Nonstationary Time Series: Spectral Estimation and Nonstationarity Tests(2)

27th April 2023

1 Introduction

To run the examples in this sheet you will need to install and load several libraries (if you have not got them already). Please refer to the separate sheet "Workshop Packages" to see what to load.

1.1 Simulated Stationary Time Series

We start with a basic time series model: an autoregressive moving average ARMA process of order (p, q)

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + \cdots + \alpha_p X_{t-p} + Z_t + \beta_1 Z_{t-1} + \cdots + \beta_q Z_{t-q}, \quad (1)$$

where $\{\alpha_i\}_{i=1}^p$ are the autoregressive coefficients and $\{\beta_j\}_{j=1}^q$ are the moving average coefficients and here we'll assume both sets are real numbers.

We can use the R function `arima.sim` to simulate from these second-order stationary time series models. Copy and paste the following commands into R:

```
#
# Set a random number generator seed value, so this example is
# reproducible for everybody
#
set.seed(423)
#
# Simulate an ARMA(0, 2) model with 256 observations
#
sim1 <- arima.sim(n=512, model=list(ma=c(0.9, -0.7)))
#
# Plot the simulated series
#
plot.ts(sim1)
```

The output you will get is shown in Figure 1.

Now plot the autocorrelation function for the `sim1` data set

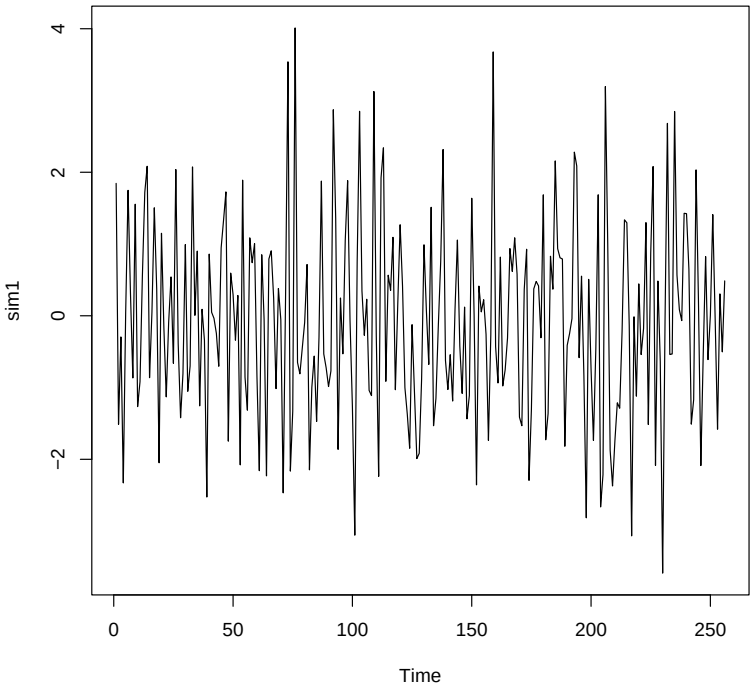

Figure 1: Simulated 512 values from an ARMA$(0, 2)$ model.

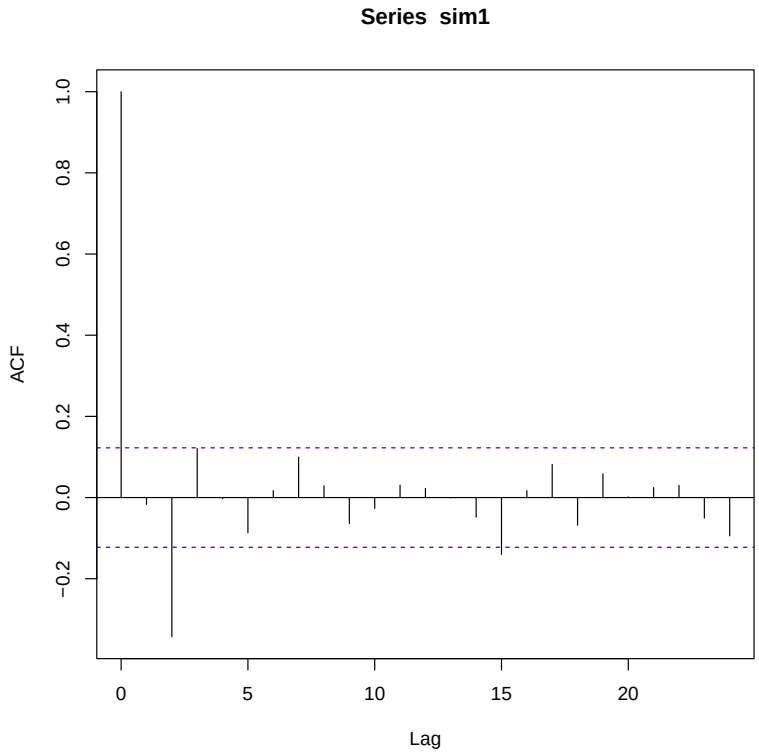

Figure 2: Autocorrelation of the `sim1` data set.

```
#
# Plot the ACF of the sim1 data set
#
acf(sim1)
```

The output of the `acf` plot is shown in Figure 2. The autocorrelation at lag 2 clearly exceeds the approximate confidence interval. So, from this plot alone we could infer that the `sim1` data set is likely to arise from an ARMA$(0, 2)$ process. (The ACF at lag 15 is also peeping above the confidence band, so we could infer a ARMA$(0, 15)$, but it is only just over the bar, and so, in the intererests of parsimonity we would probably stick with the ARMA$(0, 2)$.) We know that `sim1` is simulated from a stationary process, because we have constructed it like that.

1.2 Boris Johnson approval ratings

Figure 3 shows Boris Johnson's approval ratings during his time as Prime Minister. His ratings started off fairly low at around 30% and then increased to 66% in mid-April 2020 and then declined dramatically to the low twenties during the winter 2021–22 and through most of 2022.

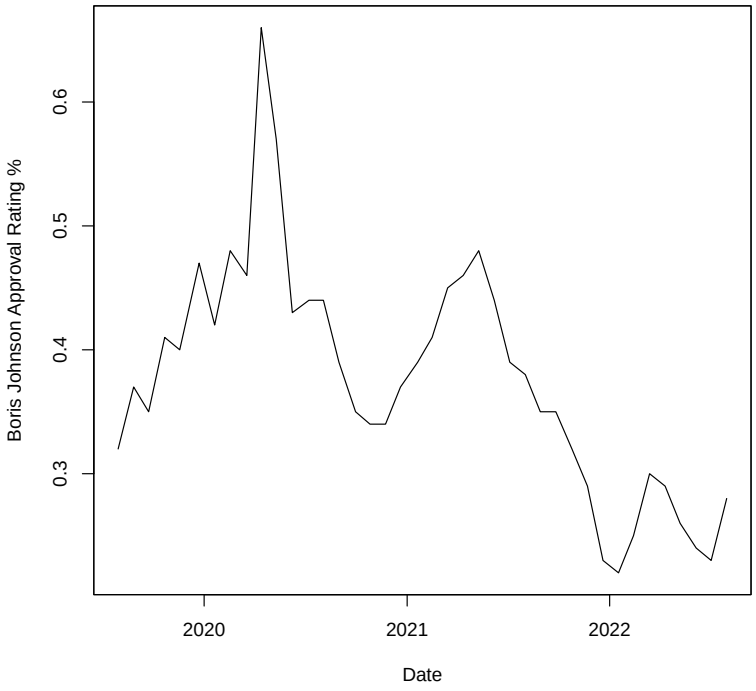

Figure 3: 'Do you think that Boris Johnson is doing well or badly as Prime Minister?'. Time series of the percentage of people who said 'Well'. YouGov poll, roughly monthly of 1623–3326 adults in Great Britain.

