Rfssa: An R Package for Functional Singular Spectrum Analysis

by Hossein Haghbin, Jordan Trinka and Mehdi Maadooliat

Abstract Functional Singular Spectrum Analysis (FSSA) is a non-parametric approach for analyzing Functional Time Series (FTS) and Multivariate FTS (MFTS) data. This paper introduces Rfssa, an R package that addresses implementing FSSA for FTS and MFTS data types. Rfssa provides a flexible container, the funts class, for FTS/MFTS data observed on one-dimensional or multi-dimensional domains. It accepts arbitrary basis systems and offers powerful graphical tools for visualizing time-varying features and pattern changes. The package incorporates two forecasting algorithms for FTS data. Developed using object-oriented programming and Rcpp/RcppArmadillo, Rfssa ensures computational efficiency. The paper covers theoretical background, technical details, usage examples, and highlights potential applications of Rfssa.

1 Introduction

In recent times, advancements in data acquisition techniques have made it possible to collect data in high-resolution formats. Due to the presence of temporal-spatial dependence, one may consider this type of data as *functional data*. Functional Data Analysis (FDA) focuses on developing statistical methodologies for analyzing data represented as functions or curves. While FDA methods are particularly well-suited for handling smooth continuum data, they can also be adapted and extended to effectively analyze functional data that may not exhibit perfect smoothness, including high-resolution data and data with inherent variability. The widely-used `R` package for FDA is **fda** (Ramsay et al., 2023), which is designed to support analysis of functional data, as described in the textbook by Ramsay and Silverman (2005). Additionally, there are over 40 other `R` packages available on CRAN that incorporate functional data analysis, such as **funFEM** (Bouveyron, 2021), **fda.usc** (Febrero-Bandle and de la Fuente, 2012), **refund** (Goldsmith et al., 2023), **fdapace** (Gajardo et al., 2022), **funData** (Happ-Kurz, 2020), **ftsspec** (Tavakoli, 2015), **rainbow** (Shang and Hyndman, 2022), and **ftsa** (Hyndman and Shang, 2023). One crucial initial requirement for any of these packages is to establish a framework for representing and storing infinite-dimensional functional observations. The **fda** package, for instance, employs the `fd` class as a container for functional data defined on a one-dimensional (1D) domain. An `fd` object represents functional data as a finite linear combination of known basis functions (e.g., Fourier, B-splines, etc.), storing both the basis functions and their respective coefficients for each curve. This representation aligns with the practical implementation found in many papers within the field of FDA. Conversely, several other `R` packages store functional data in a discrete form evaluated on grid points (e.g., **fda.usc**, **refund**, **funData**, **rainbow**, and **fdapace**). These packages also provide the capability to analyze functions beyond the one-dimensional case, such as image data treated as two-dimensional (2D) functions (e.g., **refund**, **fdasrvf**, and **funData**). To the best of our knowledge, packages that support representation beyond 1D functions utilize the grid point representation for execution and storage. Moreover, recent packages have been developed to handle multivariate functional data, which consist of more than one function per observation unit. Examples of such packages include **roahd**, **fda.usc**, and **funData**. While some recent FDA packages have focused on analyzing and implementing techniques for Functional Time Series (FTS), where sequences of functions are observed over time, none of them handle Multivariate FTS (MFTS) or multidimensional MFTS. For example, see the packages **ftsspec**, **rainbow**, and **ftsa**. In summary, there is still a need for a unified and flexible container for FTS/MFTS data, defined on either one or multidimensional domains. The `funts` class in **Rfssa** (Haghbin et al., 2023), the package discussed in this article, aims to address this gap. One of the primary contributions of the package is its capacity to handle and visualize 2-dimensional FTS, including image data. Furthermore, the package accommodates MFTS, especially when observed on distinct domains. This flexibility empowers users to analyze and visualize FTS with multiple variables, even when they do not share the same domain. Notably, the **Rfssa** package introduces novel visualization tools (as exemplified in Figure 5). These tools include heatmaps and 3D plots, thoughtfully designed to provide a deeper understanding of functional patterns over time. They enhance the ability to discern trends and variations that might remain inconspicuous in conventional plots. An additional feature of the `funts` class is its ability to accept any arbitrary basis system as input for the class constructor, including FDA basis functions or even empirical basis represented as matrices evaluated at grid points. The classes in the **Rfssa** package are developed using the S3 object-oriented programming system, and for computational efficiency, significant portions of the package are implemented using the **Rcpp**/**RcppArmadillo** packages. Notably, the package includes a shiny web application that provides a user-friendly GUI for implementing Functional Singular

Spectrum Analysis (FSSA) on real or simulated FTS/MFTS data. The **Rfssa** package was initially developed to implement FSSA for FTS, as discussed in the work of Haghbin et al. (2021). FSSA extends Singular Spectrum Analysis (SSA), a model-free procedure commonly used to analyze time series data. The primary goal of SSA is to decompose the original series into a collection of interpretable components, such as slowly varying trends, oscillatory patterns, and structureless noise. Notably, SSA does not rely on restrictive assumptions like stationarity, linearity, or normality (Golyandina and Zhigljavsky, 2013). It's worth noting that SSA finds applications beyond the functional framework, including smoothing and forecasting purposes (Hassani and Mahmoudvand, 2013; de Carvalho and Rua, 2017). The non-functional version of FSSA, known as SSA, has previously been implemented in the **Rssa** package (Golyandina et al., 2015) and the **ASSA** package (de Carvalho and Martos, 2020). The **Rssa** package provides various visualization tools to facilitate the grouping stage, and the **Rfssa** package includes equivalent functional versions of those tools (Golyandina et al., 2018). While the foundational theory of FSSA was originally designed for univariate FTS, it has since been extended to handle multidimensional FTS data, referred to as Multivariate FSSA (MFSSA) (Trinka et al., 2022). Furthermore, in line with the developments in SSA for forecasting, two distinct algorithms known as Recurrent Forecasting (FSSA R-forecasting) and Vector Forecasting (FSSA V-forecasting) were introduced for FSSA by Trinka et al. (2023). Both of these forecasting algorithms, along with the capabilities for handling MFSSA, have been seamlessly integrated into the most recent version of the **Rfssa** package. The remainder of this manuscript is organized as follows. Section 2 introduces the FTS/MFTS data preparation theory used in the `funts` class. Section 3 discusses the FSSA methodology, including the basic schema of FSSA, FSSA R-forecasting, and FSSA V-forecasting. Technical details of the **Rfssa** package are provided in Section 4, where we describe the available classes in the package and illustrate their practical usage with examples of real data. Section 5 focuses on the reconstruction stage and FSSA/MFSSA forecasting. In Section 6, we provide a summary of the embedded shiny app. Finally, we conclude the paper in Section 7.

2 Data preparation in FTS

Define $\mathbf{y}_N = (y_1, \ldots, y_N)$ to be a collection of observations from an FTS. In the theory of FTS, y_i's are considered as functions in the space $\mathbb{H} = L^2(\mathcal{T})$ where $\mathcal{T}$ is a compact subset of $\mathbb{R}$. Let $s \in \mathcal{T}$ and consider $y_i(s) \in \mathbb{R}^p$, the sequence of $\mathbf{y}_N$ is called (univariate) FTS if $p = 1$, and multivariate FTS (or MFTS) if $p > 1$. In the realm of functional data analysis, we operate under the assumption that the underlying sample functions, denoted as $y_i(\cdot)$, exhibit smoothness for each sample i, where $i = 1, \ldots, N$. Nevertheless, in practical scenarios, observations are typically acquired discretely at a set of grid points and are susceptible to contamination by random noise. This phenomenon can be represented as follows:

$$Y_{i,k} = y_i(t_k) + \varepsilon_{i,k}, \quad k = 1, \ldots, K. \tag{1}$$

In this expression, $t_k \in \mathcal{T}$, and K denotes the count of discrete grid points across all samples. The $\varepsilon_{i,k}$ terms represent i.i.d. random noise. To preprocess the raw data, it is customary to employ smoothing techniques, converting the discrete observations $Y_{i,k}$ into a continuous form, $y_i(\cdot)$. This is typically performed individually for each variable and sample. One widely used approach is finite basis function expansion (Ramsay and Silverman, 2005). In this method, a set of basis functions $\{\nu_i\}_{i\in\mathbb{N}}$ is considered (not necessarily orthogonal) for the function space $\mathbb{H}$. Each sample function $y_i(\cdot)$ in (1) is then considered as a finite linear combination of the first d basis functions:

