Chapter 3
Parameter Estimation, Forecasting, Gap Filling

Similarly to Chap. 2, this chapter is devoted to applications of SSA for one-dimensional series; that is, to 1D-SSA. The SSA analysis of time series, which is considered in Chap. 2, can be classified as model-free. In this chapter, on the contrary, we consider the methodologies within the 1D-SSA approach, which require a model. These methodologies include the common problems of forecasting, interpolation, low-rank approximation, and parameter estimation. The model used is based on properties of the approximating subspace constructed in the process of 1D-SSA analysis of Chap. 2 and so the methodologies of this chapter belong to the class of subspace-based methods of time series analysis and signal processing.

The main parametric model of 1D-SSA is a linear recurrence relation (LRR) which a time series should approximately satisfy. In Sect. 3.1, we describe how to estimate the LRR coefficients and parameters of a series component satisfying such LRR.

Section 3.2 is devoted to forecasting, the most practically important application of time series analysis. In 1D-SSA, the problem of forecasting coincides with the problem of continuation of the signal $\mathbb{S}$ extracted from the observed series $\mathbb{S} + \mathbb{R}$, where $\mathbb{R}$ is the residual (or noise). To do that, we estimate the trajectory space of $\mathbb{S}$ and make the continuation based on the estimated subspace. A straightforward manner to make a forecast is to directly use the parametric form of the signal estimated using the methods of Sect. 3.1. However, the class of series suitable for forecasting is much wider than the class of series where the parametric model is adequate and some of the forecasting methods of Sect. 3.2 only use certain features of the estimated subspaces and not the estimators of the signal parameters; hence, a medium-term forecast could be quite accurate even if the given series cannot be approximated by a signal which globally satisfies an LRR. Section 3.2 also thoroughly discusses the problem of assessing stability of forecasts, which is the key issue in understanding of how much the forecasts can be trusted.

N. Golyandina et al., *Singular Spectrum Analysis with R*, Use R!,
https://doi.org/10.1007/978-3-662-57380-8_3

Section 3.3 is devoted to the problem of imputation of missing values, or gap filling. The methods of Sect. 3.3 extend the methods of Sect. 3.2 as the problem of forecasting can be considered as a particular case of the problem of missing value imputation when the missing values are located at the end of the series.

Section 3.4 is devoted to the problem of structured low-rank approximation of Hankel matrices which arises when the parametric model of the signal is of the main interest. The most common method of solving this problem is a repeated application of the SSA algorithm. In Sect. 3.4 we stress that choosing appropriate weights in defining the matrix norm can make a significant improvement in the accuracy of the approximation, especially at the points close to both ends of the series.

In Sect. 3.5 we carefully analyze several real-world time series to illustrate the main points of previous sections. We also discuss the problem of choosing parameters of the algorithms.

As in Chap. 2, for the sake of brevity, in this chapter we will refer to 1D-SSA simply as SSA. In contrast to the SSA analysis, the input for the algorithms of this chapter is not necessarily a collection $\mathbb{X}_N = (x_1, \ldots, x_N)$ of N real numbers; it can be an estimated subspace. This subspace is, as a rule, obtained after Grouping step of any of the SSA algorithms and has the form $\mathrm{span}(U_i, i \in I)$.

3.1 Parameter Estimation

In Sect. 2.1.2, in the process of explaining the concepts related to the SSA analysis such as SSA decomposition and separability, we have described a class of series $\mathbb{S} = (s_n)$ governed by linear recurrence relations (LRRs) $s_n = \sum a_i s_{n-i}$. In this section, we describe how to estimate the LRR coefficients and parameters of a series components governed by an LRR. We will assume that the SSA method is able to approximately extract the investigated series component; that is, the component of interest is approximately separated and the window length together with the SSA modification are chosen appropriately.

A series governed by an LRR can be expressed in the parametric form (1.9). A particular case is a series $\mathbb{S} = (s_n)$ with $s_n = \sum_{i=1}^r C_i \mu_i^n$, $\mu_i \in \mathsf{C}$, or, in the real-valued form, $s_n = \sum_{i=1}^p A_i \exp(\alpha_i n) \cos(2\pi n \omega_i + \phi_i)$, where A_i, α_i, ω_i and ϕ_i $(i = 1, \ldots, p)$ are unknown parameters whose values may be (and often are) of interest to the investigator. Hence the problem of parameter estimation arises.

3.1.1 Method

We describe the so-called subspace-based methods of parameter estimation, where only the estimated subspace of the series components is of concern but Reconstruction step of the SSA algorithm is of no importance.

Let a set of indices I correspond to the component of interest in the constructed decomposition of the trajectory matrix $\mathbf{X} = \sum_{i=1}^{d} \sigma_i P_i Q_i^{\mathrm{T}}$. For simplicity of notation we assume $I = \{1, 2, \ldots, r\}$. Then the estimated subspace is $\widetilde{\mathcal{S}} = \mathrm{span}(P_1, \ldots, P_r)$. We always consider the generating set $\{P_i\}$ of $\widetilde{\mathcal{S}}$ to be orthonormal as otherwise we can orthonormalize it. Since the original vectors P_i may be linearly dependent (for example, in the method of SSA with projection), the procedure of orthogonalization may reduce the number of vectors. We will consider r to be equal to the number of vectors after orthogonalization.

We consider two kinds of parametrization; first, in the form of a governing LRR and, second, in the form (1.9). Correspondingly, we describe how to estimate the coefficients of a governing LRR and the parameters of (1.9).

3.1.1.1 Estimation of the Governing LRR

The trajectory space $\mathcal{S}$ of a signal $\mathbf{S}$ governed by a particular LRR corresponds to many LRRs. More precisely, any vector from $\mathcal{S}^{\perp}$ with the last coordinate -1 produces such an LRR; in other words, any such vector from $\mathcal{S}^{\perp}$ provides a set of coefficients for a linear combination of the first $L-1$ coordinates of a vector from $\mathcal{S}$ to obtain the last coordinate; see Golyandina et al. (2001; Section 5) and Golyandina and Zhigljavsky (2013; Chapter 3) for detailed explanations.

Among all these LRRs (generating the same trajectory space $\mathcal{S}$) there is the best LRR with minimal sum of squared coefficients (the so-called min-norm LRR). The min-norm LRR suppresses possible perturbations in the initial data as much as possible, which is important if we use this LRR for series generation or continuation, on the base of SSA approximation of the initial data.

For a chosen window length L, the signal subspace $\mathcal{S} \in \mathsf{R}^L$ and therefore the min-norm LRR has order $L-1$. For each column vector P_i of $\mathbf{P}_r$, denote π_i the last coordinate of P_i, $\underline{P_i} \in \mathsf{R}^{L-1}$ the vector P_i with the last coordinate removed, and $\nu^2 = \sum_{i=1}^{r} \pi_i^2$. Then the elements of the vector

$$\mathcal{R} = (a_{L-1}, \ldots, a_1) = \frac{1}{1-\nu^2} \sum_{i=1}^{r} \pi_i \underline{P_i} \tag{3.1}$$

provide the coefficients of the *min-norm* governing LRR: $s_n = \sum_{i=1}^{L-1} a_i s_{n-i}$.

For an estimated subspace $\widetilde{\mathcal{S}}$, the estimated LRR is calculated in the same way, on the base of an orthonormal basis of $\widetilde{\mathcal{S}}$.

3.1.1.2 Estimation of Frequencies

Let $\mathbb{X}_N = \mathbb{S}_N + \mathbb{R}_N$, where $s_n = \sum_{j=1}^{r} c_j \mu_j^n$ and the series $\mathbb{S}_N$ and $\mathbb{R}_N$ are approximately separable for a given window length L. Generally, the signal roots of the characteristic polynomial of a governing LRR allow estimation of the signal parameters μ_j, $j = 1, \ldots, r$ (see Sect. 2.1.2.2). However, the min-norm LRR is not minimal and therefore we should somehow distinguish between the signal roots and the extraneous roots. Usually, the signal roots of the min-norm LRR have maximal moduli (e.g., see Usevich (2010)). Therefore, one can find roots of the min-norm LRR, arrange them in the order of decrease, and take the first r roots.

However, the ordering is never guaranteed. Therefore, the methods that are able to separate the signal and extraneous roots could be very useful.

Let us describe one of these methods called ESPRIT (Roy and Kailath 1989). This method is implemented in two versions, LS-ESPRIT and TLS-ESPRIT, where LS means least squares, TLS means total least squares (see, e.g., the method description in Golyandina and Zhigljavsky (2013; Section 3.8.2)). Other names are HSVD (Barkhuijsen et al. 1987) and HTLS (Van Huffel et al. 1994). Here we describe the LS version (HSVD).

Denote $\{P_1, \ldots, P_r\}$ an orthonormal basis of the estimated subspace of the component under interest. Set $\mathbf{P}_r = [P_1 : \ldots : P_r]$ and let $\underline{\mathbf{P}_r}$ be the matrix with the last row removed and $\overline{\mathbf{P}_r}$ be the matrix with the first row removed. Then μ_i can be estimated by the eigenvalues of the matrix $\underline{\mathbf{P}_r}^{\dagger}\overline{\mathbf{P}_r}$, where $\dagger$ denotes pseudo-inversion. Correspondingly, the estimated frequencies are the arguments of μ_i.

Note that the matrix $\mathbf{P}_r$ conventionally consists of the chosen eigenvectors U_i in the Basic SSA algorithm. However, any basis of the subspace, which estimates the signal subspace, is suitable.

Let us mention a simple and fast method of frequency estimation which is used for identification of the eigentriples at Grouping step. Two vectors $U^{(1)}$ and $U^{(2)}$ forming an orthogonal basis of the trajectory space of an exponentially-modulated sine wave have similar forms and their phases differ by approximately $\pi/2$. Let A and B be defined by $a_n = \rho^n \sin(2\pi\omega n + \phi)$ and $b_n = \rho^n \cos(2\pi\omega n + \phi)$. Denote the angle between vectors by $\angle$. Then $\omega = \angle\left(\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}, \begin{pmatrix} a_2 \\ b_2 \end{pmatrix}\right) \Big/ (2\pi)$. Therefore, we can estimate the frequency using the basis vectors $U^{(1)}$ and $U^{(2)}$. Since these vectors do not have exactly the same form as A and B, the sequence of angles $\angle\left(\begin{pmatrix} u_i^{(1)} \\ u_i^{(2)} \end{pmatrix}, \begin{pmatrix} u_{i+1}^{(1)} \\ u_{i+1}^{(2)} \end{pmatrix}\right) \Big/ (2\pi)$, $i = 1, \ldots, L-1$, can be considered and then the mean or median can be taken as an estimate of the frequency; see Golyandina et al. (2001; Section 1.6) for details. In Rssa, the median is considered and the median of absolute deviations from the median is used as a measure of accuracy.

3.1.2 Algorithms

Although the LRR approximating the time series is usually used for forecasting, it can also be helpful for construction of the signal model. Hence we introduce an algorithm for calculation of the min-norm LRR coefficients.

Algorithm 3.1 Estimation of the signal LRR

```
Input: Matrix $\mathbf{P}_r \in \mathsf{R}^{L\times r}$ consisting of orthonormal column vectors, which form a basis of the
    estimated signal subspace.
Output: Coefficients $\mathcal{R} = (a_{L-1}, \ldots, a_1)$ of the corresponding LRR.
 1: For each column vector $P_i$ of $\mathbf{P}_r$, calculate $\pi_i$ and $\underline{P}_i$, $\nu^2 = \sum_{i=1}^{r} \pi_i^2$. If $\nu^2$ is equal to 1, then
    STOP with the error message "Verticality coefficient equals 1."
 2: Compute $\mathcal{R} = \frac{1}{1-\nu^2} \sum_{i=1}^{r} \pi_i \underline{P}_i$.
```

The next algorithm shows how the parameters μ_i in $s_n = \sum_{i=1}^{r} C_i \mu_i^n$ can be estimated from the roots of the characteristic polynomial of an LRR governing this time series (see Sect. 2.1.2.2 for a description of the relation between LRRs and their characteristic polynomials). The given LRR is an estimate of an LRR governing a series of rank r; therefore, only r roots correspond to the signal, while the other roots are extraneous. Since frequently (but not always!) the signal roots for the min-norm LRR have larger moduli than the extraneous roots, we can select signal roots with large absolute values among the whole set of roots.

Algorithm 3.2 Estimation of the signal roots through characteristic polynomial of LRR

```
Input: Coefficients $A = (a_1, \ldots, a_m)$ of the LRR $s_n = \sum_{i=1}^{m} a_i s_{n-i}$, rank $r$.
Output: Signal roots $\mu_i$, $i = 1, \ldots, r$.
 1: Construct the characteristic polynomial $P(\mu) = \mu^d - \sum_{i=1}^{m} a_i \mu^{n-i}$.
 2: Find the roots $\mu_1, \ldots, \mu_m$ of $P(\mu)$.
 3: Order the roots so that $|\mu_1| \geq \ldots \geq |\mu_m|$.
 4: The leading roots $\mu_i$, $i = 1, \ldots, r$, are the candidates for the signal roots.
```

Algorithm 3.3 is one of the most known high-resolution subspace-based algorithms of estimation of frequencies and damping factors.

Algorithm 3.3 ESPRIT

```
Input: Matrix $\mathbf{P}_r \in \mathsf{R}^{L\times r}$ consisting of orthonormal column vectors, which form a basis of the
    estimated signal space.
Output: $r$ roots in the form $(\rho_i, \omega_i)$.
 1: Using either LS or TLS method, find a matrix $\mathbf{M} \in \mathsf{R}^{r\times r}$ satisfying $\overline{\mathbf{P}_r} \approx \underline{\mathbf{P}_r}\mathbf{M}$. For the
    LS-method, $\mathbf{M} = \underline{\mathbf{P}_r}^{\dagger}\overline{\mathbf{P}_r}$.
 2: Find eigenvalues $\mu_i$, $i = 1, \ldots, r$, of $\mathbf{M}$.
 3: Set $\rho_i = \mathrm{Mod}(\mu_i)$, $\omega_i = \mathrm{Arg}(\mu_i)$.
```

The next algorithm is a complementary to Decomposition step used for helping to gather sine-waves with similar frequencies.

Algorithm 3.4 Fast ("pairs") estimation of frequencies

```
Input: Two orthonormal vectors $U^{(1)}$ and $U^{(2)}$ forming an estimated trajectory space of a sine wave.
Output: Frequency $\omega$, period $T$.
1: Compute $\phi_i = \angle\left(\begin{pmatrix} u_i^{(1)} \\ u_i^{(2)} \end{pmatrix}, \begin{pmatrix} u_{i+1}^{(1)} \\ u_{i+1}^{(2)} \end{pmatrix}\right)$, $i = 1, \ldots, L-1$.
2: Calculate $\bar{\phi}$ as the mean or median of $\{\phi_i\}$.
3: $\omega = \bar{\phi}/(2\pi)$, $T = 1/\omega$.
```

3.1.3 *Estimation in* Rssa

3.1.3.1 Description of Functions

After the `ssa` object `s` has been constructed by the call of the `ssa` (alternatively, `iossa` or `fossa`) function, the min-norm LRR can be constructed by the call of the form

```
lrr.coef <- lrr(s, groups = list(2:3))
```

Arguments:

`s` is an `ssa` object holding the full one-dimensional SSA decomposition.
`groups` is a list defining the group of selected eigentriples.

The function `lrr` returns a list of objects of the `lrr` class, which contain coefficients of the LRRs for each given group.

Complex roots of the characteristic polynomial of an LRR, which are ordered by their moduli, are calculated by the call

```
lrr.roots <- roots(lrr.coef, method = "companion")
```

Arguments:

`lrr.coef` is an `lrr` object.
`method` is a method used for calculation of the polynomial roots: via eigenvalues of the companion matrix or via R's standard `polyroot` routine.

Estimation of parameters can be performed by means of this typical call:

```
est <- parestimate(s, groups = list(c(2, 3, 5, 6)),
                   method = "esprit")
```

Arguments:

`s` is an `ssa` object holding the full one-dimensional SSA decomposition.
`groups` is a list of eigentriples groups; for `method = "pairs"` each group should consist of exactly two components.
`method` is a method of estimation of frequencies and damped factors; it can have the following values: `"esprit"`, `"pairs"`.

`subspace` indicates which space, column or row, will be used for parameter estimation by the ESPRIT method. The default value `"column"` is standard for ESPRIT.

`solve.method` is the method of shift matrix estimation; it can be set as `"ls"` for the least squares solution and `"tls"` for the total least squares approach.

For an `lrr` object, the function `print` prints the LRR coefficients and `plot` draws the produced complex roots, both signal and extraneous.

For the result of `parestimate`, the function `print` prints the estimated parameters, while `plot` draws the estimated signal roots on the complex plane.

3.1.3.2 Typical Code

We start with a simple example to show a relation between LRRs and roots (Fragment 3.1.1). For the exponential series $s_n = 1.01^n = e^{n \ln 1.01}$ of rank $r = 1$, the minimal LRR is $x_n = 1.01 x_{n-1}$; that is, the vector of its coefficients is $A = (1.01)$. The characteristic polynomial has the form $P(\mu) = \mu - 1.01$, its root is 1.01. The minimal LRR can be obtained for $L = r + 1$. Note that this choice is an inappropriate choice for noisy series, since it would most likely provide a poor separability between the signal and noise.

For $L = 6$ we have 4 extraneous roots. All five roots are depicted in Fig. 3.1. Moduli of all four extraneous roots are smaller than 1.

The second simple example produces LRR coefficients and signal roots for a linear function. In this example, rank $r = 2$, the minimal LRR does not depend on the coefficients of the linear function and is $x_n = 2x_{n-1} - x_{n-2}$; that is, $A = (2, -1)^{\mathrm{T}}$. The characteristic polynomial is $P(\mu) = \mu^2 - 2\mu + 1$; it has root 1 of multiplicity 2. Since all methods are numerical, it is impossible to obtain exactly equal roots. Therefore, a linear function numerically generates two different roots, each close to 1. In this example, the linear function is approximated by a sum of two exponentials (the case of two different real roots). In the case of two conjugate complex roots instead of one root 1 of multiplicity 2, it can be approximated by one sine wave with low frequency.

Fragment 3.1.1 (LRRs and Roots of Characteristic Polynomials)

```
> # Minimal LRR
> x <- 1.01^(1:10)
> s <- ssa(x, L = 2)
> l <- lrr(s, groups = list(1))
> print(l)
[1] 1.01
attr(,"class")
[1] "lrr"
> print(roots(l))
[1] 1.01
> # Extraneous roots
> x <- 1.01^(1:10)
> s <- ssa(x, L = 6)
> l <- lrr(s, groups = list(1))
> r <- roots(l)
> plot(l)
> # Multiple roots
> x <- 2.188 * (1:10) + 7.77
> s <- ssa(x, L = 3)
> l <- lrr(s, groups = list(1:2))
> print(l)
[1] -1  2
attr(,"class")
[1] "lrr"
> print(roots(l))
[1] 1.0000003 0.9999997
```

Fragment 3.1.2 demonstrates the methods of parameter estimation for the real-life series "CO2." For this series, ET1,4 correspond to a trend, while ET2,3 are related to a sine-wave with period 12 (annual periodicity) and ET5,6 correspond to half-year periodicity. One can see that both estimation methods provide almost equal

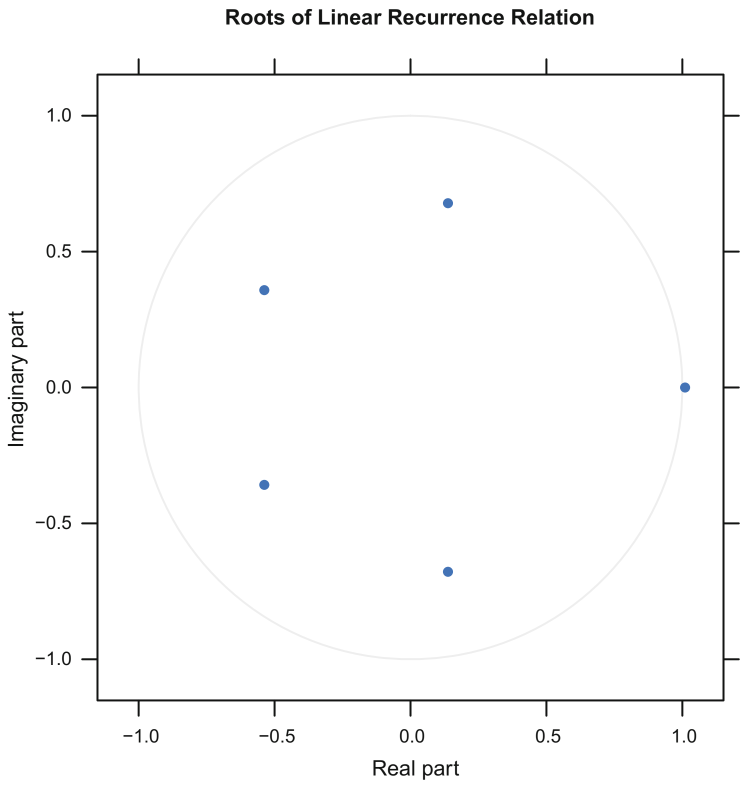

Fig. 3.1 Exponential signal: One signal and four extraneous roots

estimated periods of the annual component close to 12. Note that for the method `"pairs"`, the values `rate` and `Mod` mean nothing.

The ESPRIT method results in complex signal roots μ_j, which are presented in convenient exponential form $\mu_j = \rho_j e^{i2\pi\omega_j}$: `period` is $1/\omega_j$, `Mod` is ρ_j, `rate` is $\ln \rho_j$, `Arg` means $2\pi\omega_j$. Since for real-valued series complex roots form conjugate pairs, two rows with parameter estimates have equal absolute values but have opposite signs. One can see that the trend part of the chosen components is approximated by a sum of two real exponentials (their periods equal `Inf`), the first one is increasing (the rate is positive), while the second one is decreasing.