Produce the figure with the following command:

```
plot(boris.well.zoo, xlab="Date",
    ylab="Boris Johnson Approval Rating %")
```

and then plot the autocorrelation and partial autocorrelation plots for the Boris data:

```
oldpar <- par(mfrow=c(1,2), pty="s")

acf(coredata(boris.well.zoo), main="ACF: Boris Well %")
pacf(coredata(boris.well.zoo), main="PACF: Boris Well %")

par(oldpar)
```

The plot is shown in Figure 4.

The partial autocorrelation 'cuts off' at lag 1 and hence it looks like here that the approval rating series could be well-modelled by a kind of AR(1) process with AR parameter of $\alpha = 0.81$, roughly. The `coredata` function in the ACF and PACF commands are needed because the Boris approval rating series is not regularly spaced — `coredata` strips out the dates and just leaves the sequential

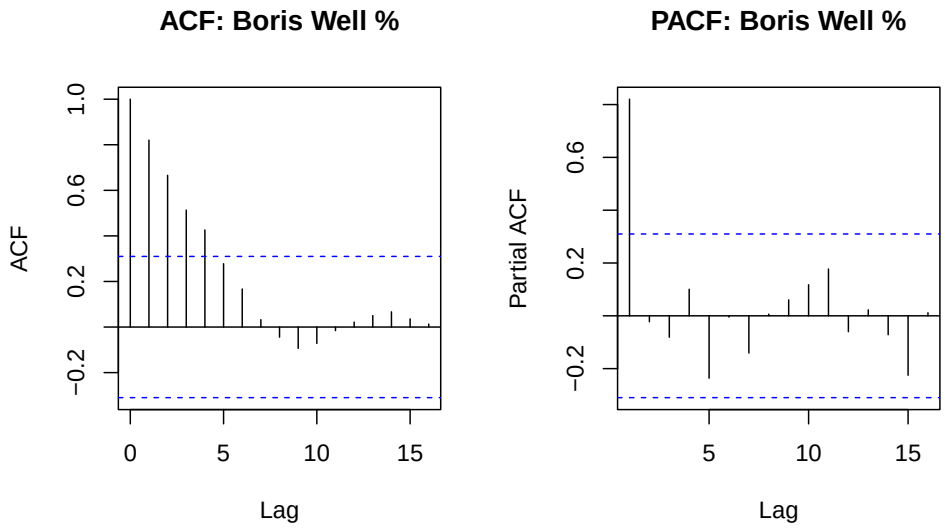

Figure 4: Autocorrelation (left) and partial autocorrelation (right) of the Boris approval time series.

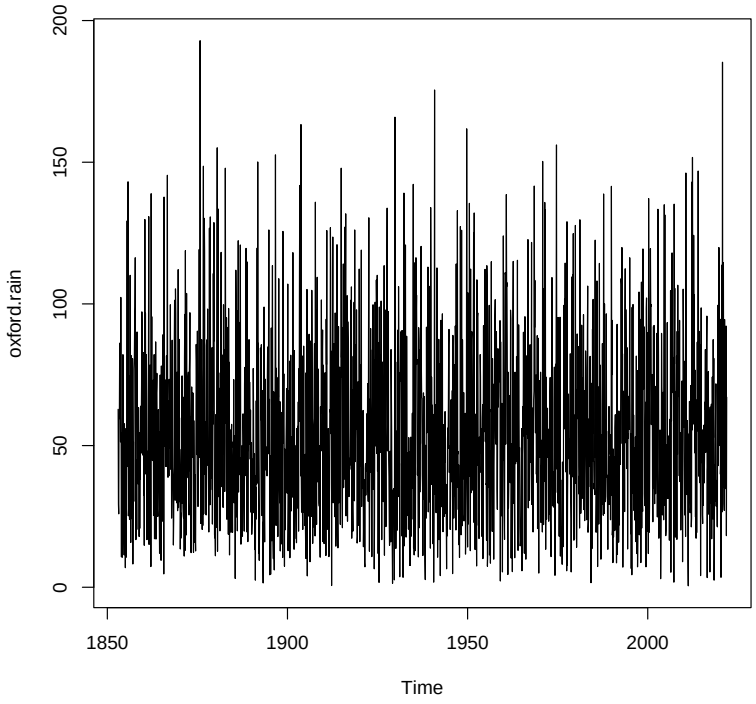

Figure 5: Oxford monthly rainfall data from January 1853 to December 2021.

observations. So, the coredatared observations might form an AR(1) process, but not the original, as its irregularly spaced.

1.3 Oxford Rain Time Series

The Oxford rain time series was obtained from the UK Met Office Historic station data set

```
https://www.metoffice.gov.uk/research/climate/
    maps-and-data/historic-station-data
```

and our set has monthly rainfall totals from January 1853 to December 2021, which consists of 2028 observations. A plot of the time series can be produced by

```
plot.ts(oxford.rain)
```

which is depicted in Figure 5. The autocorrelation and partial autocorrelation plots are shown in Figure 6. Both autocorrelation and partial autocorrelation values are small, but the partial autocorrelation plot indicates some short term autocorrelation, perhaps an AR(1) or AR(2) process.

For another way of looking at the Oxford rain series the following commands compute the spectral density estimate (or spectrum) for the Oxford rain series

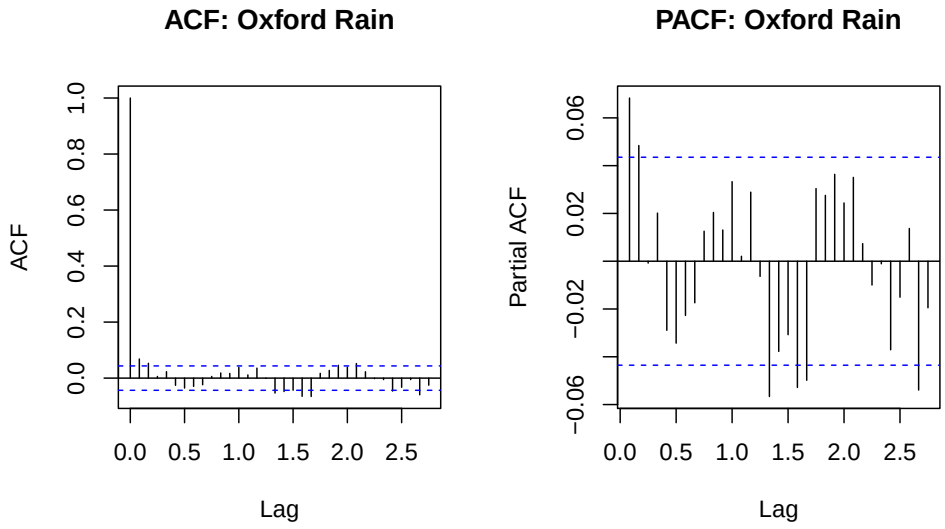

Figure 6: Autocorrelation (left) and partial autocorrelation (right) of Oxford rain data.

