Appendix A: A Short Introduction to R

A.1 Beginning with R

After R is started, the RGui windows will appear. This window has several parts, of which the R Console window is the main part of RGui where all codes must be typed, directly after the > prompt. Numerical results of calculations with R will be displayed in the R Console; however, graphical output will be depicted in the R Graphics windows. In R, the widely preferred assignment operator is an arrow made from two characters `<-`, although = can be used instead. For example typing `x<-2` will assign number `2` in name `x`. Similar to other software, there are many predefined functions in R that help users to obtain their results easily. In the next sections, several useful R functions, that will be used here, are presented. In addition, users can get help from the help menu in RGui or type `?` before the name of a command or function. For example type `?hist` in R console and press enter, and as result a page will appear in which you can find a description of the function `hist()` with relevant examples.

A.2 Types of Data in R

As in other software, logical, integer, real, complex, string (or character) and raw data can be used in R. These types of data can be defined and saved with different structures. Most applicable structures are:

© The Editor(s) (if applicable) and The Author(s) 2018

H. Hassani and R. Mahmoudvand, *Singular Spectrum Analysis*,
Palgrave Advanced Texts in Econometrics,
https://doi.org/10.1057/978-1-137-40951-5

Vectors
are one-dimensional variables with one or more values of the same type. Vectors can be defined in R by different ways such as:

```
 v1<-c(1,2,5)
 v1
[1] 1 2 5

 v2<-1:5
 v2
[1] 1 2 3 4 5

 v3<-seq(1,5,0.5)
 v3
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

 v4<-rep(c(1,5),each=2,times=3)
 v4
[1] 1 1 5 5 1 1 5 5 1 1 5 5
```

Matrices
are two-dimensional variables with one or more values of the same type. Matrices can be defined using the R function `matrix()` as in the following example:

```
 m1<-matrix(c(1,2,5,6,4,9),2,3)
 m1
     [,1] [,2] [,3]
[1,]    1    5    4
[2,]    2    6    9
```

Data frames
are similar to matrices, however they have the possibility of including different value types. The R function `data.frame()` produces the data frame, as the following example shows:

```
 sex<-c("m","m","f","m","f")
 weight<-c(2.5,3.3,2.7,4.1,2.5)
 height<-c(45,49,55,37,48)
 D<-data.frame(sex,weight,height)
 D
```

```
  sex weight height
1   m    2.5     45
2   m    3.3     49
3   f    2.7     55
4   m    4.1     37
5   f    2.5     48
```

Arrays
are one or more dimensional variables with one or more values of the same type. To create an array in R, the function `array()`, can be used as shown in the following example.

```
 a<-array(1:24,dim=c(4,3,2))
 a
, , 1

     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

Lists
are the most complete forms of data structure, which can include vectors, matrices, data frames, arrays or even other lists. A list in R gives the possibility of having a matrix or array with different lengths for rows or columns. Lists in R can be defined by the function `list()`, as shown in the following example:

```
 L1<-c(1,5,3)
 L2<-c("A","B")
 L<-list(L1,L2)
 L
```

```
[[1]]
[1] 1 5 3

[[2]]
[1] "A" "B"
```

As the output of this example shows, the components of list are numbered by `[[1]]`, `[[2]]` and so on. To determine every component, a number must be given component after the name of the list. For instance in the above example we have:

```
 L[[1]]
[1] 1 5 3
```

Note that it is possible to assign arbitrary names to the components of lists as demonstrated below:

```
 L1<-c(1,5,3)
 L2<-c("A","B")
 L<-list(Number=L1,Letter=L2)
 L
$Number
[1] 1 5 3

$Letter
[1] "A" "B"
```

As indicated, the components numbers have been replaced by names that have been defined. In these cases, the components of lists can be obtained by adding `$name` after the name of the list as shown below:

```
 L$Number
[1] 1 5 3
```

A.3 Data Entry in R

Generally speaking, data can be entered into R by using the keyboard or reading from a file. The keyboard method is suitable when there is little data for entering, whereas the data read can be much more extensive. Although it is possible to read data from different extensions of files, such as *.xls (by Excel), *.sav (by SPSS), *.mtb (by Minitab), *.txt (by Notepad), it is preferable to use only *.txt and *.xls, as they are easier to use. For reading

data from text and Excel files, it is possible to use the functions `scan()`, `read.txt()` and `read.cls()`.

- `scan()`: to use this function, type it in the command line and then press enter. R will be waiting to receive data from the user. It is then possible to enter data by keyboard and press the enter or space bar key after every filled in data. It is also possible to copy data from excel or notepad files and then paste it in the line where it is possible to enter data.
- `read.table()`: when data is entered in a *.txt file, the following code can be used to read the data of the file:

```
read.table("d:/data.txt")
```

  Note that, it is necessary to replace the address of the file instead of `d:` in the above code.
- `read.csv()`: this function is used for reading data from an Excel file. To do so, the Excel file must first be saved with extension *.csv, then use the following code:

```
read.csv("d:/data.csv")
```

A.4 Manipulating and Editing Data in R

One of the limitations of R is related to the editing of commands and data that are typed in the R Console after pressing enter. However, it is possible to access every part of the defined data and edit the value of it. It is also possible to extract one or more parts of data by determining the address of the parts. There are two ways for determining the address of the parts of data that need to be extracted:

- Method 1: by determining the index of the data as in the following examples:

```
 x<-c(-2,-1,1,2,4,8)
 x[2]
[1] -1
 x[2:4]
```

```
[1] -1  1  2
 x[c(1,3,7)]
[1] -2  1 NA
 x[c(1,2,6)]
[1] -2 -1  8
 x[-3]
[1] -2 -1  2  4  8
 x[-c(1,2)]
[1] 1 2 4 8
```

- Method 2: by defining a rule for choosing the requested data; for the defined vector `x`, the following codes can be executed:

```
 x[x>0]
[1] 1 2 4 8
 x[x>0&x<4]
[1] 1 2
 x[x^2==4]
[1] -2  2
```

For changing values of one or more parts of the data, it is possible to determine the locations of the data by the above methods and then assign new values, an example of this running code is given below:

```
x[x<0]<-0
```

This running code is to reassign zero to all negative values of the vector `x`. There are some situations where there are missing values in the data sets and those are indicated by `NA`.

A.5 Demonstration of R Functions

A.5.1 Computational Functions

A number of computational functions are available in R as in other similar software programs, such as Matlab. Figure A.1 shows the most common computational functions in R. In this figure, grey expressions are the R codes, which must be typed after > prompt in R console whereas the dark expressions next to them indicate the R codes results. For example, typing `log(x)` gives $Ln(x)$, the natural logarithm of x.

Arithmetic Operators		Logarithm		Big Operators	
x + y	$x+y$	log(x)	Ln(x)	sum(x[i], i = 1, n)	$\sum_1^n x_i$
x – y	$x-y$	log(x, y)	$\log_y(x)$	prod(x[i], i = 1, n)	$\prod_1^n x_i$
x * y	xy	Relations		Statistical Operators	
x/y	x/y	x == y	$x=y$	mean(x)	$\bar{x}$
Sub/Superscripts		x != y	$x \neq y$	var(x)	var(x)
x[i]	x_i	x <= y	$x \leq y$	min(x)	min x
x^n	x^n	x >= y	$x \geq y$	max(x)	max x

Fig. A.1 Some of the computational functions available in R

One of the most prominent capabilities of R, which helps users to do their computations quickly, is acting R functions on the whole element of data structures without using a loop; for example:

```
T<-1:5
log(T)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

A.5.2 Graphical Functions

R also has functions that can be used to produce graphs, with `plot()` the most common R function that can be used to produce graphs.

There is a variety of options designed to assist drawing relevant figures and some of these options are given in the following examples.

```
x<-1:50
plot(sin(2*x*pi/30),type="l",lwd=2,xlab="x",ylab="sin(x)")
plot(cos(2*x*pi/30),lwd=2,xlab="x",ylab="cos(x)")
```

■

These codes will produce the graphs in Fig. A.2.

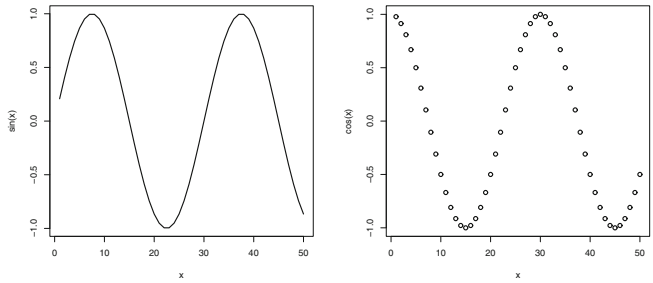

Fig. A.2 Several graphical functions

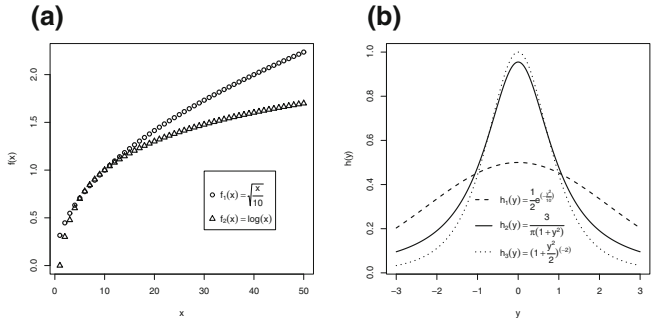

Fig. A.3 Examples of several graphical functions

Using the functions `points()` and `lines()` after `plot()` helps to add several curves in the same plot. Graphs (a) and (b) in Fig. A.3 are produced by these functions with the following codes.

```
x<-1:50
f1<-sqrt(x/10)
f2<-log(x,10)
plot(f1,pch=1,lwd=2,xlab="x",ylab="f(x)",ylim=range(f1,f2))
legend(30,.99,c(expression(f[1](x)==sqrt(frac(x,10))),
expression(f[2](x)==log(x))),pch=c(1,2))
title("(a)")
points(f2,pch=2,lwd=2)


y<-seq(-3,3,length=1000)
h1<-exp(-y^2/10)/2
```

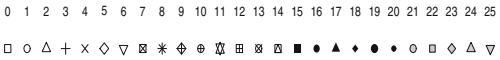

Fig. A.4 Different point characters (pch) for plot function

```
h2<-3/(pi*(1+y^2))
h3<-(1+y^2/2)^(-2)
plot(y,h1,type="l",lty=2,lwd=2,xlab="y",ylab="h(y)",ylim=range(h1,h2,h3))
legend(-1.25,.45,c(expression(h[1](y)==frac(1,2)*e^(-frac(y^2,10))),
expression(h[2](y)==frac(3,pi*(1+y^2))),
expression(h[3](y)==(1+frac(y^2,2))^(-2))),
lty=c(2,1,3),bty="n",lwd=2)
title("(b)")
lines(y,h2,lty=1,lwd=2)
lines(y,h3,lty=3,lwd=2)
```

From the options of plots, `pch` and `lty` are useful tools for creating graphs of comparison. For example, Fig. A.4 displays the results of option `pch = i` for $i = 0, \ldots, 25$ in `plot()` function. Moreover, line types (`lty`) can either be specified as an integer (0 = blank, 1 = solid (default), 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash) or as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash", where "blank" uses invisible lines (i.e., does not draw them). A full list of graphical parameters in R are available with `?par`.

Multiple Graphs on One Page

The following R functions setup multiple graphs on one page:

- `par(mfrow=c(x,y))`, and `par(mfcol=c(x,y))`, where `x,y` give the number of horizontal and vertical entities, respectively; and figures will be drawn in an x-by-y array on the device by columns (mfcol), or rows (mfrow), respectively.
- `layout()`, which allows greater flexibility.