Fragment 3.1.2 (Parameter Estimation for "CO2")

```
> # Decompose "co2" series with default window length L
> s <- ssa(co2)
> # Estimate the periods from 2nd and 3rd eigenvectors
> # using default "pairs" method
> print(parestimate(s, groups = list(c(2, 3)), method = "pairs"))
   period      rate    |     Mod      Arg   |      Re         Im
   11.995   0.000000 |  1.00000    0.52 |  0.86592    0.50019
> # Estimate the periods and rates using ESPRIT
> pe <- parestimate(s, groups = list(1:6),
+                   method = "esprit")
> print(pe)
   period      rate    |     Mod      Arg   |      Re         Im
   11.995   0.000542 |  1.00054    0.52 |  0.86638    0.50047
  -11.995   0.000542 |  1.00054   -0.52 |  0.86638   -0.50047
    5.999   0.000512 |  1.00051    1.05 |  0.50015    0.86653
   -5.999   0.000512 |  1.00051   -1.05 |  0.50015   -0.86653
      Inf   0.000375 |  1.00037    0.00 |  1.00037    0.00000
      Inf  -0.008308 |  0.99173    0.00 |  0.99173    0.00000
> plot(pe)
```

Figure 3.2 depicts six estimated signal roots on the complex plane, where two trend real-valued roots can hardly be distinguished.

3.2 Forecasting

The problem of forecasting is the problem of continuation of the signal $\mathbb{S}$ extracted from the observed series $\mathbb{X} = \mathbb{S} + \mathbb{R}$. To do that it is sufficient to estimate the trajectory space of $\mathbb{S}$ and then to construct the forecasted series based on the estimated subspace.

An obvious way to perform forecasting would be to estimate series parameters and use them for forecasting. However, the class of series suitable for forecasting is considerably wider than the class of series for parameter estimation and hence the forecasting methods considered below do not estimate series parameters but only use some features of the estimated subspace.

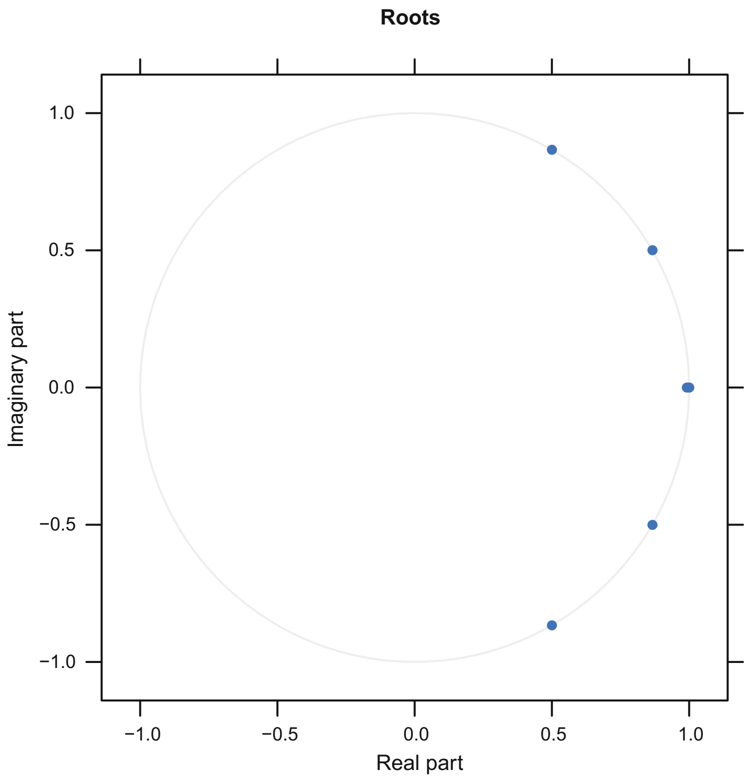

Fig. 3.2 "CO2": Six signal roots

Generally, the forecasted series should have a structure to forecast. In the framework of SSA, we say that $\mathbb{S}$ has a structure if $\mathbb{S}$ is governed by an LRR. However, it is very important to note that SSA forecasts are meaningful for a much wider class of series when an LRR gives an adequate description of the structure of the series only locally rather than globally, which is a requirement for parameter estimation. For example, as a rule, a trend does not satisfy an LRR on the whole time range but it can be locally approximated by a smooth series governed by an LRR. In particular, relatively reliable forecasts can be made for the series which can be approximated by a series of the form $s_n = \sum_{i=1}^{r} C_i(n)\mu_i^n$, $\mu_i \in \mathsf{C}$, where $C_i(n)$ are slowly varying functions of n.

In any version of the forecasting algorithm, we should assume that the series $\mathbb{S}$ is approximately separated from $\mathbb{R}$ by a chosen modification of the SSA method.

3.2.1 Method

The methods of SSA forecasting are closely related to construction of LRRs described in Sect. 3.1. Below we describe two forecasting algorithms. One of them directly uses the constructed LRR for forecasting, whereas the other algorithm does it implicitly. Both algorithms provide the same forecasts of series governed by LRRs, if parameters of SSA are chosen properly.

3.2.1.1 Approach

We will use the same notation as in Sect. 3.1.

Let $\{P_i\}$ be a basis of a subspace $\widetilde{\mathbb{S}}$ of R^L. Then we can state the problem of forecasting in this subspace.

Different modifications of SSA described in Chap. 2 (except for SSA with projection) provide an estimate of a basis of the signal subspace.

If an SSA modification is applied and a set of eigentriples $\{(\sigma_i, P_i, Q_i),\ i \in I\}$ is chosen for reconstruction, then $\widetilde{\mathbb{S}} = \mathrm{span}\{P_i, i \in I\}$. The set of vectors $\{P_i, i \in I\}$ is not necessarily an orthonormal basis. The first mandatory step is therefore the ortho-normalization of the set of vectors $\{P_i, i \in I\}$. After making this step we can construct forecasting algorithms considering an orthonormal basis as an input.

The subspace $\widetilde{\mathbb{S}}$ produces coefficients of a linear combination for reconstruction of the last coordinates of vectors from $\widetilde{\mathbb{S}}$ through their first $L-1$ coordinates. The linear combination used for forecasting has minimal Euclidean norm of coefficients among all linear combinations that correspond to the subspace $\widetilde{\mathbb{S}}$. If $\widetilde{\mathbb{S}}$ is exactly the trajectory subspace of a series governed by an LRR, then the linear combination with minimal norm corresponds to the min-norm LRR. Moreover, the continuation of the series in this subspace is unique.

However, if the series subspace is estimated approximately, several versions of forecasting can be suggested. If the estimation of the subspace was accurate enough, then different forecasting versions will be close. Otherwise, they can differ considerably.

Now we formally describe the forecasting algorithms. For detailed explanation, see Golyandina et al. (2001; Chapter 2).

3.2.1.2 Recurrent Forecasting

The recurrent SSA forecasting is performed by means of the min-norm LRR defined in (3.1).

The *recurrent forecasting method* can be formulated as follows:

1. The time series $\mathbb{Y}_{N+M} = (y_1, \ldots, y_{N+M})$ is defined by

$$y_i = \begin{cases} \widetilde{x}_i & \text{for } \; i = 1, \ldots, N, \\ \sum\limits_{j=1}^{L-1} a_j y_{i-j} & \text{for } \; i = N+1, \ldots, N+M. \end{cases} \tag{3.2}$$

2. The numbers $y_{N+1}, \ldots, y_{N+M}$ form the M terms of the recurrent forecast.

Thus, the recurrent forecasting is performed by the direct use of the forecasting LRR with coefficients taken from $\mathcal{R} = (a_{L-1}, \ldots, a_1)$.

Remark 3.1 Let us define the linear operator $\mathcal{P}_{\mathrm{Rec}} : \mathsf{R}^L \mapsto \mathsf{R}^L$ by the formula

$$\mathcal{P}_{\mathrm{Rec}} Z = \begin{pmatrix} \overline{Z} \\ \mathcal{R}^{\mathrm{T}} \overline{Z} \end{pmatrix}, \tag{3.3}$$

where $\overline{Z}$ consists of the last $L-1$ coordinates of Z. Set

$$Y_i = \begin{cases} \widetilde{X}_i & \text{for } \; i = 1, \ldots, K, \\ \mathcal{P}_{\mathrm{Rec}} Y_{i-1} & \text{for } \; i = K+1, \ldots, K+M. \end{cases} \tag{3.4}$$

It is easily seen that the matrix $\mathbf{Y} = [Y_1 : \ldots : Y_{K+M}]$ is the trajectory matrix of the series $\mathbb{Y}_{N+M}$. Therefore, (3.4) can be regarded as a vector version of (3.2).

Remark 3.2 In recurrent forecasting, the original series can be taken instead of the reconstructed series as the initial data for the forecasting LRR. This may be sensible only if the leading components are chosen for forecasting. This option can reduce the bias caused by the reconstruction inaccuracy but the volatility of forecasts may increase.

If the LRR is not minimal, then only r of the roots correspond to the signal. Other roots are extraneous and can influence the forecast. Extraneous roots that have moduli larger than 1 can lead to instability.

3.2.1.3 Vector Forecasting

Let $\mathcal{L}_r = \mathrm{span}(P_i, i \in I)$ and $\widehat{X}_i$ be the projection of the lagged vector X_i on $\mathcal{L}_r$. Consider the matrix

$$\Pi = \underline{\mathbf{P}}\,\underline{\mathbf{P}}^{\mathrm{T}} + (1 - \nu^2)\mathcal{R}\mathcal{R}^{\mathrm{T}}, \tag{3.5}$$

where $\underline{\mathbf{P}} = [\underline{P_1} : \ldots : \underline{P_r}]$ and $\mathcal{R}$ is defined in (3.1). The matrix Π defines the linear operator that performs the orthogonal projection $\mathsf{R}^{L-1} \mapsto \underline{\mathcal{L}_r}$, where $\underline{\mathcal{L}_r} =$

span($\underline{P_i}, i \in I$). Finally, we define the linear operator $\mathcal{P}_{\mathrm{Vec}} : \mathsf{R}^L \mapsto \mathcal{L}_r$ by the formula

$$\mathcal{P}_{\mathrm{Vec}} Z = \begin{pmatrix} \Pi \overline{Z} \\ \mathcal{R}^{\mathrm{T}} \overline{Z} \end{pmatrix}. \tag{3.6}$$

The *vector forecasting method* can be formulated as follows:

1. In the notation above, define the vectors

$$Y_i = \begin{cases} \widehat{X}_i & \text{for } i = 1, \ldots, K, \\ \mathcal{P}_{\mathrm{Vec}} Y_{i-1} & \text{for } i = K+1, \ldots, K+M+L-1. \end{cases} \tag{3.7}$$

2. By constructing the matrix $\mathbf{Y} = [Y_1 : \ldots : Y_{K+M+L-1}]$ and making its diagonal averaging we obtain the series $y_1, \ldots, y_{N+M+L-1}$.
3. The numbers $y_{N+1}, \ldots, y_{N+M}$ form the M terms of the vector forecast.

In recurrent forecasting, we perform diagonal averaging to obtain the reconstructed series and then apply the LRR. In the vector forecasting algorithm, these steps are applied in the reverse order. The vector forecast is typically slightly more stable. The current fast implementation of the vector forecasting makes the vector forecasting comparable with recurrent forecasting it terms of the computational cost, see Golyandina et al. (2015).

If the time series component is fully separated from the residual and is governed by an LRR, both recurrent and vector forecasting coincide and provide the exact continuation. In the case of an approximate separability, the recurrent and vector forecasting algorithms give different forecasts.

3.2.1.4 Specificity of SSA Modifications

Basic SSA, Toeplitz SSA, and Filter-adjusted SSA provide orthogonal bases. The basis obtained by Iterated O-SSA needs ortho-normalization. After the subspace is chosen, the forecasting algorithms do not depend on the modification of SSA used.

The algorithms of forecasting for SSA with projection are constructed and implemented in the RSSA package by Alex Shlemov but are not yet properly studied and even properly described. In view of this, we do not discuss these algorithms in this book. For row centering, the algorithm is a generalization of the forecasting with centering described in Golyandina et al. (2001; Section 1.7.1).

3.2.1.5 Bootstrap Confidence and Prediction Intervals

Assume again $\mathbb{X}_N = \mathbb{S}_N + \mathbb{R}_N$. Let us describe the construction of bootstrap confidence intervals for the signal $\mathbb{S}_N$ and its forecast assuming that the signal has rank r and the residuals are white noise. The algorithm consists of the following steps.

- Fix L, $I = \{1, \ldots, r\}$, apply SSA, reconstruct the signal and obtain the decomposition $\mathbb{X}_N = \widetilde{\mathbb{S}}_N + \widetilde{\mathbb{R}}_N$.
- Fix $\widetilde{\mathbb{S}}_N$, calculate the empirical distribution of the residual $\widetilde{\mathbb{R}}_N$.
- Simulate Q independent copies $\widetilde{\mathbb{R}}_{N,i}$, $i = 1, \ldots, Q$, using the empirical distribution, construct $\widetilde{\mathbb{X}}_{N,i} = \widetilde{\mathbb{S}}_N + \widetilde{\mathbb{R}}_{N,i}$.
- Apply SSA with the same L and I to $\widetilde{\mathbb{X}}_{N,i}$, reconstruct the signal, then perform M-step ahead forecasting and obtain $\widetilde{\mathbb{S}}_{N+M,i}$, $i = 1, \ldots, Q$.
- For each time point j consider the sample $\widetilde{s}_{j,i}$, $i = 1, \ldots, Q$, and construct the *bootstrap γ-confidence interval* as the interval defined by $(1-\gamma)/2$- lower and upper sample quantiles. The sample mean is called *average bootstrap forecast*.

Remark 3.3 In the same manner as for linear regression, the prediction intervals can be considered in addition to the confidence intervals. The prediction intervals are constructed as the confidence intervals enlarged by the values of quantiles of the noise distribution. While the confidence intervals show the bounds for the signal and its forecast, the prediction intervals determine the bounds for the whole series and its prediction. Note that in the case of linear regression, this is the theoretical approach; for SSA, this is an empirical approach. To estimate quantiles of the noise distribution, we use $(1-\gamma)/2$- lower and upper sample quantiles of the residuals $\widetilde{R}_N = (r_1, \ldots, r_N)$. To construct the *bootstrap γ-prediction intervals*, these quantiles are added to the lower and upper bounds of the γ-confidence interval, correspondingly.

Note that for cross-validation of SSA forecasts future values should not be involved for construction of forecasts and choice of parameters; in this respect, see a discussion in a recent paper (Du et al. 2017).

3.2.2 Algorithms

Let a version of SSA be applied to the time series $\mathbb{X}$ and let an eigentriple group $\{(\sigma_i, P_i, Q_i),\ \ i \in I\}$ be chosen for reconstruction. The suggested forecasting algorithms are formulated for forecasting in the subspace $\mathcal{L}_r = \mathrm{span}\{P_i, i \in I\} \subset \mathsf{R}^L$. For simplicity, we assume that $I = \{1, \ldots, r\}$ and the vectors P_i, $i \in I$, are orthonormal. Note that the forecasting values do not depend on the choice of basis in $\mathcal{L}_r$.

Algorithm 3.5 is written in the form, when the reconstructed series is taken as a base for forecasting. If the original series is used as the base of forecasting, x_n are taken instead of $\widetilde{x}_n$ for $n = N - L + 2, \ldots, N$ at Step 3.

Algorithm 3.5 constructs a forward recurrent forecasting. Backward recurrent forecasting is obtained by applying the forward forecasting to the reversed series.

Algorithm 3.5 Recurrent SSA forecasting

```
Input: Time series $\mathbb{X}$ of length $N$, window length $L$, orthonormal system of vectors $\{P_i\}_{i=1}^r$,
    forecast horizon $M$.
Output: Forecast values $(\widetilde{x}_{N+1}, \ldots, \widetilde{x}_{N+M})$.
 1: Construct the vector $\mathcal{R} = (a_{L-1}, \ldots, a_1)^\mathrm{T}$ of coefficients of the min-norm LRR by
    Algorithm 3.1 applied to $\{P_i, i \in I\}$.
 2: Construct the reconstructed matrix $\widehat{\mathbf{X}} = \mathbf{P}\mathbf{P}^\mathrm{T}\mathbf{X}$, where $\mathbf{P} = [P_1 : \ldots : P_r]$, and the
    reconstructed series $\widetilde{\mathbb{X}} = (\widetilde{x}_1, \ldots, \widetilde{x}_N)$ by $\widetilde{\mathbb{X}} = \mathcal{T}_{\mathrm{SSA}}^{-1} \circ \mathit{\Pi}_{\mathcal{H}}(\widehat{\mathbf{X}})$.
 3: Calculate the forecast values by applying the min-norm LRR:
        $$\widetilde{x}_n = \sum_{i=1}^{L-1} a_i \widetilde{x}_{n-i}, \; n = N+1, \ldots, N+M$$
```

The next algorithm implements the algorithm of vector forecasting, where the application of the min-norm LRR and the hankelization operation are taken in the reverse order.

Algorithm 3.6 Vector SSA forecasting

```
Input: Time series $\mathbb{X}$ of length $N$, window length $L$, orthonormal system of vectors $\{P_i\}_{i=1}^r$,
    forecast horizon $M$.
Output: Forecast values $(\widetilde{x}_{N+1}, \ldots, \widetilde{x}_{N+M})$.
 1: Obtain the vector $\mathcal{R} = (a_{L-1}, \ldots, a_1)^\mathrm{T}$ of coefficients of the min-norm LRR by Algorithm 3.1
    applied to $\{P_i, i \in I\}$.
 2: Calculate the matrix $\mathit{\Pi}$ of projection given in (3.5).
 3: Construct the reconstructed matrix $\widehat{\mathbf{X}} = \mathbf{P}\mathbf{P}^\mathrm{T}\mathbf{X}$, where $\mathbf{P} = [P_1 \ldots : P_r]$.
 4: Extend the reconstructed matrix $\widehat{\mathbf{X}} = [\widehat{X}_1 : \ldots : \widehat{X}_K]$ by column vectors:
        $$\widehat{X}_n = \mathcal{P}_{\mathrm{Vec}} \widehat{X}_{n-1} \text{ for } n = K+1, \ldots, K+M+L-1,$$
    where $\mathcal{P}_{\mathrm{Vec}}$ is given in (3.6) and uses $\mathit{\Pi}$ and $\mathcal{R}$. Denote the extended matrix $\widehat{\mathbf{X}}_{\mathrm{ext}} \in$
    $\mathsf{R}^{L \times (K+M+L-1)}$.
 5: Obtain the extended reconstructed series $\widetilde{\mathbb{X}}_{\mathrm{ext}} = (\widetilde{x}_1, \ldots, \widetilde{x}_{N+M+L-1})$ as $\widetilde{\mathbb{X}}_{\mathrm{ext}} = \mathcal{T}_{\mathrm{SSA}}^{-1} \circ$
    $\mathit{\Pi}_{\mathcal{H}}(\widehat{\mathbf{X}}_{\mathrm{ext}})$.
 6: Return the forecast values $(\widetilde{x}_{N+1}, \ldots, \widetilde{x}_{N+M})$.
```

Additional $L - 1$ vectors $\widehat{X}_n$ at Step 4 are calculated to make the forecast values independent on the forecast horizon.

In Algorithm 3.6, the reconstructed series is taken as the base for forecasting. For vector forecasting, it makes little sense to use the original series as the base for forecasting.

Note that in this straightforward form, Algorithm 3.6 has a much larger computational cost than Algorithm 3.5. However, a fast implementation described in Golyandina et al. (2015; Section 6.3) and realized in RSSA makes the vector forecasting as fast as the recurrent one.

3.2.3 *Forecasting in* Rssa

3.2.3.1 Description of Functions

Forecasting becomes available after an SSA decomposition is performed. Rssa implements two methods of SSA forecasting, the recurrent and vector ones, with construction of bootstrap confidence intervals if the signal is forecasted. The package provides different interfaces for forecasting.

Let the decomposition `s` be constructed by one of SSA modifications. For example, we construct the decomposition by Basic SSA as `s <- ssa(x)`. Then typical calls of forecasting functions are

```
# Recurrent forecasting
fr <- rforecast(s, groups = list(1, c(2:3)), len = 1,
                only.new = TRUE)

# Vector forecasting
fv <- vforecast(s, groups = list(trend = c(1,4)), len = 12,
                only.new = FALSE, drop = FALSE)
```

Arguments

`s` is an `ssa` object holding the decomposition;
`groups` is a list of groups of eigentriples to be used in the forecast;
`len` is a number of terms to forecast;
`base` is a series used as a "seed" of forecast: `"original"` or `"reconstructed"` (default) according to the value of groups argument;
`reverse` : `TRUE` means that the recurrent forecast is backward;
`only.new` : if `TRUE`, only the forecast values are returned; otherwise, the forecasted values from the parameter `base` are added;
`drop` acts only if one group is chosen; `TRUE` (default) value means that the result is transformed from the list of the forecasted series to the forecasted series itself.

The following function can perform the chosen forecasting algorithm along with construction of bootstrap confidence intervals. For example, one can call

```
# Bootstrap confidence intervals
bf <- bforecast(s, groups = list(1, c(2:3)), len = 1,
                R = 100, level = 0.95,
                type = "recurrent",
                interval = "confidence", only.intervals = FALSE)
```

In addition to parameters of `rforecast` and `vforecast`, the parameter `R` defines the number of simulations and `level` denotes the confidence level. The parameter `type`, which might take either `"recurrent"` or `"vector"` values, indicates what kind of forecasting should be used. The parameter `only.intervals` influences the construction of the forecast. If `only.intervals = TRUE`, then the forecast coincides with the result of the function `rforecast` (or `vforecast`, in the case of vector forecast). For the default value `only.intervals = FALSE`, the forecasting values are obtained by averaging the forecasts, which were performed during the construction of bootstrap confidence intervals. Since these forecasts are constructed for simulated series, both the bootstrap intervals and the forecasting values can differ for different calls of `bforecasts`.

The argument `interval` takes the value from `c("confidence","prediction")`. The value `"confidence"` means that the bootstrap confidence bounds for the reconstructed signal/forecast are calculated; the value `"prediction"` means that the bootstrap prediction intervals for the whole series are computed.

One of the parameters of `rforecast` and `vforecast`, which can be added to the arguments of the functions `bforecast`, is the parameter `only.new`. If `only.new = FALSE`, then the bootstrap intervals are constructed for both the signal/series and the forecast.

The following all-in-one function is designed to form an input for visualization of the forecast results by means of the FORECAST package (Hyndman 2017).

```
# All-in-one forecasting
f <- forecast(s,
              groups = list(trend = 1:4), len = 12,
              method = "recurrent",
              interval = "confidence",
              level = c(0.8, 0.99))
```

The function `predict` is exactly the same as `forecast` except for the form of the returned value.