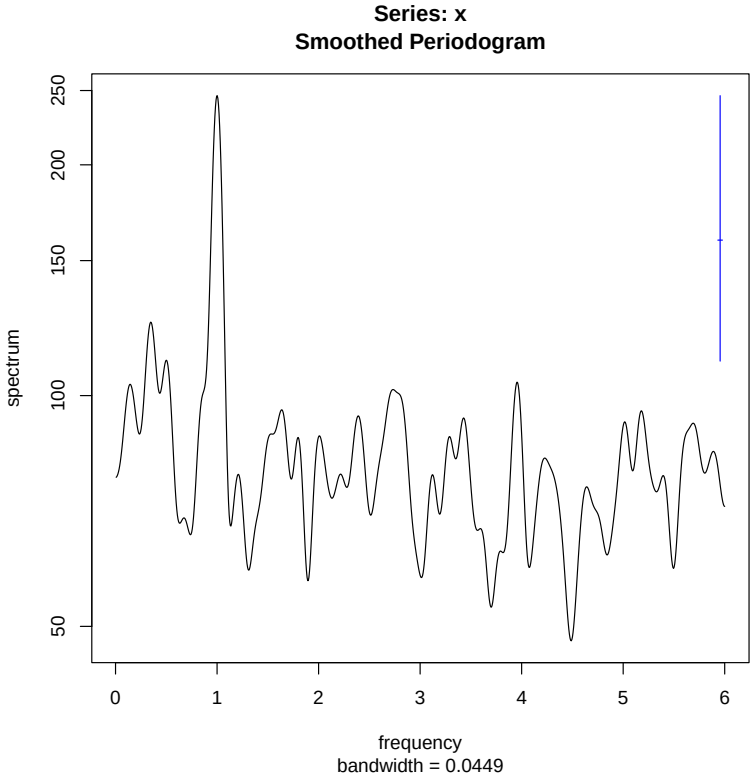

Figure 7: Spectrum of Oxford rain data. The units of frequency are in 'oscillations per year'. So, the largest peak is at one oscillation per year.

```
stats::spectrum(oxford.rain, spans=c(17,15,13,11))
```

and the output from this command is shown in Figure 7. The spectrum shows a clear peak at $t = 12$ months, which is what you'd expect for data that shows annual seasonality and it is a much clearer peak than anything that can be seen in the autocorrelation plots in Figure 6.

1.4 BabyECG Series

The following command plots the BabyECG series

```
plot.ts(BabyECG)
```

This is an ECG recording from a 66 day old infant recorded at the Bristol Institute of Child Health, Royal Hospital for Sick Children. The observations were taken every 16 seconds recorded from 21:17:59 until 06:27:18 and are described in Nason et al. (2000). The plot from these commands is shown in Figure 8. Already, just from the plot, it seems unlikely that this time series would be stationary or second-order stationary. There appear to be distinct 'pulses' in behaviour and these are related to baby movements and/or the baby moving in and out of different sleep states.

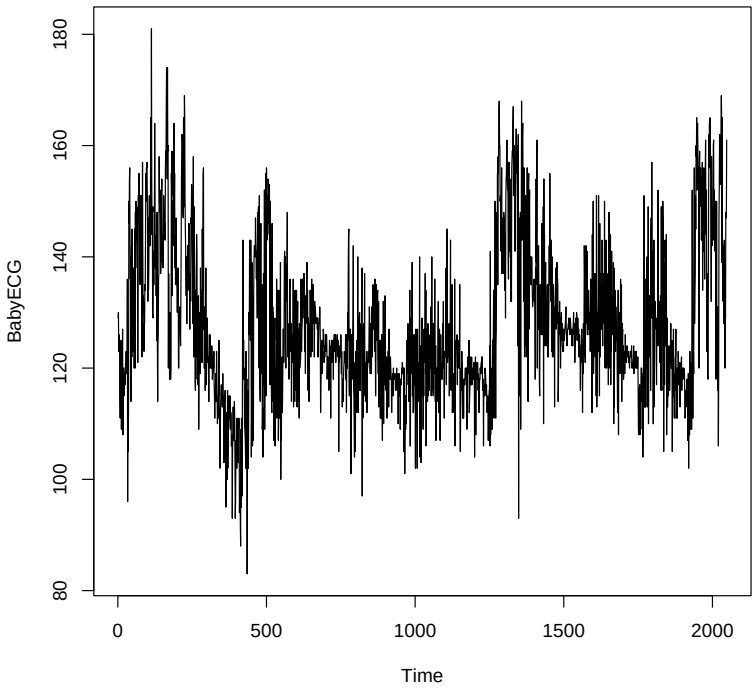

Figure 8: BabyECG time series

Let us examine the autocorrelation function of the BabyECG series during time periods. The first period is the first 256 observations (just over one hour in duration) and the second period is the observations ranging from $t = 1001$ to $t = 1256$, taken about four and a half hours later.

The commands to do this are:

```
oldpar <- par(mfrow=c(1,2), pty="s")

acf(BabyECG[1:256], main="a.")
acf(BabyECG[1001:1256], main="b.")

par(oldpar)
```

and the plots are shown in Figure 9. The key point here is that the *plots are very different*. The left-hand plot shows significant autocorrelation, but the right-hand plot shows hardly any. This is direct evidence of second-order NON-stationarity.

1.5 Haar concat simulation

The commands

```
set.seed(345)
plot.ts(HaarConcat())
```

plots the concatenated Haar simulated process as described in the associated earlier talk in this course. These commands result in the plot shown in Figure 10. We know that this process is designed to deliberately not be second-order stationary. To show this in more detail the left- and right-hand plots of Figure 11 show the autocorrelation functions of the HaarConcat series on the first and last 128 observations. They are, as you'd expect, since the series is not second-order stationary, very different.

2 Tests of Stationarity

This section primarily uses the `hwtos2` function from the `locits` package to test time series for second-order stationarity. The `hwtos2` function only accepts time series whose length is a power of two.