For example, the following R codes generate Fig. A.5:

```
par(mfrow=c(1,3))
hist(rnorm(1000))
boxplot(rnorm(1000))
plot(density(rnorm(1000)))
```

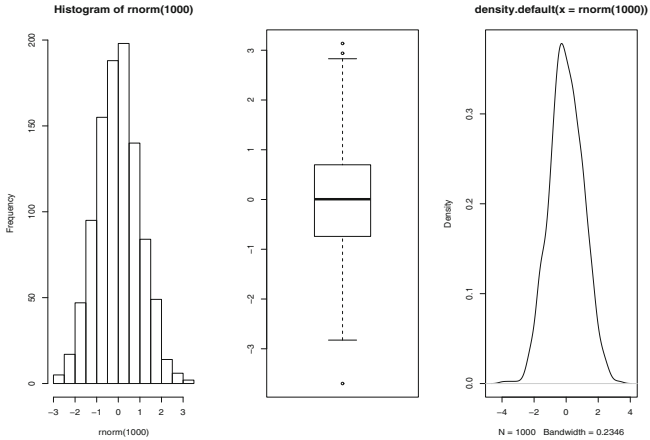

Fig. A.5 Using the mfrow function

Note that the multifigure design, which can be specified by `par()`, is very rigid. In this function all panels of the plots have the same size; however, in some situations it is better to draw plots with different panel sizes. There is more flexibility with `layout()`, for instance, it is possible to combine some parts of the page as illustrated in the following R code, which produces Fig. A.6:

```
layout(matrix(c(1,1,2,3),nrow=2,byrow=T))
boxplot(rnorm(1000),horizontal=T)
hist(rnorm(1000))
plot(density(rnorm(1000)))
```

As can be seen in Fig. A.6, the first and second graph's place are combined to produce a better figure than Fig. A.5.

Multiple Graphic Windows

It is useful in some situations to have more than one graphic window at the same time, but note that only one graphics windows can accept graphics commands at any one time, and this is known as the current device. When multiple devices are open, they form a numbered sequence with names giving the kind of device at any position. The R function `windows()` is one of the main commands used for operating with multiple devices

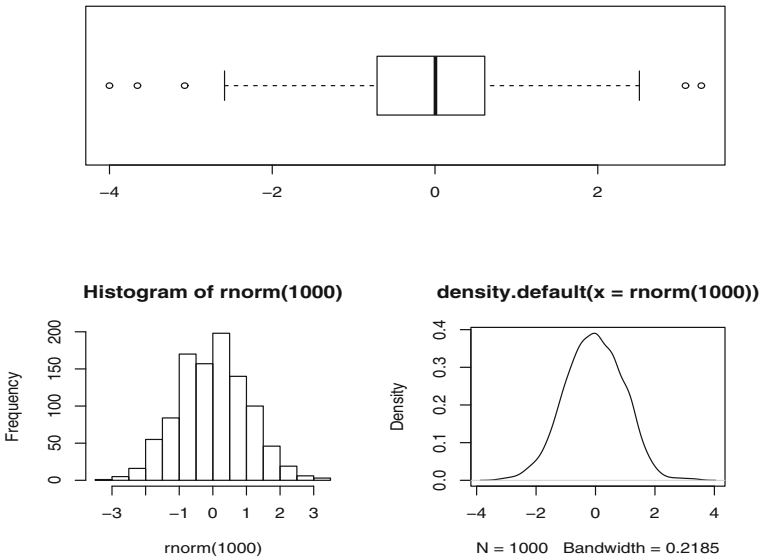

Fig. A.6 Using the layout function

and their meanings. Similar to other R functions, `windows()` has several different arguments: arguments `width, height`, which determine the (nominal) width and height of the canvas of the plotting window in inches by default of 7. A full list of arguments can be obtained by `?windows`. An example of using `windows()` for generating a better representation of Fig. A.5 can be constructed by:

```
windows(width=7,height=3)
par(mfrow=c(1,3))
hist(rnorm(1000))
boxplot(rnorm(1000))
plot(density(rnorm(1000))),
```

which produces Fig. A.7.

Saving as Graphs in R

R can generate graphics on almost any type of display such as `jpeg`, `bmp`, `tiff`, `gif`, `pdf` and `eps`. To convert graphical instructions from R into a form mentioned, use the menu `File > Save as` and then select the

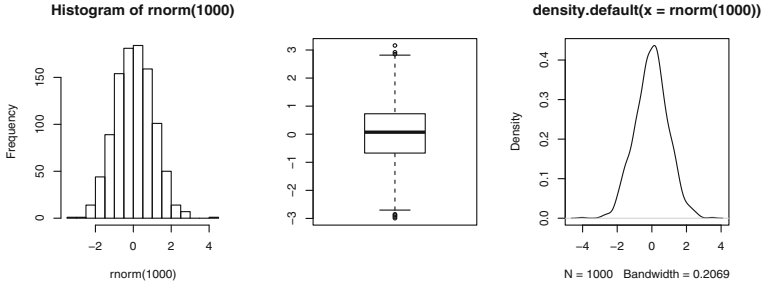

Fig. A.7 Using the mfrow function

suitable format. Another way is using a right click on the graph and choosing a suitable format to export the graph into a file.

A.5.3 Probability Distribution Functions and Their Components in R

Random, density, cumulative and quantile data from known probability distribution functions are the key components, and the way of obtaining them is described here. It is possible to get these values in R by adding the following prefix letters to the name of distribution:

- `r`: for generating random data.
- `d`: for computing density function.
- `p`: for computing cumulative distribution function.
- `q`: for obtaining quantiles of distribution function.

As an example, the following codes show the output for the normal distribution with mean 0 and standard deviation 1. Note that the statements after # are just explanations for the codes and have not been executed.

```
 d1<-dnorm(2,0,1)#computes the density value at point 1.
 d1
[1] 0.05399097
 p1<-pnorm(1,0,1)#computes the cumulative probability value to the point 1.
 p1
[1] 0.8413447
 q1<-qnorm(0.975,0,1)#computes the quantile for the level 0.975.
 q1
[1] 1.959964
 r1<-rnorm(5,0,1)#generates 5 random data.
 r1
[1] -0.7007135  0.6981410 -0.3570325  0.8213821  2.2474506
```

■

A.5.4 Random Samples and Permutations in R

R can also be used to generate arbitrary random data from known distributions. The function `sample()` produces a sample from a vector of numeric data. This function takes a sample of the specified size from the elements of a numeric data vector, either, sampled with or without replacement, as specified in the code. The following example illustrates the use of `sample()` with replacement:

```
 x<-c(1,6,2,9,8)
 sample(x,10,replace=T)
 [1] 1 6 1 1 8 1 1 8 1 2
 sample(x,3,replace=F)
[1] 9 8 2
```

A.6 Writing New Functions in R

R also has the option to define new functions or provide some changes as the existing functions. The general form of the new function for R is as follows:

```
 funcname<-function(arguments){
 body of function
 }
```

where `funcname` is an arbitrary name for the function, `arguments` are the input parameters of the function and `body of the function` indicates the computations and output of the function. After defining the function, code `funcname(arguments)`, with given values for arguments, is used to get the results of function. Several examples are provided below.