The parameter `method` can have values `"recurrent"` (by default) and `"vector"`. The value of `interval` is from `c("none", "confidence", "prediction")`. If `interval = "none"`, then bootstrap intervals are not constructed. Opposite to the function `bforecast`, the default value of the parameter `only.intervals` in the function `forecast` is `TRUE`. Parameters of `rforecast`, `vforecast`, or `bforecast` can be added to the parameters of the functions `forecast` and `predict` depending on the values of `method` and `interval`. One of the parameters, which can be formally added, is `only.new`. However, for the function `forecast` the value of this parameter is forced to `TRUE`.

Note that the help information for `forecast` and `predict` can be obtained in R as `?forecast.ssa` and `?predict.ssa`.

Like the function `reconstruct`, all the forecasting routines try to use the attributes of the initial series for the resulting series (in particular, they try to add to the result the time index of the series). Unfortunately, this cannot be done in class-neutral way as it is done in the `reconstruct` case and needs to be handled separately for each possible type of time series. The forecasting routines know how to impute the time indices for some standard time series classes like `ts` and `zooreg`.

3.2.3.2 Typical Code

Let us demonstrate the result of application of the family of forecasting functions to the series "CO2," see Fragment 3.2.1.

Fragment 3.2.1 (Forecasting of "CO2")

```
> # Decomposition stage
> s <- ssa(co2, L = 120)
> # Recurrent forecast, the result is the forecast values only
> # The result is the set of forecasts for each group
> for1 <- rforecast(s, groups = list(1, c(1,4), 1:4, 1:6),
+                    len = 12)
> matplot(data.frame(for1), type = "b",
+         pch = c("1", "2", "3", "4"), ylab = "")
> # Vector forecast, the forecasted points are
> # added to the base series
> for1a <- vforecast(s,
+                     groups = list(1, trend = c(1,4), 1:4, 1:6),
+                     len = 36, only.new = FALSE)
> # Plot of the forecast based on the second group c(1,4)
> plot(cbind(co2, for1a$trend), plot.type = "single",
+      col = c("black", "red"), ylab = NULL)
> # Reverse recurrent forecast
> len <- 60
> for2 <- rforecast(s, groups = list(1:6), len = len,
+                    only.new = TRUE, reverse =  TRUE)
> initial <- c(rep(NA, len), co2)
> forecasted <- c(for2, rep(NA, length(co2)))
> matplot(data.frame(initial, forecasted), ylab = NULL,
+         type = "l", col = c("black", "red"), lty = c(1, 1))
> set.seed(3)
> for3 <- forecast(s, groups = list(1:6),
+                   method = "recurrent", interval = "confidence",
+                   only.intervals = FALSE,
+                   len = 24, R = 100, level = 0.99)
> plot(for3, include = 36, shadecols = "green", type = "l",
+      main = "Confidence intervals")
> set.seed(3)
> for4 <- forecast(s, groups = list(1:6),
+                   method = "recurrent", interval = "prediction",
+                   only.intervals = FALSE,
+                   len = 24, R = 100, level = 0.99)
> plot(for4, include = 36, shadecols = "green", type = "l",
+      main = "Prediction intervals")
```

Analysis of the Basic SSA decomposition (see Sect. 2.1 for recommendations) shows that ET1,4 can be referred to a trend, while ET2-3,5–6 make the seasonality group. Figure 3.3 shows a set of the forecast values for different eigentriple groups. The forecast for trend (ET1 and ET4) is shown in Fig. 3.4 together with the reconstructed series. Figure 3.5 demonstrates backward recurrent forecast. Figure 3.6 shows the bootstrap confidence and prediction intervals for the forecasts; it uses the graphical tools from the FORECAST package. Recall that the confidence intervals are constructed for the signal forecast, while the prediction intervals are constructed for the forecast of the whole signal. Therefore, prediction intervals are wider.

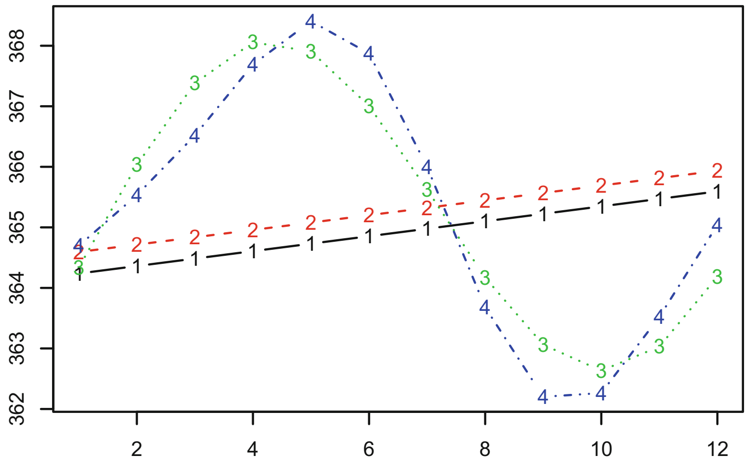

Fig. 3.3 "CO2": A set of recurrent forecasts

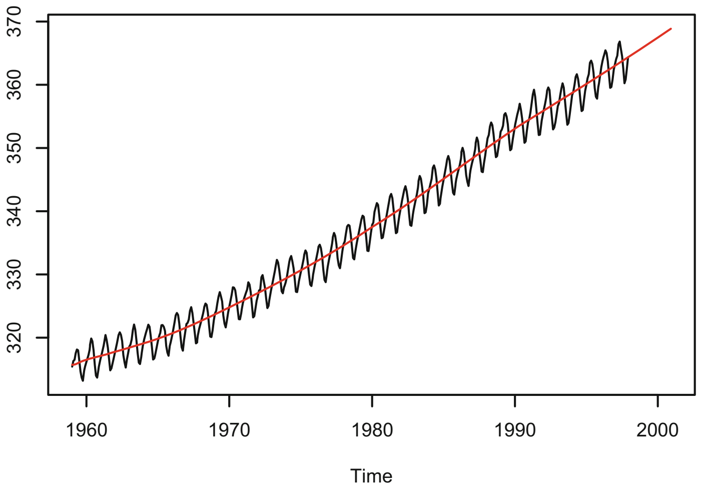

Fig. 3.4 "CO2": Forecast of trend

3.3 Gap Filling

This section is devoted to the extension of the SSA forecasting algorithms for the analysis of time series with missing data.

There are three approaches for solving this problem. The first approach was suggested in Schoellhamer (2001). This approach is suitable for stationary time series only and uses the following simple idea: in the process of the calculation of the inner products of vectors with missing components we use only pairs of valid

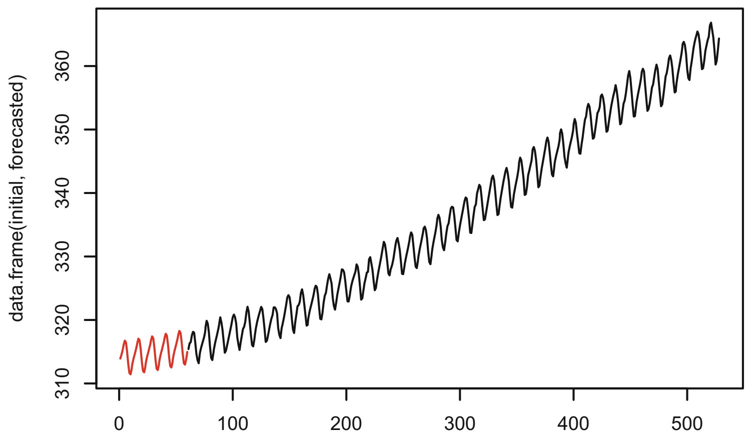

Fig. 3.5 "CO2": Backward forecast of the signal

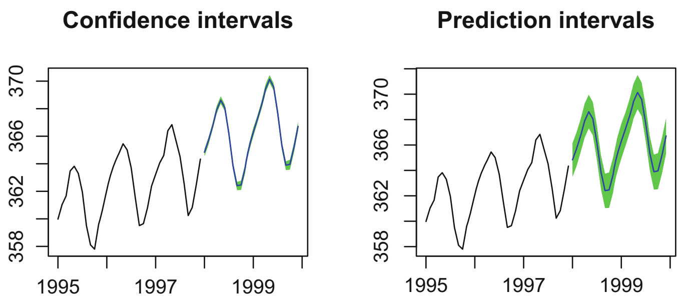

Fig. 3.6 "CO2": Plots of confidence and prediction intervals for the forecast

vector components and omit the others. The Rssa package does not implement this approach in view of its limitations. We hence concentrate on the other two approaches, the subspace-based (Golyandina and Osipov 2007) approach and the iterative (Kondrashov and Ghil 2006) one.

Usually, the problem of missing data imputation is stated as the problem of filling-in the signal data. However, the problem of imputation is more general. For example, one can be interested in imputation of missing data in the trend or seasonality only. To do it, the structure, which we are interested in, should be detected by the method. From the viewpoint of SSA, it means that the interesting series component should be separated from the residual and also the rule for the component extraction should be fixed (e.g., the indices of the eigentriples for reconstruction should be set in Basic SSA). Therefore, it makes sense to combine the considered methods with the SSA modifications described in Chap. 2 that improve separability.

For detection of structure prior to performing gap filling, Shaped SSA can be applied to the series if the location of the gaps allows the decomposition (see

Sect. 2.6). If Shaped SSA gives unsatisfactory results (for example, if the number of complete lagged vectors is too small and therefore detection of the structure is impossible), then the subspace-based approach is not applicable. However, the following general technique can be applied in the framework of the iterative approach: artificial gaps can be added and parameters of the method of gap filling can be chosen to minimize the error of imputation.

3.3.1 Method

3.3.1.1 Subspace-Based Approach

The subspace-based method of gap filling suggested in Golyandina and Osipov (2007) (see also Golyandina and Zhigljavsky (2013; Section 3.7)) is an extension of SSA forecasting algorithms. For forecasting, the last vector coordinate in a chosen subspace can be uniquely imputed as a linear combination of the first $L - 1$ coordinates. The approach can be extended for imputing a set of unknown (missing) vector coordinates as linear combinations of known coordinates. Here we use a found signal structure (in the form of a subspace) to fill the gaps. In a particular case, when missing values are located at the end of the series, the problem of filling-in of these values coincides with the problem of forecasting.

The assumptions for the gap filling are the same as for forecasting; that is, SSA should be able to approximately separate the series component of interest.

Note that imputation of gaps in separate signal components can be performed as the following two-step procedure: first, we fill gaps in the whole signal and then decompose the reconstructed signal into desired components.

Clusters of Missing Data

In the subspace-based approach, the gap filling method can be applied to different groups of missing data independently. To introduce such independent clusters, let us give several definitions following (Golyandina and Osipov 2007).

Definition 3.1 For a fixed L, a sequence of missing data of a time series is called a *cluster of missing data* if every two adjacent missing values from this sequence are separated by less than L non-missing values and there is no missing data among L neighbors (if they exist) of the left/right element of the cluster.

Thus, a group of not less than L successive non-missing values of the series separates clusters of missing data.

A cluster is called *left/right* if its left/right element is located at a distance of less than L from the left/right end of the series. A cluster is called *continuous* if it does not contain non-missing data.

The Layout of the Algorithm

Let us describe the algorithm layout. Note that the description below is different from the descriptions provided in Golyandina and Osipov (2007) and Golyandina and Zhigljavsky (2013; Section 3.7); we present it here in the form that is implemented in the Rssa package.

Assume that we have the initial time series $\mathbb{X}_N = (x_1, \ldots, x_N)$ consisting of N elements, some part of which is unknown. Let us describe the scheme of the algorithm assuming that we are reconstructing the first component $\mathbb{X}_N^{(1)}$ of the observed series $\mathbb{X}_N = \mathbb{X}_N^{(1)} + \mathbb{X}_N^{(2)}$.

The scheme of the method is as follows. Parameters are the window length L and a group I of components in the SSA decomposition. We assume that the location of missing data allows application of Shaped SSA for the chosen L. Two versions, "sequential" and "simultaneous" are suggested. These versions correspond to sequential recurrent forecasting and simultaneous vector forecasting, respectively.

Scheme of Subspace-Based Gap Filling

1. **Shaped SSA.** For the series, the shaped version of SSA is applied for the given window length L and group I. Any modification described in Chap. 2 and consistent with Shaped SSA can be used. As a result, we obtain a reconstructed series and a set of orthonormal vectors providing a basis for the approximated signal subspace.
2. **Detection of clusters of missing data.** All missing entries are split into clusters. For sequential version, each cluster is transformed into a continuous one; that is, non-missing values within the cluster are changed to NA.
3. **Forecasting.** For forecasting or filling-in several values, two approaches can be used, sequential and simultaneous. Both of these approaches can use either the recurrent or vector forecasting methods. In the current version of the Rssa package, the sequential approach uses the recurrent forecasting method, while the simultaneous approach uses a method similar to the vector forecasting one.
   Sequential gap filling-in makes forecasting from the left and from the right for each cluster with subsequent weighted averaging of the forecasting results; the subspace used for forecasting is estimated by Shaped SSA. If a cluster is left or right, then only one forecast is used for gap-filling.
   For simultaneous gap filling the so-called simultaneous forecasting is used. In Golyandina and Zhigljavsky (2013; Section 3.1), simultaneous forecasting is described in its recurrent version. Since here we use the vector version, the simultaneous gap filling coincides with the description in Golyandina and Osipov (2007), where the method consists of two operations called "Π-projection" and "simultaneous filling-in."

Discussion

Note that for successful imputation, an approximate separability of the imputed component is necessary. For exactly separated component, the missing values can be reconstructed with no error. The location of missing data is very important for the possibility of imputation by the subspace method, since the number of non-missing values should be large enough for achieving separability by Shaped SSA. At least, the number of the complete lagged vectors should be larger than the rank of the imputed time series component.

3.3.1.2 Iterative Approach

A natural and simple idea for filling-in missing values is the iterative approach, when the missing entries are initially filled-in using some reasonable values and then these values are iteratively improved by updating the SSA approximations for underlying structure of the object. This idea was suggested in Beckers and Rixen (2003) for the imputation of missing values in noisy rank-deficient matrices and was later extended to time series in Kondrashov and Ghil (2006).

For a rank-deficient matrix, the structure is defined by its rank and therefore the improvement is performed by the SVD, where the first r SVD components describe this structure. Time series of finite rank r can be considered in the form of its trajectory matrix, which has rank r and also is a Hankel matrix. Therefore, the improvement can be obtained with the help of the SVD of the trajectory matrix with subsequent hankelization. Note that this is exactly the Basic SSA algorithm with reconstruction. Also, Toeplitz SSA or SSA with projection can be used at iterations, if the series is stationary or we partly know the series model.

At each iteration, we insert the improved values at the places of missing entries and restore the initially used data at the places of non-missing entries. The initially used data may be of two types. First, the original values are used. Second, if application of Shaped SSA is possible, then the reconstructed values can be used instead of the original ones.

The approach described above can be formally applied for almost any location of missing values. Numerical experiments shows that the iterative approach can fail if missing data are located at the ends of the time series.

The iterative approach has no rigorous proof of convergence. Another drawback of the iterative approach is its impossibility to fill-in the gaps exactly even for noiseless signals. Moreover, the iterative method has large computational cost.

3.3.2 *Algorithms*

Let us start with describing a simpler iterative gap-filling algorithm. For a collection $\mathbb{Y}$ and a set of indices P we denote by $\mathbb{Y}\big|_P$ the part of the collection with the indices from P. Set $\mathcal{N} = \{1, \ldots, N\}$.

Algorithm 3.7 Iterative gap filling

```
Input: Time series $\mathbb{X}$ of length $N$ containing gaps, set of indices of missing values $P$, window length $L$, version of SSA, series $\mathbb{G}$ of length $N$ as the source of initial values for gaps, rank for reconstruction $r$, stop criterion STOP.
Output: Reconstructed series component $\widetilde{\mathbb{X}}$ with no gaps.
1: $k \leftarrow 0$, $\widetilde{\mathbb{G}}^{(k)}\big|_P = \mathbb{G}\big|_P$, $I = \{1, \ldots, r\}$.
2: Set $\widetilde{\mathbb{X}}^{(k+1)}$ such that $\widetilde{\mathbb{X}}^{(k+1)}\big|_{\mathcal{N}\setminus P} = \mathbb{X}\big|_{\mathcal{N}\setminus P}$ and $\widetilde{\mathbb{X}}^{(k+1)}\big|_P = \mathbb{G}^{(k)}\big|_P$.
3: Apply the selected version of SSA with the chosen $L$ and $I$ to $\widetilde{\mathbb{X}}^{(k+1)}$ and obtain the reconstructed series $\mathbb{G}^{(k+1)}$.
4: $k \leftarrow k + 1$
5: If not STOP, go to Step 2; else $\widetilde{\mathbb{X}} = \mathbb{G}^{(k)}$.
```

Input of the algorithm can contain several groups of indices I_k, $k = 1, \ldots m$. Then the iterations are performed for $r = \max\{i : i \in I_k, k = 1, \ldots, m\}$. In this case, the reconstruction at the last step before STOP is performed for each group I_k separately.

There is a modification of Algorithm 3.7, where $\widetilde{\mathbb{X}}^{(k+1)}\big|_{\mathcal{N}\setminus P}$ at step 2 is taken from the reconstructed series, which is calculated by Shaped SSA applied to the initial time series $\mathbb{X}$.

Below we only provide a short description of the algorithms of subspace-based filling-in. More comprehensive description and mathematical details of the algorithms can be found in Golyandina and Osipov (2007) and Golyandina and Zhigljavsky (2013; Section 3.7). The algorithms can deal with several gaps. We only describe their versions for one internal gap.

The following algorithm corresponds to a combination of methods "sequential filling-in from the left" and "sequential filling-in from the right."

Algorithm 3.8 Sequential recurrent subspace-based gap filling

```
Input: Time series $\mathbb{X}$ of length $N$ containing a gap, which starts from $i$th and finished in $j$th points, set of gap indices $P = \{i, \ldots, j\}$, $p = |P|$, window length $L$, version of SSA, group of eigentriples $I$.
Output: Reconstructed series component $\widetilde{\mathbb{X}}$ with a filled gap.
1: Apply the shaped form of the chosen SSA version to $\widetilde{\mathbb{X}}$ (Algorithms 2.13 and 2.14) and obtain the subspace $\mathcal{L} = \mathrm{span}\{P_i, i \in I\}$ and the reconstructed series $\widetilde{\mathbb{X}}$ with gaps.
2: Apply the forward recurrent forecasting algorithm in the subspace $\mathcal{L}$ starting from $(\widetilde{x}_{i-L+1}, \ldots, \widetilde{x}_{i-1})$ and construct the $p$-step recurrent forecast $\mathbb{G}^{\mathrm{left}}$.
3: Apply the backward recurrent forecasting algorithm in the subspace $\mathcal{L}$ starting from $(\widetilde{x}_{j+L-1}, \ldots, \widetilde{x}_{j+1})$ and construct the $p$-step recurrent forecast $\mathbb{G}^{\mathrm{right}}$.
4: Combine $\mathbb{G}^{\mathrm{left}}$ and $\mathbb{G}^{\mathrm{right}}$ to obtain $\mathbb{G}$. For example, $g_i = (1-\alpha_i)g_i^{\mathrm{left}} + \alpha_i g_i^{\mathrm{right}}$, $i = 1, \ldots, p$, where $\alpha_i = i/(p+1)$.
5: Set $\widetilde{\mathbb{X}}\big|_P = \mathbb{G}$.
```

Note that if the gap is right (or left), then only forward (or backward) recurrent forecasting is applied.

There is a modification of the algorithm, when the initial data for the forecasting formula at Steps 2 and 3 is taken from the initial series but not from the reconstructed series as in Algorithm 3.8.

The following algorithm corresponds to the combination of the method "Π-projector" and "simultaneous filling-in" introduced in Golyandina and Osipov (2007).

Algorithm 3.9 Simultaneous vector subspace-based gap filling

```
Input: Time series $\mathbb{X}$ of length $N$ containing a gap, which starts from $i$th and finishes in $j$th
    points, set of gap indices $P = \{i, \dots, j\}$, $p = |P|$, window length $L$, version of SSA, group
    of eigentriples $I$.
Output: Reconstructed series component $\widetilde{\mathbb{X}}$ with a filled gap.
1: Apply the shaped form of the chosen SSA version to $\widetilde{\mathbb{X}}$ (Algorithms 2.13 and 2.14) and obtain
   the subspace $\mathcal{L} = \mathrm{span}\{P_i, i \in I\}$ and the reconstructed matrix $\widehat{\mathbf{X}}$ consisting of vectors with
   non-missing values at all positions.
2: Continue the values of the complete reconstructed vectors according to Hankel structure of the
   matrix to obtain partly filled reconstructed vectors.
3: Project the valid parts of vectors by means of the $\Pi$-projector.
4: Simultaneously fill-in the missing parts of the vectors. If it is impossible, then put NA ("not
   available").
5: Hankelize the matrix $\widehat{\mathbf{X}}$ to obtain the series $\widetilde{\mathbb{X}}$. Hankelization is performed by averaging by
   non-missing values. If there are no non-missing values, then the result is NA.
```

3.3.3 Gap-Filling in Rssa

3.3.3.1 Description of Functions

Since subspace-based gap filling can be considered as interpolation (that is, as forecasting of the time series to gaps), the call of `gapfill` is similar to a call of the forecasting functions.

Similarly to forecasting, one should estimate the trajectory space of an interpolated series component by an SSA-modification; for example, by a call `s <- ssa(ts, L=120)`. Since series contains gaps, the shaped version of SSA is applied. Shaped SSA results in reconstruction of a set of series points, which can be covered by the chosen window length. Therefore, the set of uncovered points can be wider than the set of missing values. The sequential method considers uncovered points as gaps, while the simultaneous method imputes the initial gaps.

Subspace-based gap filling-in is applied to each cluster of missing values, which are detected automatically. A typical call is as follows:

```
# Subspace-based gap filling
g <- gapfill(s, groups = list(c(1,4),c(2:3,5:6)),
        base = "reconstructed",
        method = "sequential",
        alpha = function(len) seq.int(0, 1, length.out = len))
```

Arguments:

`s` is a shaped `ssa` object holding the decomposition; this kind of the object is obtained for a series with `NA` values.

`groups` is a list of groups of eigentriples used for estimation of the component subspaces.

`base` is a series used as a "seed" for gap filling: original or reconstructed according to the value of groups argument; e.g., for sequential filling-in, this is simply the series a forecasting LRR is applied to.

`method` is a method used for gap filling, `"sequential"` or `"simultaneous"`.