2.1 Haar concat simulation

The Haar concatenation already has a power of two (512) number of observations. It is a good time series to run the test of stationarity on, as well already know it is from a nonstationary process. So, we can run `hwtos2` directly on this time series as follows:

```
set.seed(345)
x <- HaarConcat()
hwtos2(x)
```

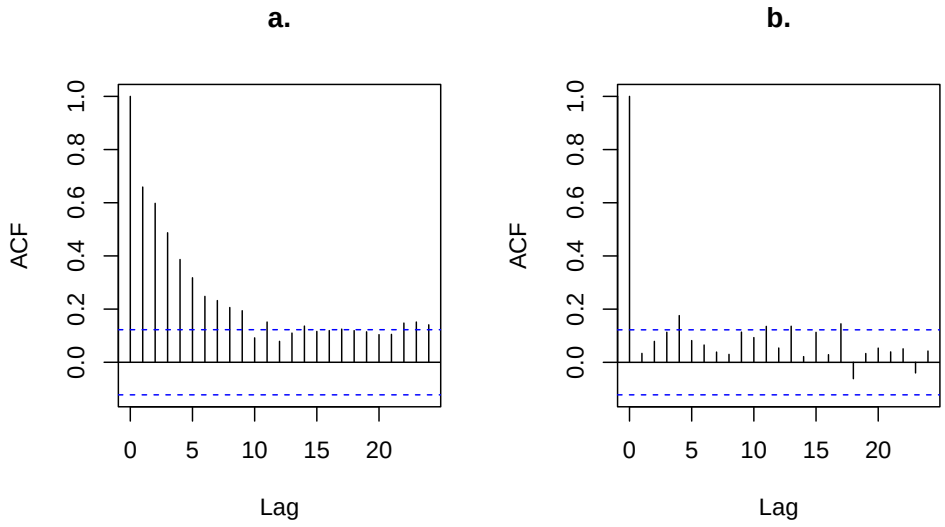

Figure 9: BabyECG autocorrelation functions for two time periods: (a.) from $t = 1$ to $t = 256$; (b.) from $t = 1001$ to $t = 1256$.

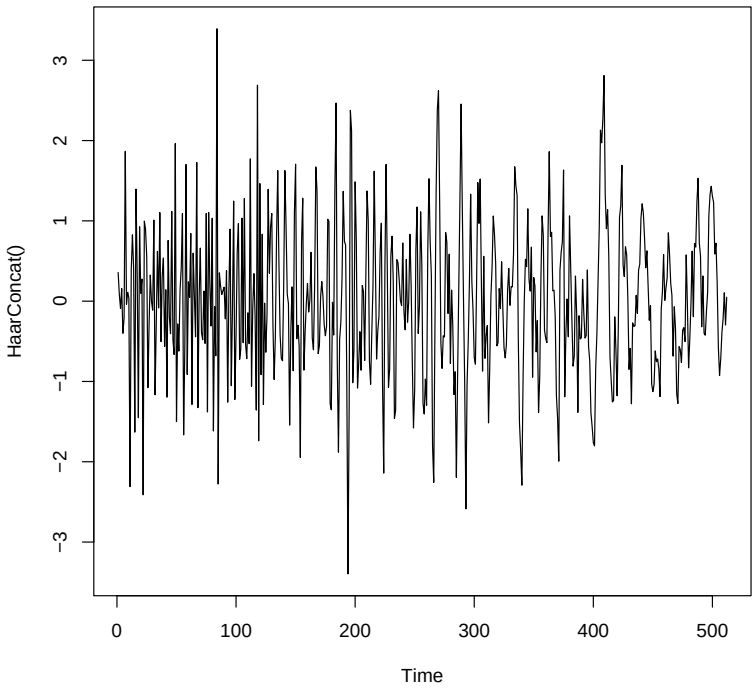

Figure 10: Concatenated Haar simulation.

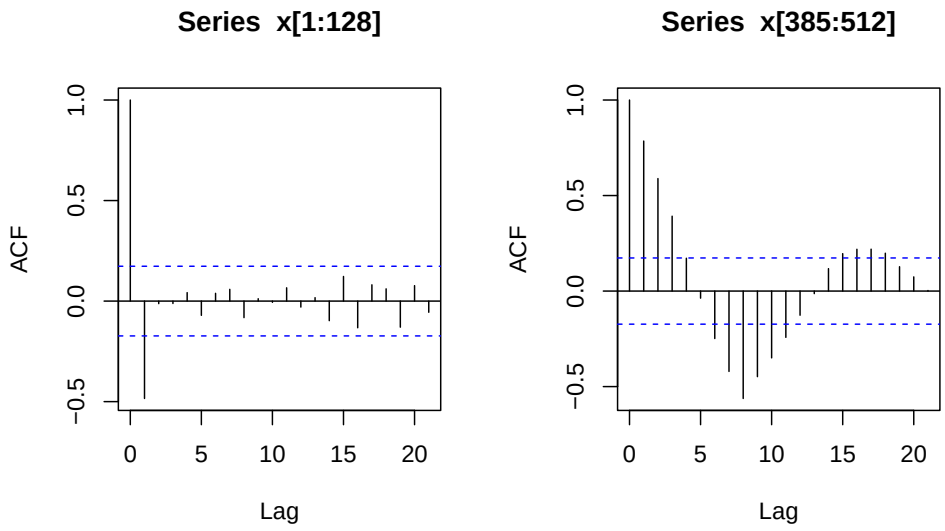

Figure 11: Autocorrelation functions of the first (left) and last (right) 128 observations from the HaarConcat series.

The output from this set of comments is:

```
  8    7    6    5    4    3
Class 'tos' : Stationarity Object :
       ~~~~  : List with 9 components with names
              nreject rejpval spvals sTS AllTS AllPVal alpha x xSD


summary(.):
----------
There are  186  hypothesis tests altogether
There were  6  FDR rejects
The rejection p-value was  0.0008956097
Using Bonferroni rejection p-value is  0.0002688172
And there would be  5  rejections.
Listing FDR rejects... (thanks Y&Y!)
P: 5 HWTlev:  0  indices on next line...[1] 1
P: 5 HWTlev:  1  indices on next line...[1] 2
P: 5 HWTlev:  2  indices on next line...[1] 4
P: 5 HWTlev:  4  indices on next line...[1] 13
P: 6 HWTlev:  0  indices on next line...[1] 1
P: 8 HWTlev:  0  indices on next line...[1] 1
```

This output shows that the test examined 186 separate hypothesis tests and, in the multiple hypothesis test situation, applied a false-discovery rate approach and decided that 6 tests reached a level of significance that enabled the null hypothesis to be rejected (and even the conservative Bonferroni approach rejected 5).

Exercise

Run the `hwtos2` test of stationarity function on `x` again, but this time add the argument `filter.number=2` or any other integer between two and ten. See what the result of the hypothesis test is.

The remaining text indicates at which wavelet resolution levels and locations the individual tests were rejected. The test is applied to each level of the wavelet periodogram — which level is indicated by the 'P' in the text output. A full Haar wavelet transform is applied to those levels (controlled by the `lowlev` argument of `hwtos2`) and then, the scales and locations of those that are significant by each individual test are listed.

For example, with the above output, on the wavelet periodogram level $P = 5$, the full Haar wavelet transform was applied (but only scales coarser than argument `WTscale` are examined) and on the secondary Haar wavelet transform, indices at scales $0, 1, 2, 4$ were found to be significant at indices $1, 2, 4, 13$ respectively.