Example A.1 Assume `x` is a numeric vector and it is desired to obtain the standardized values of `x`. This can be achieved with the following function:

```
 stand<-function(x){
 (x-mean(x))/sd(x)
 }
```

After writing this function in the R console, predefined R functions can be used. For example:

```
 e<-c(-1.5,-.5,0,2)
 stand(e)
[1] -1.0190493 -0.3396831  0.0000000  1.3587324
```

It is possible to get several different outputs from a function. For example, function `stand()` is changed in such a way that original values, standardized values, mean and variance of data are returned as output.

```
 stand<-function(x){
 z=(x-mean(x))/sd(x)
 data<-data.frame(x,z)
 dimnames(data)[[2]]<-c("Original","Standardized")
 list(data=data,mean=mean(x),variance=var(x))
 }

 stand(e)
$data
  Original Standardized
1     -1.5   -1.0190493
2     -0.5   -0.3396831
3      0.0    0.0000000
4      2.0    1.3587324

$mean
[1] 0

$variance
[1] 2.166667
```

A.6.1 Loops and Conditions in R

R functions can be applied to all elements of a vector, matrix or array, without using loops and conditional commands. Although, in some cases, it may be necessary to use a loop or condition to write a new R functions. Note that using loop and condition functions in new functions increases the time of execution and it is better to avoid using loops and conditions wherever possible.

As in other software programs, the function `for()` is the most common loop command in R. The general structure of `for` is as follows:

```
for(counter in setcounter){
commands must be repeated
}
```

Here are several examples:

Example A.2 An identity matrix with dimension 5×5 can be constructed by:

```
 I<-matrix(rep(0,25),5,5)
 for(i in 1:5)
 I[i,i]<-1
 I
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

Example A.3 The centralized moments of a vector of numeric data can be obtained by the following function:

```
 moment<-function(x,k){
 for(i in 1:k){
 M<-mean((x-mean(x))^i)
 cat("M",i,"=",M,sep="","\n")}}
```

As an example:

```
 x<-1:20
 moment(x,10)
M1=0
M2=33.25
M3=0
M4=1983.362
M5=0
```

Table A.1 A discrete distribution

Value	x_1	x_2	x_k
Prob	p_1	p_2	p_k

```
M6=140371.7
M7=0
M8=10781840
M9=0
M10=868289604
```

The R function `if()` is the usual conditional command like as in other software programs.

```
if(cond1) expr1
else expr2
```

Example A.4 Here we write a function that produces the power of a square matrix. The following code is an example for obtaining the power of a matrix.

```
M.power<-function(M,p){
size<-nrow(M)
M1<-diag(rep(1,size))
if(p==1)
return(M)
else{
for(i in 1:p)
M1<-M1%*%M
return(M1)
}}
```

Example A.5 In this example, the new function produces a sample with size n from the following discrete distribution.

To solve this problem, first recall that generating data from such distributions is possible by the following steps:

- Generate samples $u_1, \ldots, u_n$ from the uniform $(0, 1)$ distribution,
- Let $p_1 + ... + p_{i-1} < u_j \leq p_1 + ... + p_i$ then x_i is a random sample from the given distribution.
- Do the previous step for $j = 1, ..., n$ and determine $x_1, ..., x_n$

Applying this algorithm, the following function generates random samples from the specified discrete distribution:

```
rdisc<-function(n,p,x){
k<-length(x)
ran<-array(dim=n)
u<-runif(n)
P<-cumsum(p)
for(i in 1:n){
if(u[i]<P[1])
ran[i]<-x[1]
else{
for(j in 1:(k-1)){
if(u[i]<P[(j+1)]& u[i]>P[j])
ran[i]<-x[(j+1)]
}}}
ran
}
```

To use `rdisc()` vectors `x`, `p` and sample size `n` must be determined. For instance it is possible to use the following code:

```
p<-c(.1,.3,.4,.2)
x<-c(-1,0,1,2)
rdisc(10,p,x)
[1]  1  2  0  1 -1  1  1  1  0 -1
```

Example A.6 A colour scaling plot is useful in certain situations to show the rate of association between two variables. The following code is an example showing how a scaling plot can be created:

```
Plt.Img <- function(x){
min <- min(x)
max <- max(x)
yLabels <- rownames(x)
xLabels <- colnames(x)
```

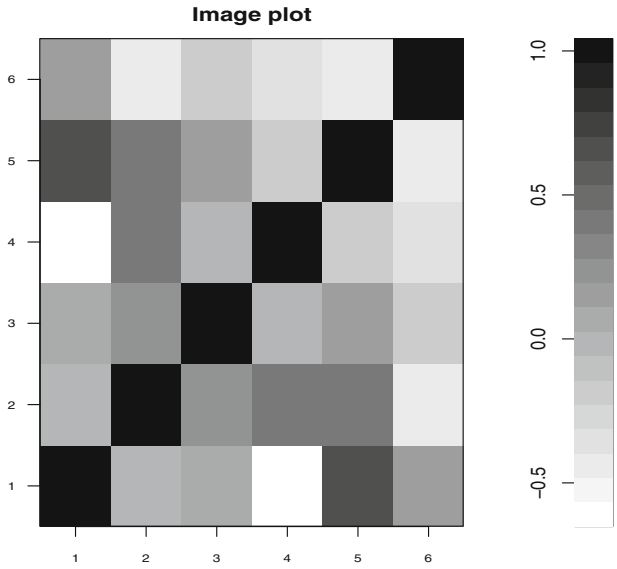

Fig. A.8 Examples of several graphical functions