`alpha` is used for `method = "sequential"` and sets weights used for combining forecasts from the left and from the right. It can have the following values: 0 (1) means that only the forecast from the left (respectively, from the right) is used; 0.5 means that the forecasts are averaged. It can be a function of `len`, where `len` is the number of missing data in one gap. By default, the function provides linearly decreasing weights from 1 to 0 from both sides.

Note that the computational cost of subspace-based gap filling is very low.

Iterative gap filling is used when the signal subspace is hard (or even impossible) to estimate. Suppose that we know the rank of the signal r. Then the short call has the form

```
# Iterative gap filling
ig <- igapfill(s, groups=list(1:r),  base = "original")
```

Arguments:

`s` is a shaped `ssa` object holding the decomposition; at least, the window length L is taken from `s`. Decomposition in `s` can be empty if `base = "original"` is used.

`groups` is a list of groups of eigentriples to be used for reconstruction at iterations. Each group I is used at the first iteration. Next iterations use groups $\{1, \ldots, |I|\}$.

`fill` is a time series of the length coinciding with the length of the original series; initial values for the missing entries at the first iteration are taken from this series; if `fill = NULL`, the average of the base series is used.

`tol` is a tolerance for the difference between reconstructions at subsequent iterations.

`maxiter` is an upper bound for the number of iterations; if `maxiter = 0`, the upper bound is not used.

`norm` is a distance function used in the convergence criterion;

`base` is a series used for forced values at non-missing positions at each iteration. `"original"` is used if a shaped decomposition is impossible; `"reconstructed"` is used if Shaped SSA yields an appropriate estimation of the component subspace.

`trace` is logical; specifies whether the convergence process should be traced. For example, the number of iterations for convergence can be found in the trace.

3.3.3.2 Typical Code

Let us demonstrate the methods of gap filling by inserting artificial gaps into the time series "CO2." The methods fill gaps in a series component such as signal or trend; noise component can not be recovered.

Fragment 3.3.1 demonstrates how the subspace-based filling method is able to reconstruct a gap. We take one gap of length 100 in the middle of the time series and use the window length $L = 72$. In this case we have enough data to estimate the signal subspace.

Fragment 3.3.1 (Subspace-Based Gap Filling)

```
> F <- co2
> F[201:300] <- NA
> s <- ssa(F, L = 72)
> g0 <- gapfill(s, groups = list(c(1, 4)), method = "sequential",
+              alpha = 0, base = "reconstructed")
> g1 <- gapfill(s, groups = list(c(1, 4)), method = "sequential",
+              alpha = 1, base = "reconstructed")
> g <- gapfill(s, groups = list(c(1, 4)), method = "sequential",
+             base = "reconstructed")
> plot(co2, col = "black")
> lines(g0, col = "blue", lwd = 2)
> lines(g1, col = "green", lwd = 2)
> lines(g, col = "red", lwd = 2)
```

Figure 3.7 shows the result of sequential filling-in. Sequential filling-in is constructed as a linear combination of forecasts from the left and from the right in the subspace estimated by Shaped SSA applied to the whole series. To choose the subspace, identification of eigentriples should be performed as it was demonstrated in Sect. 2.6. Here ET1,4 correspond to a trend.

It can be seen that the accuracy of forecasts from the left and from the right gets worse as the distance from the corresponding edge increases. Therefore, one should

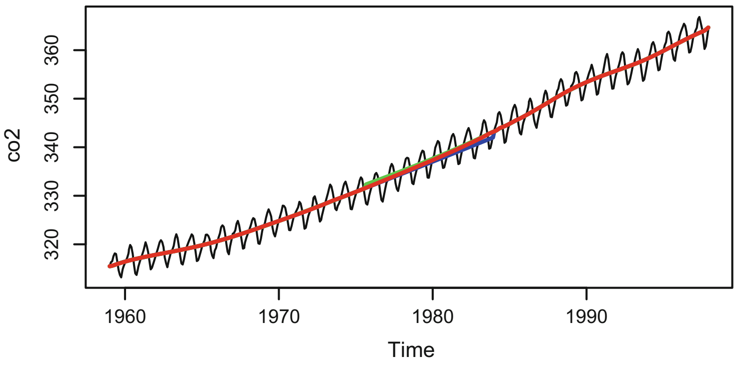

Fig. 3.7 "CO2": Subspace-based gap filling, from the left, from the right, and their combination

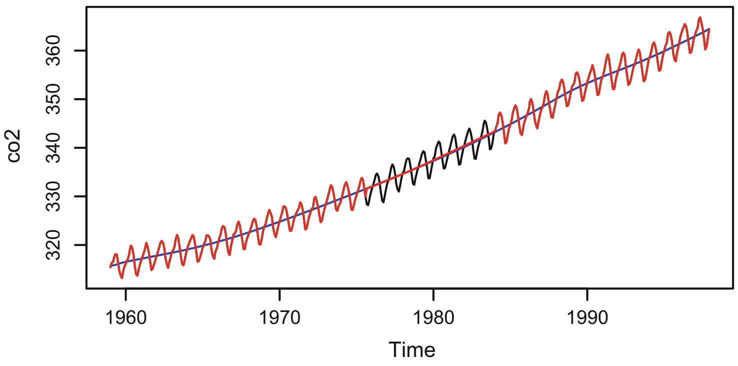

Fig. 3.8 "CO2": Iterative gap filling of trend

combine these forecasts with assigning weights which decrease while moving far from the edges (this combination is discussed in Rodrigues and de Carvalho (2013)). We take the sets of weights linearly decreasing from 1 to 0 with the sum of weights equal to 1. The linear combination of forecasts becomes more accurate.

Since for the recurrent forecast the forecasting LRR is applied to reconstructed series by default, here we also choose `base = "reconstructed"`. The mode `base = "original"` can be used only if the gaps in the signal are imputed, while `base = "reconstructed"` provides a reasonable result for reconstruction of any separable series component, e.g., seasonality.

Simultaneous gap filling cannot fill-in the considered missing data, since the window length is smaller than the size of the gap.

Let us demonstrate how the iterative method works (Fragment 3.3.2). The first example of gaps location is the same as in Fragment 3.3.1 and Fig. 3.7. At each iteration, non-missing values are returned to either reconstructed or original values. The mode with reconstructed values is available only if Shaped SSA for estimation of the component subspace is applicable. The mode with original series values is used when the component subspace cannot be estimated by Shaped SSA because of the gap location. Note that the latter cannot be used for filling-in gaps in components, which are not the leading ones. Figure 3.8 shows that both options yield approximately the same result as the subspace method.

Fragment 3.3.2 (Iterative Gap Filling, One Gap)

```
> F <- co2
> F[201:300] <- NA
> is <- ssa(F, L = 72)
> ig <- igapfill(is, groups = list(c(1,4)),
+                base = "reconstructed")
> igo <- igapfill(is, groups = list(c(1,4)),
+                 base = "original")
> # Compare the result
> plot(co2, col="black")
> lines(ig, col = "blue", lwd = 1)
> lines(igo, col = "red", lwd = 1)
> ig1 <- igapfill(is, groups = list(c(1, 4)),
+                 base = "original", maxiter = 1)
> ig5 <- igapfill(is, groups = list(c(1, 4)), fill = ig1,
+                 base = "original", maxiter = 4)
> ig10 <- igapfill(is, groups = list(c(1, 4)), fill = ig5,
+                  base = "original", maxiter = 5)
> init.lin <- F
> init.lin[200:301] <- F[200] + (0:101) / 101 * (F[301] - F[200])
> ig.lin <- igapfill(s,
+                    fill = init.lin,
+                    groups = list(c(1, 4)),
+                    base = "original", maxiter = 10)
> # Compare the result
> plot(co2, col = "black")
> lines(ig1, col = "green", lwd = 1)
> lines(ig5, col = "blue", lwd = 1)
> lines(ig10, col = "red", lwd = 1)
> lines(ig.lin, col = "darkred", lwd = 1)
```

By default, iterations are run until the difference between filled-in values becomes smaller than the given accuracy. The reconstructed trend in Fig. 3.8 was obtained with accuracy `tol = 1e-6` and used about 200 iterations; that is, 200 calls of SSA (to obtain this information, one can add `trace = TRUE` to the function parameters).

Results of filling-in after performing 1, 5 and 10 iterations are depicted in Fig. 3.9. It seems that performing 10 iterations is probably enough. The difference between filled-in values in consecutive iterations is 0.17. Note that we continue the iterations ($1 + 4 = 5$, $5 + 5 = 10$) using the results of the previous iterations as the initial values for the next iterations. These iterations start from the initial values by default; these initial values are the mean for the whole time series. Evidently, for

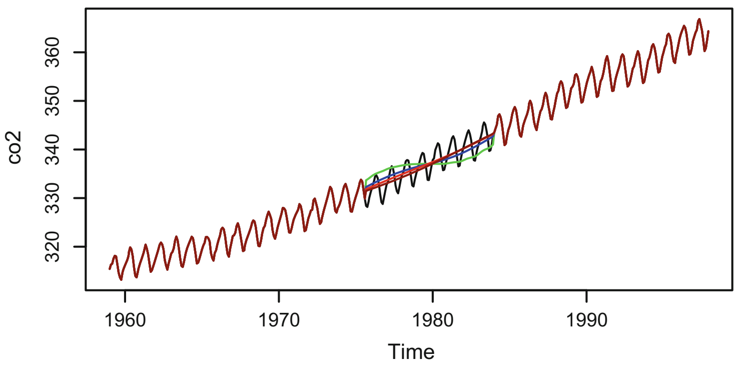

Fig. 3.9 "CO2": Iterative gap filling of trend: convergence

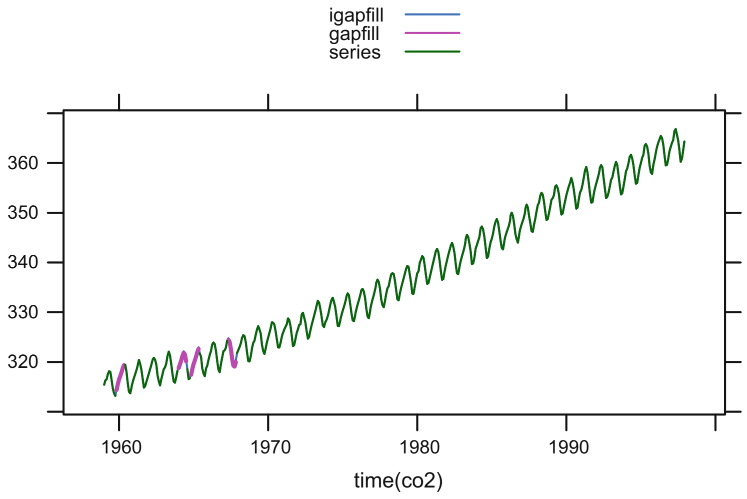

Fig. 3.10 "CO2": Iterative and simultaneous subspace-based gap filling of trend: randomly located gaps

non-stationary time series it is not a good choice. If we take a linear combination of edge values, then 10 iterations give the difference 0.1.

In the second example (Fragment 3.3.3), gaps are located arbitrarily and their location may make it difficult to estimate the signal subspace. Here we use an additional information that the components ET1–6 correspond to a signal. The result of imputation is shown in Fig. 3.10. Recall that in the case of unknown rank of the signal, one can add artificial gaps and choose the number of components to minimize errors for artificial gaps (Kondrashov and Ghil 2006).

Fragment 3.3.3 (Iterative Gap Filling, Several Gaps)

```
> F <- co2
> loc <- c(11:17, 61:67, 71:77, 101:107)
> F[loc] <- NA;
> sr <- ssa(F, L = 200)
> igr <- igapfill(sr, groups = list(c(1:6)), fill = 320,
+                 base = "original", maxiter = 10)
> gr <- gapfill(sr, groups = list(c(1:6)),
+               method = "simultaneous", base = "original")
> G <- rep(NA, length(F)); G[loc] = gr[loc]
> print(mean((gr[loc] - co2[loc])^2)) #MSE of gapfill
[1] 0.1132225
> print(mean((igr[loc] - co2[loc])^2)) #MSE of igapfill
[1] 0.1425962
> xyplot(igr + G + F ~ time(co2), type = "l",
+        lwd = c(1, 2, 1), ylab = NULL,
+        auto.key = list(lines = TRUE, points = FALSE,
+                        text = c("igapfill", "gapfill", "series")))
```

Note that it is typical for the iterative approach to use the parameter value `force.decompose = FALSE` in the preliminary call of `ssa` to avoid the decomposition, which may be impossible due to the gap location. Here we used the default value `TRUE` of `force.decompose`, since it was necessary for demonstration of `gapfill` and `igapfill` with `base = "reconstructed"`.

In addition to iterative gap filling-in, we depict the result of simultaneous subspace-based gap filling, with the original series as the base series. One can see that the results almost coincide. Moreover, the mean-squared error is slightly smaller for the simultaneous filling-in.

3.4 Structured Low-Rank Approximation

3.4.1 Cadzow Iterations

Let us consider the problem of extraction of a finite-rank signal $\mathbb{S}_N$ of rank r from an observed noisy signal $\mathbb{X}_N = \mathbb{S}_N + \mathbb{R}_N$.

The problem of finite-rank approximation can be reduced to the problem of approximation of the L-trajectory matrix $\mathbf{X}$ of the observed time series $\mathbb{X}$ by a Hankel matrix of rank r. This problem belongs to the class of problems of structured low-rank approximation (SLRA), see, e.g., Markovsky et al. (2006), Markovsky (2012).

The Hankel SLRA problem can be stated in two forms: (a) vector form and (b) matrix form. The vector (time series) form of this problem is

$$f_w(\mathbb{Y}) \to \min_{\mathbb{Y}:\mathrm{rank}\,\mathbb{Y}\le r}, \quad f_w(\mathbb{Y}) = \|\mathbb{X} - \mathbb{Y}\|_w^2 = \sum_{i=1}^{N} w_i (x_i - y_i)^2, \tag{3.8}$$

where $\mathbb{Y} = (y_1, \ldots, y_N)$ and $w_1, \ldots, w_N$ are some non-negative weights.

The Hankel SLRA problem in the matrix form is the following optimization problem:

$$f_{\mathbf{M}}(\mathbf{Y}) \to \min_{\mathbf{Y}\in\mathcal{M}_r\cap\mathcal{H}}, \quad f_{\mathbf{M}}(\mathbf{Y}) = \|\mathbf{X}-\mathbf{Y}\|_{\mathbf{M}}^2 = \sum_{l=1}^{L}\sum_{k=1}^{K} m_{lk}(x_{lk}-y_{lk})^2, \tag{3.9}$$

where $\mathbf{M} = [m_{lk}]$ is a matrix of size $L \times K$, $\mathcal{H} = \mathcal{M}_{L,K}^{(\mathrm{H})} \subset \mathsf{R}^{L\times K}$ is the space of Hankel matrices of size $L \times K$, $\mathcal{M}_r \subset \mathsf{R}^{L\times K}$ is the set of matrices of rank not larger than r. Matrices $\mathbf{X}$ and $\mathbf{Y}$ are related to vectors (time series) $\mathbb{X}$ and $\mathbb{Y}$ by, respectively, $\mathbf{X} = \mathcal{T}(\mathbb{X})$ and $\mathbf{Y} = \mathcal{T}(\mathbb{Y})$, where $\mathcal{T} = \mathcal{T}_{\mathrm{SSA}}$ is the SSA embedding operator defined in (2.1).

Each $\mathbf{M}$ in (3.9) generates a set of weights w_i in (3.8) by the equality $f_{\mathbf{M}}(\mathbf{Y}) = f_w(\mathbb{Y})$. If $m_{lk} = 1$ for all l and k, then the weights w_i are as in (2.2).

The iterative step of the method of alternating projections for solving (3.9) has the following form:

$$\mathbf{Y}_{k+1} = \breve{\Pi}_{\mathcal{H}} \circ \breve{\Pi}_{\mathcal{M}_r}(\mathbf{Y}_k), \tag{3.10}$$

where $\mathbf{Y}_0 = \mathbf{X}$ and $\breve{\Pi}_{\mathcal{H}}$ and $\breve{\Pi}_{\mathcal{M}_r}$ are projectors to the corresponding sets with respect to the norm $\|\cdot\|_{\mathbf{M}}$.

For the matrix $\mathbf{M}$ with $m_{lk} = 1$ for all l and k, we obtain the well-known method of Cadzow iterations (Cadzow 1988). In this case, $\|\cdot\|_{\mathbf{M}}$ is the conventional Frobenius norm and $\breve{\Pi}_{\mathcal{H}} = \Pi_{\mathcal{H}}$. We will keep the same name "Cadzow iterations" for general $\mathbf{M}$.

The projector $\breve{\Pi}_{\mathcal{H}}$ is calculated in a straightforward way: for $\widehat{\mathbf{Y}} = \breve{\Pi}_{\mathcal{H}}\mathbf{Y}$ we have

$$\hat{y}_{ij} = \frac{\sum_{l,k:\, l+k=i+j} m_{lk} y_{lk}}{\sum_{l,k:\, l+k=i+j} m_{lk}}. \tag{3.11}$$

The projector $\breve{\Pi}_{\mathcal{M}_r}$ is easily calculated if there exist $\mathbf{P}$ and $\mathbf{Q}$ such that $\|\mathbf{X} - \mathbf{Y}\|^2_{\mathbf{M}}$ corresponds to the Frobenius norm with respect to the oblique inner products $(X, Y)_{\mathbf{P}} = X^{\mathrm{T}}\mathbf{P}Y$ in R^L and $(X, Y)_{\mathbf{Q}} = X^{\mathrm{T}}\mathbf{Q}Y$ in R^K. Then $\breve{\Pi}_{\mathcal{M}_r}$ is calculated as the r leading terms of the $(\mathbf{P}, \mathbf{Q})$-SVD defined by (2.14) in Definition 2.5.

One of examples of such $\mathbf{M}$ is the case, when $\mathbf{P} = \mathrm{diag}(p_1, \ldots, p_L)$, $\mathbf{Q} = \mathrm{diag}(q_1, \ldots, q_K)$ and therefore $m_{ij} = p_i q_j$. In this case, as explained in Zhigljavsky et al. (2016a), $\|\cdot\|_{\mathbf{M}} = \|\cdot\|_w$ if and only if $W = P \star Q$, where $W = (w_1, \ldots, w_N)$, $P = (p_1, \ldots, p_L)$, $Q = (q_1, \ldots, q_K)$, and $\star$ denotes convolution of vectors.

A natural choice of equal weights w_i in (3.8) corresponds to the ordinary least-squares method. As there are no vectors P and Q with positive elements (in this case $(\cdot,\cdot)_{\mathbf{P}}$ and $(\cdot,\cdot)_{\mathbf{Q}}$ would be norms) such that $(1, 1, \ldots, 1) = P \star Q$ only approximately equal weights w_i, $i = 1, \ldots, N$, can be achieved; see, e.g., Zhigljavsky et al. (2016b) or Gillard and Zhigljavsky (2016).

As a rule, Cadzow iterations do not converge to the optimal solution, see, e.g., Gillard and Zhigljavsky (2011, 2013). There are some partial results on convergence of Cadzow iterations to the set $\mathbf{Y} \in \mathcal{M}_r \cap \mathcal{H}$, but even this question is hard, see, e.g., Zvonarev and Golyandina (2017). One may try to solve (3.8) by applying standard techniques of global optimization in a parametric space (assuming the model of the sum of damped sinusoids, for example) but the arising optimization problem is far too difficult, see Gillard and Kvasov (2016). Another general approach to the numerical solution of the weighted SLRA problems can be found in Markovsky and Usevich (2014).

Assume we have stopped after k iterations of (3.10) and have mapped the matrices to R^N. Then the resulting series can be written as

$$\widetilde{\mathbb{S}}_N = \mathcal{T}^{-1} \circ \left(\breve{\Pi}_{\mathcal{H}} \circ \breve{\Pi}_{\mathcal{M}_r}\right)^k \circ \mathcal{T}(\mathbb{X}_N). \tag{3.12}$$

The vector (series) $\widetilde{\mathbb{S}}_N$ obtained by (3.12) can be considered as an estimator of the signal. If $k = 1$ and $m_{lk} = 1$ for all l and k, then (3.12) is simply the Basic SSA reconstruction with $I = \{1, \ldots, r\}$.

Let the series length N be divisible by the window length L. Choose some $\alpha > 0$ and set $p_i = 1$ for each i and

$$q_k = q_k(\alpha) = \begin{cases} 1, & \text{if } k = jL + 1 \text{ for some } j, \\ \alpha, & \text{otherwise.} \end{cases} \tag{3.13}$$

According to Zvonarev and Golyandina (2017), we will call the method (3.12) with these $P = (p_1, \ldots, p_L)$ and $Q = (q_1, \ldots, q_K)$ "Cadzow(α) iterations." In this notation, Cadzow(1) corresponds to the conventional Cadzow iterations, while Cadzow(0) corresponds to the Cadzow iterations that would attempt to solve the problem (3.8) with $w_i = 1$ ($i = 1, \ldots, N$). Cadzow(0) does not solve the problem (3.8) in view of the degeneracy of some weights in Q. As an approximation to Cadzow(0) iterations, Cadzow(α) iterations with small $\alpha > 0$ are considered.

It is shown in Zvonarev and Golyandina (2017) that Cadzow(α) with smaller α has slower convergence rate and better accuracy in the limit than Cadzow(α) with larger α.

3.4.2 Algorithms

We call Algorithm 3.10 Cadzow iterations. It implements the iterations (3.12), where projections are performed with respect to the norm, induced by the left and right weight vectors P and Q.

Algorithm 3.10 Cadzow iterations

Input: Time series $\mathbb{X}$ of length N, window length L, version of SSA, weight vectors $P \in \mathsf{R}^L$ and $Q \in \mathsf{R}^K$, rank for reconstruction r, stop criterion STOP.

Output: Approximation $\widetilde{\mathbb{X}}$ of rank r for the series $\mathbb{X}$.

```
1: $k \leftarrow 0$, $\mathbf{X}^{(0)} = \mathcal{T}(\mathbb{X})$.
2: $\mathbf{X}_r^{(k+1)} = \breve{\Pi}_{\mathcal{M}_r}\mathbf{X}^{(k)}$ taking the leading $r$ components of the $(\mathbf{P}, \mathbf{Q})$-SVD, see Algorithm 2.6.
3: $\mathbf{X}^{(k+1)} = \breve{\Pi}_{\mathcal{H}}\mathbf{X}_r^{(k)}$ by (3.11) with $m_{ij} = p_i q_j$.
4: $k \leftarrow k + 1$;
5: If not STOP, go to Step 2; else $\widetilde{\mathbb{X}} = \mathcal{T}^{-1}(\mathbf{X}^{(k)})$.
```

Note that in the RSSA package the input data are partly taken from the `ssa` object (e.g., the window length L).