It is easier to see the scale and location of these rejections in a plot constructed by the following commands:

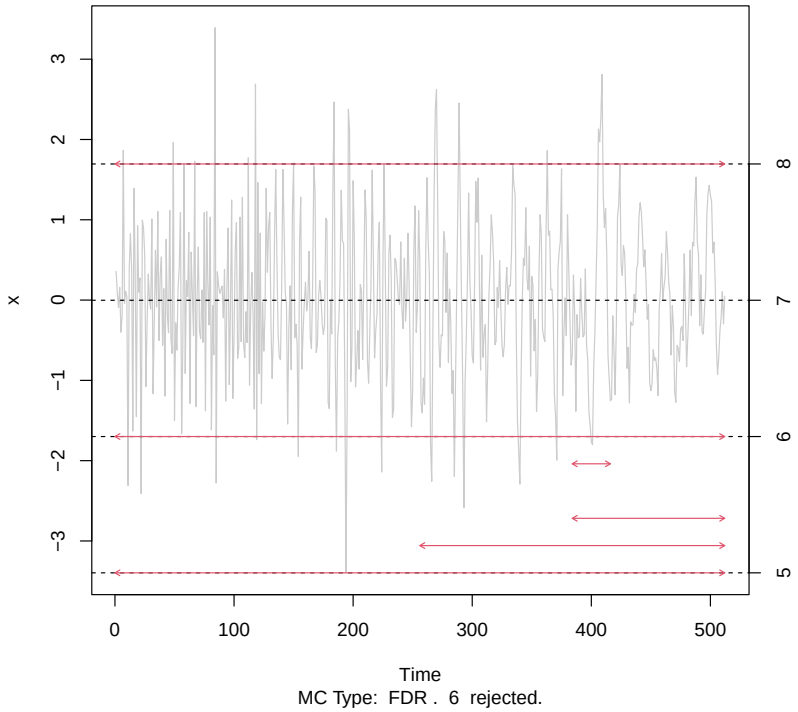

Figure 12: hwtos2 test of stationarity plot for the Haar concat time series, indicating location and scale of nonstationarities.

```
set.seed(345)
x <- HaarConcat()
plot(hwtos2(x))
```

which results in the plot shown in Figure 12. The red double-headed arrows indicate the scale (right-hand axis), extent and location of the nonstationarities. So, following the significant tests picked out of the text above:

- The `HWTlev` of 0 at index 1 is the coarsest scale Haar wavelet, which covers the whole extent of the time series and there is only one Haar wavelet are this scale, so its index is 1. This corresponds to the bottom-most red double-headed arrow in Figure 12 which covers the whole extent of the time series. This arrow and test indicates that the test judges there to be nonstationarities affecting the whole of the series.

- `HWTlev` of 1 at index 2 is the second-from-bottom red double-headed arrow. This wavelet is half the width of the coarsest wavelet, and hence covers half the extent of the series. There are only two indices at this scale: 1 and 2, and since this wavelet's index is 2 the arrow is located on the right-hand side of the plot. This test indicates that there are identified nonstationarities in the second-half of the series.

- `HWTlev` of 2 at index 4 is the third-from bottom red double-headed arrow. This wavelet is one-quarter of the width of the coarsest wavelet, and hence covers one-quarter of the extent of the series. There are four indices at this scale, $1, 2, 3, 4$ and since this wavelet's index is 4, the arrow is located at the far right-hand side of the plot. This test indicates that there are nonstationarities operating at scale $1/4$ of the whole extent at that location.
- Similarly, the `HWTlev` of 4 at index 13 is the last smaller double-headed red arrow before the level 6 (on the right-hand side axis) at location 13 out of 16.

2.2 BabyECG test of stationarity

Run the `hwtos2` test of stationarity on the BabyECG series

```
plot(hwtos2(BabyECG))
print(hwtos2(BabyECG))
```

The output is

```
  10    9    8    7    6    5    4    3
Class 'tos' : Stationarity Object :
       ~~~~  : List with 9 components with names
               nreject rejpval spvals sTS AllTS AllPVal alpha x xSD


summary(.):
----------
There are  504  hypothesis tests altogether
There were  9  FDR rejects
The rejection p-value was  0.0008239662
Using Bonferroni rejection p-value is  9.920635e-05
And there would be  7  rejections.
Listing FDR rejects... (thanks Y&Y!)
P: 4 HWTlev:  5  indices on next line...[1] 31
P: 5 HWTlev:  5  indices on next line...[1] 31
P: 6 HWTlev:  1  indices on next line...[1] 1
P: 6 HWTlev:  5  indices on next line...[1] 31
P: 7 HWTlev:  1  indices on next line...[1] 1
P: 7 HWTlev:  2  indices on next line...[1] 4
P: 8 HWTlev:  1  indices on next line...[1] 1
P: 8 HWTlev:  2  indices on next line...[1] 4
P: 9 HWTlev:  1  indices on next line...[1] 1
```

and the plot is shown in Figure 13. This time series rejects the null hypothesis of second-order stationarity.

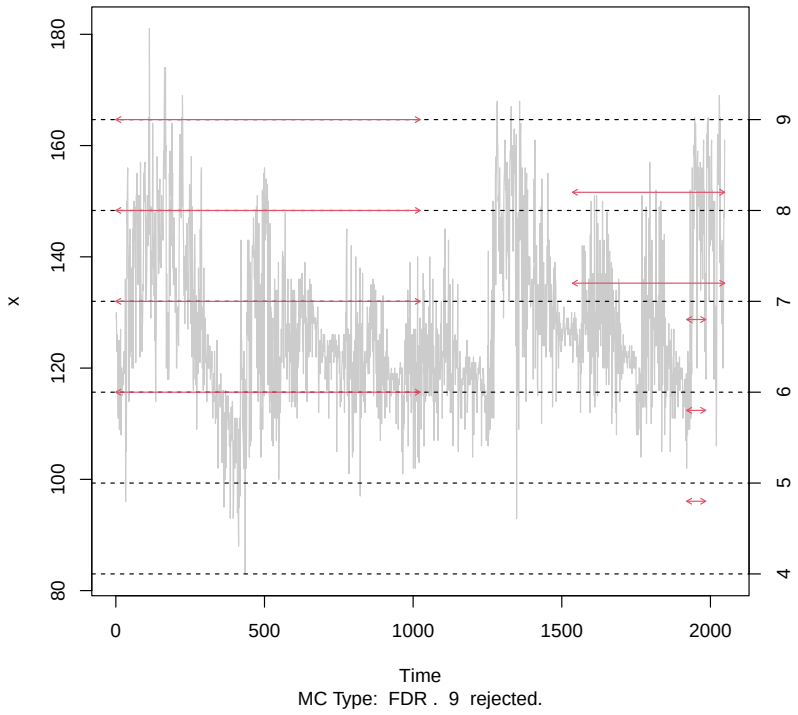

Figure 13: hwtos2 test of stationarity plot for BabyECG time series.