$$y_i(s) = \sum_{j=1}^{d} c_{ij}\nu_j(s). \tag{2}$$

Subsequently, the coefficients c_{ij} can be estimated using least square techniques. By adopting the linear representation form for the functional data in (2), we establish a correspondence between each function $y_i(\cdot)$ and its coefficient vector $\boldsymbol{c}_i = (c_{ij})_{j=1}^{d}$. As a result, the coefficient vectors $\boldsymbol{c}_i$ can serve to store and retrieve the original functions, $y_i(\cdot)$'s. This arises from the inherent isomorphism between two finite vector spaces of the same dimension (in this case, d). Consequently, $\boldsymbol{c}_i$'s are stored as the primary attribute of `funts` objects within the **Rfssa** package. Take two elements $x, y \in \mathbb{H}$ with corresponding coefficient vectors $\boldsymbol{c}_x$ and $\boldsymbol{c}_y$. Then, the inner product of x, y can be computed in matrix form as $\langle x, y\rangle = \boldsymbol{c}_x^\top \mathbf{G}\boldsymbol{c}_y$, where $\mathbf{G} = [\langle \nu_i, \nu_j\rangle]_{i,j=1}^{d}$ is the Gram matrix. It is important to note that $\mathbf{G}$ is Hermitian. Furthermore, because the basis functions $\{\nu_i\}_{i=1}^{d}$ are linearly independent, $\mathbf{G}$ is positive definite, making it invertible (Horn and Johnson, 2012, Thm. 7.2.10). Moreover, let $A : \mathbb{H} \to \mathbb{H}$ be a linear operator and $y = A(x)$. Then, $\boldsymbol{c}_y = \mathbf{G}^{-1}\mathbf{A}\boldsymbol{c}_x$, where $\mathbf{A} = [\langle A(\nu_j), \nu_i\rangle]_{i,j=1}^{d}$ is called the corresponding matrix of the operator A. It is worth noting that while the FSSA theory extends to arbitrary dimensions, practical implementation for dimensions greater than 2 introduces considerable computational complexity. Moreover, high-dimensional FTS data are relatively rare in real-world applications. Therefore, within the **Rfssa** package, we have chosen to confine the `funts` object to support functions observed over domains that are one or two-dimensional. In the **Rfssa** package, the task of preprocessing the raw discrete observations and converting those to the `funts` object is assigned to the `funts(·)` constructor.

3 An overview of the FSSA methodology

FSSA is a nonparametric technique to decompose FTS and MFTS, and the methodology can also be used to forecast such data (Haghbin et al., 2021; Trinka et al., 2022, 2023); it can also be used as a visualization tool to illustrate the concept of seasonality and periodicity in the functional space over time.

3.1 Basic schema of FSSA

Basic FSSA consists of two stages where each stage includes two steps. We outline the four steps of the FSSA algorithm here.

I) **First stage: decomposition**

1. *Embedding*
For a positive integer, $L < N/2$, let $\mathbb{H}^L$ be the Cartesian product of L copies of $\mathbb{H}$ and define the *trajectory* operator $\boldsymbol{\mathcal{X}} : \mathbb{R}^K \to \mathbb{H}^L$ with

$$\boldsymbol{\mathcal{X}}\boldsymbol{a} := \sum_{j=1}^{K} a_j \boldsymbol{x}_j, \qquad \boldsymbol{a} = (a_1, \dots, a_K)^\top \in \mathbb{R}^K. \tag{3}$$

where $K = N - L + 1$ and

$$\boldsymbol{x}_j(s) := \Big(y_j(s), y_{j+1}(s), \dots, y_{j+L-1}(s)\Big)^\top, \ j = 1, \dots, K, \tag{4}$$

are called *lag-vectors*. One may consider the trajectory operator $\boldsymbol{\mathcal{X}}$ in (3) as an $L \times K$ matrix with functional entries in the form of

$$\boldsymbol{\mathcal{X}}(s) = \begin{bmatrix} y_1(s) & y_2(s) & y_3(s) & \cdots & y_K(s) \\ y_2(s) & y_3(s) & y_4(s) & \cdots & y_{K+1}(s) \\ y_3(s) & y_4(s) & y_5(s) & \cdots & y_{K+2}(s) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_L(s) & y_{L+1}(s) & y_{L+2}(s) & \cdots & y_N(s) \end{bmatrix}. \tag{5}$$

Note that the antidiagonal elements of the matrix in (5) are all equal. Such matrices are called Hankel, and since $\boldsymbol{\mathcal{X}}(s)$ is a Hankel matrix for any s in the domain, we call $\boldsymbol{\mathcal{X}}$ a Hankel operator. In practice, a main challenge is how to use the original basis of $\mathbb{H}$ to represent the lag-vectors in $\mathbb{H}^L$. To do this, one may define a quotient sequence q_k, and a remainder sequence r_k, by $k = (q_k - 1)L + r_k$, where $1 \le r_k \le L$, and $1 \le q_k \le d$. Now, consider $\boldsymbol{\phi}_k$ as a functional vector of length L with all zero functions, except the r_k-th element, which is ν_{q_k}. Using this definition, the lag-vector $\boldsymbol{x}_j$ given in (4) can be represented as a linear combination of $\{\boldsymbol{\phi}_k\}_{k=1}^{Ld}$ with the corresponding coefficient vector $\mathbf{b}_j = (c_{1j}, \dots, c_{1,j+L-1}, c_{2j}, \dots, c_{2,j+L-1}, \dots, c_{d,j+L-1})^\top$.

2. *FSVD*
We apply the functional singular value decomposition (FSVD) to $\boldsymbol{\mathcal{X}}$ and obtain a collection of singular values, $\{\sqrt{\lambda_i}\}_{i=1}^r$, orthonormal right singular vectors $\{\mathbf{v}_i\}_{i=1}^r$ (that are elements of $\mathbb{R}^K$), and orthonormal left singular functions $\{\boldsymbol{\psi}_i\}_{i=1}^r$ (that are elements of $\mathbb{H}^L$). The collection $(\sqrt{\lambda_i}, \boldsymbol{\psi}_i, \mathbf{v}_i)$ will be called the i^{th} eigentriple of the FSVD, and r is the rank of $\boldsymbol{\mathcal{X}}$:

$$\boldsymbol{\mathcal{X}} = \sum_{i=1}^{r} \boldsymbol{\mathcal{X}}_i = \sum_{i=1}^{r} \sqrt{\lambda_i}\mathbf{v}_i \otimes \boldsymbol{\psi}_i, \tag{6}$$

where $\boldsymbol{\mathcal{X}}_i : \mathbb{R}^K \to \mathbb{H}^L$ is a rank one elementary operator. To implement the FSVD of $\boldsymbol{\mathcal{X}}$ given in (6), let $\mathbf{B} := [\mathbf{b}_1, \cdots, \mathbf{b}_K]$, $\mathbf{G} := [\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle_{\mathbb{H}^L}]_{i,j=1}^{Ld}$, $\mathbf{X} := \mathbf{G}^{1/2}\mathbf{B}$, and $(\sqrt{\lambda_i}, \boldsymbol{u}_i, \mathbf{v}_i)$'s be the eigentriples of the SVD of the matrix $\mathbf{X}$. It can be shown that $(\sqrt{\lambda_i}, \boldsymbol{\psi}_i, \mathbf{v}_i)$ is the i^{th} eigentriple of the FSVD of $\boldsymbol{\mathcal{X}}$, where the left singular function, $\boldsymbol{\psi}_i$, is corresponding to the coefficient vector $\mathbf{G}^{-1/2}\boldsymbol{u}_i$. See (Haghbin et al., 2021) for more details.

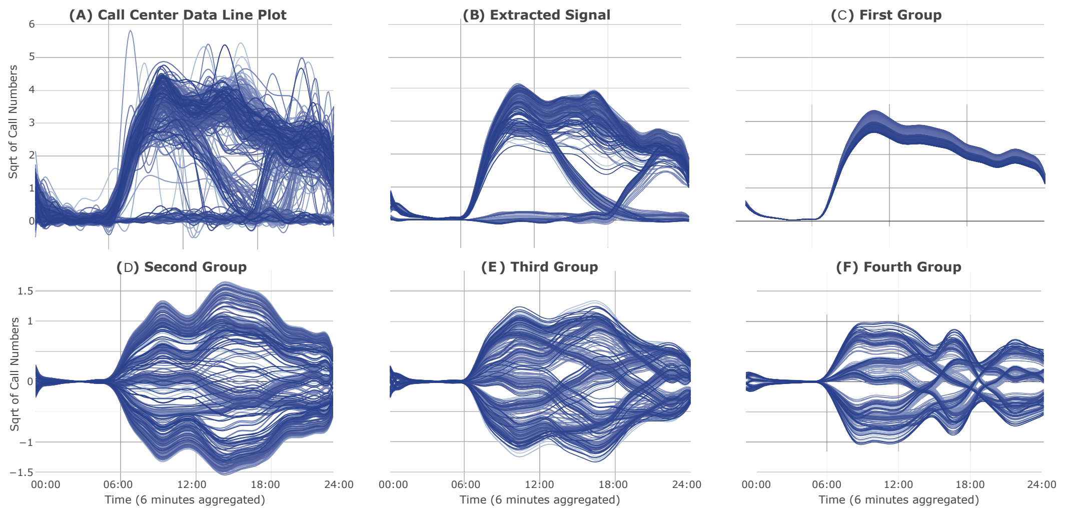


Figure 1: (A): Line plot of `Callcenter` data; (B): FTS reconstructed using $I_s = \{1, \cdots, 7\}$; (C): FTS reconstructed by setting $I_1 = \{1\}$; (D): FTS reconstructed using $I_2 = \{2, 3\}$; (E): FTS reconstructed from $I_3 = \{4, 5\}$; (F): FTS reconstructed from $I_4 = \{6, 7\}$.