3.4.3 *Structured Low-Rank Approximation in* RSSA

3.4.3.1 Description of Functions

A typical call is as follows:

```
c <- cadzow(s, rank = 2, correct = FALSE, tol = 1e-6,
       maxiter = 100, norm = function(x) max(abs(x)))
```

Arguments:

`s` is an `ssa` object holding the decomposition parameters. Decomposition in `s` can be empty.
`tol` is a tolerance for the difference between reconstructions at sequential iterations.
`maxiter` is an upper bound for the number of iterations; if `maxiter = 0`, this upper bound is not used.
`norm` is a distance function used for the convergence criterion;
`trace` is logical; it indicates whether the convergence process should be traced. For example, the number of iterations for convergence can be found in the trace, which is printed at the output window.

There are no parameters related to weights in Cadzow iterations. This is because the weights are fully specified in the preliminary call of the `ssa` function. For example, in the call

```
s <- ssa(x, column.oblique = 'identity', row.oblique = weights)
```

the parameter `row.oblique = weights` provides the vector of right weights of length K.

3.4.3.2 Typical Code

Let us demonstrate how to perform forecasting by means of the result of weighted Cadzow iterations. We take the first 360 points of the “CO2” series and $L = 60$. Small values of α provide better accuracy of the signal reconstruction but slow convergence, see Zvonarev and Golyandina (2017). Therefore, we take 10 iterations with small $\alpha = 0.01$ and then continue iterations with $\alpha = 1$ to reach convergence. Fragment 3.4.1 contains the code for such iterations. The weights are constructed according to (3.13).

Fragment 3.4.1 (Weighted Cadzow Approximation)

```
> cut <- 49 + 60
> x <- window(co2, end = time(co2)[length(co2) - cut + 1])
> L <- 60
> K <- length(x) - L + 1
> alpha <- 0.01
> weights <- vector(len = K)
> weights[1:K] <- alpha
> weights[seq(K, 1, -L)] <- 1
> xyplot(weights ~ 1:K, type = "l")
> s1 <- ssa(x, L = L) #to detect the series rank
> ncomp <- 6
> s01 <- ssa(x, L = L, column.oblique = "identity",
+            row.oblique = weights)
> c01 <- cadzow(s01, rank = ncomp, maxiter = 10)
> s01.1 <- ssa(c01, L = L, column.oblique = NULL,
+             row.oblique = NULL)
> c01.1 <- cadzow(s01.1, rank = ncomp, tol = 1.e-8 * mean(co2))
> print(t(ssa(c01.1, L = ncomp + 1)$sigma), digits = 5)
      [,1]   [,2]   [,3]  [,4]  [,5]     [,6]       [,7]
[1,] 16472 73.537 41.096 12.29 4.674 0.044457 2.2465e-06
> ss01.1<- ssa(c01.1, L = ncomp + 1)
> fr <- rforecast(ss01.1, groups = list(1:ncomp), len = cut)
> xyplot(cbind(Original = co2, Cadzow1and01 = c01.1,
+             ForecastCadzow = fr), superpose = TRUE)
> print(parestimate(ss01.1, groups = list(1:ncomp),
+                  method = "esprit"))
   period      rate   |   Mod     Arg  |     Re        Im
   11.998   0.000525 |  1.00053   0.52 |  0.86643   0.50035
  -11.998   0.000525 |  1.00053  -0.52 |  0.86643  -0.50035
      Inf   0.000450 |  1.00045  -0.00 |  1.00045  -0.00000
    5.998   0.000287 |  1.00029   1.05 |  0.49986   0.86644
   -5.998   0.000287 |  1.00029  -1.05 |  0.49986  -0.86644
      Inf  -0.004656 |  0.99535  -0.00 |  0.99535  -0.00000
```

The resultant signal estimate `c01.1` has rank $r = 6$: the 7-th singular value is practically zero. The trajectory subspace of this signal for any $L > 6$ uniquely defines the LRR, which governs the estimated signal and provides the forecasting formula. To obtain this formula by means of RSSA, we apply Basic SSA with $L = r + 1$ and then construct forecast by any forecasting method, see Fig. 3.11. In addition to forecasting, parameter estimation can be performed by applying the corresponding function to the `ssa` object.

3.4.3.3 Simulated Example

One example presented in Sect. 3.4.3.2 cannot show if Cadzow(α) iterations are better than the conventional Cadzow(1) iterations. Therefore, we perform simulations for a noisy sine wave. The parameters are chosen to repeat one of the numerical results of Zvonarev and Golyandina (2017). Simulations show that

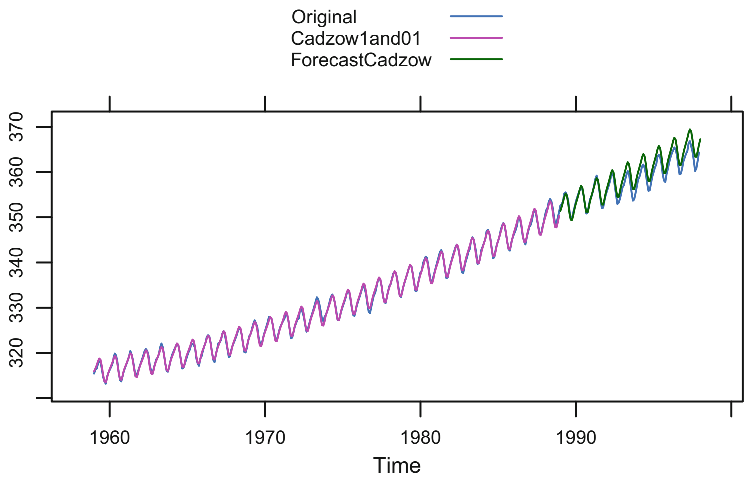

Fig. 3.11 "CO2": Approximation of rank 6 by the weighted Cadzow method and its forecast

$\alpha = 0.01$ provides approximately 10% improvement in the MSE of the signal estimator.

Fragment 3.4.2 (Accuracy of Weighted Cadzow Approximation)

```
> SIMUL <- FALSE
> set.seed(3)
> L <- 20
> N <- 2 * L
> K <- N - L + 1
> alpha <- 0.01
> sigma <- 1
> signal <- 5 * sin(2 * pi * (1:N) / 6)
> weights <- vector(len = K)
> weights[1:K] <- alpha
> weights[seq(1, K, L)] <- 1
> M <- 1000
> norm.meansq <- function(x) mean(x^2)
> if (SIMUL) {
+   RMSE <- sqrt(rowMeans(replicate(M, {
+     x <- signal + sigma * rnorm(N)
+     s.alpha <- ssa(x, L = L, column.oblique = NULL,
+                        row.oblique = weights)
+     c.alpha <- cadzow(s.alpha, rank = 2, tol = 1.e-8,
+                           norm = norm.meansq,
+                           correct = FALSE)
+     s <- ssa(x, L = L)
+     cc <- cadzow(s, rank = 2, norm = norm.meansq, tol = 1.e-8,
+                  correct = FALSE)
+     c("err" = mean((cc - signal)^2),
+       "err.alpha" = mean((c.alpha - signal)^2))
+   })))
+
+   cadzow.sim <- as.data.frame(t(RMSE))
+ } else {
+   data("cadzow.sim", package = "ssabook")
+ }
> print(cadzow.sim)
        err err.alpha
1 0.3753331 0.3222088
```

3.5 Case Studies

3.5.1 Forecasting of Complex Trend and Seasonality

Let us consider the series "Elec," which was analyzed in Sect. 2.8.8. We will construct forecasts obtained using the subseries with the last two years removed and then compare the two-year forecasts with the known data. The series "Elec" has complex trend and therefore the trend can be extracted by means of Basic SSA with a small window length. Unfortunately, the window length $L = 12$ is too small to obtain a stable forecast. However, larger window lengths would make the signal and residual to be badly mixed. Iterated O-SSA can help to better separate the trend from the residual. Fragment 3.5.1 contains the code, which performs two forecasts, on the base of the whole time series and on the base of the second half of the series. Figure 3.12 shows that Iterative O-SSA allows to obtain accurate forecasts. Moreover, both forecasts are almost the same.

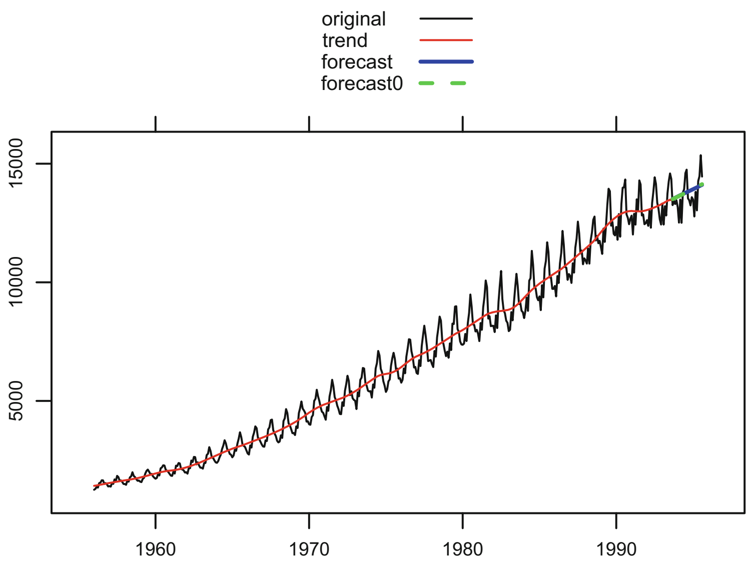

Fig. 3.12 "Elec": Trend forecasting

Fragment 3.5.1 ("Elec": Trend Forecasting and `iossa`)

```
> data("elec", package = "fma")
> N <- length(elec)
> len <- 24
> L <- 24
> s <- ssa(window(elec, end = c(1993, 8)), L = L)
> si <- iossa(s, nested.groups = list(c(1, 4), c(2, 3, 5:10)))
> fi <- rforecast(si, groups = list(trend = c(1:2)),
+                 len = len, only.new = FALSE)
> s0 <- ssa(window(elec, start = c(1972, 8), end = c(1993, 8)),
+           L = L)
> f0 <- vforecast(s0, groups = list(trend = c(1)),
+                 len = len, only.new = TRUE)
> si0 <- iossa(s0, nested.groups = list(c(1,4), c(2,3,5:10)))
> fi0 <- vforecast(si0, groups = list(trend = c(1:2)),
+                  len = len, only.new = TRUE)
> theme <- simpleTheme(col = c("black", "red", "blue", "green"),
+                      lwd = c(1, 1, 2, 2),
+                      lty = c("solid", "solid",
+                              "solid", "dashed"))
> xyplot(cbind(elec,
+              window(fi, end = c(1993, 8)),
+              window(fi, start = c(1993, 9)),
+              fi0),
+        superpose = TRUE, type ="l", ylab = NULL, xlab = NULL,
+        auto.key = list(text = c("original", "trend",
+                                 "forecast", "forecast0"),
+                        type = c("l", "l", "l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme)
```

Unlike the trend, seasonality has a stable structure. The window length L equal to two periods is still too small to obtain an accurate forecast. Therefore, a combined forecast similar to Sequential SSA briefly described in Sect. 2.1.3.2 can be recommended. In Fragment 3.5.2, the trend estimated in Fragment 3.5.1 is subtracted from the series and the residual is forecasted with large window $L = 240$. Figure 3.13 contains the forecast obtained by the sum of trend and seasonality forecasts. Note that the choice of eigentriples for forecasting was performed by the standard technique, which involves the analysis of eigenvectors as demonstrated in Sect. 2.1.5.3.

Fragment 3.5.2 ("Elec": Combined Forecasting)

```
> L <- 240
> elec_sa <- elec - fi
> s_sa <- ssa(window(elec_sa, end = c(1993, 8)), L = L)
> f_sa <- rforecast(s_sa, groups = list(trend = c(1:13)),
+                   len = len, only.new = FALSE)
> theme <- simpleTheme(col = c("black", "red", "green"),
+                      lwd = c(1, 2, 2),
+                      lty = c("solid", "solid", "solid"))
> xyplot(cbind(window(elec, start = c(1985, 12)),
+             window(fi, start = c(1985, 12), end = c(1993, 8)),
+             window(fi, start = c(1993, 9)) +
+               window(f_sa, start = c(1993, 9))),
+       superpose = TRUE, type = "l", ylab = NULL, xlab = NULL,
+       auto.key = list(text = c("original", "trend",
+                                "forecast"),
+                       type = c("l", "l", "l"),
+                       lines = TRUE, points = FALSE),
+       par.settings = theme)
```

3.5.2 Different Methods of Forecasting

In the example considered in Sect. 3.5.1, it was not important which forecasting method to apply, since the series “Elec” has a reasonably stable structure. Generally, the closeness of vector and recurrent forecasts can be considered as an indication of structural stability of the series component we have chosen to forecast. This does not guarantee, however, that the forecasts are accurate, since the structure of the series may change in the future.

Here we demonstrate a difference between recurrent and vector forecasts. From the theoretical point of view, the recurrent forecast is simpler, since it is just a continuation by an LRR. An explicit formula for this continuation can be constructed in a similar manner to what is done in Sect. 3.5.5 below. Vector forecasting, however, can be more accurate for continuation of a stable structure; in particular, it performs extra $L-1$ steps; the extra steps are done for making the

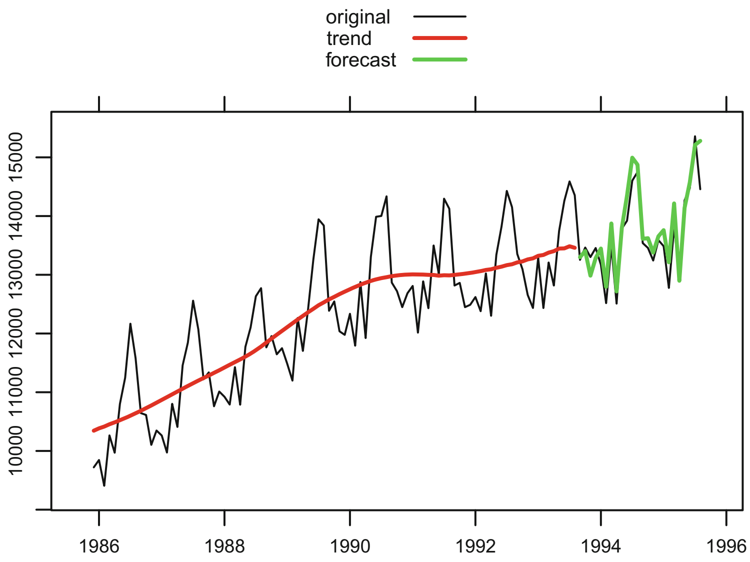

Fig. 3.13 “Elec”: Combined forecasting

coincidence between an M-step forecast and the first M points of $(M + 1)$-step forecast.

Let us consider a small example "Cowtemp" (daily morning temperature of a cow).

We have removed the last 14 points (2 weeks) and constructed the recurrent and vector forecasts of the 61-term series. Window length was chosen to be equal to 4 weeks. It is large and therefore we can expect the global tendency to be captured well. In addition to the two main forecasts, we consider the forecast based on Toeplitz SSA (see Sect. 2.2). Recall that Toeplitz SSA is designed to analyze stationary series. The considered series probably has a trend. Therefore, we take a small window length for Toeplitz SSA. The code, which implements three methods of forecasting, is contained in Fragment 3.5.3. In all cases, we will construct forecasts based on one leading eigentriple.

Figure 3.14 shows several typical effects. First, if the separability is not good, then the recurrent forecast can have a jump at its first point. We can see that the vector forecast (a) has no jumps and (b) changes the last $L - 1$ points of the reconstructed series. Nevertheless, both forecasts are similar. They have the form $C \cdot 0.995^n$.

The estimator of the signal root by Toeplitz SSA is much closer to 1. Note that `parestimate` for Toeplitz SSA force unit moduli of roots by default. For the option `normalize.roots = FALSE`, which we have chosen, the root estimate is almost 1. The root estimator obtained as the maximum-modulus root of the characteristic polynomial of the estimated LRR (which is the forecasted LRR for the recurrent forecast) is equal to 0.999. Thus, we see that the vector and recurrent Basic SSA

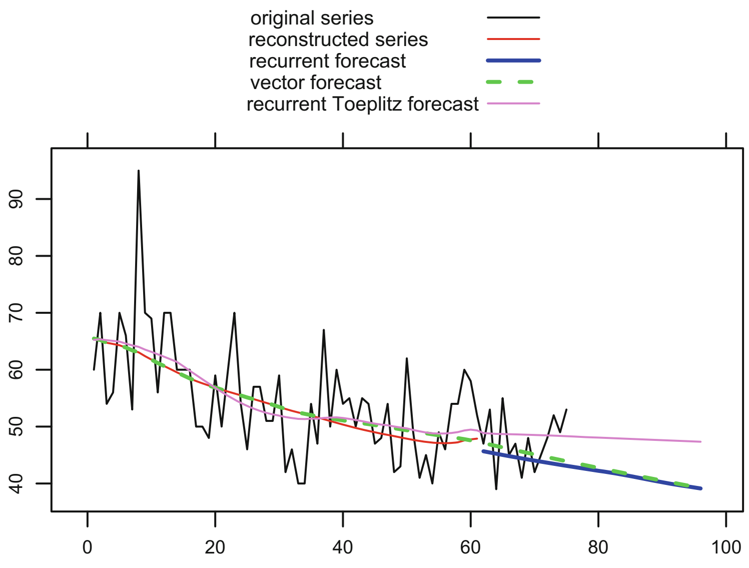

Fig. 3.14 "Cowtemp": Basic SSA and Toeplitz SSA forecasting

forecasts decrease as 0.995^n, while the recurrent Toeplitz SSA forecast decreases slower as 0.999^n.

On the time interval [62, 75], the forecasts are more or less similar. However, the long-term forecast is much more appropriate in the case of Toeplitz SSA, since the temperature is likely to oscillate around a constant and cannot rapidly decrease. Therefore, the method, which adds the limitation of stationarity, wins in this particular example.

Fragment 3.5.3 (“Cowtemp”: Different Methods of Forecasting)

```
> data("cowtemp", package = "fma")
> series <- cowtemp
> N <- length(series)
> cut <- 14
> future <- 21
> len <- cut + future
> r <- 1
> L <- 28
> Lt <- 14
> s <- ssa(window(series, end = N - cut), L = L)
> parestimate(s, groups = list(trend = c(1:r)),
+             method = "esprit")$moduli
[1] 0.9946241
> roots(lrr(s, groups = list(trend = c(1:r))))[1]
[1] 0.9953519+0i
> rec <- reconstruct(s, groups = list(1:r))
> st <- ssa(window(series, end = N - cut),
+           kind = "toeplitz-ssa", L = Lt)
> parestimate(st, groups = list(trend = c(1:r)),
+             method = "esprit")$moduli
[1] 1
> parestimate(st, groups = list(trend = c(1:r)),
+             normalize.roots = FALSE,
+             method = "esprit")$moduli
[1] 0.9999984
> roots(lrr(st, groups = list(trend = c(1:r))))[1]
[1] 0.9990458+0i
> fr <- rforecast(s, groups = list(trend = c(1:r)),
+                 len = len, only.new = TRUE)
> fv <- vforecast(s, groups = list(trend = c(1:r)),
+                 len = len, only.new = FALSE)
> ftr <- rforecast(st, groups = list(trend = c(1:r)),
+                  len = len, only.new = FALSE)
> print(sqrt(mean((window(fr, start = 62) -
+              window(series, start = 62))^2)))
[1] 5.711485
> print(sqrt(mean((window(fv, start = 62) -
+              window(series, start = 62))^2)))
[1] 5.253602
> print(sqrt(mean((window(ftr, start = 62) -
+              window(series, start = 62))^2)))
[1] 4.785783
```

```
> theme <- simpleTheme(col = c("black", "red", "blue",
+                              "green", "violet"),
+                      lwd = c(1, 1, 2, 2, 1),
+                      lty = c("solid", "solid", "solid",
+                              "dashed", "solid"))
> future.NA <- rep(NA, future)
> xyplot(cbind(series, rec$F1, fr, fv, ftr),
+        superpose = TRUE, type = "l", ylab = NULL, xlab = NULL,
+        auto.key = list(text = c("original series",
+                                 "reconstructed series",
+                                 "recurrent forecast",
+                                 "vector forecast",
+                                 "recurrent Toeplitz forecast"),
+                        type = c("l", "l", "l", "l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme)
```

3.5.3 Choice of Parameters and Confidence Intervals

Since SSA is a model-free method, there are no theoretical confidence intervals. As shown in Sect. 3.2.1.5 we can naturally construct bootstrap confidence intervals for the signal. The assumption for the adequacy of bootstrap confidence intervals is that the signal is extracted and a model for the residual distribution is built. In practice, these assumptions are not valid (or we cannot check them) and therefore bootstrap confidence intervals can only be considered as indicative measures of accuracy of the forecasts.

By considering bootstrap confidence intervals, the user investigates forecasting stability. As an alternative to bootstrap confidence intervals, one can study forecast response to a perturbation of the initial series. To do this, we add to the initial series a perturbation (e.g., a white noise with some standard deviation σ) and look at the resulting variability of forecasts. Similarly to the case of bootstrap confidence intervals, for a confidence level γ, we will consider the intervals between $(1-\gamma)/2$ lower and upper quantiles and call them *perturbed forecasting intervals* (see Fragment 3.5.4). Note that the influence of perturbation on the signal subspace estimation is theoretically studied in Nekrutkin (2010).