2.3 Oxford Rain

The length of the Oxford rain time series is 2028 and not a power of two. The next largest power of two is 2048, only another twenty observations. So, let us run the test of stationarity on an extended version of the Oxford rain series, by adding twenty identical observations equal to the sample mean of the series to the beginning of the series. In other words

```
oxford.rain.extend <- c(rep(mean(oxford.rain), 20), oxford.rain)
```

and now perform the test of stationarity:

```
hwtos2(oxford.rain.extend)
```

which gives output

```
  10    9    8    7    6    5    4    3
Class 'tos' : Stationarity Object :
       ~~~~  : List with 9 components with names
              nreject rejpval spvals sTS AllTS AllPVal alpha x xSD


summary(.):
----------
There are  504  hypothesis tests altogether
There were  0  FDR rejects
No p-values were smaller than the FDR val of:
Using Bonferroni rejection p-value is  9.920635e-05
And there would be  0  rejections.
```

Since there are no rejections, we do not have evidence to reject H_0 at this point. So, until we receive evidence to the contrary we will assume the series is second-order stationary.

2.4 Truncation and Padding

Some wavelet-related functions require time series to have a dyadic length (e.g. $64, 128, 256, 512$, etc). Most time series are not a dyadic length. To enable us to use some of these length-restricted wavelet functions we can try one of two things:

1. truncate the series,
2. pad out the series with values.

in both cases making the length of the modified series a power of two. Truncation is easy. Just work out what the largest power of two is that is less than the current length of the series and use the R `:` operator to select those values. E.g. if the series length if 267, then the largest power of two less than that is 256. So, we can create a new series from the old one by truncating as `series[1:256]` or something selection of 256 consecutive values.

Padding can be a bit trickier. Basic locally stationary series are assumed to have a constant mean (often zero). So, it's possible to augment the series with values at that mean value. Usually, it's a good idea to augment the series by adding new values to the beginning of the series. This is because one often wishes to forecast values from a time series and we don't want to do this if we've added fictitious values to the end of the series. An example of padding was carried out for the Oxford Rain series in the previous section where we added twenty new values to the front of the series, those values being the mean of the series before padding.

In general, it's best not to truncate or pad too many values. It's also a good idea to not rely to heavily on estimates computed near to where the series was padded. E.g. for a wavelet spectral estimate for the extended Oxford Rain it'd be wise to downweight your confidence of the estimate near to the left-hand end of the series, as it's based on padded data.

3 Wavelet Spectrum Estimation

3.1 Concatenated Haar process

We can plot the estimated spectrum from a realization from the concatenated Haar process with the following commands:

```
set.seed(345)
#
# Generate concatenated Haar series
#
x <- HaarConcat()
#
# Calculate "best" bandwidth for spectrum estimation
#
HC.bw <- AutoBestBW(x)
#
# What is the "best" bandwith?
#
cat("AutoBestBW is ", HC.bw, "\n")
#
# Compute the evolutionary wavelet spectrum using Haar analysis
# wavelets, and using the "best" bandwidth for smoothing.
#
HC.S <- ewspec3(x, filter.number=1, family="DaubExPhase",
    binwidth=HC.bw)$S
#
# Plot the smoothed evolutionary wavelet spectrum
#
plot(HC.S)

# MORE COMMANDS ON NEXT PAGE...
```

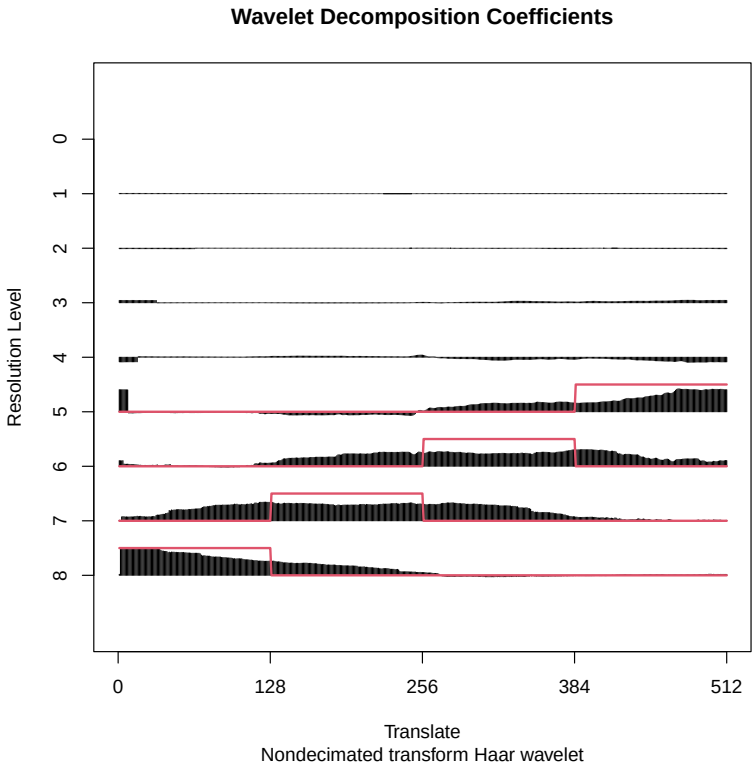

Figure 14: Evolutionary wavelet spectral estimate from a realization from the concatenated Haar process. The true spectrum is overlaid in red.

```
# Compute the true underlying spectrum
#
hc1 <- c(rep(0.5, 128), rep(0, 512-128))
hc2 <- c(rep(0, 128), rep(0.5, 128), rep(0, 256))
hc3 <- rev(hc2)
hc4 <- rev(hc1)

# Plot the true, underlying spectrum
#
lines(1:512, 1+hc1, lwd=2, col=2)
lines(1:512, 2+hc2, lwd=2, col=2)
lines(1:512, 3+hc3, lwd=2, col=2)
lines(1:512, 4+hc4, lwd=2, col=2)
```

If you run these commands, you'll find that the "best" bandwidth is 305 and the estimated spectrum, and underlying true spectrum (in red) is shown in the plot given in Figure 14. The spectral peaks are in roughly the right place in Figure 14, but the power is spread out too much horizontally. This is because the automatic choice of smoothing binwidth (also known as bandwidth or smoothing parameter) is too large (and it generally overestimates.

Let's repeat the spectral analysis, but choose a smaller bandwidth, say 100.