These steps can also be extended to the multivariate case, i.e. MFSSA. See Trinka et al. (2022) for more details. In the **Rfssa** package, the results of the decomposition stage are held in an object from the `fssa` class. The constructor, `fssa(·)`, performs the decomposition for both FSSA and MFSSA algorithms and returns an object of class `fssa`. Further discussion about the attributes and methods of the `fssa` class is given in the technical details section.

II) **Second stage: reconstruction**

3. *Grouping*
   We partition the set of indices $\{1, \ldots, r\}$ into disjoint sets I_q, where $q \in \{1, \ldots, m\}$ and $m \leq r$. From here, we obtain the group $\boldsymbol{\mathcal{X}}_{I_q}$ by combining the respective elementary operators accordingly:
   $$\boldsymbol{\mathcal{X}}_{I_q} = \sum_{i \in I_q} \boldsymbol{\mathcal{X}}_i.$$
   Exploratory plots of singular values, right singular vectors, and left singular functions that investigate the different modes of variation extracted in the decomposition stage are used to decide how to form the sets, I_q, and we discuss such plots in further detail in section five.

4. *Hankelization*
   Since each $\boldsymbol{\mathcal{X}}_{I_q}$ is not necessarily Hankel, we perform diagonal averaging of the entries to Hankelize each operator. From each Hankelized $\boldsymbol{\mathcal{X}}_{I_q}$, we obtain an FTS, $\mathbf{y}_N^q$, that describes main characteristics of $\mathbf{y}_N$ such as mean, seasonal, trend, and noise behaviors.

For the reconstruction stage, the **Rfssa** package provides the function `freconstruct(·)` which returns a list of objects of class `funts` associated with the groups specified by the user. If the supplied input to the `fssa(·)` function is an MFTS, the signal extraction process is almost identical as compared to the univariate case with the exception that now, we have that each element of the time series is a tuple of functions comprised of elements observed over one or two-dimensional domains.

As a motivating example, consider the `Callcenter` dataset in Figure 1(A), which has been previously discussed in Maadooliat et al. (2015) and is included in the package. This dataset records the number of calls received by a call center in 1999. Each functional observation corresponds to the square root of the daily call count, and the entire FTS represents all the days in 1999. When dealing with time series data exhibiting known periodic patterns, it is a common practice in SSA to choose the window length, L, as a multiple of this underlying periodicity. This choice ensures that the method effectively extracts the corresponding periodic components (Golyandina et al., 2001). For our illustrative example using the `Callcenter` dataset, we specifically opt for a window length of $L = 28$. This selection allows us to effectively capture the weekly periodic patterns that inherently exist within this functional time series (Haghbin et al., 2021). After applying FSSA, we obtain reconstructed FTS representations under different grouping considerations, as illustrated in Figures 1(B-F). Particularly noteworthy is the reconstructed FTS obtained using the leading 7 eigentriples of the FSVD, as shown in Figure 1(B). It is important to mention that the selection of groups in FTS is typically guided by

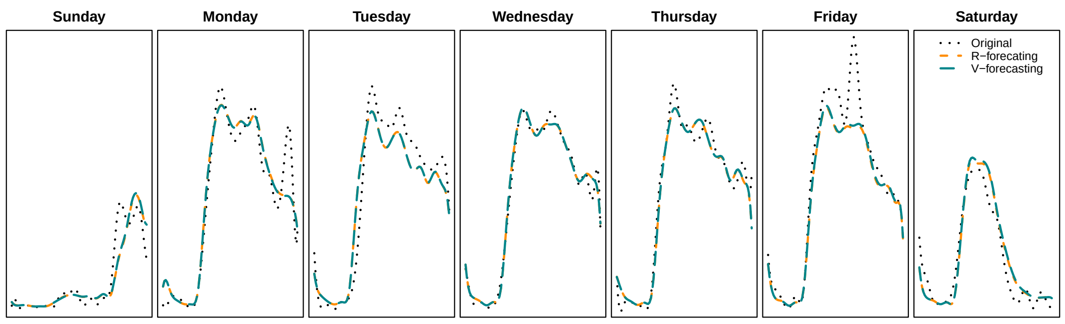


Figure 2: The prediction of the last 7 days of the year 1999 for the `Callcenter` data, based on FSSA R-forecasting and V-forecasting, and comparing the results with the observed functions.

various SSA-type plotting tools, which rely on the similarity in the harmonic structure of extracted elementary components. Such SSA-type plots have been available for non-functional time series in the **Rssa** package, and analogous tools for the functional case have been developed in **Rfssa** (Figure 7). However, it's worth noting that this extension presented its own set of challenges, both in theory and implementation. Detailed code for generating the reconstructed functions shown in Figure 1 can be found in the subsequent sections of this paper.

3.2 FSSA Forecasting

After the decomposition stage, one may perform forecasting instead of reconstruction (Figure 3). R-forecasting and V-forecasting are two main defined approaches in FSSA/MFSSA (Trinka et al., 2023). In both forecasting methods, the goal is to obtain the FTS, $\mathbf{g}_{N+M}^q = \left(g_1^q, \dots, g_{N+M}^q\right)$, where the first N terms are close to $\mathbf{y}_N^q$ and the goal is to forecast the remaining elements, $\{g_i^q\}_{i=N+1}^{N+M}$. Forecasts in the R-forecasting method are obtained using a linear combination of the previous $L-1$ elements of $\mathbf{g}_{N+M}^q$. Consider a positive integer $k < r$ and define $\boldsymbol{\mathcal{V}} = \sum_{i=1}^k \pi_i \otimes \pi_i$, where $\pi_i \in \mathbb{H}$ is the last element of the left singular function $\boldsymbol{\psi}_i$. Define $\boldsymbol{\mathcal{A}}_j : \mathbb{H} \to \mathbb{H}$ such that $\boldsymbol{\mathcal{A}}_j = \sum_{i=1}^k \psi_{j,i} \otimes (\boldsymbol{\mathcal{I}} - \boldsymbol{\mathcal{V}})^{-1} \pi_i$, where $\psi_{j,i}$ is the j^{th} component of $\boldsymbol{\psi}_i$, and $\boldsymbol{\mathcal{I}} : \mathbb{H} \to \mathbb{H}$ is the identity operator. Then the R-forecasting can be obtained using the following equation

$$g_i^q = \begin{cases} y_i^q, & i = 1, \dots, N \\ \sum_{j=1}^{L-1} \boldsymbol{\mathcal{A}}_j g_{i+j-L}^q, & i = N+1, \dots, N+M. \end{cases} \tag{7}$$

Forecasts in the V-forecasting method are obtained by predicting functional vectors. One may define an orthogonal projection, $\mathbf{\Pi} : \mathbb{H}^{L-1} \to \mathbb{H}^{L-1}$, that projects onto the space spanned by $\{\boldsymbol{\psi}_i^\nabla\}_{i=1}^k$, where $\boldsymbol{\psi}_i^\nabla \in \mathbb{H}^{L-1}$ is formed from the first $L-1$ elements of $\boldsymbol{\psi}_i$. Now let $\boldsymbol{\mathcal{Q}} : \mathbb{H}^L \to \mathbb{H}^L$ be given by

$$\boldsymbol{\mathcal{Q}}(\boldsymbol{x}) = \begin{pmatrix} \mathbf{\Pi}\left(\boldsymbol{x}^\Delta\right) \\ \sum_{j=1}^{L-1} \boldsymbol{\mathcal{A}}_j x_j^\Delta \end{pmatrix}, \qquad \boldsymbol{x} \in \mathbb{H}^L \tag{8}$$

where $\boldsymbol{x}^\Delta$ contains the last $L-1$ components of $\boldsymbol{x}$, and x_j^Δ is the j^{th} component of $\boldsymbol{x}^\Delta$. The V-forecasting algorithm is given in the following steps:

1. Define $\boldsymbol{w}_j^q$ as
$$\boldsymbol{w}_j^q = \begin{cases} \boldsymbol{x}_j^q, & j = 1, \dots, K \\ \boldsymbol{\mathcal{Q}} \boldsymbol{w}_{j-1}^q, & j = K+1, \dots, K+M, \end{cases}$$
where $\{\boldsymbol{x}_j^q\}_{j=1}^K$ spans the range of operator $\boldsymbol{\mathcal{X}}_{I_q}$.
2. Form the operator $\boldsymbol{\mathcal{W}}^q : \mathbb{R}^{K+M} \to \mathbb{H}^L$ whose range is linearly spanned by the set $\{\boldsymbol{w}_i^q\}_{i=1}^{K+M}$.
3. Hankelize $\boldsymbol{\mathcal{W}}^q$ in order to extract the FTS $\mathbf{g}_{N+M}^q$.
4. The functions, $g_{N+1}^q, \dots, g_{N+M}^q$, form the M terms of the FSSA vector forecast.

