fArgs=seq(0, 10, length=5))
> plot(basis)
```

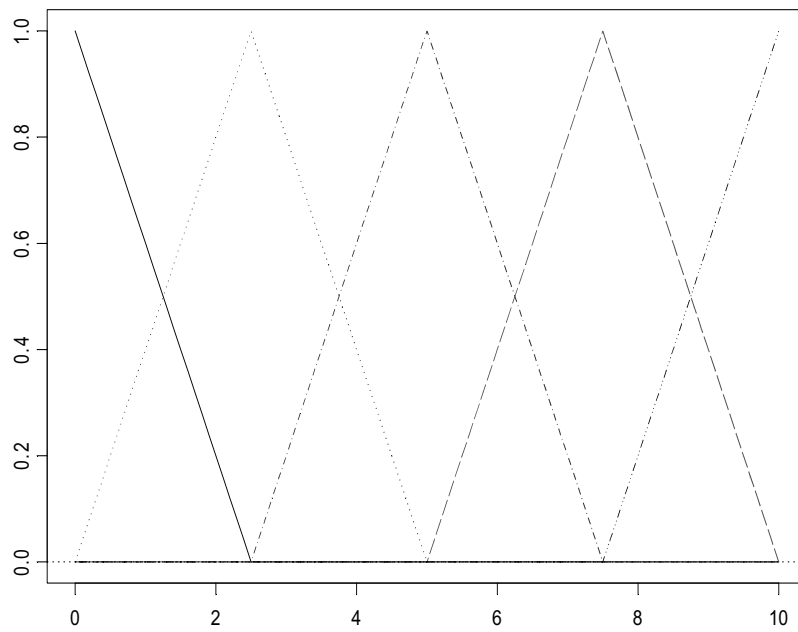


Figure 2.5: Polygonal basis functions on the domain $(0, 10)$.

exponentialBasis

Objects of class `exponentialBasis` consist of terms of the form $\exp(k_i t)$ where the k_i are user-specified rate constants. As with polynomial bases away from their center, exponential bases do not adapt well away from the origin. For this reason, exponential bases should only be chosen in special circumstances. A plot of five exponential basis functions is obtained as follows (see Figure 2.6):

```
> basis <- exponentialBasis(fDomain=c(0, 10),
                             ratevec=c(-2, -1, -0.5, -0.25, -0.1))
```

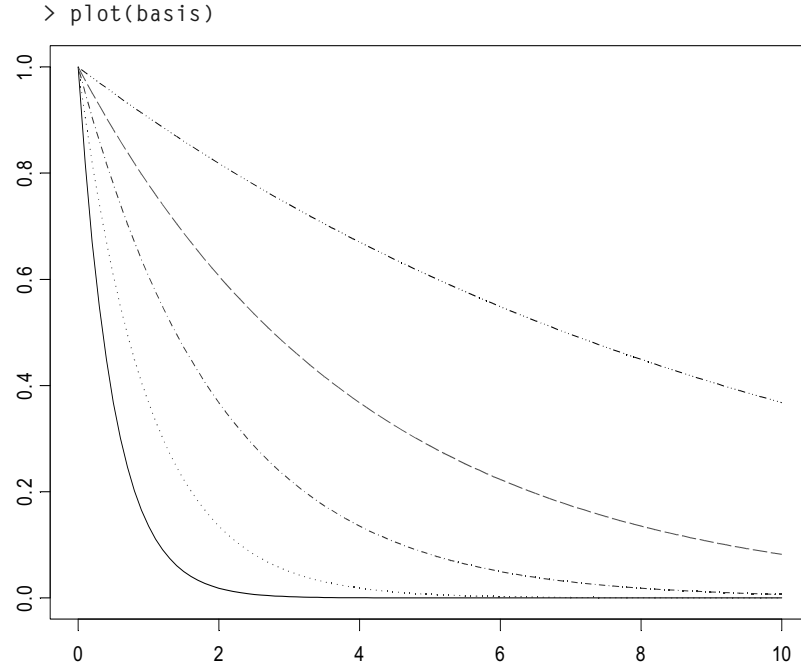


Figure 2.6: Exponential basis functions on the domain $(0,10)$.

compositeBasis

Objects of class `compositeBasis` represent bases whose terms are selected from one or more fundamental bases. A composite basis is a sum of terms of the form:

$$b_k(x) = \sum_{j=1}^{n_b} \beta_{kj} b_{kj}(x)$$

each of which is a basis expansion. The following is an example of the representation of a function in terms of a composite basis consisting of three different fundamental bases:

$$f(x) = \sum_{j=1}^{n_1} \beta_{1j} b_{1j}(x) + \sum_{j=1}^{n_2} \beta_{2j} b_{2j}(x) + \sum_{j=1}^{n_3} \beta_{3j} b_{3j}(x)$$

(there are n_k basis functions of the k th type in this composite basis).

The advantage of a composite basis is the ability to adapt to more complex functions with fewer basis functions. As a simple example, consider a time series with a baseline and a linear trend in time. Such a function can be represented by the composite of a polynomial basis with two terms (a constant plus a linear term) to account for the linear trend, and a Fourier basis to represent the detrended time series.

In a more complex example of a composite basis, we use a constant basis to account for a baseline, an exponential basis to account for exponential decay, and a Fourier basis to account for periodic behavior. Such a basis can be constructed as follows:

```
> basis1 <- constantBasis(fDomain=c(0, 10))
> basis2 <- exponentialBasis(fDomain=c(0, 10), ratevec=-1)
> basis3 <- FourierBasis(fDomain=c(0, 10))
> basis <- compositeBasis(basis1, basis2, basis3)
> plot(basis)
```

The resulting plot is shown in Figure 2.7.

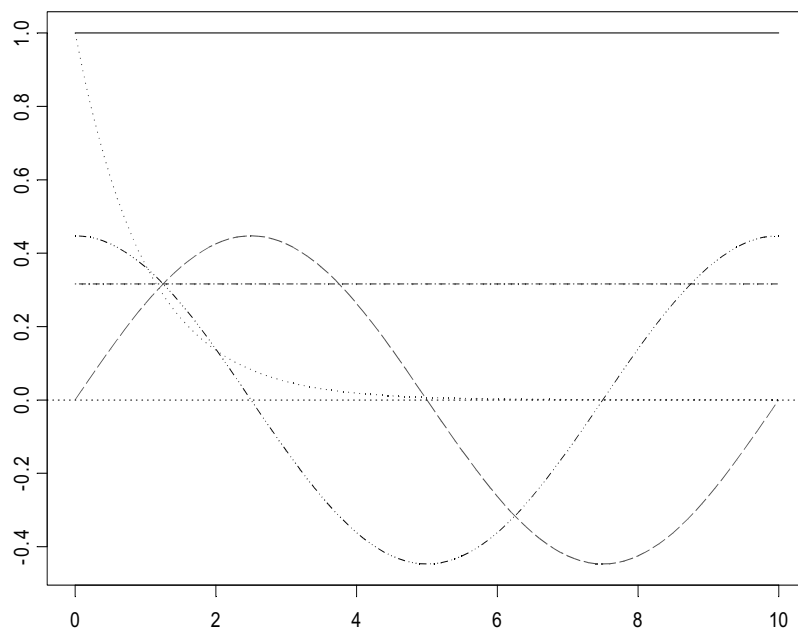


Figure 2.7: A composite basis consisting of a constant, an exponential decay, and a Fourier series over the domain $(0, 10)$.

Chapter 2 Basis Objects and Operations

Although we explicitly included a constant basis for a baseline (mean) term, in fact a constant basis functions is already included in the Fourier basis.

CREATING BIVARIATE BASES

Separable Bases

An S-PLUS constructor function `fProdBasis` creates product basis functions for bivariate functional data. Simply supply the two univariate bases that form the product.

For example, to create a product basis consisting of a Fourier and a B-spline univariate basis:

```
> basis1 <- FourierBasis(fDomain=c(0,365), nbasis=11)
> basis2 <- bsplineBasis(fDomain=c(0,52), nbasis=20)
> basisProd <- fProdBasis(basis1, basis2)
```

The result is an object of class `fProdBasis`, which contains as two components the univariate bases:

```
> names(basisProd)
[1] "basis1" "basis2"
```

Finite Element Bases

An S-PLUS constructor function `fFinElemBasis` creates finite element basis functions for bivariate functional data.

```
> args(fFinElemBasis)

function(xDomain, yDomain, params)
```

For example:

```
> basisFinElem <- fFinElemBasis(c(0, 10), c(0, 10),
                               c(10, 10))
> basisFinElem
```

```
Linear Basis for 2D Finite Element Method:
```

```
Domain x: 0 10
        y: 0 10
```

```
Number of Basis: 121
Number of Element: 200
```

The result is an object of class `fFinElemBasis` which contains the following components:

```
> names(basisFinElem)
```

Chapter 2 Basis Objects and Operations

```
[1] "fDomain" "Vnode" "Velem" "basis.coe"
```

The plot method for the object of class `fFinElemBasis` is also available for graphic view of the *i*-th basis function, as shown in Figure 2.8:

```
> plot(basisFinElem)
```

By default, the method picks one basis function in the middle of the domain. In this example, it chooses the 60th basis function.

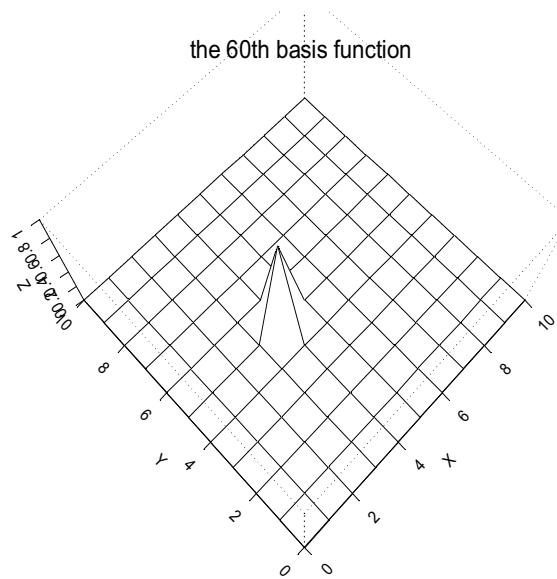


Figure 2.8: The 60th basis function of *basis2D*, an object of class *fFinElement*.

OPERATIONS ON UNIVARIATE BASES

Derivatives

The function `fEval` produces values of basis functions and their derivatives (when applicable).

Applied to an object that inherits from class `fBasis`, the required input to `fEval` includes the basis object, the points at which the basis is to be evaluated, and the desired order of the derivative. The output is the evaluated derivative (or function value) for all basis functions at the specified points.

As an example, evaluate and plot the first derivatives of the exponential basis (displayed in Figure 2.6) over a sequence of numbers of length 100 over the domain (0, 10) using code:

```
> basis <- exponentialBasis(fDomain=c(0,10),  
                           ratevec=c(-2, -1, -0.5, -0.25, -0.1))  
> dBasis <- fEval(basis, fArg=seq(0, 10, length=100),  
                  linDop=fDop(1))  
> matplot(seq(0, 10, length=100), dBasis, type="l")
```

The resulting plot is given in Figure 2.9.

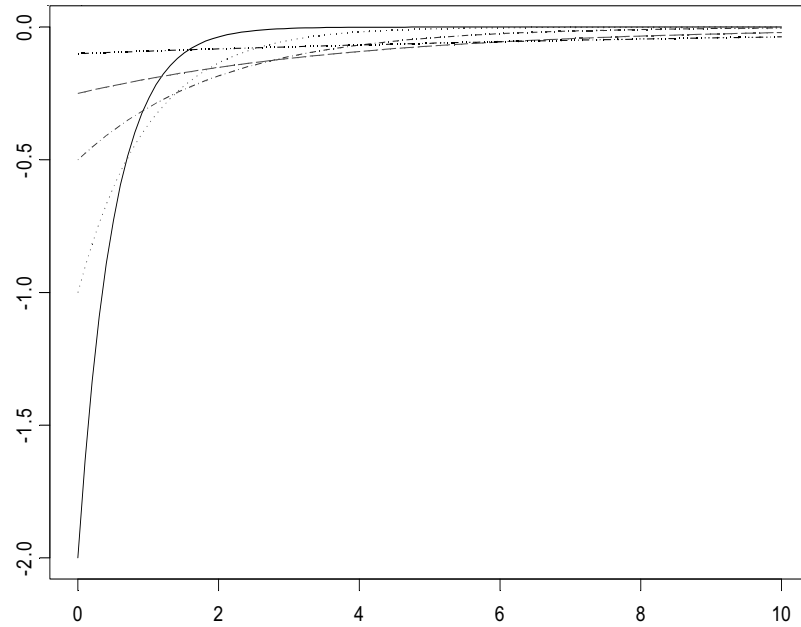


Figure 2.9: First derivatives of Figure 2.6.

Integrals

It is also possible to integrate the basis functions. The function `fInt` integrates all of the functions in a basis over a specified range. For example, the exponential basis in Figure 2.6 can be integrated as follows:

```
> basis <- exponentialBasis(fDomain=c(0,10),  
                             ratevec=c(-2, -1, -0.5, -0.25, -0.1))  
> fInt(basis, limits=c(0, 10))
```

This results in the following vector of integral values:

```
1.000 0.999 0.999 0.999 0.998 0.994 0.981 0.950 0.864 0.632
```

It is often less computationally expensive to integrate the basis functions over a specified range, and then use these integrals to evaluate the integrals of the functions of interest.

Inner Products Inner products of basis functions are required in many of the analyses provided in S+FDA. These inner products are integrals of the form:

$$\int b_1(s)b_2(s)ds$$

for any two basis functions $b_1(s)$ and $b_2(s)$, where the two bases are assumed to have the same domain and the integration is over the entire domain. For example, inner products of the basis functions within a particular basis are computed using the function `fInProd` as follows:

```
> fInProd(basis, basis)
```

The result is a square matrix containing the inner products with dimension equal to the number of functions in `basis`. This inner product matrix is used, for example, in functional regression models. Inner products of the derivatives of basis functions are also possible.

OPERATIONS ON BIVARIATE BASES

Derivatives

Evaluating bivariate basis functions of class `fFinElemBasis` and their derivatives (when applicable) are the main operations of interest. Integrals of functions are not calculated as the linear combination of the integrals of basis functions, therefore S+FDA does not implement integrals of basis functions.

Evaluate the basis functions at specified arguments by

```
> fEval(basis, fArg1, fArg2=fArg1, xDeriv=0, yDeriv=0)
```

where arguments `xDeriv` and `yDeriv` can be specified by 1 for the first order derivative on x and y . It returns a matrix of basis function values $\beta_s(x_j, y_j)$ with $s = 1, \dots, nBasis$ and $j = 1, \dots, nPoint$.

<http://www.springer.com/978-0-387-24969-8>

S+Functional Data Analysis

User's Manual for Windows ®

Clarkson, D.B.; Fraley, C.; Gu, C.; Ramsay, J.

2005, X, 192 p., Softcover

ISBN: 978-0-387-24969-8