```
if( is.null(xLabels) ){
xLabels <- c(1:ncol(x))
}
if( is.null(yLabels) ){
yLabels <- c(1:nrow(x))
}
layout(matrix(data=c(1,2), nrow=1, ncol=2),
widths=c(4,1), heights=c(1,1))
ColorRamp <- gray( seq(1,0,length=20))
ColorLevels <- seq(min, max, length=length(ColorRamp))
par(mar = c(3,5,2.5,2))
image(1:length(xLabels), 1:length(yLabels),
t(x), col=ColorRamp, xlab="",
ylab="", axes=FALSE, zlim=c(min,max))
title(main=c("Image plot"))
axis(BELOW<-1, at=1:length(xLabels),
labels=xLabels, cex.axis=0.7)
axis(LEFT <-2, at=1:length(yLabels),
labels=yLabels, las= HORIZONTAL<-1,
cex.axis=0.7)
```

```
box()
par(mar = c(3,2.5,2.5,2))
image(1, ColorLevels,
matrix(data=ColorLevels, ncol=length(ColorLevels),nrow=1),
col=ColorRamp,
xlab="",ylab="",
xaxt="n")
layout(1)
}
```

Now considering matrix *m* as below, the results of function `Plt.Img()` is similar to w-correlation plot used in SSA:

```
y<-matrix(rbinom(60,10,.5),12,5)
m<- cor(y)
Plt.Img(m)
```

Appendix B: Theoretical Explanations

In this appendix, we provide details of a number of theorems referred in Chapters 1 and 2. We also use the opportunity to elaborate some concepts used in the previous chapters.

Singular Value Decomposition (SVD)

The singular value decomposition (SVD) is a factorization of a real or complex matrix. It is the generalization of the eigendecomposition of a positive semidefinite normal matrix (e.g., a symmetric matrix with positive eigenvalues) to any $L \times K$ matrix $\mathbf{X}$ via an extension of polar decomposition.

The SVD of $L \times K$ matrix $\mathbf{X}$ is a factorization of the form $\mathbf{U}\Sigma\mathbf{V}^T$, where $\mathbf{U}$ is an $L \times L$ real or complex unitary matrix, Σ is a $L \times K$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and $\mathbf{V}$ is a $K \times K$ real or complex unitary matrix. The diagonal entries λ_i of Σ are known as the singular values of $\mathbf{X}$. The columns of $\mathbf{U}$ and the columns of $\mathbf{V}$ are called the left-singular vectors and right-singular vectors of $\mathbf{X}$, respectively.

Note that the left-singular values of $\mathbf{X}$ are a set of orthonormal eigenvectors of $\mathbf{X}\mathbf{X}^T$, the right-singular vectors of $\mathbf{X}$ are a set of orthonormal eigenvectors of $\mathbf{X}^T\mathbf{X}$, and the nonzero singular values of $\mathbf{X}$ (which are on the diagonal entries of Σ) are the square roots of the nonzero eigenvalues of both $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}^T\mathbf{X}$.

© The Editor(s) (if applicable) and The Author(s) 2018

H. Hassani and R. Mahmoudvand, *Singular Spectrum Analysis*,
Palgrave Advanced Texts in Econometrics,
https://doi.org/10.1057/978-1-137-40951-5

Theorem B.1 *Let d denotes the rank of the trajectory matrix $\mathbf{X}$ in SSA. The maximum rank of the trajectory matrix $\mathbf{X}$ is achieved at $L = L_{\max}$.*

Proof The maximum value of d is obtained as follows:

$$\begin{aligned}\max_{L\in\{2,\ldots,N-1\}} d &= \max_{L\in\{2,\ldots,N-1\}} \min\{L, N-L+1\} \\ &= \max_{L\in\{2,\ldots,N-1\}} \frac{N+1-2L-(N+1)}{2},\end{aligned}$$

which shows that the maximum rank of $\mathbf{X}$ is attained when $2L-(N+1)$ is minimized. Thus, $L = L_{\max}$. ■

Theorem B.2 *Let d denote the rank of the trajectory matrix $\mathbf{X}$ in multivariate SSA (either VMSSA or HMSSA). The maximum rank of the trajectory matrix $\mathbf{X}$ is achieved at $L = \left\lceil \frac{1}{M+1}(N+1) \right\rceil$ and $L = \left\lceil \frac{M}{M+1}(N+1) \right\rceil$ for VMSSA and HMSSA, respectively, where $\lceil u \rceil$ denotes the closest integer number to u.*

Proof Let us, for example, consider the VMSSA version. The maximum value of d is obtained as follows:

$$\begin{aligned}\max d &= \max_{L\in\{2,\ldots,N-1\}} \min\{LM, N-L+1\} \\ &= \max_{L\in\{2,\ldots,N-1\}} \frac{L(M-1)+N+1-|L(M+1)-(N+1)|}{2}\end{aligned} \tag{B.1}$$

which shows that the maximum rank of $\mathbf{X}$ is attained when $|L(M+1)-(N+1)|$ is minimized. Thus, $L = \left\lceil \frac{N+1}{M+1} \right\rceil$. Similarly, it is straightforward to prove that $L = \left\lceil \frac{M}{M+1}(N+1) \right\rceil$ provides the optimal L for VMSSA. Moreover, these values of L are the first $(M+1)$-quantile and the Mth $(M+1)$-quantile of the integer set $\{1, \ldots, N\}$. The optimal values of L in the reconstruction stage of MSSA, having M series with length N, are as follows:

$$L = \begin{cases} \left\lceil \frac{1}{M+1}) \right\rceil (N+1) & \text{VMSSA;} \\ \left\lceil \frac{M}{M+1}) \right\rceil (N+1) & \text{HMSSA.} \end{cases} \tag{B.2}$$

■

Theorem B.3 *Consider the Hankel matrix* $\mathbf{X}$ *as defined in Eq. (1.2), then,*

$$T_{\mathbf{X}}^{L,N} = \sum_{j=1}^{N} w_j^{L,N} y_j^2,$$

where $w_j^{L,N} = \min\{\min\{L, K\}, j, L + K - j\} = w_j^{K,N}$.

Proof Applying the definition of $\mathbf{X}$, we have:

$$T_{\mathbf{X}}^{L,N} = \sum_{i=1}^{L} \sum_{j=i}^{N-L+i} y_j^2. \tag{B.3}$$

Changing the order of the summations in Eq. (B.3), we observe

$$T_{\mathbf{X}}^{L,N} = \sum_{j=1}^{N} C_{j,L,N} y_j^2,$$

where $C_{j,L,N} = \min\{j, L\} - \max\{1, j - N + L\} + 1$. We only need therefore to show that $C_{j,L,N} = w_j^{L,N}$, for all j and L. We consider two cases: $L \leq K$ and $L > K$. For the first case, we have

$$C_{j,L,N} = \begin{cases} j, & 1 \leq j \leq L, \\ L, & L + 1 \leq j \leq K, \\ N - j + 1, & K + 1 \leq j \leq N, \end{cases}$$

which is exactly equal to $w_j^{L,N}$. Similarly for the second case, we have

$$C_{j,L,N} = \begin{cases} j, & 1 \leq j \leq K, \\ K, & K + 1 \leq j \leq L, \\ N - j + 1, & L + 1 \leq j \leq N, \end{cases}$$

and again is equal to $w_j^{L,N}$, for $L > K$. ■

Theorem B.4 *Let* $T_{\mathbf{X}}^{L,N}$ *be defined as before, then,* $T_{\mathbf{X}}^{L,N}$ *is an increasing function of* L *on* $\{2, \ldots, [(N + 1)/2]\}$ *and a decreasing function on* $\{[(N + 1)/2] + 1, \ldots, N - 1\}$, *and*

$$\max T_{\mathbf{X}}^{L,N} = T_{\mathbf{X}}^{\left[\frac{N+1}{2}\right],N}.$$

Proof First, we show that $w_j^{L,N}$ is an increasing function of L on $\{2, \ldots, [(N+1)/2]\}$. Let L_1 and L_2 be two arbitrary values where $L_1 < L_2 \leq [(N+1)/2]$. From the definition of $w_j^{L,N}$, we have

$$w_j^{L_2,N} - w_j^{L_1,N} = \begin{cases} 0, & 1 \leq j \leq L_1, \\ j - L_1, & L_1 + 1 \leq j \leq L_2, \\ L_2 - L_1, & L_2 + 1 \leq j \leq N - L_2 + 1, \\ N - j + 1 - L_1, & N - L_2 + 2 \leq j \leq N - L_1 + 1, \\ 0, & N - L_1 + 2 \leq j \leq N. \end{cases}$$

Therefore, $w_j^{L_2,N} - w_j^{L_1,N} \geq 0$, for all j, and the inequality is strict for some j. Thus,

$$T_{\mathbf{X}}^{L_2,N} - T_{\mathbf{X}}^{L_1,N} = \sum_{j=1}^{N} \left(w_j^{L_2,N} - w_j^{L_1,N} \right) h_j^2 > 0. \tag{B.4}$$

This confirms that w_j^L is an increasing function of L on $\{2, \ldots, [(N+1)/2]\}$. A similar approach for the set $\{[(N+1)/2]+1, \ldots, N-1\}$ implies that $w_j^{L,N}$ is a decreasing function of L on this interval. Note also that $T_{\mathbf{X}}^{L_2,N} - T_{\mathbf{X}}^{L_1,N}$ in Eq. (B.4) increases as we increase the value of L_2 proving that $T_{\mathbf{X}}^{L,N}$ is an increasing function on $\{2, \ldots, [(N+1)/2]\}$. The maximum value of $T_{\mathbf{X}}^{L_2,N}$ is therefore attained at the maximum value of L, which is $[(N+1)/2]$. ■

