Nonstationary Time Series: Spectral Estimation and Nonstationarity Tests (4)

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.

2 Local ACVF for Earthquake

Here, we use ACVF as an abbreviation for autocovariance function. The `locits` package contains a functions for various sorts of local autocovariance (ACVF) and autocorrelation (ACF) estimation as well as a test for second-order stationarity.

2.1 Earthquake ACVF & ACF

The Earthquake data comes from the `astsa` package. It can be viewed with the following commands

```
plot.ts(Earthquake, ylab="Earthquake", xlab="Time")
```

shown in Figure 1. The series looks very nonstationary. For example, the variance seems to change over time. We can use the methods from the earlier session to run a test of second-order stationarity using the `hwtos2` function from Nason (2013) via

```
plot(hwtos2(Earthquake))
```

which results in Figure 2 indicating multiple nonstationarities with the red double-headed arrows. Clearly, the statistical properties of the Earthquake series are changing over time. What about quantities such as the (local) autocovariance, autocorrelation and partial autocorrelation?

Let us use the `lacf` function to estimate the local autocorrelation. We can compute this using the commands

```
plot(lacf(Earthquake), lags=0:8)
```

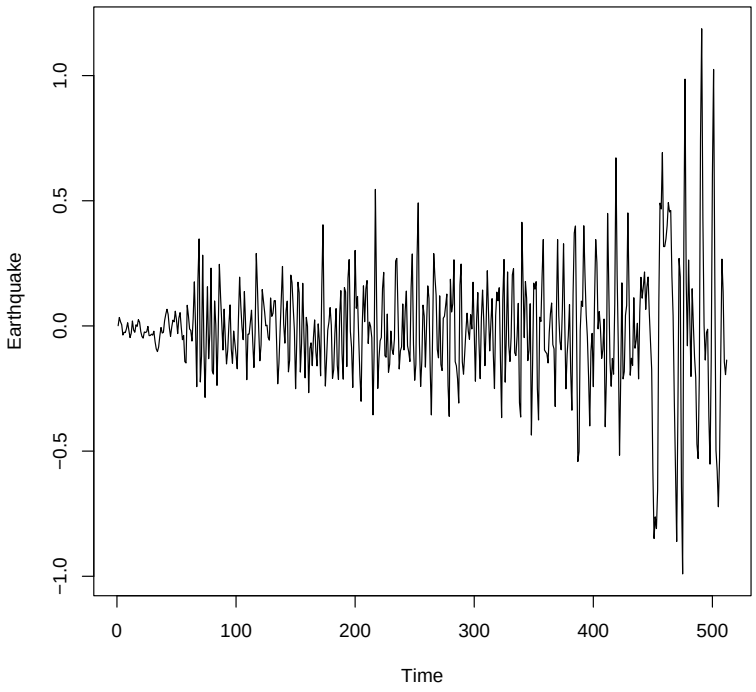

Figure 1: Plot of Earthquake time series.