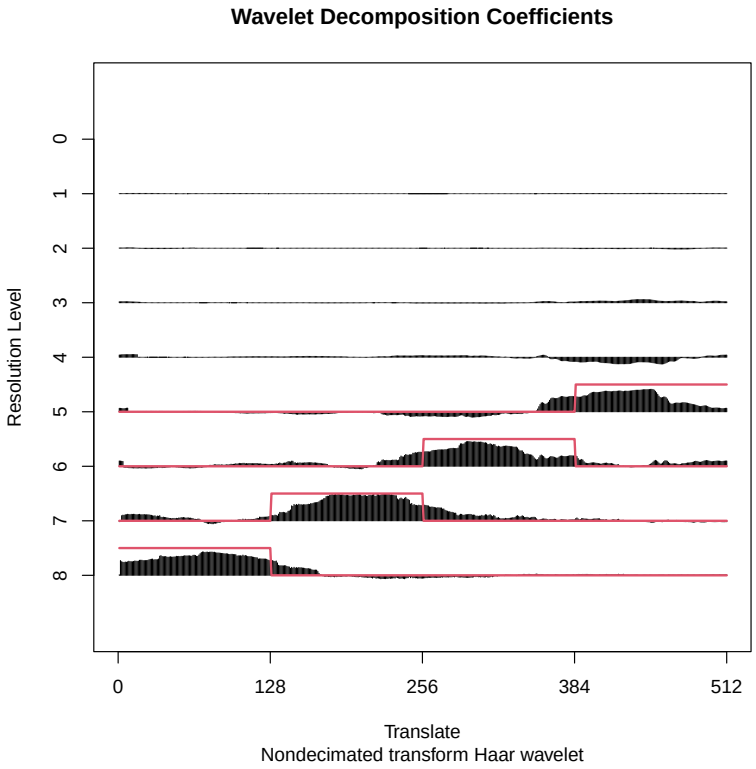

Figure 15: Evolutionary wavelet spectral estimate from a realization from the concatenated Haar process with bin width of 100. The true spectrum is overlaid in red.

```
HC.S <- ewspec3(x, filter.number=1, family="DaubExPhase",
    binwidth=100)$S
#
# Plot the smoothed evolutionary wavelet spectrum
#
plot(HC.S)

hc1 <- c(rep(0.5, 128), rep(0, 512-128))
hc2 <- c(rep(0, 128), rep(0.5, 128), rep(0, 256))
hc3 <- rev(hc2)
hc4 <- rev(hc1)

lines(1:512, 1+hc1, lwd=2, col=2)
lines(1:512, 2+hc2, lwd=2, col=2)
lines(1:512, 3+hc3, lwd=2, col=2)
lines(1:512, 4+hc4, lwd=2, col=2)
```

This gives the plot shown in Figure 15 and the power looks to be better localised where it "should" be.

Table 1: Sleep states and their interpretation.

Sleep State	Meaning
1	Quiet sleep
2	Between Quiet and Active
3	Active sleep
4	Awake

Exercise

For the exercises below, your time series needs to be of dyadic (power of two, i.e. $128, 256, 512$, etc) length. This is easy to achieve for the simulation, but for a real time series you might need to cut part of the series or pad it with some other values.

1. Use your own time series to test whether it is stationary or not second-order stationary. Calculate the evolutionary wavelet spectrum for your series.

2. Simulate a stationary series (e.g. use `arima.sim`) and then test it for stationarity. Calculate the evolutionary wavelet spectrum of the simulated series.

3.2 BabyECG wavelet spectrum

We can plot the evolutionary wavelet spectrum of the BabyECG time series with the following commands

```
best.bw <- AutoBestBW(BabyECG, filter.number=10)

plot(ewspec3(BabyECG, filter.number=10, binwidth=best.bw)$S)
```

and these result in the plot in Figure 16. The spectrum in Figure 16 does not look that impressive (!) nor particularly nonstationary at the finer to medium levels. However, as well as the BabyECG time series, the researchers also recorded the sleep state of the infant. This is an integer between 1 and 4 with the following meanings shown in Table 1. Let us plot the 10th scale from the wavelet spectum and superimpose the sleep state (suitably scaled) on top using the following:

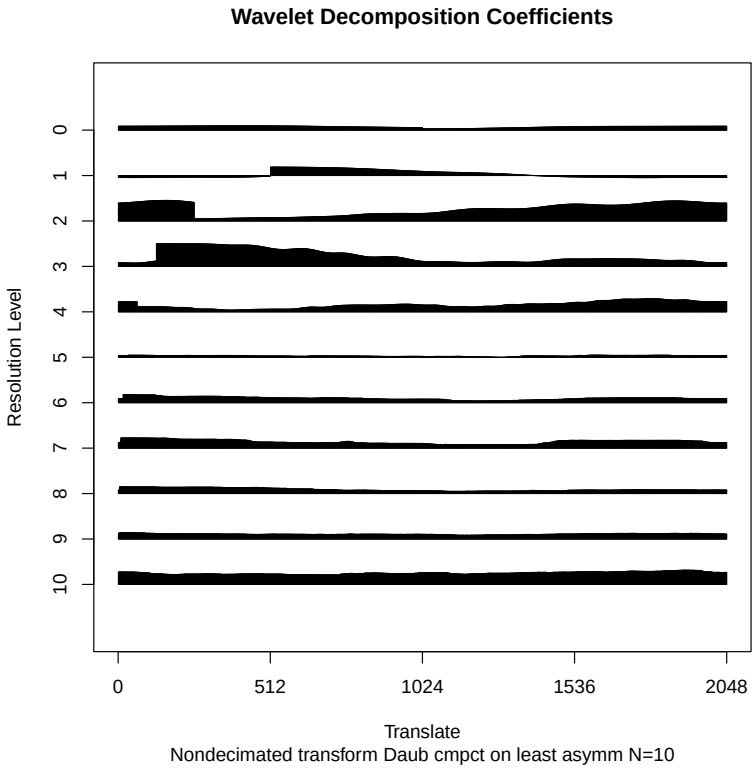

Figure 16: Evolutionary wavelet spectral estimate of the BabyECG time series using Daubechies' wavelet 10.

However, if we extract the finest (10th) level and superimpose the BabySS variable

```
best.bw <- AutoBestBW(BabyECG, filter.number=10)

baby.S <- ewspec3(BabyECG, filter.number=10, binwidth=best.bw)$S

baby.fine <- accessD(baby.S, level=10)

plot(1:2048, baby.fine, type="l")

scaled.BabySS <- min(baby.fine) +
    (max(baby.fine)-min(baby.fine))*(BabySS-1)/3

scaled.axis <- min(baby.fine) +
    (max(baby.fine)-min(baby.fine))*(0:3)/3

lines(1:2048, scaled.BabySS, lty=2, col=2)

axis(4, at=scaled.axis,
    lab=c("Quiet", "Quiet/Active", "Active", "Awake"),
    col=2, col.ticks=2, col.axis=2)
```

This results in the plot shown in Figure 17. The plot shows that there seems to be some association between the baby's heart rate and its sleep state (this was actually exploited further in Nason et al. (2001)).

Exercise

Compute and plot the evolutionary wavelet spectrum of the `oxford.rain.extend` data set.

4 More Advanced Stationarity Tests

We looked at the `hwtos2` test of stationarity in Section 2. This evaluated stationarity by examining a set of Haar wavelet coefficients of a wavelet periodogram (blurred spectral estimate). There are generalizations of wavelets, called wavelet packets. Wavelet packets form a large collection of bases, which are similar to wavelets. However, they have different oscillatory properties. It is possible to select a basis from a wavelet packet library and one such basis is the wavelet basis. A full exposition is beyond the scope of this document. However, to show how the stationarity test works we apply an example to the BabyECG data.