Continuing from the previous example, we utilized the first 358 days of the year to train the FSSA model on the `Callcenter` dataset. Subsequently, we employed the FSSA R-forecasting and V-forecasting methods to make predictions for the final week ($len = 7$). Figure 2 illustrates both the actual and forecasted curves for each day of the week, employing each respective method. Detailed code for

Function	Descriptions	Main arguments	Returns
`funts(·)`	Create FTS/MFTS objects	Discretely sampled data or coefficients, basis system specifications, a set of argument values corresponding to the observations in X, the time specifications arguments.	An object of class `funts`.
`fssa(·)`	Performs the decomposition (including embedding and FSVD steps) stage for FTS/MFTS data.	An object of class `funts` and window length L.	An object of class `fssa`.
`freconstruct(·)`	Performs the reconstruction (including grouping and Hankelization steps) stage.	An object of class `fssa` and a list of numeric vectors includes indices of elementary components of a group.	A list of `funts` objects reconstructed according to the specified groups.
`fforecast(·)`	Performs FSSA R-forecasting or FSSA V-forecasting.	An object of class `fssa`, a list of numeric vectors includes indices of elementary components of a group used for reconstruction and forecasting, and forecast horizon h.	An object of class `fforecast`.

Table 1: A summary of FSSA written functions in the **Rfssa** package.

this process can be found in the subsequent sections. In the **Rfssa** package, we offer the `fforecast(·)` function designed for the execution of R-forecasting or V-forecasting algorithms. This function expects an input argument of class `fssa` and yields an output of class `fforecast`. The latter comprises a list of objects of class `funts`, with each `funts` representing a forecasted group.

4 Technical details of the Rfssa package

The roadmap of the main functions used in the **Rfssa** package is given in Figure 3. The inputs and outputs of these functions are described in Table 1. As it can be seen from Table 1, three classes (`funts`, `fssa` and `fforecast`) are used to support the return objects of these functions. The `funts(·)`, `fssa(·)` and `fforecast(·)` functions are the constructors of the `funts`, `fssa` and `fforecast` classes, respectively. In the rest of this section we present these classes with illustrative examples, and later we describe the reconstruction and forecasting functions in detail.

4.1 The `funts` class

The `funts(·)` constructor is used to create an `S3` object of class `funts`. This object is designed to encapsulate various forms of FTS, including both univariate and multivariate types. It offers a versatile framework for the creation and manipulation of `funts` objects, accommodating different basis systems and dimensions. It accepts the following arguments:

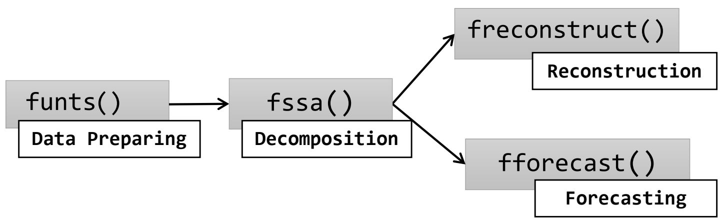
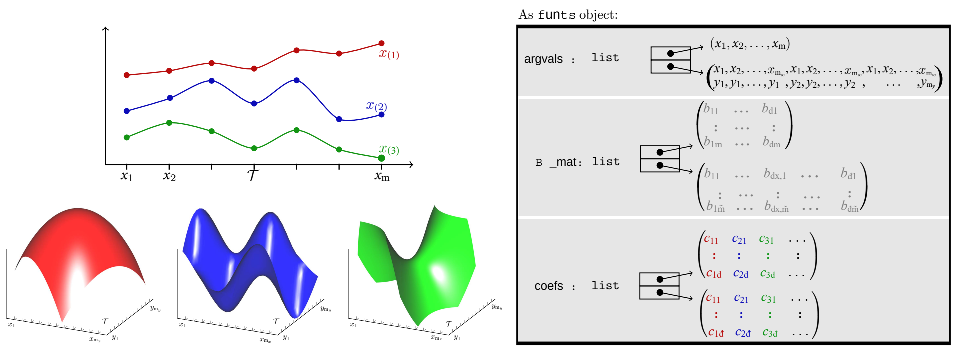


Figure 3: The roadmap of the **Rfssa** package.

- X: A matrix, three-dimensional array, or a list of matrix or array objects. When method="data", it represents the observed curve values at discrete sampling points or argument values. When method="coefs", X specifies the coefficients corresponding to the basis system defined in basisobj. If X is a list, it defines a multivariate FTS, with each element being a matrix or three-dimensional array object. In matrix objects, rows correspond to argument values, and columns correspond to the length of the FTS. In three-dimensional array objects, the first and second dimensions correspond to argument values, and the third dimension to the length of the FTS.
- basisobj: This argument should be an object of class basisfd, a matrix of empirical basis, or a list of basisfd or empirical basis objects. In the case of empirical basis, rows correspond to basis functions, and columns correspond to grid points.
- argval: A vector list of length p, representing a set of argument values corresponding to the observations in X. Each entry in this list should either be a numeric value or a list of numeric elements, depending on the dimension of the domain over which the variable is observed. It's worth noting that these values can vary from one variable to another. If argval is set to NULL, the default values are the integers from 1 to n, where n is the size of the first dimension in the X argument.
- method: This parameter determines the type of the X matrix, and it can take one of two values: coefs or data.
- start and end: Specify the time of the first and last observations. They can be a single positive integer or an object of classes Date, POSIXct, or POSIXt, representing a natural time unit.
- tname, vnames and dnames: These parameters accept strings or lists of strings to specify the names of time, variables, and domains.

The funts(·) constructor offers flexibility to users. Users can either provide their custom basis or request **Rfssa** to generate the basis for them, leveraging the capabilities of the **fda** package. It is assumed that each variable is observed over a regular and equidistant grid. Furthermore, each variable in the funts object is assumed to be observed over either a one or two-dimensional domain, as illustrated in Figure 4. To enhance the representation of time, the funts function introduces two parameters, namely start and end, which capture the time series duration. This design allows users to specify time information in a structured and standardized manner. Users have the flexibility to set start and end using various time and date classes, such as Date, POSIXct, or POSIXt. If users do not provide the start and end arguments, default values are used, with start=1 and end=N, where N represents the length of the time series. This default approach aligns with common practices, as seen in classes like ts in the **stats** package, as well as fts classes in **rainbow** or **ftsa**. An object of class funts is a list encompassing the following elements:

- N: Represents the length of the time series.
- dimSupp: A list specifying the dimensions of the support domain of the variables.
- time: The time object.
- coefs: A list containing basis coefficients.
- basis: A list of basis systems.
- B_mat: Evaluated basis functions on initial arguments.
- argval: Initial arguments of the observed functions.



As funts object:
argvals : list
$(x_1, x_2, \dots, x_m)$
$\begin{pmatrix} x_1, x_2, \dots, x_{m_x}, x_1, x_2, \dots, x_{m_x}, x_1, x_2, \dots, x_{m_x} \\ y_1, y_1, \dots, y_1, y_2, y_2, \dots, y_2, \dots, y_{m_y} \end{pmatrix}$
B_mat: list
coefs : list
$x_{(1)}$
$x_{(2)}$
$x_{(3)}$
x_1
x_2
$\mathcal{T}$
x_m

Figure 4: The main roadmap of funts objects.

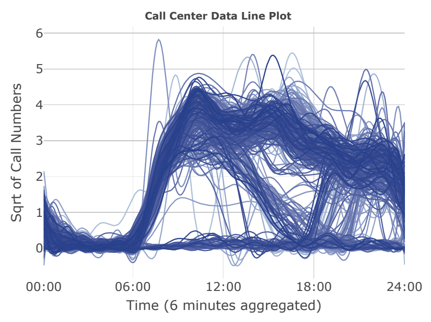
The funts class provides essential functionalities for managing FTS objects, ensuring users have well-defined basic operations. It supports arithmetic operations like addition and multiplication, along with indexing methods. Additionally, three generic methods, `length(·)`, `print(·)`, and `plot(·)`, are available. The `eval.funts(·)` method allows users to evaluate a `funts` object on a specified grid. To determine if an object belongs to the `funts` class, the `is.funts(·)` method is provided. Converting objects from the **fda** package's `fd` class or the **rainbow** package's `fts` class to a `funts` object is made easy using the `as.funts(·)` function. The strength of `funts` objects lies in their powerful visualization capabilities within the field of FTS. Users can create two types of plots using the graphical commands `plot(·)` and `plotly_funts(·)`. The `plot(·)` method utilizes base graphics objects to generate FTS plots. On the other hand, the `plotly_funts(·)` function offers a versatile **plotly** platform for visualizing FTS data, providing several plot types: `line`, `3Dline`, `3Dsurface`, and `heatmap`. These plots make it easier to detect trends or patterns within each curve's behavior over time. Furthermore, users can directly apply the `plotly_funts(·)` function to objects from the `fd`, `fds`, or `fts` classes in packages like **rainbow**, **fds**, **ftsa**, or **fda**, without the need for conversion to `funts` objects. Additionally, when converting objects from packages like `fds` or `fts`, the `xname` and `yname` arguments are automatically captured and used as the `xlab` and `zlab` arguments, ensuring that resulting plots are informative and intuitive. To demonstrate the capabilities of the **Rfssa** package for handling FTS, two illustrative examples are provided. The first example showcases the `Callcenter` dataset consisting of curves observed over a one-dimensional domain. In contrast, the second example involves a bivariate FTS dataset, which includes a sequence of two types of remote sensing images (two-dimensional functional data domain).