Fragment 3.5.4 (Function for Perturbed Forecasting Intervals)

```
> perturbation <-
+   function(s, noise, R, Qfor, num.comp, L, level, template) {
+   r <- reconstruct(s, groups = list(1:num.comp))
+   stopifnot(length(r) == 1)
+
+   delta <- sd(residuals(r))
+   res <- matrix(nrow = Qfor, ncol = R)
+
+   ser <- s$F; attributes(ser) <- s$Fattr
+   for (j in 1:R) {
```

```
+     si <- ssa(ser + delta * noise[, j], L = L)
+     res[, j] <- rforecast(si, groups = list(1:num.comp),
+                           len = Qfor)
+   }
+
+   cf <- apply(res, 1, quantile,
+               probs = c((1 - level) / 2, (1 + level) / 2))
+   out <- template
+   out$x[] <- ser
+   out$fitted[] <- r[[1]]
+   out$residuals[] <- residuals(r)
+   out$lower[] <- cf[1, ]
+   out$upper[] <- cf[2, ]
+   out$level[] <- 100 * level
+   out$mean[] <- rowMeans(res)
+
+   out
+ }
```

Fragment 3.5.5 shows how the bootstrap and perturbed intervals depend on the number of eigentriples, which are used for reconstruction, for the total wine sales "Total" from the dataset "AustralianWine".

Fragment 3.5.5 ("Total": Stability of Forecasting)

```
> data("AustralianWine", package = "Rssa")
> wine <- window(AustralianWine, end = time(AustralianWine)[174])
> ser0 <- wine[, "Total"]
> Q <- 66
> l <- length(ser0)
> ser <- window(ser0, end = time(ser0)[l-Q])
> include <- min(1000, l - Q)
> L <- 48
> s <- ssa(ser, L = L)
> plot(wcor(s, groups = 1:min(nu(s), 50)),
+      scales = list(at = c(10, 20, 30, 40, 50)))
> set.seed(1)
> R <- 100
> noise <- matrix(rnorm(length(ser) * R), nrow = length(ser))
> range <- 1:30
> err.pert <- numeric(length(range))
> err <- numeric(length(range))
> k <- 1
> for (num.comp in range) {
+   bf0 <- forecast(s, groups = list(1:num.comp),
+                   method = "recurrent",
+                   interval = "confidence",
+                   len = Q, R = R, level = 0.9)
+
+   bf0.pert <- perturbation(s, noise, R, Q, num.comp, L,
+                            level = 0.9, bf0)
+   err.pert[k] <- sqrt(mean((bf0.pert$upper - bf0.pert$lower)^2))
+   err[k] <- sqrt(mean((bf0$upper - bf0$lower)^2))
+   k <- k + 1
+ }
> bf0.pert1 <- perturbation(s, noise, R, Q, 1, L,
+                           level = 0.9, bf0)
> plot(bf0.pert1, include = include, shadecols = "green",
+      main = paste("Perturbed SSA forecast, 1 component"))
> bf0.pert12 <- perturbation(s, noise, R, Q, 12, L,
+                            level = 0.9, bf0)
> plot(bf0.pert12, include = include, shadecols = "green",
+      main = paste("Perturbed SSA forecast, 12 components"))
> bf0.pert14 <- perturbation(s, noise, R, Q, 14, L,
+                            level = 0.9, bf0)
> plot(bf0.pert14, include = include, shadecols = "green",
+      main = paste("Perturbed SSA forecast, 14 components"))
> start <- 48
> theme <- simpleTheme(col = c("black","red","blue"),
+                      lwd = c(2, 1, 2),
+                      lty = c("solid", "solid", "solid"))
> xyplot(cbind(window(ser0, start = c(1984, 1)),
+              bf0.pert12$mean,
+              bf0.pert14$mean),
+        superpose = TRUE, type = "l", ylab = NULL, xlab = NULL,
+        auto.key = list(text = c("'Total'",
+                                 "forecast, ET1-12",
+                                 "forecast, ET1-14"),
+                        type = c("l", "l", "l"),
+                        lines = TRUE, points = FALSE),
+        par.settings = theme)
> xyplot(err + err.pert ~ range, type = "l",
+        ylab = NULL, xlab = NULL,
+        auto.key = list(text = c("Bootstrap errors",
+                                 "Perturbation errors"),
+                        type = c("l", "l"),
+                        lines = TRUE, points = FALSE),
+        scales = list(y = list(log = TRUE)))
```

We take the first 108 points and consider 66-term forecasting. Window length is $L = 48$. Figure 3.15 contains the graph of **w**-correlations. One can see that there are several natural candidates for the number of leading components which we can choose. For example, if we choose the number of leading components to be equal to 12, 16, 17, or 19, then the reconstructed signal looks to be almost unmixed with the residual. Let us calculate the mean size of bootstrap and perturbed 90% confidence intervals for the forecasts performed for different numbers of leading components. For construction of the perturbed intervals, we will take σ equal to the standard deviation of the residuals after the reconstruction by the corresponding number of components. Figure 3.16 contains square roots of mean squared confidence ranges as a function of the number of components. One can see that after 12 components the variability of forecasts sharply increases. On the base of this, the choice of 12 components can be recommended. It is interesting that there is only a small difference between bootstrap and perturbed intervals.

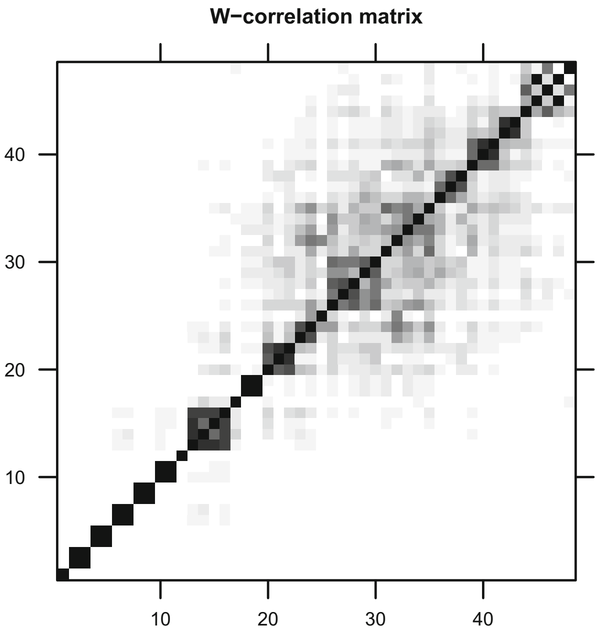

Fig. 3.15 "Total": **w**-Correlations

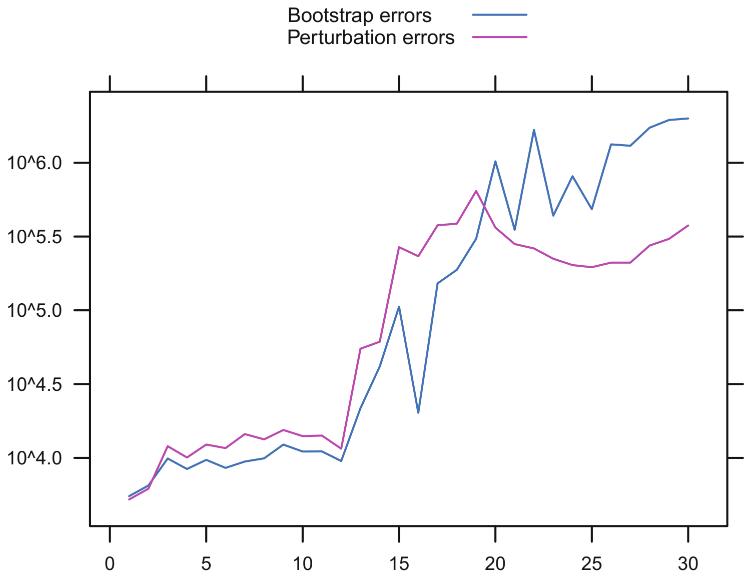

Fig. 3.16 "Total": Sizes of 90% forecasting intervals in dependence on the number of components

Let us demonstrate how the confidence intervals look like for forecasts which use ET1, ET1–12, and ET1–14 (Figs. 3.17, 3.18, and 3.19, respectively). Figure 3.19 shows that the long-term forecasting by 14 components is likely to be wrong.

Let us compare the obtained forecasts with the known series values. The mean forecasts which are calculated by averaging simulated forecast values are depicted. Figure 3.20 shows that, first, the long-term forecast by ET1–12 is more or less adequate, but the forecast by ET1–14 fails. On the other hand, short-term forecasts by ET1–12 and 1–14 are similar. Moreover, the "wrong" forecast (by ET1–14) is slightly more accurate at short horizons.

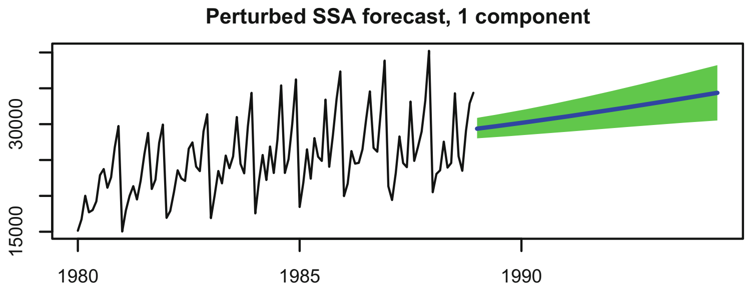

Fig. 3.17 "Total": Perturbed forecasting intervals, ET1

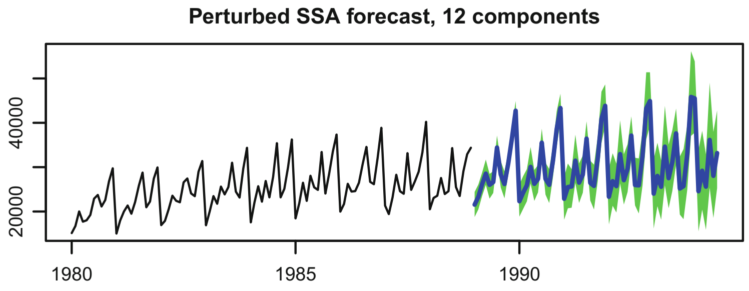

Fig. 3.18 "Total": Perturbed forecasting intervals, ET1–12

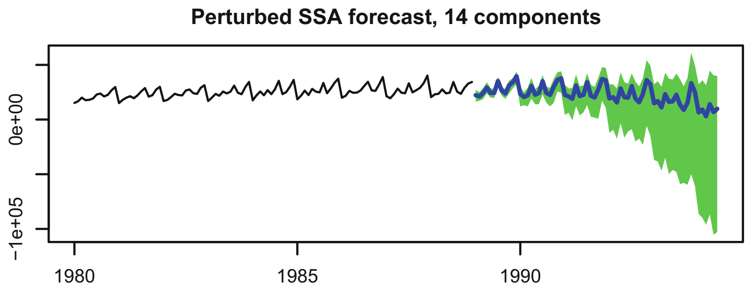

Fig. 3.19 "Total": Perturbed forecasting intervals, ET1–14

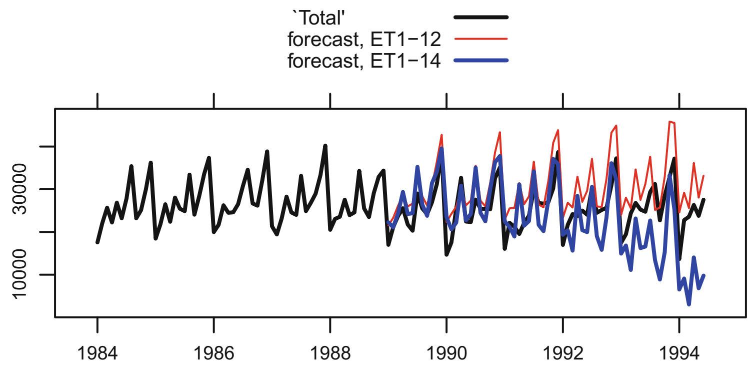

Fig. 3.20 "Total": Comparison of forecasts by ET1–12 and ET1–14

3.5.4 *Gap Filling*

Let us consider the data "Glonass" provided by the satellite navigation system Glonass (the investigated data `igs<wwww><d>.clk.Z` contains final corrections of time in the format Clock_RINEX obtained in IGS). Data are presented with the step 5 min (300 s) so that any 24-h period consists of 288 points. This data can be used for correcting future time values. However, the data contain gaps. Trends of corrections can be of different form. In Fragment 3.5.6, we demonstrate how RSSA can help for data imputation on a simple example with an almost linear trend. Data consists of 104832 points, taken from 02/01/2014 to 31/12/2014, the GLONASS satellite number 15.

Fragment 3.5.6 ("Glonass": Gap Filling)

```
> data("g15", package = "ssabook")
> xyplot(g15 ~ 1:length(g15), type = "l",
+        ylab = NULL, xlab = NULL)
> range1 <- 14950:15050
> g15_short <- g15[range1]
> g15_un <- na.omit(as.vector(g15))
> g15_un_short <- g15_un[range1]
> p1 <- xyplot(g15_short ~ range1, type = "l",
+              ylab = NULL, xlab = NULL)
> p2 <- xyplot(g15_un_short ~ range1, type = "l",
+              ylab = NULL, xlab = NULL)
> plot(p1, split = c(1, 1, 2, 1), more = TRUE)
> plot(p2, split = c(2, 1, 2, 1), more = FALSE)
> L <- 72
> neig <- min(L, 100)
> s <- ssa(g15, L = 72, neig = neig)
> plot(s, type = "vectors", idx = 1:8, plot.contrib = FALSE)
> g <- gapfill(s, groups = list(1:2))
> xyplot(g[range1] + g15[range1] ~ range1, type = "l",
+        ylab = NULL, xlab = NULL,
+         par.settings = simpleTheme(col = c("red", "black")))
> spec.pgram(g15_un, detrend = FALSE, log = "no",
+            xlim = c(0.00, 0.02), ylim = c(0, 1e-14),
+            main = "", sub = "")
> axis(1, at = c(1/144, 1/72), labels = c("1/144", "1/72"),
+      las = 2)
> spec.pgram(as.vector(g), detrend = FALSE, log = "no",
+            xlim = c(0.00, 0.02), ylim = c(0, 1e-14),
+            main = "", sub = "")
> axis(1, at = c(1/144, 1/72), labels = c("1/144", "1/72"),
+      las = 2)
```

The whole time series is depicted in Fig. 3.21. It contains several gaps; the total number of missing points is 895.

Let us consider two ways for dealing with missing data: suppressing it (simply by removing time points with missing observations and thus reducing the sample size) or properly treating it as missing and imputing the missing observations. A part of the data in these two cases is depicted in Fig. 3.22.

We start with filling-in the gaps and then show why suppressing the missing data is a wrong strategy, at least for this data set.

Since the time series is very long and the missing data has several compact locations, we will use the subspace-based method of gap filling, which is implemented in the function `gapfill`. To cover by the window all the points of the time series, we

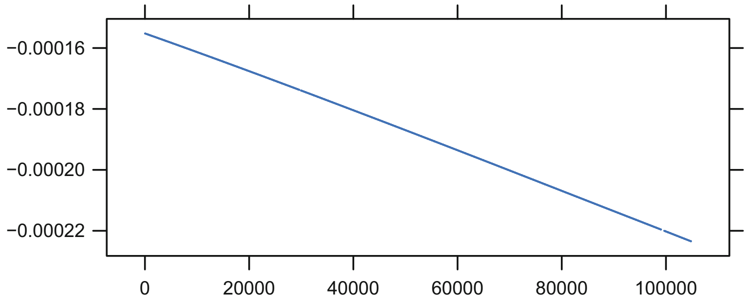

Fig. 3.21 "Glonass": Initial series with gaps

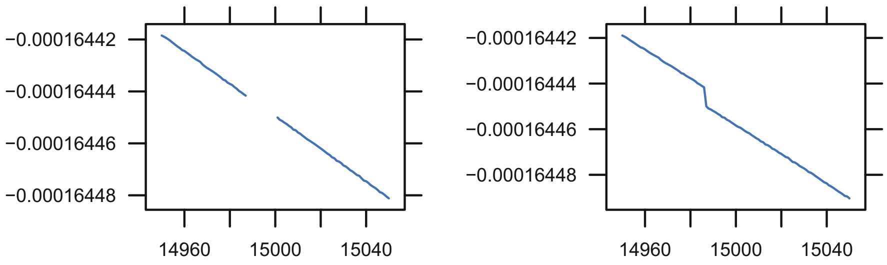

Fig. 3.22 "Glonass": A subseries with a gap (left) and with the suppressed gap (right)

will take a moderate window length L equal to 120. Figure 3.23 shows the leading eight eigenvectors. One can see that the first two eigenvectors correspond to a linear trend. Let us implement the gaps on the base of ET1,2. Figure 3.24 demonstrates imputation for one of the gaps. Certainly, for a trend as simple as this one, many methods can impute the gaps. An advantage of SSA is that the method does not use any trend model and therefore can be applied to trends of other shapes in exactly the same way.

Periodograms of the series with suppressed gaps (Fig. 3.25) and with filling-in gaps (Fig. 3.26) clearly demonstrate that the suppression of gaps corrupts the trend and hides periodicities, while the time series with properly filled gaps is

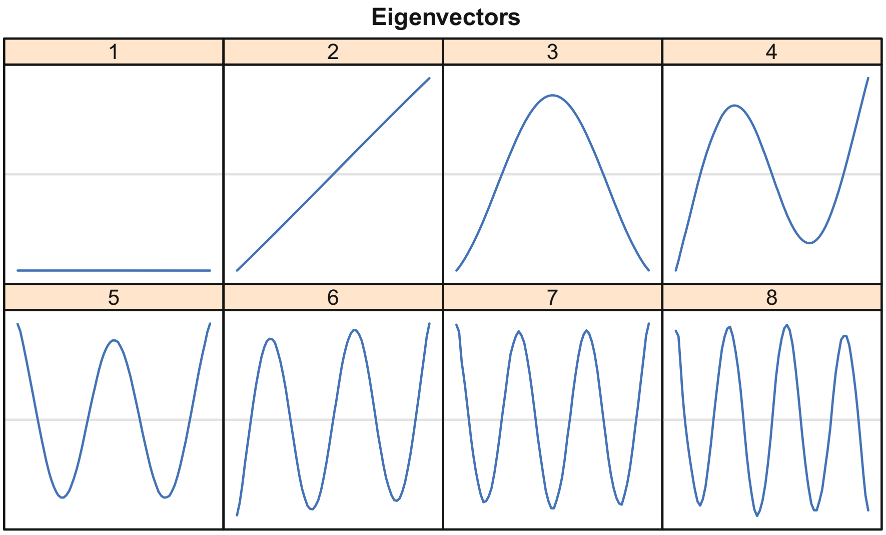

Fig. 3.23 "Glonass": Eigenvectors for the series with gaps, $L = 72$

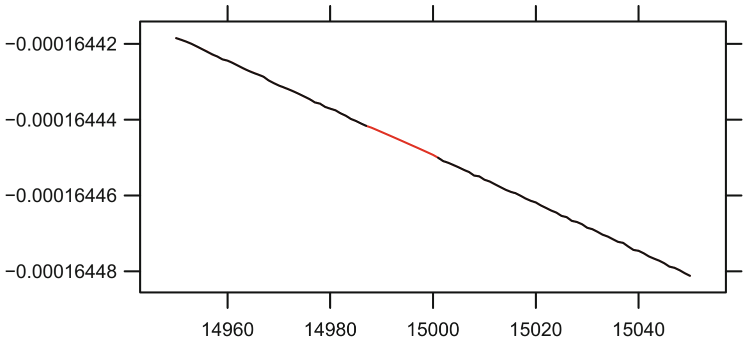

Fig. 3.24 "Glonass": A subseries with the filled gap

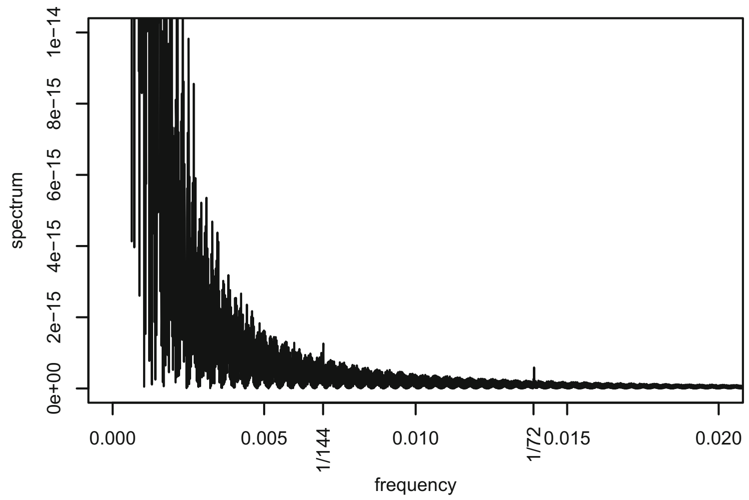

Fig. 3.25 "Glonass": Periodogram of the series with suppressed gaps

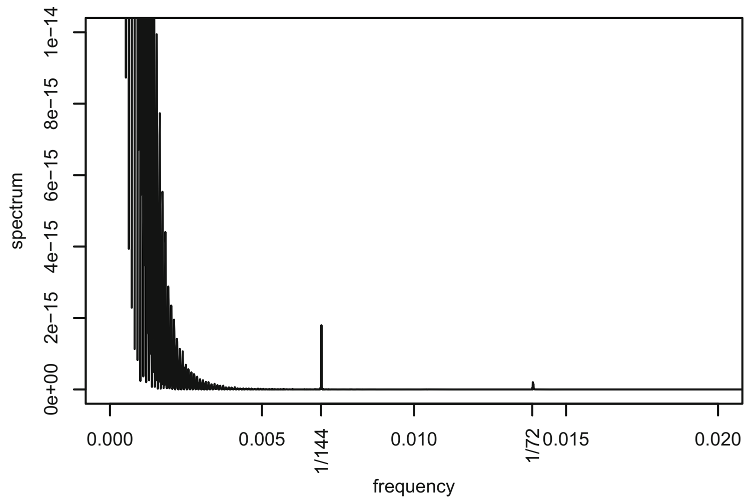

Fig. 3.26 "Glonass": Periodogram of the series with filled gaps

well-structured. Figure 3.26 shows that there are strong peaks at frequencies 1/144 and 1/72, which corresponds to a 12-h periodicity.

Let us extract the periodicity (Fragment 3.5.7) from the obtained time series with no gaps. Since the trend is simple, we take a large window length $L = 52416$ equal to the half of the time series length and divisible by 288. As a guess, we use the period estimates obtained from all pairs of eigenvectors with numbers $(i, i + 1)$.