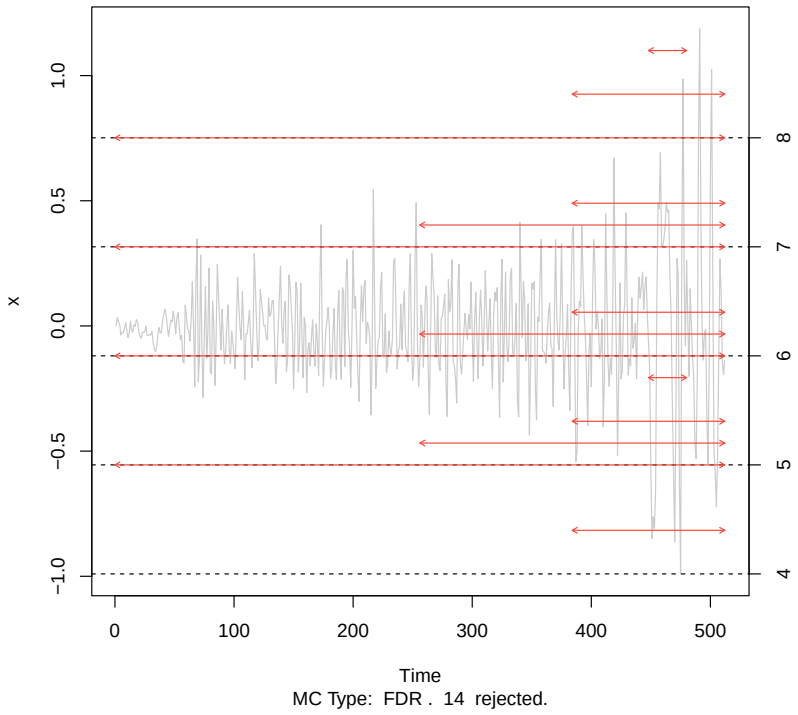

Figure 2: Test of stationarity plot of earthquake data

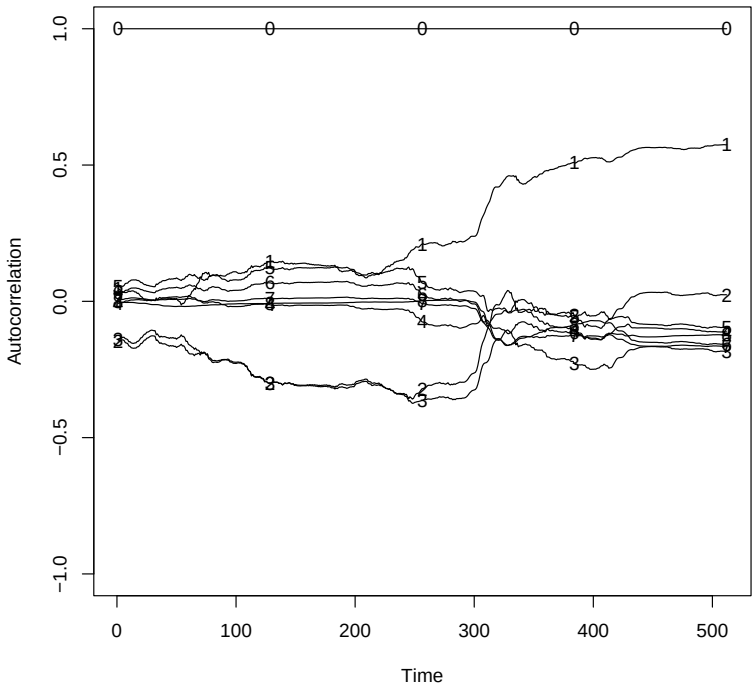

Figure 3: Local autocorrelation of Earthquake series

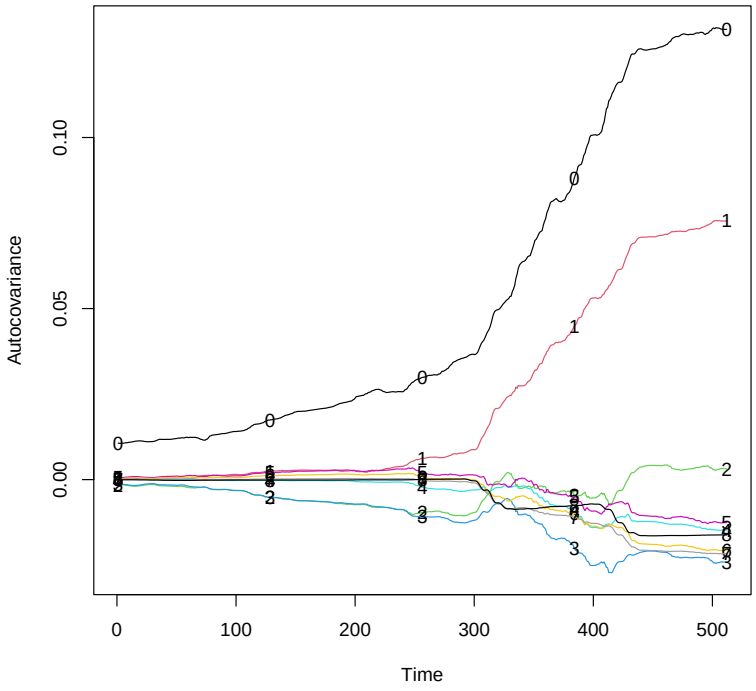

Figure 4: Local autocovariance of Earthquake series

which results in Figure 3. It is very clear that the autocorrelation is changing over time. For example, focus on the lag one autocorrelation, which increases from near zero at the beginning of the series to nearly 0.5 at the end and is more rapidly increasing after $t = 250$. The autocorrelations at lags two and three are also changing in lockstep. The higher lags' autocorrelations are smaller and changing less. The lag zero autocorrelation is always equal to one, as with stationary series.