Example: Creating `funts` objects for FTS

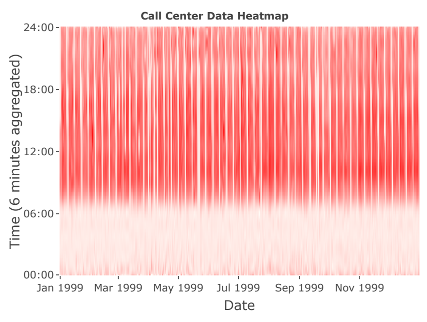
The first example uses the raw `Callcenter` dataset which was discussed before. The `funts` object can be made and plotted using the following codes. The generated plots are shown in Figure 5.

```
# Load necessary libraries
require(Rfssa)
require(fda)
# Load Callcenter data
Call_data <- loadCallcenterData()
# Prepare the data
D <- matrix(sqrt(Call_data$calls)), nrow = 240)
bs1 <- create.bspline.basis(c(0, 23), 22)
# Create a 'funts' object
Call_funts <- funts(D, bs1, start = as.Date("1999-1-1"),
        vnames = "Sqrt of Call Numbers",
        dnames = "Time (6 minutes aggregated)",
        tname = "Date")
xtlab <- list(c("00:00", "06:00", "12:00", "18:00", "24:00"))
xtloc <- list(c(1, 60, 120, 180, 240))
# Generate a line plot using Plotly
plotly_funts(Call_funts, main = "Call Center Data Line Plot",
        xticklabels = xtlab, xticklocs = xtloc)
# Generate a heatmap plot using Plotly
plotly_funts(Call_funts, type = "heatmap", main = "Call Center Data Heatmap",
        xticklabels = xtlab, xticklocs = xtloc)
# Generate a 3D line plot using Plotly
plotly_funts(Call_funts, type = "3Dline", main = "Call Center Data 3Dline plot",
        xticklabels = xtlab, xticklocs = xtloc);
# Generate a 3D surface plot using Plotly
plotly_funts(Call_funts, type = "3Dsurface", main = "Call Center Data 3Dsurface plot",
        xticklabels = xtlab, xticklocs = xtloc);
```

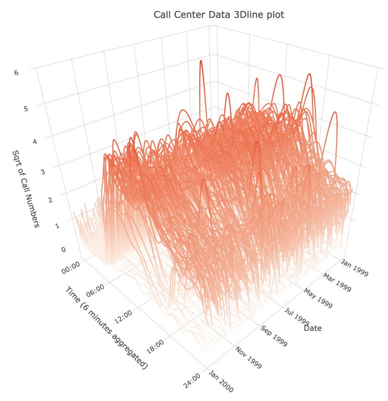
As one can see from Figure 5(A), the overlapping line plot is a common way to view FTS data observed over a one-dimensional domain, where the curves that are recorded on dates closer to the start of 1999 are given in light blue while functions obtained on later dates are plotted in a darker blue. The heatmap plot in Figure 5(B) is a newer technique used to visualize FTS data observed over a one-dimensional domain, allowing the user to see the evolution of the FTS over time instead of relying on different colorings of the curves to specify date. Such one-dimensional FTS can also be represented in a interactive 3D view. To obtain such plots the user simply needs to specify `type="3Dsurface"` or `type="3Dline"`.



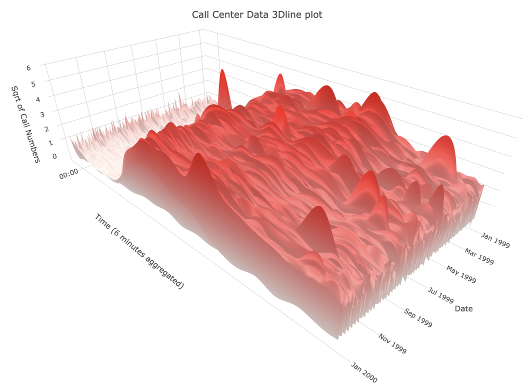
(a) The type of "`line`"



(b) The type of "`heatmap`"



(c) The type of "`3Dline`"



(d) The type of "`3Dsurface`"

Figure 5: The plot types of the generated plots by `plotly_funts(·)`

Example: Creating `funts` objects for MFTS

The second example considers two collections of images where each image is drawn from a region southeast of the city of Jambi, Indonesia, located between latitudes of 1.666792° S - 1.598042° S and longitudes of 103.608963° E - 103.677742° E. The images were recorded using the MODerate Resolution Imaging Spectroradiometer (MODIS) Terra satellite with a resolution of 250 meters every 16 days starting from February 18, 2000 and ending July 28, 2019. The output of each image is the normalized difference vegetation index (NDVI) which is used to quantify how much vegetation is present and enhanced vegetation index (EVI). The NDVI values closer to one being indicative of more vegetation and values closer to zero indicate less vegetation (see Haghbin et al., 2021, for more details). The following code can be used to load the data, define a `funts` object for a MFTS, slice the `funts` object to select a specific variable (here NDVI), and plot the smoothed images over time in an animation, that a snapshot is given in Figure 6.

```
# Load the Jambi dataset
Jambi <- loadJambiData()
# Extract NDVI and EVI array data
NDVI <- Jambi$NDVI
EVI <- (Jambi$EVI)
# Create a list containing NDVI and EVI array data
Jambi_Data <- list(NDVI, EVI)
# Create a multivariate B-spline basis
require(fda)
bs2 <- create.bspline.basis(c(0, 1), 13)
bs2d <- list(bs2, bs2)
bsmv <- list(bs2d, bs2d)
# Create a funts object
Y_J <- funts(X = Jambi_Data,
        basisobj = bsmv,
        start = as.Date("2000-02-18"), end = as.Date("2019-07-28"),
        vnames = c("NDVI", "EVI"), tname = "Date",
        dnames = list(c("Latitude", "Longitude"), c("Latitude", "Longitude")))
# Create a Plotly-based visualization of the NDVI Image
```

```
plotly_funts(Y_J[, 1],
        main = "NDVI Image (Jambi)",
        xticklabels = list(c("103.61\u00B0 E", "103.68\u00B0 E")),
        yticklabels = list(c("1.67\u00B0 S", "1.60\u00B0 S")),
        xticklocs = list(c(0, 1)),
        yticklocs = list(c(0, 1)),
        color_palette = "RdYlGn");
```

4.2 The fssa Class

Once the funts object is created and L is chosen, one can apply the fssa(·) constructor to obtain an S3 object of class fssa that contains our singular values, left singular functions, and right singular vectors. An object of class fssa is a list of right singular functions, which is packed in an object of class funts and the following components:

- values: A numeric vector of singular values.
- L: The specified window length.
- N: The length of the functional time series.
- Y: The original funts object.

The generic plot(·) is developed for the fssa class to help the user make decisions on how to do the grouping stage of FSSA/MFSSA. This method provides a complete set of visualization tools for the user to check the quality of the decomposition stage. These SSA-type plots encompass various types, each providing unique insights into the data:

- "values": Plots the singular values (default).
- "paired": Visualizes pairs of right singular vectors, which is particularly useful for detecting periodic components.
- "wcor": Generates a plot of the W-correlation matrix for the reconstructed objects.
- "vectors": Displays the right singular vectors, aiding in the detection of period length.
- "lcurves": Showcases the left singular functions, assisting in period length detection.
- "lheats": Heatmap plots of left singular functions are available, designed for funts variables observed over one or two-dimensional domains. These plots are valuable for identifying meaningful patterns.
- "periodogram": Periodogram plots of the right singular vectors can be generated, which help detect the frequencies of oscillations in functional data.

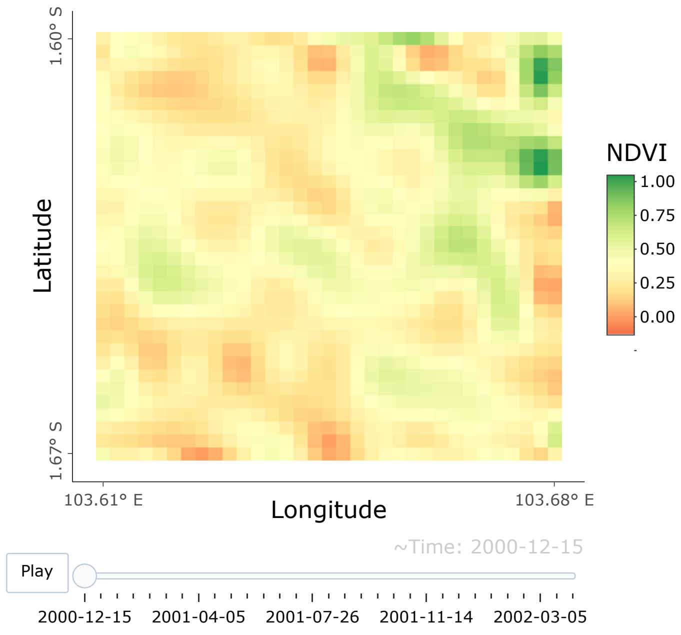


Figure 6: An NDVI snapshot from the city of Jambi, Indonesia.