Corollary B.1 *Let $L_{\max}$ denote the value of L such that $T_{\mathbf{X}}^{L,N} \leq T_{\mathbf{X}}^{L_{\max},N}$, for all L, and the inequality to be strict for some values of L then,*

$$L_{\max} = \begin{cases} \frac{N+1}{2}, & \text{if } N \text{ is odd}, \\ \frac{N}{2}, \frac{N}{2} + 1, & \text{if } N \text{ is even}. \end{cases}$$

Corollary B.1 shows that $L = \text{median}\{1, \ldots, N\}$ maximizes the sum of squares of the Hankel matrix singular values with fixed values of N. Applying Corollary B.1 and the definition of $w_j^{L,N}$, we can show that

$$w_j^{L_{\max},N} = \frac{N+1}{2} - \left| \frac{N+1}{2} - j \right|. \tag{B.5}$$

Equation (B.5) shows that $y_{[(N+1)/2]}$ has maximum weight at $T_{\mathbf{X}}^{L,N}$.

Theorem B.5 *$T_{\mathbf{X}}^{L,N}/K$ is an increasing function of L on $\{2, ..., N-1\}$.*

Proof L_1 and L_2 are two arbitrary values where $L_1 < L_2$. It is necessary to show that $K_1 T_{\mathbf{X}}^{L_2,N} - K_2 T_{\mathbf{X}}^{L_1,N} > 0$, where $K_j = N - L_j + 1$ $(j = 1, 2)$. To this end, the following four cases are considered:

- Case 1: $L_2 \leq L_{\max}$. In this case, the results are easily obtained from Theorem B.4.
- Case 2: $L_1 \leq L_{\max} < L_2 < N - L_1 + 1$. Using the equality $T_{\mathbf{X}}^{L_2,N} = T_{\mathbf{X}}^{N-L_2+1,N}$ can be obtained. Now, applying the inequality $N-L_2+1 > L_1$ for this case, the results are again obtained from Theorem B.4.
- Case 3: $L_1 \leq L_{\max} \leq N - L_1 + 1 < L_2$. In this case:

  $$K_1 w_j^{L_2,N} - K_2 w_j^{L_1,N} =$$

  $$\begin{cases} j(L_2 - L_1), & 1 \leq j \leq K_2, \\ K_2(K_1 - j), & K_2 + 1 \leq j \leq L_1, \\ K_2(K_1 - L_1), & L_1 + 1 \leq j \leq K_1, \\ K_2(j - L_1), & K_1 + 1 \leq j \leq L_2, \\ (N - j + 1)(L_2 - L_1), & L_2 + 1 \leq j \leq N, \end{cases}$$

  Therefore, $K_1 w_j^{L_2,N} - K_2 w_j^{L_1,N} > 0$ for all j, which gives the required result.
- Case 4: $L_{\max} \leq L_1 < L_2$. Similar to the case 3 there is:

  $$K_1 w_j^{L_2,N} - K_2 w_j^{L_1,N} =$$

  $$\begin{cases} j(L_2 - L_1), & 1 \leq j \leq K_2, \\ K_2(K_1 - j), & K_2 + 1 \leq j \leq K_1, \\ 0, & K_1 + 1 \leq j \leq L_1, \\ K_2(j - L_1), & L_1 + 1 \leq j \leq L_2, \\ (N - j + 1)(L_2 - L_1), & L_2 + 1 \leq j \leq N, \end{cases}$$

  Therefore, $K_1 w_j^{L_2,N} - K_2 w_j^{L_1,N} \geq 0$ for all j and inequality is strict for some j. ■

Theorem B.6 *The matrix* $\mathbf{\Pi}$ *is the matrix of the linear operator that performs the orthogonal projection* $\mathbb{R}^{L_{sum}-M} \mapsto \mathfrak{L}_r^{\nabla}$, *where* $\mathfrak{L}_r^{\nabla} = Span\{U_1^{\nabla}, \ldots, U_r^{\nabla}\}$.

Proof Let $\mathbf{U}$ be a matrix with columns $U_1, \ldots, U_r$. By definition of U_j we can write:

$$\mathbf{I}_{r\times r} = \mathbf{U}^T\mathbf{U} = \mathbf{U}^{\nabla T}\mathbf{U}^{\nabla} + \mathbf{W}^T\mathbf{W}. \tag{B.6}$$

Thus, $\mathbf{U}^{\nabla T}\mathbf{U}^{\nabla} = \mathbf{I}_{r\times r} - \mathbf{W}^T\mathbf{W}$. Using Eq. (B.6), we can write:

$$\begin{aligned}
&\left(\mathbf{U}^{\nabla T}\mathbf{U}^{\nabla}\right)\left(\mathbf{I}_{r\times r} + \mathbf{W}^T\left(\mathbf{I}_{M\times M} - \mathbf{W}\mathbf{W}^T\right)^{-1}\mathbf{W}\right)\\
&= \left(\mathbf{I}_{r\times r} - \mathbf{W}^T\mathbf{W}\right)\left(\mathbf{I}_{r\times r} + \mathbf{W}^T\left(\mathbf{I}_{M\times M} - \mathbf{W}\mathbf{W}^T\right)^{-1}\mathbf{W}\right)\\
&= \mathbf{I}_{r\times r} - \mathbf{W}^T\mathbf{W} + \mathbf{W}^T\left(\mathbf{I}_{M\times M} - \mathbf{W}\mathbf{W}^T\right)\left(\mathbf{I}_{M\times M} - \mathbf{W}\mathbf{W}^T\right)^{-1}\mathbf{W} = \mathbf{I}_{r\times r}
\end{aligned}$$

Therefore $\left(\mathbf{U}^{\nabla T}\mathbf{U}^{\nabla}\right)^{-1} = \left(\mathbf{I}_{r\times r} + \mathbf{W}^T\left(\mathbf{I}_{M\times M} - \mathbf{W}\mathbf{W}^T\right)^{-1}\mathbf{W}\right)$ and the perpendicular projection onto the column space of $\mathbf{U}^{\nabla}$ can be obtained by the following equation:

$$\mathbf{U}^{\nabla}\left(\mathbf{U}^{\nabla T}\mathbf{U}^{\nabla}\right)^{-1}\mathbf{U}^{\nabla T} = \mathbf{U}^{\nabla}\mathbf{U}^{\nabla T} + \mathbf{U}^{\nabla}\mathbf{W}^T\left(\mathbf{I}_{M\times M} - \mathbf{W}\mathbf{W}^T\right)^{-1}\mathbf{W}\mathbf{U}^{\nabla T}, \tag{B.7}$$

which completes the proof by using the definition of $\mathcal{R}$. ■

Strong separability

Fix the window length L, consider a certain SVD of the trajectory matrix $\mathbf{X}$ of the initial series Y_N of length N, and assume that this series is a sum of two series $Y_N^{(1)}$ and $Y_N^{(2)}$, that is, $Y_N = Y_N^{(1)} + Y_N^{(2)}$. Moreover, let $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$ be the corresponding trajectory matrices of the series $Y_N^{(1)}$ and $Y_N^{(2)}$, respectively.

Separability of the series $Y_N^{(1)}$ and $Y_N^{(2)}$ means that the matrix terms of the SVD can be split from the trajectory matrix $\mathbf{X}$ into two different groups, so that the sums of terms within the groups give the trajectory matrices $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$ of the series $Y_N^{(1)}$ and $Y_N^{(2)}$, respectively. In other words assume $\mathbf{X} = \mathbf{X}_{\mathrm{SVD}}^{(1)} + \mathbf{X}_{\mathrm{SVD}}^{(2)}$ by using SVD and grouping. Then strong separability occurs if $\mathbf{X}_{\mathrm{SVD}}^{(1)} = \mathbf{X}^{(1)}$ and $\mathbf{X}_{\mathrm{SVD}}^{(2)} = \mathbf{X}^{(2)}$.

Although, strong separability only rarely occurs in practice; it happens theoretically if the columns (rows) of $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$ are orthogonal and the singular values are isolated. Zokaei and Mahmoudvand (2011) have considered the coefficient of variation for the differentiation of singular values and showed that the maximum separation for a wide class of series will be obtained when $L = L_{\max}$.

References

Alonso, F., Castillo, J., & Pintado, P. (2005). Application of singular spectrum analysis to the smoothing of raw kinematic signals. *Journal of Biomechanics*, *38*(2), 1085–1092.

Alvarez-Meza, A., Acosta-Medina, C., & Castellanos-Domnguez, G. (2013). Automatic singular spectrum analysis for time-series decomposition. In *ESANN 2013 Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (pp. 131–136). Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