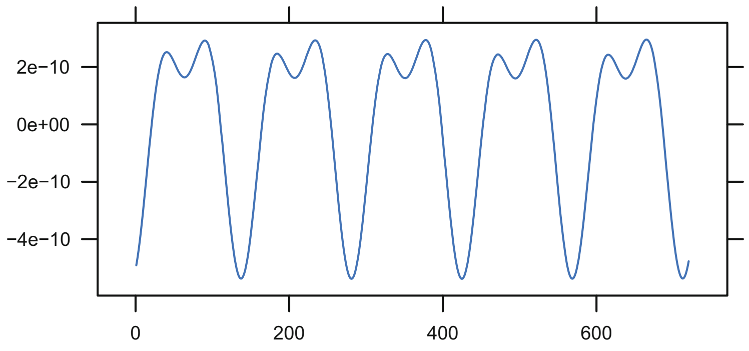

Fig. 3.27 "Glonass": A subseries with the 12-h periodicity; it is extracted from the series with filled gaps and $L = 52416$

The pairs (32, 33) and (59, 60), where the estimated period is smaller than 300, have the expected periods 144 and 72, approximately.

Figure 3.27 demonstrates the extracted 12-h periodicity.

Fragment 3.5.7 ("Glonass": Periodicity Extraction After the Gap Filling)

```
> s_filled = ssa(g, L = 52416, neig = 100)
> pg <- vector(length = 99)
> for (i in 1:99) {
+   pg[i] <- parestimate(s_filled, groups = list(i:(i + 1)),
+                        method = "esprit")$period[1]
+ }
> print(ind <- which(pg < 1/0.003))
[1] 32 58
> print(pg[ind], digits = 0)
[1] 144  72
> r <- reconstruct(s_filled, groups = list(day = c(ind, ind+1)))
> xyplot(r$day[1:720] ~ 1:720, type = "l",
+        ylab = NULL, xlab = NULL)
```

3.5.5 *Parameter Estimation and Low-Rank Approximation*

Low-rank approximation works well if the signal is governed by an LRR exactly. Such kind of signals, e.g., a sum of modulated sinusoids, is a common case in engineering.

To demonstrate the method, we choose a simple series "FORT" from the dataset "AustralianWine," which is very similar to a noisy signal of finite rank. First, let us apply the Cadzow(α) method described in Sect. 3.4 (see Fragment 3.5.8). In Zvonarev and Golyandina (2017), the value $\alpha = 0.2$ is recommended for the

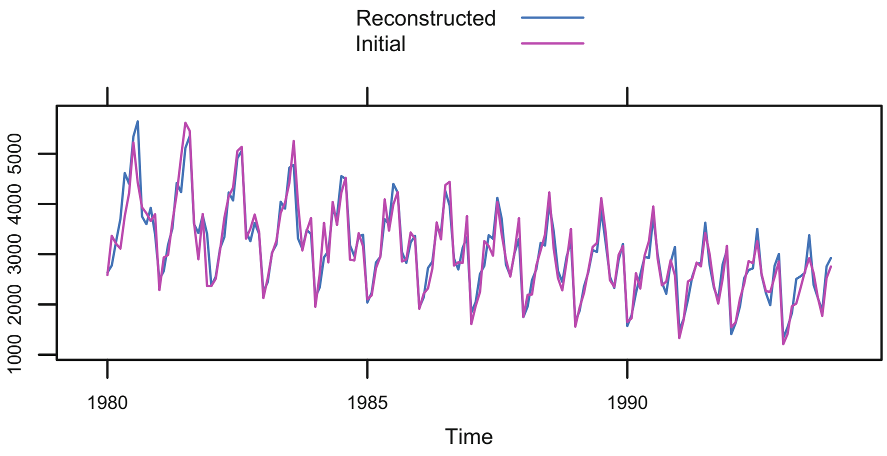

Fig. 3.28 "FORT": Approximation by a series of finite rank

reconstruction of the signal in "FORT." The result of the Cadzow(0.2) iterations is depicted in Fig. 3.28.

Fragment 3.5.8 ("FORT": Cadzow Iterations)

```
> wine <- window(AustralianWine, end = time(AustralianWine)[168])
> ser <- wine[, "Fortified"]
> N <- length(ser)
> L <- 84
> K <- N - L + 1
> rank <- 11
> # Basic SSA
> s0 <- ssa(ser, L = L)
> r0 <- reconstruct(s0, groups = list(signal = 1:rank))$signal
> # Cadzow iterations with series weights close to equal.
> alpha <- 0.1
> weights <- numeric( K)
> weights[1:K] <- alpha
> weights[seq(from = K, to = 1, by = -L)] <- 1
> s <- ssa(ser, L = L, column.oblique = "identity",
+          row.oblique = weights, decompose.force = FALSE)
> c <- cadzow(s, rank = rank)
> sc <- ssa(c, L = rank + 1)
> rc <- reconstruct(sc, groups = list(signal = 1:rank))$signal
> xyplot(cbind(rc, ser), type = "l",
+        superpose = TRUE,
+        auto.key = list(text = c("Reconstructed",
+                                  "Initial"),
+                        type = c("l", "l"),
+                        lines = TRUE, poinwts = FALSE))
```

Let us describe how an explicit form of the estimated signal can be obtained. We consider two cases: (a) estimation of the signal parameters by means of one Cadzow iteration (this iteration coincides with the Basic SSA reconstruction) in Fragment 3.5.9, and (b) estimation of the signal parameters using a finite-rank approximation in Fragment 3.5.10.

To estimate r signal roots of a signal of rank r, it is sufficient to take the window length equal to $r + 1$. Thus, we apply Basic SSA to the limit series of the Cadzow iterations and then call the function `parestimate` for the group consisting of ET1–r. Since we apply Basic SSA to a noisy signal, we should take a large window length L to separate the signal from noise. The results are quite similar for different choices of L, as long as L is large enough. Negative periods are calculated formally as $1/\pm\omega$.

Fragment 3.5.9 ("FORT": Estimation of Parameters by Basic SSA)

```
> # Estimation by means of the first iteration of Cadzow
> # iterations (SSA)
> par <- parestimate(s0, groups = list(1:rank),
+                    method = "esprit")
> print(par)
   period      rate   |    Mod      Arg  |     Re        Im
    5.972   0.004238  | 1.00425    1.05  |  0.49781   0.87218
   -5.972   0.004238  | 1.00425   -1.05  |  0.49781  -0.87218
    2.388   0.001744  | 1.00175    2.63  | -0.87403   0.48945
   -2.388   0.001744  | 1.00175   -2.63  | -0.87403  -0.48945
    4.000   0.000359  | 1.00036    1.57  | -0.00008   1.00036
   -4.000   0.000359  | 1.00036   -1.57  | -0.00008  -1.00036
      Inf  -0.003318  | 0.99669   -0.00  |  0.99669  -0.00000
   12.006  -0.005931  | 0.99409    0.52  |  0.86104   0.49680
  -12.006  -0.005931  | 0.99409   -0.52  |  0.86104  -0.49680
    3.015  -0.011285  | 0.98878    2.08  | -0.48570   0.86127
   -3.015  -0.011285  | 0.98878   -2.08  | -0.48570  -0.86127
> o <- order(abs(par$periods), decreasing = TRUE)
> periods <- (par$periods[o])
> moduli <- par$moduli[o]
> len <- rank
> vars <- matrix(nrow = len, ncol = rank)
> for (i in 1:rank) {
+   if (periods[i] == Inf)
+     vars[, i] <- moduli[i]^(1:len)
+   else if (periods[i] == 2)
+     vars[, i] <- (-moduli[i])^(1:len)
+   else if (periods[i] > 0)
+     vars[, i] <-
+       moduli[i]^(1:len) * sin(2 * pi * (1:len) / periods[i])
+   else
+     vars[, i] <-
+       moduli[i]^(1:len) * cos(2 * pi * (1:len) / periods[i])
+ }
> lm0 <- lm(r0[1:len] ~ 0 + ., data = data.frame(vars))
> coefs0 <- coef(lm0)
> print(round(coefs0[1:6], digits = 2))
```

```
     X1      X2      X3      X4      X5      X6 
3969.23 -717.10 -927.57  105.52  137.98 -287.11 
> print(round(coefs0[7:11], digits = 2))
     X7      X8      X9     X10     X11 
 215.64 -254.51 -205.12   90.44   10.95 
```

Fragment 3.5.10 ("FORT": Estimation of Parameters by Cadzow Iterations)

```
> # Estimation by means of the limit of Cadzow iterations
> parc <- parestimate(sc, groups = list(1:rank),
+                     method = "esprit")
> print(parc)
   period     rate   |    Mod     Arg  |     Re        Im
    5.975   0.004986 |  1.00500   1.05 |  0.49863   0.87258
   -5.975   0.004986 |  1.00500  -1.05 |  0.49863  -0.87258
    2.389   0.003402 |  1.00341   2.63 | -0.87506   0.49102
   -2.389   0.003402 |  1.00341  -2.63 | -0.87506  -0.49102
    3.998   0.000311 |  1.00031   1.57 | -0.00076   1.00031
   -3.998   0.000311 |  1.00031  -1.57 | -0.00076  -1.00031
      Inf  -0.003356 |  0.99665   0.00 |  0.99665   0.00000
   12.009  -0.005985 |  0.99403   0.52 |  0.86105   0.49669
  -12.009  -0.005985 |  0.99403  -0.52 |  0.86105  -0.49669
    3.018  -0.010620 |  0.98944   2.08 | -0.48394   0.86301
   -3.018  -0.010620 |  0.98944  -2.08 | -0.48394  -0.86301
> oc <- order(abs(parc$periods), decreasing = TRUE)
> periodsc <- (parc$periods[o])
> modulic <- parc$moduli[o]
> lenc <- rank
> varsc <- matrix(nrow = lenc, ncol = rank)
> for (i in 1:rank) {
+   if (periodsc[i] == Inf)
+     varsc[, i] <- modulic[i]^(1:lenc)
+   else if (periodsc[i] == 2)
+     varsc[, i] <- (-modulic[i])^(1:lenc)
+   else if (periodsc[i] > 0)
+     varsc[, i] <-
+       modulic[i]^(1:lenc) * sin(2 * pi * (1:lenc) / periodsc[i])
+   else
+     varsc[, i] <-
+       modulic[i]^(1:lenc) * cos(2 * pi * (1:lenc) / periodsc[i])
+ }
> lm.c <- lm(rc[1:lenc] ~ 0 + ., data = data.frame(varsc))
> #lm.c
> coefs.c <- coef(lm.c)
> print(round(coefs.c[1:6], digits = 2))
     X1      X2      X3      X4      X5      X6 
4005.56 -721.77 -940.64   68.30  184.45 -269.52 
> print(round(coefs.c[7:11], digits = 2))
     X7      X8      X9     X10     X11 
 325.92 -251.10 -255.41  154.28   61.90 
```

By looking at the signal root estimates, we suggest the following model for the signal:

$$\begin{aligned} s_n &= C_0\rho_0^n + \sum_{k=1}^{5} C_k\rho_k^n \sin\left(\frac{2\pi n}{T_k} + \phi_k\right) \\ &= C_0\rho_0^n + \sum_{k=1}^{5} \rho_k^n \left(A_k \sin\left(\frac{2\pi n}{T_k}\right) + B_k \cos\left(\frac{2\pi n}{T_k}\right)\right). \end{aligned} \tag{3.14}$$

Results of `parestimate` are the estimates of ρ_k and T_k in (3.14). To estimate C_k and ϕ_k, one can first estimate A_k and B_k by the least-squares method and then find $C_k = \sqrt{A_k^2 + B_k^2}$, $\phi_k = \arctan(B_k/A_k)$ (Fragment 3.5.11). Note that for Basic SSA we take the whole time series to estimate the coefficients, while in the case of series of finite rank it is sufficient to take any r sequential series points to find r unknown parameters.

Fragment 3.5.11 ("FORT": Estimation of Parametric Real-Valued Form)

```
> idx <- seq(2, 11, 2)
> coefs.c.phase <- numeric(length(idx))
> phases.c <- numeric(length(idx))
> periods.c.phase <- numeric(length(idx))
> moduli.c.phase <- numeric(length(idx))
> for (i in seq_along(idx)) {
+   periods.c.phase[i] <- periodsc[idx[i]]
+   moduli.c.phase[i] <- modulic[idx[i]]
+   coefs.c.phase[i] <- sqrt(coefs.c[idx[i]]^2 +
+                            coefs.c[idx[i] + 1]^2)
+   phases.c[i] <- atan2(coefs.c[idx[i] + 1], coefs.c[idx[i]])
+ }
> print("trend:")
[1] "trend:"
> print("coefficient * modulus^n")
[1] "coefficient * modulus^n"
> print(data.frame(coefficients = coefs.c[1],
+                  moduli = modulic[1]))
  coefficients    moduli
X1     4005.561 0.9966493
> print("periodics:")
[1] "periodics:"
> print("coefficient * modulus^n * cos(2 * pi* n/period + phase)")
[1] "coefficient * modulus^n * cos(2 * pi* n/period + phase)"
> print(data.frame(periods = periods.c.phase, phases = phases.c,
+                  coefficients = coefs.c.phase,
+                  moduli = moduli.c.phase))
    periods    phases coefficients    moduli
1 12.008804 -2.225294    1185.6458 0.9940328
2  5.974637  1.216171     196.6835 1.0049988
3  3.998061  2.261753     422.9253 1.0003111
4  3.018060 -2.347688     358.1681 0.9894363
5  2.388825  0.381569     166.2372 1.0034081
```

3.5.6 Subspace Tracking

Let us consider the problem of finding changes in a time series by the SSA subspace tracking. There are many algorithms for change-point detection in time series, see Basseville et al. (1993), Tartakovsky et al. (2014). We advocate SSA for change-point detection and structure monitoring. The principal technique of using SSA for sequential detection of structural changes was developed in Moskvina and Zhigljavsky (2003); in what follows, we pursue the approach thoroughly described in Golyandina et al. (2001; Chapter 3). We take the annual sunspots data 1700–2015 and try to find changes in the oscillations. Trend needs to be extracted as a preprocessing step. Trend extraction and further analysis are performed in Fragment 3.5.12.

Fragment 3.5.12 ("Sunspots": Subspace Tracking)

```
> data("sunspot2", package = "ssabook")
> s <- ssa(sunspot2, L = 11)
> r <- reconstruct(s, groups = list(Trend = 1))
> plot(r, plot.method = "xyplot", superpose = TRUE)
> sun.oscill <- residuals(r)
> N <- length(sun.oscill)
> rank <- 2
> periods <- function(M, L) {
+   ts(sapply(1:(N - M),
+             function (i) {
+               s <- ssa(sun.oscill[i:(i + M - 1)], L = L)
+               par <- parestimate(s, groups = list(c(1:rank)),
+                                  method = "esprit")
+               abs(par$periods[1])
+             }),
+      start = time(sunspot2)[M + 1], delta = 1)
+ }
> per22 <- periods(22, 11)
> per44 <- periods(44, 22)
> xyplot(cbind(per22, per44), type = "l", xlim = c(1677, 2040),
+        strip = strip.custom(factor.levels =
+                              c("B = 22", "B = 44")))
> M <- 22; L <- M / 2
> hm <- hmatr(sun.oscill, B = M, T = M, L = L, neig = rank)
> plot(hm)
> M <- 44; L <- M / 2
> hm <- hmatr(sun.oscill, B = M, T = M, L = L, neig = rank)
> plot(hm)
```

Figure 3.29 (top) contains the extracted trend and the residual. We choose the window length $L = 11$ as approximately equal to the main period. This choice corresponds to the extraction of a more detailed trend, which can be reconstructed by the leading eigentriple.

Further we consider the residual. Let us check the behavior of the main frequency. First, we consider the sliding subseries $\mathbb{X}_n = (x_{n-B+1}, \ldots, x_n)$, $n = B, \ldots N$, with

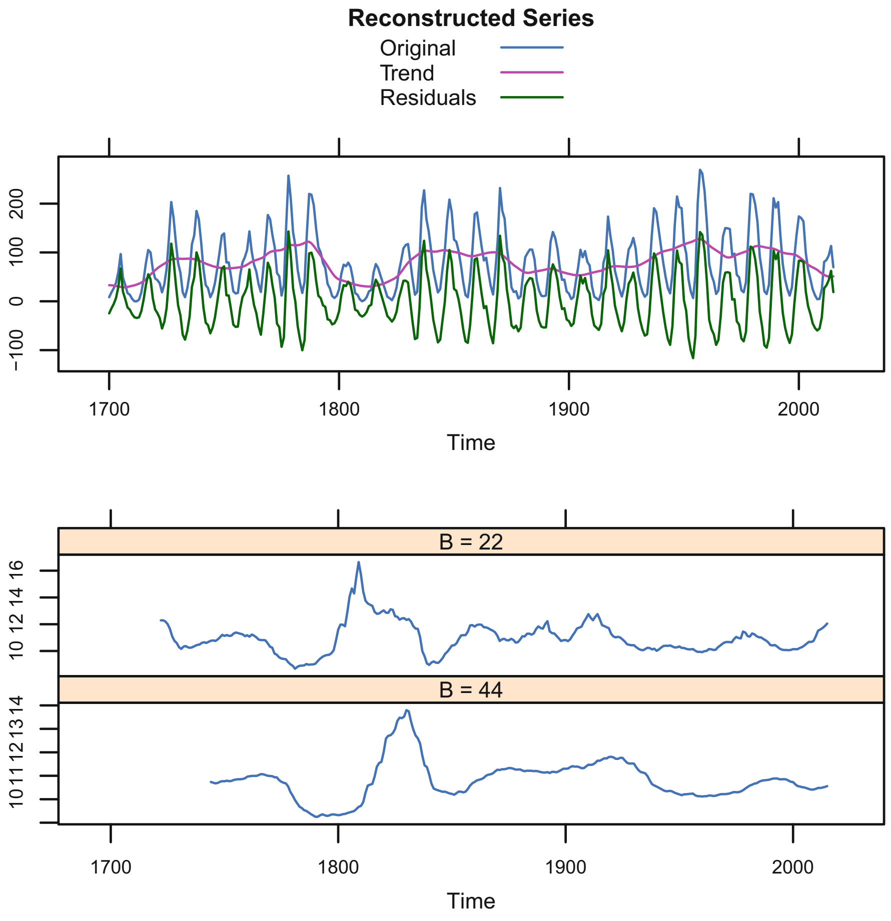

Fig. 3.29 “Sunspots”: Trend extraction (top), subspace tracking of residuals with $B = 22$ (middle) and $B = 44$ (bottom)

$B = 22$ and $B = 44$, choosing window length L equal to $B/2$. Then, we apply SSA to each series $\mathbb{X}_n$, find the subspace produced by the first two eigenvectors and estimate the main period P_n by the LS-ESPRIT method. Graphs of periods P_n as functions of n are shown in Fig. 3.29 (middle and bottom). One can see that the period is oscillating around $P = 11$. However, after 1800 we see that the period sharply increases. This corresponds to small values of sun activities (Fig. 3.29, top). If we use the tracking of frequencies for a-priori change detection, then we can clearly see that the delay for $B = 44$ (bottom) is larger than that for $B = 22$ (middle).

Let us apply the techniques suggested in Golyandina et al. (2001; Chapter 3) for a visualization of a-posteriori change-point detection. Visual change-point detection can be performed by means of the so-called heterogeneity matrix. The

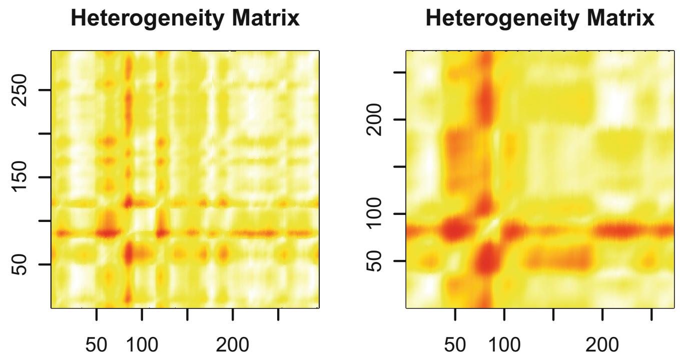

Fig. 3.30 "Sunspots": Heterogeneity matrices $B = 22$ (left) and $B = 44$ (right)

rows correspond to sliding base subseries of length B. We chose the same two values, $B = 22$ and $B = 44$. Each subseries produces a subspace, which is exactly the subspace used for frequency tracking. The columns correspond to sliding test subseries of length T. We take $T = B$. Each test series produces a set of L-lagged vectors. The heterogeneity index is defined as the sum of distances from the L-lagged vectors of the test series to the base subspace, see (3.1) in Golyandina et al. (2001). Large values of the heterogeneity index correspond to large dissimilarity between the test and base subseries. In Fig. 3.30, large values of the heterogeneity index are depicted by red color, while small values of this index are depicted by white and yellow colors. If we consider rows or columns of the heterogeneity matrix, then we see that there is a change-point starting from the subseries $(x_{80}, \ldots, x_{80+B-1})$. This corresponds to the same change at around year 1800, which we have found by the frequency tracking. Note that after the subseries passes the intervals that include the change-point, the values of the heterogeneity index become smaller again. This means that the change in the series was temporal.

Fast algorithms of subspace tracking were developed in many papers (e.g., Real et al. (1999), Badeau et al. (2004), among others), since it can be expected that the subspace estimate for the $(n + 1)$th subseries can be calculated more effectively if one uses the information obtained at the previous nth step. However, our experience shows that despite the fast algorithms implemented in the RSSA and SVD packages cannot be improved on the base of the use of previously constructed subspaces, these implementations of RSSA and SVD can be faster than the improved conventional algorithms.

3.5.7 *Automatic Choice of Parameters for Forecasting*

Since SSA can be applied without requiring validity of any model for the time series, the choice of parameters should be non-specific. The most conventional model-free way of parameter choice is the minimization of forecasting errors within the validation (training) period. This approach can be applied if the time series length is large enough.

Fragment 3.5.13 contains the code of functions, which may help in finding the optimal parameters (which are the window length and the number of leading components). The function `forecast.mse` performs forecasting on the base of a given `ssa` object `x` and calculates the mean square of the difference between the forecast and a given time series `F.check`. The function `forecast.sliding.mse` call `forecast.mse` for sliding subseries, given set of window lengths L, and set of numbers of components *ncomp* used for forecasting. The function `forecast.mse` is designed to be applied to one series (subseries), one window length, one number of leading components. In `forecast.sliding.mse`, `forecast.mse` is applied to many (`K.sliding`) subseries of a given series, many window lengths (stored in the vector `L`), many numbers of components (stored in the vector `ncomp`), which is done in an effective manner. In this way, we obtain a 3D array of MSE errors. Note that the number of SSA decompositions is equal to the number of sliding windows `K.sliding` multiplied by the number of window lengths.