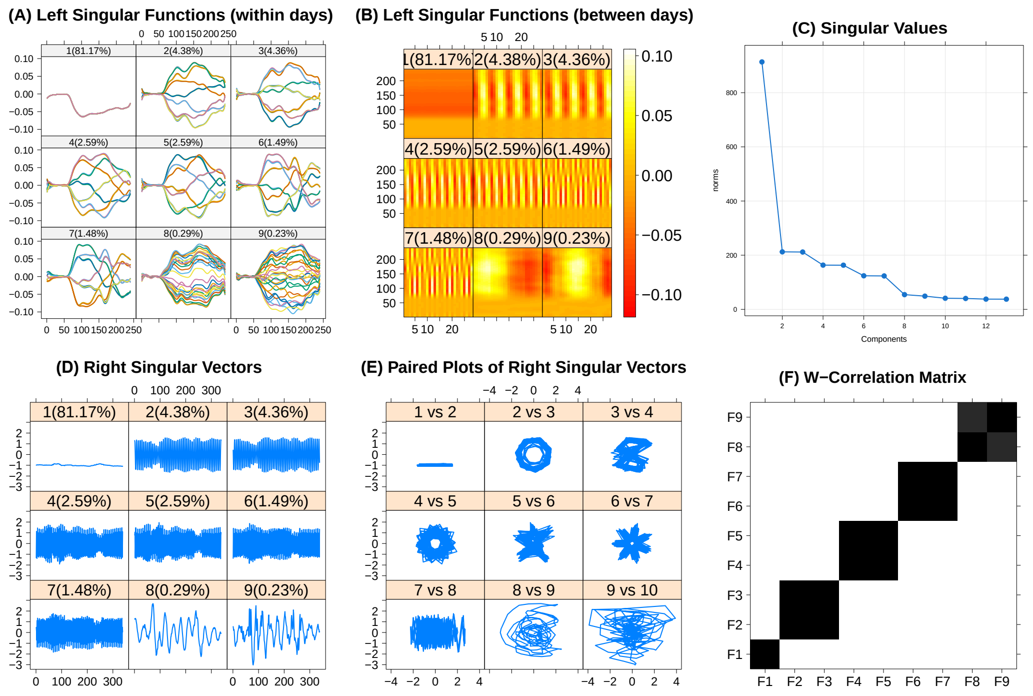


Figure 7: (A): Line plot of the left singular functions, used to identify periodic and trend components; (B): Heatmap of the left singular functions; (C): Scree plot of the singular values often used for grouping; (D): Right singular vectors, used to identify periodic and trend components; (E): Paired plots of the right singular vectors, used for grouping and identifying periodicity in the FTS; (F): Weighted correlation (W-correlation) matrix used for grouping.

While efforts have been made to align these plots in Rfssa with the non-functional versions available in the Rssa package, there are some fundamental differences. Notably, "lheats" and "lcurves" are novel plot types introduced in the Rfssa package for functional data. Additionally, "paired" and "vectors" types are developed based on the right singular vectors. All these plot types utilize the lattice graphics engine. The following example codes generate these plots for the Callcenter dataset.

Example: performing decomposition stage of FSSA

In the rest of the paper, we will use the pre-generated funts class object datasets such as Callcenter which are included within the package. These ready-to-use datasets serve as practical examples and templates, allowing users to test the FSSA procedure without the need to start from scratch with data preprocessing. As mentioned previously, our approach involves performing FSSA with a lag window of $L = 28$. We then generate a variety of SSA-type plots, as illustrated in Figure 7, using the following code:

```
# Load the Callcenter dataset
data("Callcenter")
# Perform FSSA:
fssa_results <- fssa(Callcenter, L = 28)
# FSSA plots:
plot(fssa_results, d = 9, type = "lcurves",
        main = "(A) Left Singular Functions (within days)")
plot(fssa_results, d = 9, type = "lheats",
        main = "(B) Left Singular Functions (between days)")
plot(fssa_results, d = 13, main = "(C) Singular Values")
plot(fssa_results, d = 9, type = "vectors",
        main = "(D) Right Singular Vectors")
plot(fssa_results, d = 10, type = "paired",
        main = "(E) Paired Plots of Right Singular Vectors")
plot(fssa_results, d = 9, type = "wcor",
        main = "(F) W-Correlation Matrix")
```

The W-correlation matrix in Figure 7(F) is built by measuring the correlation between FTS that are reconstructed using the grouping of indices by setting $m = r$ (see the discussion on grouping in section 2). We also have that Figure 7(E) is a plot of successive right singular vectors against one another. Using Figures 7(C, E-F), we identify four groups in the `Callcenter` data such that $I_1 = \{1\}$, $I_2 = \{2,3\}$, $I_3 = \{4,5\}$, and $I_4 = \{6,7\}$. Specifically, as discussed in Golyandina et al. (2001), periodic components in FTS typically exhibit a rank of 2, consisting of pairs of harmonic elementary components (sine and cosine functions with the same frequency). To identify these pairs, we can examine pairwise scatterplots of the right singular vectors, as illustrated in Figure 7(E). In such scatterplots, components with identical frequencies, amplitudes, and phases form points that lie along a circular path. Additionally, the number of vertices in the resulting regular T-vertex polygon corresponds to the periodicity of the component. In Figure 7(E), subplots 2 vs. 3, 4 vs. 5, and 6 vs. 7 reveal harmonic factors with frequencies of 1/7, 8/14, and 4/7, respectively. For a more detailed exploration of extracting meaningful components from the extracted signal, additional references in the SSA literature, such as Golyandina and Zhigljavsky (2013), can be consulted. In addition, using Figures 7(A-B, D-E), we see clear weekly periodic patterns captured in the decomposition, for example, the seven distinct curves found in various subplots of Figure 7(A) and the seven corners seen in subplot 2 vs. 3 of Figure 7(E). For interested readers, we provide further results for the decomposition stage and the `fssa` object in the GitHub repository of the **Rfssa** package (`https://github.com/haghbinh/Rfssa`).

5 Reconstruction and forecasting

After obtaining an object of class `fssa`, the user may then choose to perform reconstruction using the `freconstruct(·)` function or perform forecasting using `fforecast(·)`. The reconstruction and forecasting functions both return a list of `funts` objects with length m (number of groups). We note that even though it is common to perform forecasting using a combination of groups that best reconstruct the original signal, the user may try forecasting using several different combinations of groups.

5.1 Reconstruction

We start by reconstructing the `Callcenter` data using the grouping suggested from the FSSA decomposition. The following code implements the reconstruction methodology and gives the plots of the reconstruction in Figure 1.

```
# Define groups and their labels
groups <- list(1, 2:3, 4:5, 6:7, 1:7)
group_labels <- c("(B) First Group",
        "(C) Second Group",
        "(D) Third Group",
        "(E) Fourth Group",
        "(F) Extracted Signal")
# Perform FSSA reconstruction
reconstructed_data  <- freconstruct(fssa_results, groups)
# Create and visualize plots
for (i in 1:length(groups)) {
  print(plotly_funts(reconstructed_data[[i]], main = group_labels[i],
                xticklocs = xtlab, xticklabels = xtloc))
}
```

One may observe the mean behavior (from group I_1) in Figure 1(C), and the weekly behaviors (from groups I_2, I_3 and I_4) in Figures 1(D-F). Note that the weekly trajectories are more well-separated in I_4 as opposed to the groups I_2 and I_3. We consider the last group, $I_5 = \{1, \cdots, 7\}$, as the set of indices corresponding to the leading eigentriples that capture more than 98% of the variation in the signal, to reconstruct the original FTS in Figure 1(B).

5.2 The `fforecast` Class

As previously mentioned, in addition to performing reconstruction of the FTS, the package offers the capability to perform forecasting using an object of class `fssa`. The constructor for this class, i.e., the `fforecast(·)` function, accepts the following arguments:

- `U`: An object of class `fssa` holding the decomposition.
- `groups`: A list of numeric vectors where each vector is used for reconstruction and forecasting.
- `len`: An integer representing the desired length of the forecasted FTS.

- `method`: A character string specifying the type of forecasting to perform, with options including:
  - "recurrent" for FSSA R-forecasting.
  - "vector" for FSSA V-forecasting.
- `only.new`: A logical argument, when set to `TRUE`, returns only the forecasted FTS, otherwise the entire FTS is returned.

The `fforecast(·)` function returns an S3 class named `fforecast`, which has been introduced to encapsulate the output of the function. This class is designed to provide a more organized and intuitive structure for handling FTS data. The `fforecast` class includes the following attributes:

- `original_funts`: This attribute stores the original FTS object, allowing users to maintain a clear reference to the original data.
- `predicted_time`: Stores the forecast time index.
- `groups`: Contains a list of numeric vectors, where each vector includes indices of elementary components of a group used for reconstruction and forecasting.
- `method`: A character string specifying the type of forecasting performed.