What about local auto*covariance*? This can be plotted using a similar command, but adding in the `plotcor=FALSE` argument to the plotting command via

```
plot(lacf(Earthquake), lags=0:8, plotcor=FALSE, lcol=1:9)
```

which results in Figure 4. We have also used the `lcol` argument to the `lacf` plotting function to generate lines of different colours to make the individual lags' autocovariances stand out. Figure 4 shows that the local variance (lag zero local autocovariance) increases from about 0.01 to 0.13 across the extent of the series and the lag one tracks this. Overall, this series seems to be nonstationary, exemplified by the changing local autocovariance and autocorrelation as well as the hypothesis test.

Exercise

Select your own time series. Remember some commands here, such as `lacf`, can only analyse time series that have a power of two length.

1. Calculate and plot the local autocorrelation of your series.
2. Calculate and plot the local auto*covariance* of your series.
3. (Optional) play around with some of the options of the `lacf` function, such as the type of wavelet, or the smoothing parameter `binwidth`.

2.2 Local ACVF with confidence

The regular R autocovariance function is fairly rudimentary when it comes to confidence intervals for time series. We can ask `locits` to give us a local autocovariance estimate at different time locations and calculate associated confidence intervals for them. For example, the `Rvarlacf` function can calculate the local autocovariance around $t = 100$, with approximate 95% confidence intervals, and can be plotted by

```
plot(Rvarlacf(Earthquake, nz=100, var.lag.max=10), type="acf",
    plotcor=FALSE)
```

which results in Figure 5. The 95% confidence intervals are drawn as red shaded bars. Figure 5 shows the local autocovariance (in black) and (by default) 95% confidence intervals individually for each lag. From the plot, it can seen that the autocovariance for lag one, and three and higher (up to ten, but *probably* higher) are not significantly different from zero. The lag two autocovariance does seem to be significantly different from zero.

Let us next compute a similar plot, but centred around $t = 400$ computed using the following command:

```
plot(Rvarlacf(Earthquake, nz=400, var.lag.max=10), type="acf",
    plotcor=FALSE)
```

which results in Figure 6. Figure 6 clearly shows that all autocovariances up to and including lag eight are significantly different from zero.

That the two (local) autocovariances at times $t = 100$ and $t = 400$ are so different is strong evidence that the Earthquake time series is not second-order stationary, which agrees with our assessment in Section 2.1.

Exercise

With your own time series, compute the local autocovariance in the format of a regular ACVF plot for different time locations using the `Rvarlacf` function and generate local confidence intervals. Explore different options for the `Rvarlacf` function, such as wavelet filter, or the number of lags you wish to compute confidence intervals for (`var.lag.max`).

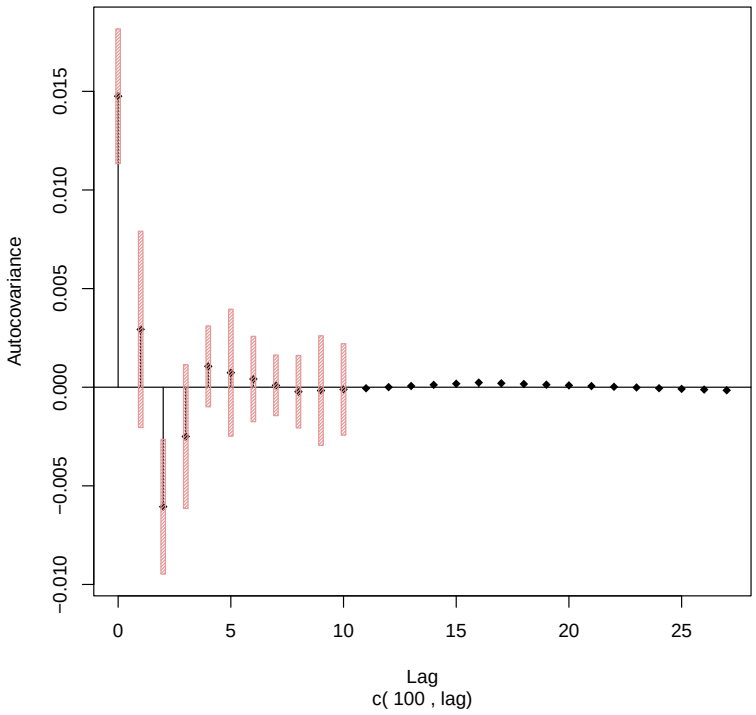

Figure 5: Local autocovariance of Earthquake series around $t = 100$, with approximate 95% confidence intervals (red bars).