Broomhead, D. S., & King, G. P. (1986a). Extracting qualitative dynamics from experimental data. *Physica D*, *20*, 217–236.

Broomhead, D. S., & King, G. P. (1986b). *On the qualitative analysis of experimental dynamical systems.* Bristol: Adam Hilger.

Broomhead, D. S., King, G. P., & Pike, E. R. (1987). *Singular spectrum analysis with application to dynamical systems.* Bristol: IOP Publication.

Elsner, J. B., & Tsonis, A. A. (1996). *Singular spectrum analysis: A new tool in time series analysis.* New York: Plenum Press.

Ghodsi, M., Hassani, H., Sanei, S., & Hichs, Y. (2009). The use of noise information for detection of temporomandibular disorder. *Biomedical Signal Processing and Control*, *4*(2), 79–85.

Golyandina, N. (2010). On the choice of parameters in singular spectrum analysis: And related subspace-based methods. *Statistics and Inference*, *3*(3), 257–279.

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

Golyandina, N., & Osipov, E. (2007). The caterpillar-SSA method for analysis of time series with missing values. *Journal of Statistical Planning and Interface*, *137*(8), 2642–2653.

Golyandina, N., & Zhigljavsky, A. (2013). *Singular spectrum analysis for time series.* Springer brief in statistics. Berlin: Springer.

Hamilton, J. D. (2017). Why you should never use the Hodrick–Prescott filter. *Review of Economics and Statistics.*

Harris, T., & Yan, H. (2010). Filtering and frequency interpretations of singular spectrum analysis. *Physica D*, *239*, 1958–1967.

Hassani, H. (2010). Singular spectrum analysis based on the minimum variance estimator. *Nonlinear Analysis: Real World Applications*, *11*(3), 2065–2077.

Hassani, H., & Mahmoudvand, R. (2013). Multivariate singular spectrum analysis: A general view and new vector forecasting algorithm. *International Journal of Energy and Statistics*, *1*(1), 55–83.

Hassani, H., Mahmoudvand, R., Nabe Omer, H., & Silvia, E. S. (2014). A preliminary investigation into the effect of outlier(s) on singular spectrum analysis. *Fluctuation and Noise Letters*, *13*(4), https://doi.org/10.1142/S0219477514500291.

Hassani, H., Mahmoudvand, R., & Yarmohammadi, M. (2010). Filtering and denoising in the linear regression model. *Fluctuation and Noise Letters*, *9*(4), 343–358.

Hassani, H., Mahmoudvand, R., & Zokaei, M. (2011a). Separability and window length in the singular spectrum analysis. *Compets Rendes Mathematiques*, *349*, 987–990.

Hassani, H., Mahmoudvand, R., Zokaei, M., & Ghodsi, M. (2012). On the separability between signal and noise in singular spectrum analysis. *Fluctuation and Noise Letters*, *11*(2), 1–11.

Hassani, H., & Thomakos, D. (2010). A review on singular spectrum analysis for economic and financial time series. *Statistics and Its Interface*, *3*(3), 377–397.

Hassani, H., Zhigljavsky, A., & Zhengyuan, X. (2011b). Singular spectrum analysis based on the perturbation theory. *Nonlinear Analysis: Real World Applications*, *2*(5), 2752–2766.

Khan, M. A. R., & Poskitt, D. S. (2010). *Description length based signal detection in singular spectrum analysis* (Working Paper).

Khan, M. A. R., & Poskitt, D. S. (2013a). Moment tests for window length selection in singular spectrum analysis of short and long memory processes. *Journal of Time Series Analysis*, *34*(2), 141–155.

Khan, M. A. R., & Poskitt, D. S. (2013b). A note on window length selection in singular spectrum analysis. *Australian & New Zealand Journal of Statistics*, *55*(2), 87–108.

Kondrashov, D., & Ghil, M. (2006). Spatio temporal filling of missing points in geophysical data sets. *Nonlin Processes Geophys*, *13*, 151–159.

Kume, K., & Nose-Togawa, N. (2014). Multidimensional extension of singular spectrum analysis based on filtering interpretation. *Advances in Adaptive Data Analysis*, *6*, 1–17.

Mahmoudvand, R., Najari, N., & Zokaei, M. (2013). On the optimal parameters for reconstruction and forecasting in the singular spectrum analysis. *Communication in Statistics: Simulations and Computations*, *42*, 860–870.

Mahmoudvand, R., & Rodrigues, P. (2016a). Correlation analysis in contaminated data by singular spectrum analysis. *Quality and Reliability Engineering International*, *32*, 2127–2137.

Mahmoudvand, R., & Rodrigues, P. (2016b). Missing value imputation in time series using singular spectrum analysis. *International Journal of Energy and Statistics,4,* 1650005 (6 pages).

Mahmoudvand, R., & Zokaei, M. (2012). On the singular values of the Hankel matrix with application in singular spectrum analysis. *Chilean Journal of Statistics*, *3*(1), 43–56.

Mohammad, Y., & Nishida, T. (2011). On comparing SSA-based change point discovery algorithms. *IEEE SII* (pp. 938–945).

Moskvina, V., & Zhigljavsky, A. (2003). An algorithm based on singular spectrum analysis for change-point detection. *Communication in Statistics: Simulation and Computation*, *32*(2), 319–352.

Patterson, K., Hassani, H., Heravi, S., & Zhigljavsky, A. (2011). Forecasting the final vintage of the index of industrial production. *Journal of Applied Statistics*, *38*(10), 2183–2211.

Rendall, R., & Reis, M. (2014). A comparison study of single-scale and multi-scale approaches for data-driven and model-based online denoising. *Quality and Reliability Engineering International*, *30*(7), 935–950.

Rodrigues, P., & de Carvalho, M. (2013). Spectral modeling of time series with missing data. *Applied Mathematical Modelling*, *37*, 4676–4684.

Rodrigues, P., & Mahmoudvand, R. (2017). The benefits of multivariate singular spectrum analysis over the univariate version. *Journal of the Franklin Institute,355,* 544–564.

Sanei, S., & Hassani, H. (2015). *Singular spectrum analysis of biomedical signals.* Boca Raton: CRC Press.

Schoellhamer, D. (2001). Singular spectrum analysis for time series with missing data. *Geophysical Research Letters*, *28*(16), 3187–3190.

Shen, Y., Peng, F., & Li, B. (2015). Improved singular spectrum analysis for time series with missing data. *Nonlinear Processes in Geophysics*, *22*, 371–376.

Index

© The Editor(s) (if applicable) and The Author(s) 2018

H. Hassani and R. Mahmoudvand, *Singular Spectrum Analysis*,
Palgrave Advanced Texts in Econometrics,
https://doi.org/10.1057/978-1-137-40951-5