To streamline user interactions further, we have developed a `print(·)` method for the `fforecast` class, making it more convenient to view and assess forecasted FTS data. To illustrate these enhancements, consider the continuation of the Callcenter data example in the following:

```
# Perform FSSA R-forecasting
pr_R <- fforecast(U = fssa_results, groups = c(1:3), len = 14, method = "recurrent")
# Perform FSSA V-forecasting
pr_V <- fforecast(U = fssa_results, groups = list(1,1:7), len = 14, method = "vector",
        only.new = FALSE)
plot(pr_R, main = 'R-Forecast (only.new = TRUE)')
plot(pr_V, main = 'V-Forecast (only.new = FALSE)')
print(pr_V)
# FSSA Forecast (fforecast) class:
# Groups: List of 2
# : num 1
# : int [1:7] 1 2 3 4 5 6 7
# Prediction method:  vector
# Predicted series length:  14
# Predicted time:  Date[1:14], format: "2000-01-01" "2000-01-02" ...
# ---------The original series-----------
# Functional time series (funts) object:
# Number of variables:  1
# Lenght:  365
# Start:  10592
# End:  10956
# Time:  Date[1:365], format: "1999-01-01" "1999-01-02" "1999-01-03"  ...
```

The resulted figures are shown in Figure 8. The next example will forecast the `Callcenter` FTS one

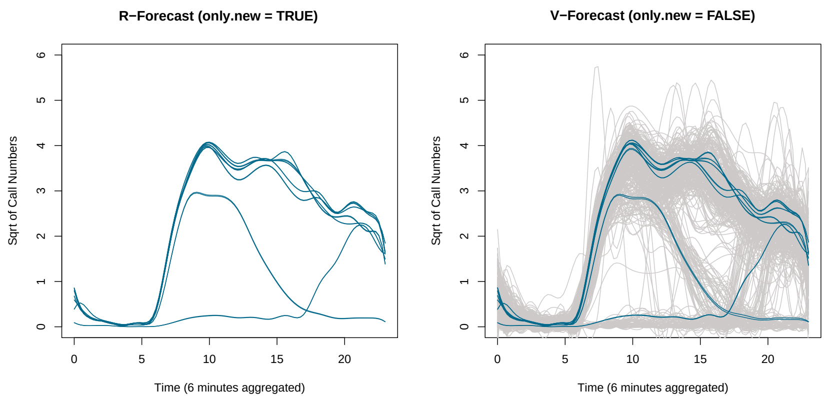


Figure 8: `plot(·)` method of `fforecast` class.

week into the future, based on the first 358 days of the year, by leveraging the FSSA R-forecasting and V-forecasting methods as the results that had been given in Figure 2.

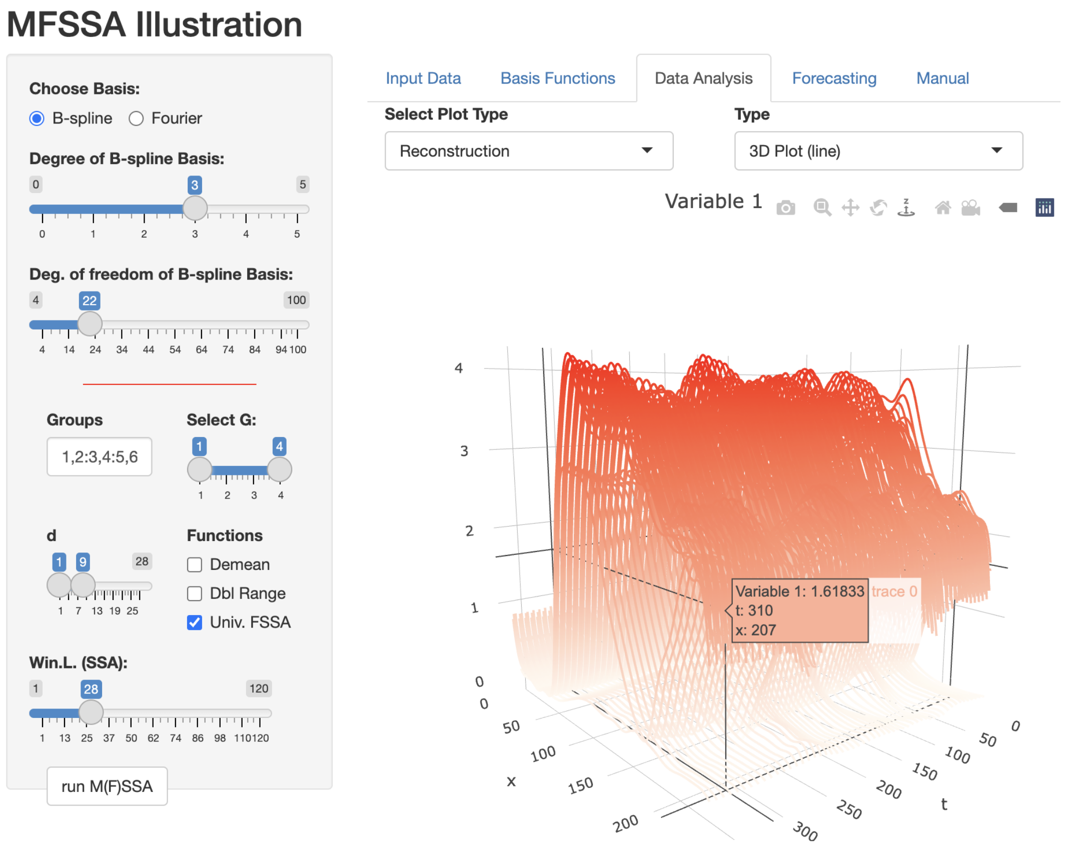
```
# Define the data length
N <- Callcenter$N
U1 <- fssa(Callcenter[1:(N-7)], 28)
# Perform recurrent forecasting using FSSA
fore_R = fforecast(U1, groups = list(1:7), method = "recurrent", len = 7)[[1]]
# Perform vector forecasting using FSSA
fore_V = fforecast(U1, groups = list(1:7), method = "vector", len = 7)[[1]]
# Extract the true call data
true_call <- Callcenter[(N-7+1):N]
# Define weekdays and colors
wd <- c('Sunday', 'Monday', 'Tuesday', 'Wednesday','Thursday', 'Friday', 'Saturday')
clrs <- c("black", "turquoise4", "darkorange")
argvals <- seq(0, 23, length.out = 100);
par(mfrow = c(1,7), mar = c(0.2, 0.2, 2, 0.2));
# Iterate over the days of the week
for(i in 1:7) {
        plot(true_call[i], col = clrs[1], ylim = c(0, 5.3),
                lty = 3, yaxt = "n", xaxt = "n", main = wd[i]);
        plot(fore_R[i], col = clrs[2], lty = 2, add = TRUE);
        plot(fore_V[i], col = clrs[3], lty = 5, add = TRUE);
}
legend("top", c("Original", "R-forecasting", "V-forecasting"), col = clrs,
        lty = c(3, 2, 5));
```

More information on these results may be found in the associated literature of Haghbin et al. (2021) and Trinka et al. (2023).

6 A shiny application

The **Rfssa** package contains shiny apps for both FSSA and MFSSA methods. Those shiny apps can be called using the launchApp(·) function, and are also available at http://sctc.mscs.mu.edu/fssa.htm and http://sctc.mscs.mu.edu/mfssa.htm. Here we present the features of the MFSSA app since the FSSA app would be a special case of that. The MFSSA shiny app was developed to visualize and extract the information related to MFTS in a non-parametric framework. The proposed shiny app provides a friendly GUI for users to implement the **Rfssa** functionalities and even compare the results with the non-functional version (**Rssa**). Figure 9 provides a snapshot of the features available in the MFSSA shiny app. At the top of the sidebar panel, users may specify the basis functions (B-spline or Fourier) and the associated degrees of freedom to represent the funts object. Those basis functions can be visualized in the sub-panel 'Basis Functions' under the main panel. The remaining inputs in the sidebar will be used in other sub-panels. Specifically, 'Groups' is an input box for the third step of the MFSSA algorithm. Each group is specified via a vector (e.g. 'c(1,2,4)' or '1:3') and separated from other groups by a comma (','). The slider 'd' is used to specify the dimensions used in MFSSA (scree, W-correlation, paired, singular vectors & functions, and periodogram plots). The checkbox (a) 'Demean' is used to subtract the mean to obtain mean-zero functions; (b) 'Dbl Range' is used to extend the y-axis to cover all potential mirror functions (e.g. sometimes FPCs may get multiplied by a negative sign); and (c) 'Univ. FSSA' to compare the MFSSA results with marginal FSSA ones respectively. The 'Win.L.' slider specifies the window lengths for the MSSA and the MFSSA. The 'run M(F)SSA' button runs MSSA and MFSSA using the specified parameters for the given dataset. In general, for the sidebar, the top inputs (above the red line) are mostly to describe the basis functions. The bottom inputs (below the red line) are used to specify SSA and FSSA parameters. The main panel includes five sub-panels. Here we briefly describe the features in each of these sub-panels:

- 'Input Data': In this sub-panel, the user can either (a) use the functional datasets available in the **Rfssa** package (e.g., Callcenter data or remote sensing datasets); (b) simulate MFTS (see Haghbin et al., 2021, for details on simulation setup); or (c) upload any arbitrary FTS matrix (where FTS are given in common grid points and are represented in the columns of the data matrix) to provide the dataset and then analyze it.
- 'Basis Functions': As described before, we illustrate the basis functions selected by the user in this sub-panel.
- 'Data Analysis': In this sub-panel, the user can call various tools to
  - visualize the MFTS.
  - obtain the optimal number of basis functions based on the GCV criteria.