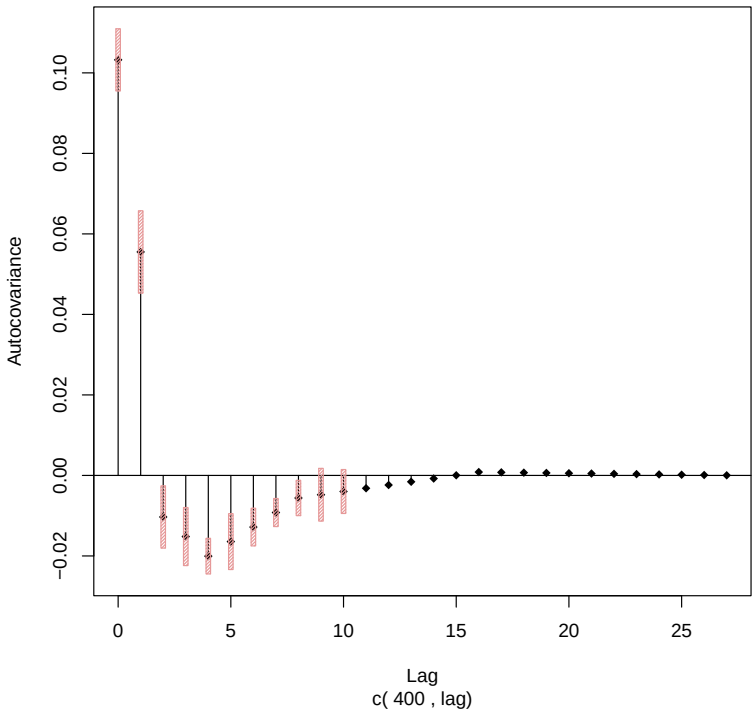

Figure 6: Local autocovariance of Earthquake series around $t = 400$.

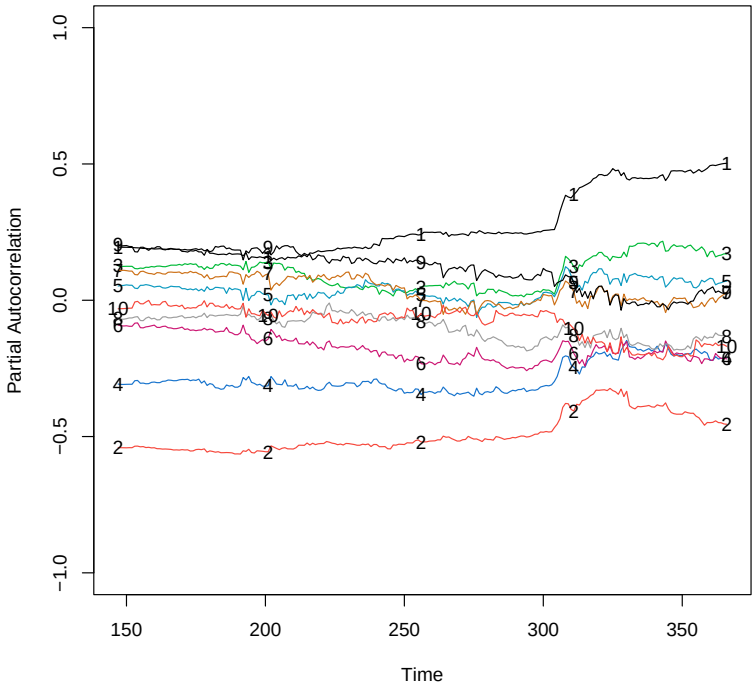

Figure 7: Local partial autocorrelation of Earthquake series

2.3 Local Partial Autocovariance

Local partial autocovariance at a lag τ is the association (covariance) between lagged versions of the time series, taking in account (or "taking out") autocovariances at lags lower than τ. So, the local partial autocovariance at lag τ is, if you like, the excess autocovariance at lag τ that did not arise from chaining together of lower-lag autocovariances.

The `lpacf` package's `lpacf` function computes the local partial autocorrelation. So, for example, the local partial autocorrelation for the Earthquake data can be computed and plotted using

```
plot(lpacf(Earthquake), lags=1:10, lcol=1:10)
```

which results in Figure 7. Again, these partial autocorrelations are time-varying, again reflecting the nonstationary nature of the Earthquake time series.