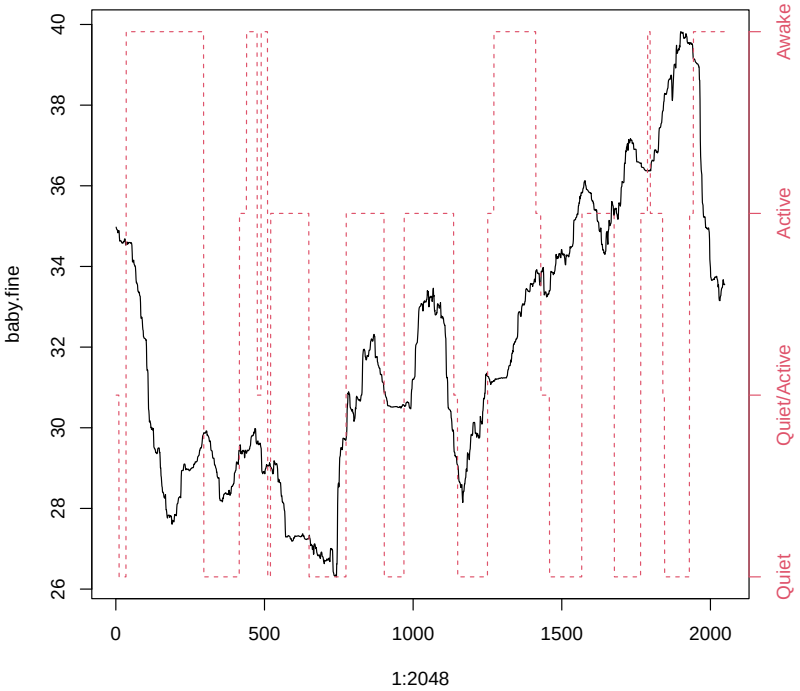

Figure 17: Evolutionary wavelet spectrum at level 10 with BabySS sleep state superimposed in red, with axis to the right.

Issue the following commands

```
#
# Load parallel library and set number core on machine
#
library("parallel")
options(mc.cores=6)
#
# Compute test of stationarity using level 7 wavelet packets
set.seed(1026)
Baby.7 <- BootWPTOS(BabyECG, levs=c(rep(7,15)), indices=c(1:15),
    lapplyfn=mclapply)
print(Baby.7)
```

The output is

```
WPBootTOS test of stationarity

data:  BabyECG
= 38195, p-value = 0.02
```

Hence, the p-value of this test of stationarity is 0.02, which is certainly significant at the 5% level (your p-value might vary slightly).

Two key arguments of the test of stationarity here are the `levs` and `indices` arguments. These control which wavelet packets are used in the testing. The `levs` is just the same as the scale levels in the regular wavelet transform. Indices correspond to the particular wavelet packet within that scale. For every scale and index of 0 corresponds to father wavelets and 1 to mother wavelets. If the length of the time series is $n = 2^J$, then the finest scale is at $j = J - 1$ and there are two packets at index 0 and 1 (mother and father). At scale $j - 2$ (one coarser than $J - 1$) there are four packets labelled 0, 1, 2 and 3.

The BabyECG series is of length $2048 = 2^1 1$ so $J = 11$. Hence at acale $j = 10$ there are two packets, at scale $j = 9$ there are four, at scale $j = 8$ there are eight, and at scale $j = 7$ there are 15, indexed 1 to 15.

We can pick on any particular wavelet packet for a more detailed analysis. Let us select the third wavelet packet and compute the test only with respect to that.

```
plot(WPTOSpickout(BabyECG, level=7, index=3))
```

and the associated plot is shown in Figure 18. This plot is similar to the one above for the `hwtos2` test, but this one is for a wavelet packet, not a wavelet. The benefit of using wavelet packets is that they can pick up different kinds of non-stationarity than the wavelets (or Fourier) alone and give a more diverse way of looking for non-stationarities.

Exercise

Try using `WPTOSpickout` on your own time series — try some different levels and indices and plot the results.

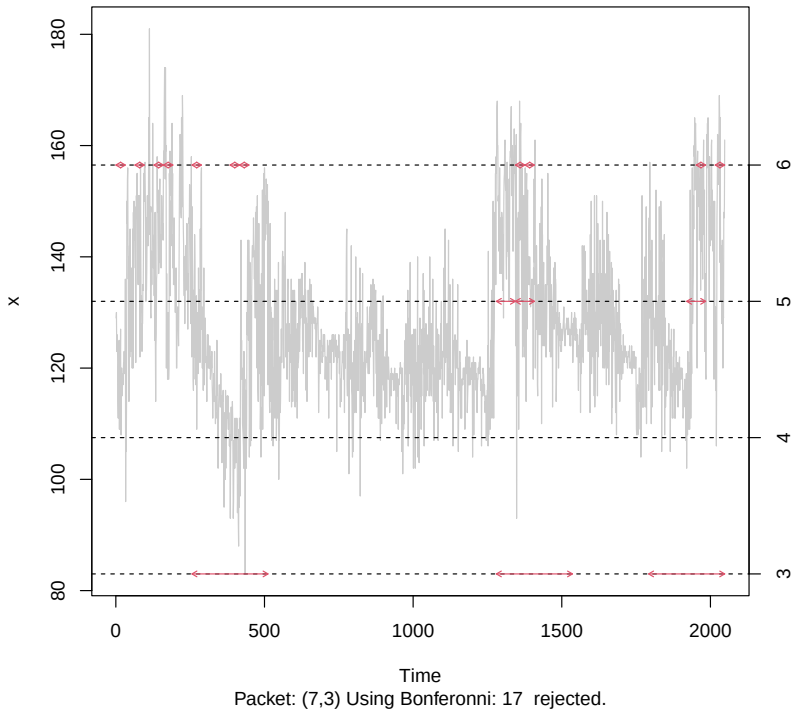

Figure 18: Wavelet packet test for packet 3 at level 7 for the BabyECG data.

A Useful Commands Summary

`BootWPTOS::BootWPTOS` calculate omnibus wavelet packet bootstrap test of stationarity

`BootWPTOS::WPTOSpickout` do specific wavelet packet test of stationarity and generate associated plot (with plot function on result).

`locits::AutoBestBW` calculate reasonable bandwidth for evolutionary wavelet spectrum estimation (and routines that make use of that0.

`locits::hwtos2` Test for second-order stationarity based on Haar wavelet analysis

`locits::ewspec3` Calculate evolutionary wavelet spectrum

`wavethresh::HaarConcat` Produces simulated concatenated Haar process

References

G. P. Nason, R. von Sachs, and G. Kroisandt. Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *J. R. Statist. Soc.* B, 62: 271–292, 2000.

G. P. Nason, T. Sapatinas, and A. Sawczenko. Wavelet packet modelling of infant sleep state using heart rate data. *Sankhyā* B, 63:199–217, 2001.