MFSSA Illustration

Choose Basis: B-spline [✓] Fourier

Degree of B-spline Basis: 3

Deg. of freedom of B-spline Basis: 22

Groups: 1,2:3,4:5,6

Select G: 1 – 4

d: 1 – 9

Functions: Demean [ ] Dbl Range [ ] Univ. FSSA [✓]

Win.L. (SSA): 28

run M(F)SSA

Input Data | Basis Functions | Data Analysis | Forecasting | Manual

Select Plot Type: Reconstruction

Type: 3D Plot (line)

Variable 1

Variable 1: 1.61833 trace 0
t: 310
x: 207

Figure 9: Snapshot of the MFSSA shiny app

- select a variety of outputs under MSSA and MFSSA, that includes scree plot, W-correlation plot, paired plots, singular vectors plots, periodogram plots, singular functions (heat or regular plots), and reconstruction of FTS using different types of plots (heat, regular, 3Dline and 3Dsurface). An example of the 3Dline reconstruction plot for the `Callcenter` data is given in Figure 9.

- 'Forecasting': This sub-panel would be accessible after the user runs the MFSSA procedure, and it includes the functionalities of R-forecasting and V-forecasting algorithms.
- 'Manual': This sub-panel provides a brief instruction manual to use the MFSSA shiny app.

7 Summary and conclusion

In summary, **Rfssa** is a pioneering package that brings the power of SSA to the realm of functional time series, offering novel techniques for decomposition, reconstruction, multivariate analysis, and functional forecasting. Its flexible data representation and functional context for SSA make it a valuable addition to the CRAN ecosystem, providing unique capabilities not readily available in other packages. Notably, the package offers extensive capabilities for analyzing FTS/MFTS data, allowing joint analysis of smoothed curves and image data across different dimensional domains. The implementations of the methodologies in the package have been optimized for speed by leveraging the functionalities of **RcppEigen** and **RSpectra** R packages, along with custom C++ code. By utilizing the **Rfssa** package, researchers and practitioners can easily apply advanced FSSA-based techniques to their data, yielding informative results that can significantly enhance decision-making across various applied domains. The intuitive nature and computational efficiency of the package make it a valuable asset for the FTS analysis toolkit.

Acknowledgments

The authors would like to express their sincere gratitude to the anonymous reviewers for their valuable feedback and constructive comments, which greatly contributed to the improvement of this work.

Additionally, we would like to acknowledge the significant contributions of Dr. S. Morteza Najibi during the development stages of the first version of the **Rfssa** package.

References

C. Bouveyron. *funFEM: Clustering in the Discriminative Functional Subspace*, 2021. URL https://CRAN.R-project.org/package=funFEM. R package version 1.2. [p82]

M. de Carvalho and G. Martos. *ASSA: Applied Singular Spectrum Analysis*, 2020. URL https://CRAN.R-project.org/package=ASSA. R package version 2.0. [p83]

M. de Carvalho and A. Rua. Real-time nowcasting the us output gap: Singular spectrum analysis at work. *International Journal of Forecasting*, 33(1):185–198, 2017. [p83]

M. Febrero-Bandle and M. O. de la Fuente. Statistical computing in functional data analysis: The R package **fda.usc**. *Journal of Statistical Software*, 51(4):1–28, 2012. doi: 10.18637/jss.v051.i04. [p82]

A. Gajardo, S. Bhattacharjee, C. Carroll, Y. Chen, X. Dai, J. Fan, P. Z. Hadjipantelis, K. Han, H. Ji, C. Zhu, H.-G. Müller, and J.-L. Wang. *fdapace: Functional Data Analysis and Empirical Dynamics*, 2022. URL https://CRAN.R-project.org/package=fdapace. R package version 0.5.9. [p82]

J. Goldsmith, F. Scheipl, L. Huang, J. Wrobel, C. Di, J. Gellar, J. Harezlak, M. W. McLean, B. Swihart, L. Xiao, C. Crainiceanu, P. T. Reiss, Y. Chen, S. Greven, L. Huo, M. G. Kundu, S. Y. Park, D. L. Miller, A.-M. Staicu, E. Cui, and R. Li. *refund: Regression with Functional Data*, 2023. URL https://CRAN.R-project.org/package=refund. R package version 0.1.32. [p82]

N. Golyandina and A. Zhigljavsky. *Singular Spectrum Analysis for Time Series*. Springer Science & Business Media, Berlin, Heidelberg, 2013. [p83, 93]

N. Golyandina, V. Nekrutkin, and A. A. Zhigljavsky. *Analysis of Time Series Structure: SSA and Related Techniques*. Chapman and Hall/CRC, Boca Raton, FL, 2001. [p85, 93]

N. Golyandina, A. Korobeynikov, A. Shlemov, and K. Usevich. Multivariate and 2D extensions of the Rssa package. *Journal of Statistical Software*, 67(2):1–78, 2015. doi: 10.18637/jss.v067.i02. URL http://dx.doi.org/10.18637/jss.v067.i02. [p83]

N. Golyandina, A. Korobeynikov, and A. Zhigljavsky. *Singular Spectrum Analysis with R*. Springer Berlin Heidelberg, 2018. [p83]

H. Haghbin, S. Morteza Najibi, R. Mahmoudvand, J. Trinka, and M. Maadooliat. Functional singular spectrum analysis. *Stat*, page e330, 2021. doi: https://doi.org/10.1002/sta4.330. e330 STAT-20-0240.R1. [p83, 84, 85, 90, 95]

H. Haghbin, J. Trinka, S. M. Najibi, and M. Maadooliat. *Rfssa: Functional Singular Spectrum Analysis*, 2023. URL https://CRAN.R-project.org/package=Rfssa. R package version 3.0.2. [p82]

C. Happ-Kurz. Object-oriented software for functional data. *Journal of Statistical Software*, 93(5):1–38, 2020. doi: 10.18637/jss.v093.i05. [p82]

H. Hassani and R. Mahmoudvand. Multivariate singular spectrum analysis: A general view and new vector forecasting approach. *International Journal of Energy and Statistics*, 01(01):55–83, 2013. doi: 10.1142/S2335680413500051. URL https://doi.org/10.1142/S2335680413500051. [p83]

R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 2012. [p83]

R. Hyndman and H. L. Shang. *ftsa: Functional Time Series Analysis*, 2023. URL https://CRAN.R-project.org/package=ftsa. R package version 6.3.0. [p82]

M. Maadooliat, J. Z. Huang, and J. Hu. Integrating data transformation in principal components analysis. *Journal of Computational and Graphical Statistics*, 24(1):84–103, 2015. [p85]

J. O. Ramsay and B. W. Silverman. *Functional Data Analysis*. Springer Series in Statistics. New York, NY, 2005. ISBN 038740080X. URL http://0-search.ebscohost.com.libus.csd.mu.edu/login.aspx?direct=true&db=cat06952a&AN=mul.b2395232&site=eds-live. [p82, 83]

J. O. Ramsay, H. Wickham, S. Graves, and G. Hooker. *fda: Functional Data Analysis*, 2023. URL https://CRAN.R-project.org/package=fda. R package version 6.1.4. [p82]

H. L. Shang and R. Hyndman. *rainbow: Rainbow Plots, Bagplots and Boxplots for Functional Data*, 2022. URL https://CRAN.R-project.org/package=rainbow. R package version 3.7. [p82]

S. Tavakoli. *ftsspec: Spectral Density Estimation and Comparison for Functional Time Series*, 2015. URL https://CRAN.R-project.org/package=ftsspec. R package version 1.0.0. [p82]

J. Trinka, H. Haghbin, and M. Maadooliat. Multivariate functional singular spectrum analysis: A nonparametric approach for analyzing multivariate functional time series. In *Innovations in Multivariate Statistical Modeling: Navigating Theoretical and Multidisciplinary Domains*, pages 187–221. Springer, 2022. [p83, 84, 85]

J. Trinka, H. Haghbin, H. L. Shang, and M. Maadooliat. Functional time series forecasting: Functional singular spectrum analysis approaches. *Stat*, 12(1):e621, 2023. doi: 10.1002/sta4.621. URL https://doi.org/10.1002/sta4.621. [p83, 84, 86, 95]

Hossein Haghbin
Artificial Intelligence and Data Mining Research Group, ICT Research Institute
Faculty of Intelligent Systems Engineering and Data Science, Persian Gulf University
Boushehr, Iran
(ORCiD 0000-0001-8416-2354)
haghbin@pgu.ac.ir

Jordan Trinka
Department of Mathematical and Statistical Sciences, Marquette University,
Wisconsin, USA
(ORCiD 0000-0001-9118-5781)
jordantrinka4@hotmail.com

Mehdi Maadooliat
Department of Mathematical and Statistical Sciences, Marquette University,
Wisconsin, USA
(ORCiD: 0000-0002-5408-2676)
mehdi.maadooliat@mu.edu