Exercise

Calculate the local partial autocorrelation with your own time series. Note, `lpacf` works with any length time series, not just those which are a power of two length. Again, you might wish to explore using different smoothing `binwidth` arguments and seeing the difference with using the `allpoints` argument set to `TRUE` or `FALSE`.

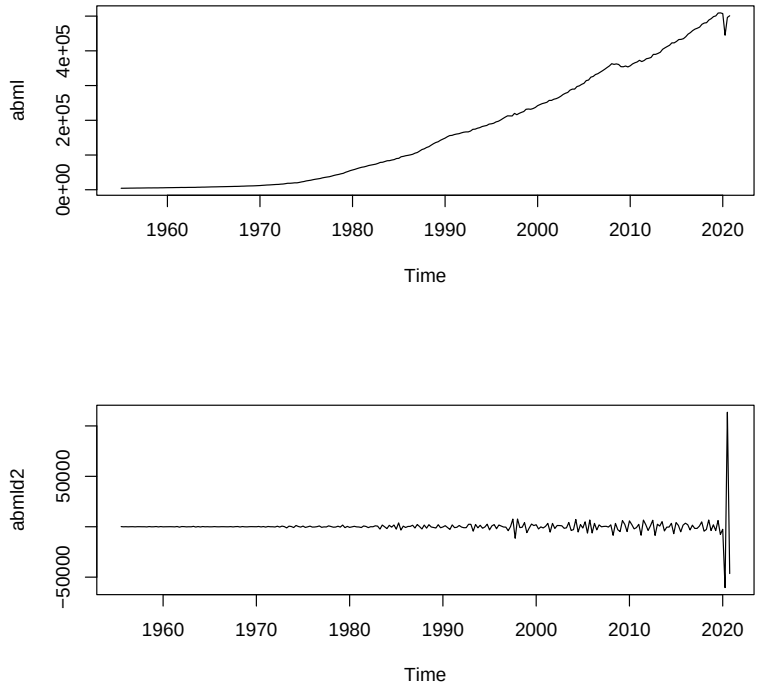

Figure 8: Top: ABML time series. Bottom: Second differences of ABML.

3 ABML Series

3.1 Introduction and Exploratory Plots

A country's gross domestic product (GDP) is a statistic that measures the value of all goods and services produced and sold by that country in a specified time period. A sub-component of the GDP is gross value added (GVA), which is the difference between the gross value of economic output minus the value of the intermediate consumption (costs of materials, services or supplies used to make the goods). We examine the UK's ABML time series, which is the gross value added series at current prices, seasonally adjusted. We obtained this data set from the UK Office for National Statistics data store `www.ons.gov.uk`.

Figure 8 shows the both ABML time series and its second differences. Produce the figure with these commands

```
oldpar <- par(mfrow=c(2,1))
ts.plot(abml)
ts.plot(abmld2)
par(oldpar)
```

The large drop in ABML towards the right end is as the result of the COVID19 pandemic and this shows up clearly at the right-hand end of the second difference plot. The lengths of the ABML series and the second differences are 264

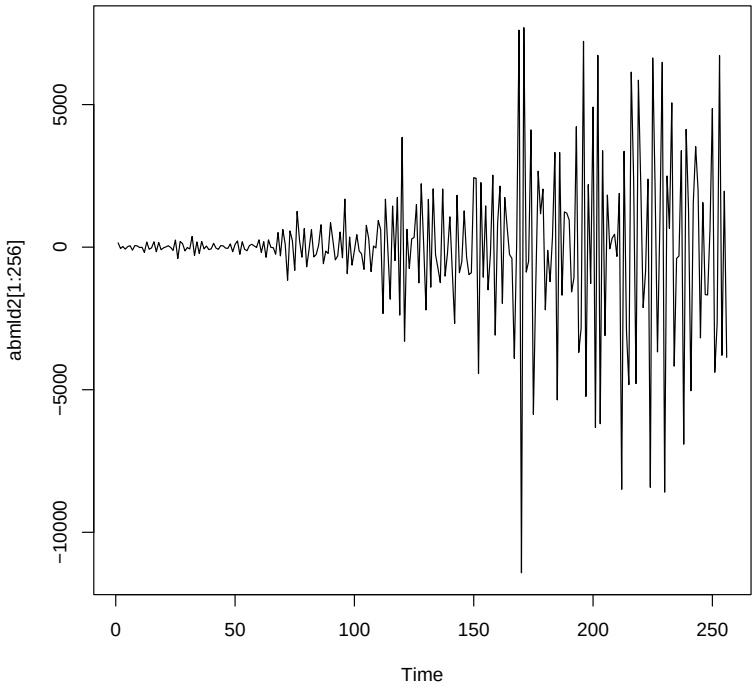

Figure 9: First 256 observations of second differences of ABML. The increasing variance in the series is clearly seen.

and 262, respectively.