Finally, the function `optim.par` on the base of this 3D array calculates the matrix of mean MSE errors, which are obtained by computing the average of the MSE errors corresponding to different sliding subseries, and finds the optimal window length and the number of components, which correspond to the minimal mean MSE. The matrix of mean MSE errors provides a possibility to plot the dependence of accuracy as a function of parameters and hence to check if this accuracy is stable (that is, does not change much) in the chosen range of parameters.

To use the function `optim.par`, the user should choose the length of the validation period. This validation period may correspond to the whole series. In the example considered (see Fragment 3.5.13) the whole series is divided into training and test periods to check the forecast accuracy.

Fragment 3.5.13 (Functions for the Search of Optimal Parameters)

```
> library("plyr")
> forecast.mse <- function(x, F.check,
+                          forecast.len = 1, ...) {
+   stopifnot(length(F.check) == forecast.len)
+   f <- forecast(x, h = forecast.len, ...)
+   mean((f$mean - F.check)^2)
+ }
> forecast.sliding.mse <- function(F,
+                                  L, ncomp,
+                                  forecast.len = 1,
+                                  K.sliding = N %/% 4,
+                                  .progress = "none",
+                                  .parallel = FALSE,
+                                  ...) {
+   N <- length(F)
+   sliding.len <- N - K.sliding - forecast.len + 1
+   L.max = max(L); L.min = min(L); ncomp.max = max(ncomp)
+   stopifnot(sliding.len > L.max)
+   stopifnot(ncomp.max + 1 < min(L.min, N - L.max + 1))
+   g <- expand.grid(L = L, i = 1:K.sliding)
+   aaply(g, 1,
+         splat(function(L, i) {
+             F.train <- F[seq(from = i, len = sliding.len)]
+             F.check <- F[seq(from = sliding.len + i,
+                              len = forecast.len)]
+             s <- ssa(F.train, L = L)
+             sapply(ncomp,
+                    function(ncomp) {
+                        res <- forecast.mse(s, F.check,
+                                            forecast.len =
+                                              forecast.len,
+                                            groups =
+                                              list(1:ncomp),
+                                            ...)
+                        names(res) <- as.character(ncomp)
+                        res
+                    })
+         }),
+         .progress = .progress, .parallel = .parallel)
+ }
> optim.par <- function(m0) {
+   m <- apply(m0, c(1, 3), mean)
+   mpos <- which(m == min(m), arr.ind = TRUE)
+   L.opt <- Ls[mpos[1]]
+   ncomp.opt <- ncomp[mpos[2]]
+   list(L.opt = L.opt, ncomp.opt = ncomp.opt, m = m)
+ }
```

Fragment 3.5.14 shows how this function is applied to “Bookings” data to obtain optimal parameters. The data contains the numbers of hourly hotel bookings through a particular web-site during 6 months. We can expect the main period of the data to be equal to $168 = 24 \cdot 7$ (`frequency` of the data is equal to 168 observations per week). We forecast the series for 2 weeks. To find the parameters, we minimize the RMSE errors of two forecasts for $336 = 2 \cdot 168$ steps ahead (`K.sliding = 2`).

Fragment 3.5.14 (“Bookings”: Search for Optimal Parameters)

```
> data("bookings", package = "ssabook")
> K.sliding <- 2
> forecast.base.len <- 2*frequency(bookings)
> base.len <- length(bookings)
> sliding.len <- base.len - K.sliding - forecast.base.len + 1
> print(sliding.len)
[1] 4007
> ncomp <- 1:100
> L.min <- frequency(bookings)
> Ls <- seq(L.min, 10*L.min, by = frequency(bookings))
> m0 <- forecast.sliding.mse(bookings,
+                            K.sliding = K.sliding,
+                            L = Ls, ncomp = ncomp,
+                            method = "recurrent",
+                                  forecast.len = forecast.base.len,
+                                  .progress = "none")
> p <- optim.par(m0)
> print(c(p$L.opt, p$ncomp.opt, sqrt(min(p$m))))
[1] 504.0000  90.0000 116.0134
> matplot(Ls, sqrt(p$m), ylab = "", xlab = "Window lengths",
+          type = "l", col = topo.colors(100))
```

The dependence of RMSE errors on window lengths for different number of components r chosen for decomposition is depicted in Fig. 3.31. Colors are changing from blue to yellow; this corresponds to values of r from 1 to 100. The optimal window length us equal to 504, the optimal number of components is equal to 90.

In Fragment 3.5.15, the forecast is constructed with the parameters found. The forecasts are shown in Fig. 3.32 for the whole series. One can see that the series has some irregularities. The last points are depicted in Fig. 3.33.

Fragment 3.5.15 ("Bookings": Forecast with Optimal Parameters)

```
> forecast.len <- 2*frequency(bookings)
> ssa.obj <- ssa(bookings, L = p$L.opt)
> ssa.for <- rforecast(ssa.obj, groups = list(1:p$ncomp.opt),
+                        len = forecast.len)
> xyplot(cbind(bookings, ssa.for),
+         type = "l", superpose = TRUE)
> xyplot(cbind(bookings, ssa.for), type = "l",
+         superpose = TRUE, xlim = c(21,29))
```

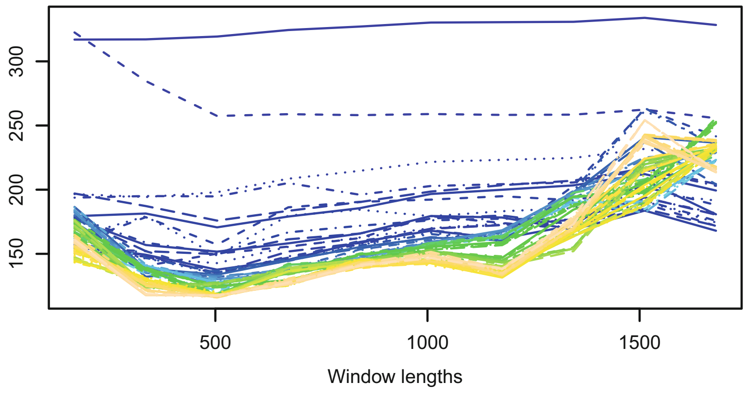

Fig. 3.31 "Bookings": Dependence of RMSE on L for different numbers of components

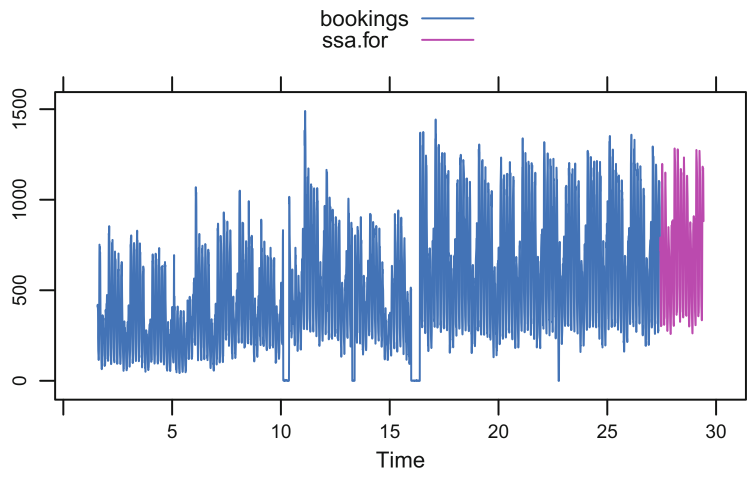

Fig. 3.32 “Bookings”: Forecast with optimal parameters

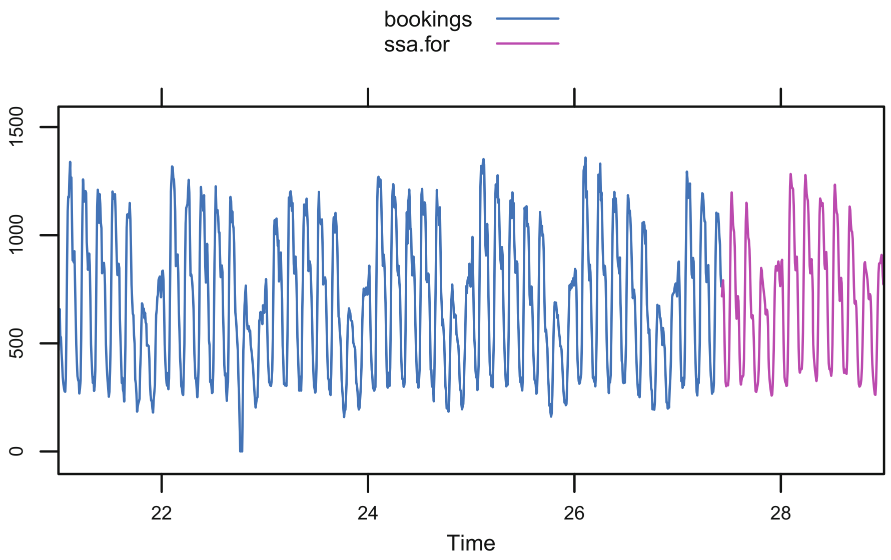

Fig. 3.33 “Bookings”: Forecast with optimal parameters for last points

3.5.8 Comparison of SSA, ARIMA, and ETS

As was mentioned in Sect. 1.6.3, real-world time series do not satisfy any model exactly. Therefore, comparing application of ARIMA and SSA with totally different models, we compare approximations by these models.

Seasonal ARIMA and exponential smoothing (ETS) methods correspond to concrete model families. In particular, the frequency of the periodic component (e.g., seasonality) should be specified. Therefore, different information criteria are developed for these models. The idea of these criteria is to use some measure of correspondence between the model and the time series and then adjust it by the number of parameters in the model. In real-world problems, these information criteria can be formally applied for the choice of model from the corresponding family.

For SSA, we will use the approach described in Sect. 3.5.7. Since the minimized forecasting errors are calculated for series points which do not participate in the construction of the forecasts, this procedure partly avoids over-fitting. Minimization of reconstruction errors is senseless, since the minimum can be reached by over-fitting (the larger is the number of the reconstructed components, the smaller is the reconstruction error).

ARIMA and ETS models provide theoretical prediction intervals for the whole series. SSA provides bootstrap confidence intervals for the signal and bootstrap prediction for the whole series in the model "signal + noise," see Sect. 3.2.1.5. For comparability, we will consider prediction intervals for all considered methods. To compare accuracy of the methods, we consider the mean squared difference between the series and mean forecasts for ARIMA and ETS and the mean squared difference between the series and signal forecasts for SSA.

Let us consider the series "Sweetwhite" from the dataset "AustralianWine." This time series contains monthly sales of sweet white wines and has a changing structure. Therefore, it does not suit any model. We divide the series into two parts: training (base) and test ones (Fragment 3.5.16). Models will be constructed by means of the training part, while the methods will be compared by the forecasting of the test part values.

Fragment 3.5.16 ("Sweetwhite": Training and Test Periods)

```
> name <- "Sweetwhite"
> wine <- window(AustralianWine, end = time(AustralianWine)[174])
> series <- wine[, name]
> set.seed(1)
> forecast.len <- 12
> base.len <- length(series) - forecast.len
> F.base <- window(series,
+                  end = time(series)[base.len]) # training
> F.new <- window(series,
+                 start = time(series)[base.len + 1]) # test
```

Fragment 3.5.17 contains the code, which seeks the parameters of SSA, the window length and the number of eigentriples for forecasting. Similar to the example in Sect. 3.5.7, the parameters correspond to minimal MSE forecasting errors on the training part. Since the series is short enough and has a changing behavior, we minimize the mean MSE of 12 two-step ahead forecasts. The obtained values are $L = 108$, $r = 8$.

Fragment 3.5.17 ("Sweetwhite": Search for SSA Parameters)

```
> K.sliding <- 12
> forecast.base.len <- 2
> ns <- base.len - K.sliding - forecast.base.len + 1
> ncomp <- 1:15
> L.min <- 24
> Ls <- seq(L.min, ns - L.min, by = 12)
> method <- "recurrent"
> m0 <- forecast.sliding.mse(F.base, K.sliding = K.sliding,
+                            L = Ls, ncomp = ncomp,
+                            method = method,
+                            forecast.len = forecast.base.len)
> p <- optim.par(m0)
> print(c(p$L.opt, p$ncomp.opt, sqrt(min(p$m))))
[1] 108.00000   8.00000  24.80634
> # These parameters provides the best forecast
> # L.opt <- 132; ncomp.opt <- 13
```

In Fragment 3.5.18, the forecasts are constructed and the methods are compared by RMSE. Since the parameters of SSA were constructed by forecasting of sliding shortened time series of length 149, we will use the last 149 points of the base series to construct the forecast for comparison. One can see that the forecast accuracy is approximately the same. Since ARIMA and ETS models provides prediction intervals for the whole series, we consider the SSA prediction intervals too. Figures 3.34, 3.35, and 3.36 depict the series, the forecast (in blue color), and the prediction intervals. Note that for ARIMA forecasting the prediction intervals are very large.

Fragment 3.5.18 ("Sweetwhite": Comparison of SSA, ARIMA and ETS)

```
> # SSA forecast
> F.base.short <-
+   window(F.base, start =
+            time(series)[K.sliding + forecast.base.len])
> ssa.obj <- ssa(F.base.short, L = p$L.opt)
> ssa.for <- forecast(ssa.obj,
+                     groups = list(1:p$ncomp.opt),
+                     method = method, h = forecast.len,
+                     interval = "prediction",
+                     level=c(0.8, 0.95))
> err.ssa <- (ssa.for$mean - F.new)^2
> # ARIMA forecast
> sarima.fit <- auto.arima(F.base, trace = FALSE,
+                          lambda = 0, stepwise = FALSE)
> sarima.for <- forecast(sarima.fit, h = forecast.len)
> err.sarima <- (sarima.for$mean - F.new)^2
> # ETS forecast
> ets.fit <- ets(F.base)
> ets.for <- forecast(ets.fit, h = forecast.len)
> err.ets <- (ets.for$mean - F.new)^2
> # Models
> print(sarima.fit)
Series: F.base
ARIMA(1,1,0)(2,0,0)[12]
Box Cox transformation: lambda= 0
Coefficients:
          ar1    sar1    sar2
      -0.4165  0.4847  0.2123
s.e.   0.0722  0.0765  0.0813
sigma^2 estimated as 0.03897:  log likelihood=30.78
AIC=-53.55   AICc=-53.29   BIC=-41.22
> print(ets.fit)
ETS(M,N,M)
Call:
 ets(y = F.base)
  Smoothing parameters:
    alpha = 0.5571
    gamma = 1e-04
  Initial states:
    l = 123.5798
    s=1.4262 1.2408 1.0236 1.0546 1.1188 1.0852
           0.7795 0.8136 0.8903 0.8834 0.8241 0.8598
  sigma:  0.1732
     AIC     AICc      BIC
2026.600 2029.887 2072.913
> print(c("SSA(L,r)", p$L.opt, p$ncomp.opt))
[1] "SSA(L,r)" "108"      "8"
> # RMSE for test periods
> print(c("ssa", sqrt(mean(err.ssa))))
[1] "ssa"               "55.3572366330604"
> print(c("sarima", sqrt(mean(err.sarima))))
[1] "sarima"            "60.1534785331151"
> print(c("ets", sqrt(mean(err.ets))))
[1] "ets"               "54.2338009066311"
> # Plot of forecasts with confidence intervals
> plot(ets.for); lines(series,col="black");
> plot(sarima.for); lines(series,col="black");
> plot(ssa.for); lines(series,col="black");
```

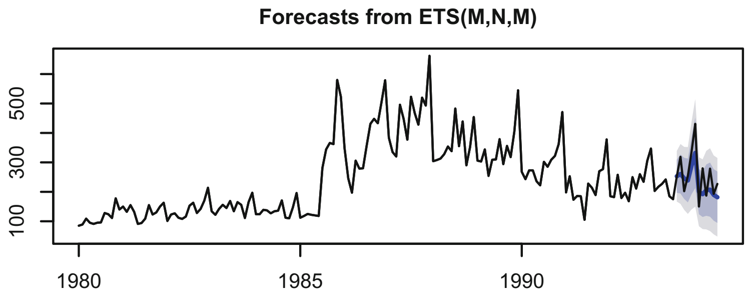

Fig. 3.34 "Sweetwhite": ETS forecast with optimal parameters

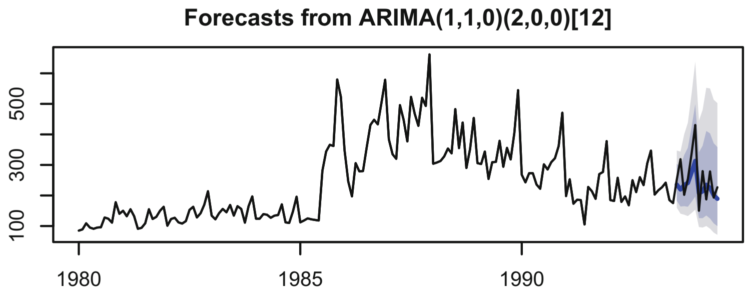

Fig. 3.35 "Sweetwhite": ARIMA forecast with optimal parameters

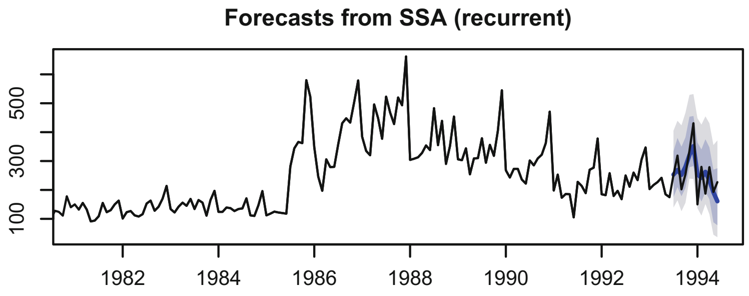

Fig. 3.36 "Sweetwhite": SSA forecast with optimal parameters

References

Badeau R, Richard G, David B (2004) Sliding window adaptive SVD algorithms. IEEE Trans Signal Process 52(1):1–10

Barkhuijsen H, de Beer R, van Ormondt D (1987) Improved algorithm for noniterative time-domain model fitting to exponentially damped magnetic resonance signals. J Magn Reson 73:553–557

Basseville M, Nikiforov IV, et al (1993) Detection of abrupt changes: theory and application, vol 104. Prentice Hall, Englewood Cliffs

Beckers J, Rixen M (2003) EOF calculations and data filling from incomplete oceanographic data sets. Atmos Ocean Technol 20:1839–1856

Cadzow JA (1988) Signal enhancement: a composite property mapping algorithm. IEEE Trans Acoust 36(1):49–62

Du K, Zhao Y, Lei J (2017) The incorrect usage of singular spectral analysis and discrete wavelet transform in hybrid models to predict hydrological time series. J Hydrol 552:44–51

Gillard J, Kvasov D (2016) Lipschitz optimization methods for fitting a sum of damped sinusoids to a series of observations. Stat Interface 10(1):59–70

Gillard J, Zhigljavsky A (2011) Analysis of structured low rank approximation as an optimization problem. Informatica 22(4):489–505

Gillard J, Zhigljavsky A (2013) Optimization challenges in the structured low rank approximation problem. J Global Optim 57(3):733–751

Gillard J, Zhigljavsky AA (2016) Weighted norms in subspace-based methods for time series analysis. Numer Linear Algebra Appl 23(5):947–967

Golyandina N, Osipov E (2007) The “Caterpillar”-SSA method for analysis of time series with missing values. J Stat Plan Inference 137(8):2642–2653

Golyandina N, Zhigljavsky A (2013) Singular spectrum analysis for time series. Springer briefs in statistics. Springer

Golyandina N, Nekrutkin V, Zhigljavsky A (2001) Analysis of time series structure: SSA and related techniques. Chapman&Hall/CRC

Golyandina N, Korobeynikov A, Shlemov A, Usevich K (2015) Multivariate and 2D extensions of singular spectrum analysis with the Rssa package. J Stat Softw 67(2):1–78

Hyndman RJ (2017) FORECAST: Forecasting functions for time series and linear models. URL http://CRAN.R-project.org/package=forecast, R package version 8.1, with contributions from Slava Razbash and Drew Schmidt

Kondrashov D, Ghil M (2006) Spatio-temporal filling of missing points in geophysical data sets. Nonlinear Process Geophys 13(2):151–159

Markovsky I (2012) Low rank approximation. Springer

Markovsky I, Usevich K (2014) Software for weighted structured low-rank approximation. J Comput Appl Math 256:278–292

Markovsky I, Willems JC, Van Huffel S, De Moor B (2006) Exact and approximate modeling of linear systems: A behavioral approach, vol 11. SIAM

Moskvina V, Zhigljavsky A (2003) An algorithm based on singular spectrum analysis for change-point detection. Commun Stat Simul Comput 32(2):319–352

Nekrutkin V (2010) Perturbation expansions of signal subspaces for long signals. Stat Interface 3:297–319

Real E, Tufts D, Cooley JW (1999) Two algorithms for fast approximate subspace tracking. IEEE Trans Signal Process 47(7):1936–1945

Rodrigues PC, de Carvalho M (2013) Spectral modeling of time series with missing data. Appl Math Modell 37(7):4676–4684

Roy R, Kailath T (1989) ESPRIT: estimation of signal parameters via rotational invariance techniques. IEEE Trans Acoust 37:984–995

Schoellhamer D (2001) Singular spectrum analysis for time series with missing data. Geophys Res Lett 28(16):3187–3190

Tartakovsky A, Nikiforov I, Basseville M (2014) Sequential analysis: Hypothesis testing and changepoint detection. CRC Press

Usevich K (2010) On signal and extraneous roots in singular spectrum analysis. Stat Interface 3(3):281–295

Van Huffel S, Chen H, Decanniere C, van Hecke P (1994) Algorithm for time-domain NMR data fitting based on total least squares. J Magn Reson Ser A 110:228–237

Zhigljavsky A, Golyandina N, Gillard J (2016a) Analysis and design in the problem of vector deconvolution. In: Kunert J, Müller HC, Atkinson CA (eds) mODa 11 - advances in model-oriented design and analysis. Springer International Publishing, pp 243–251

Zhigljavsky A, Golyandina N, Gryaznov S (2016b) Deconvolution of a discrete uniform distribution. Stat Probab Lett 118:37–44

Zvonarev N, Golyandina N (2017) Iterative algorithms for weighted and unweighted finite-rank time-series approximations. Stat Interface 10(1):5–18