We will analyse the second differences as the original series is highly integrated (in the economics sense) in that it has a distinctive and strong trend, which persists after first differencing. The second differences have a near zero (fairly) constant mean and, hence, suitable for analysis using a locally stationary process with zero mean assumptions.

To begin with, we will analyse the second differences of ABML before the COVID19 pandemic. Since some of the commands below only accept time series with power of two length, we will analyse the first 256 observations of the ABML second differences, which can be plotted by the command

```
ts.plot(abmld2[1:256])
```

which produces Figure 9. Figure 9 more clearly shows the increasing variance of the series compared to Figure 8 (bottom). The increasing variance is hard to see in the latter, since the extreme second differences near to the end of the series cause the earlier oscillations to be drawn very small.

3.2 Testing ABML for second-order stationarity

Given that the above plots suggest ABML is not second-order stationary. Let us first test the non-COVID ABML second differences series for stationarity.

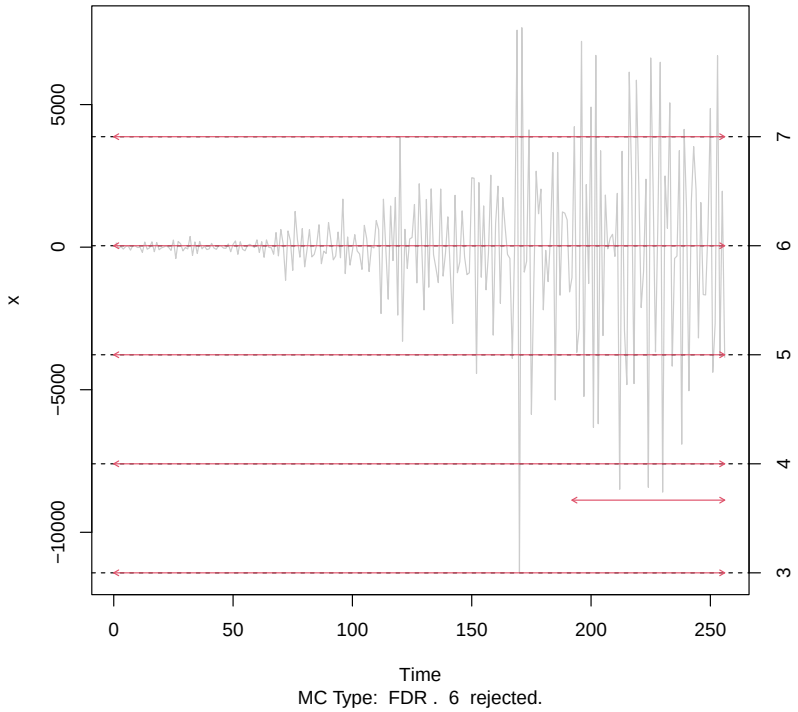

Figure 10: Results of plot of `hwtos2` test of stationarity applied to first 256 second differences of ABML series.

This is testing the null hypothesis:

$$H_0 : \text{ABML series is second-order stationary,}$$

versus the alternative that it is not second-order stationary.

We will first use the `hwtos2` test to carry out a test of second-order stationarity:

```
plot(hwtos2(abmld2[1:256]))
```

carries out the test and plots it in Figure 10. Figure 10 has six red double-headed arrows, which indicates that there is strong evidence to reject H_0 and view the ABML series as not second-order stationary.

Another test for stationarity is the `BootWPTOS` test from the package of the same name. We use a companion function `WPTOSpickout` from `BootWPTOS`. This tests against the same null and alternative as above, but does so against different wavelet-like oscillatory functions called wavelet packets. The `WPTOSpickout` function tests against a specific family member from a large set of possible packet members from a large wavelet packet library. These tests are described in Cardinali and Nason (2018).

For this example, we choose wavelets (as `index` is equal to one) according to wavelets at scale five and apply the test with the commands

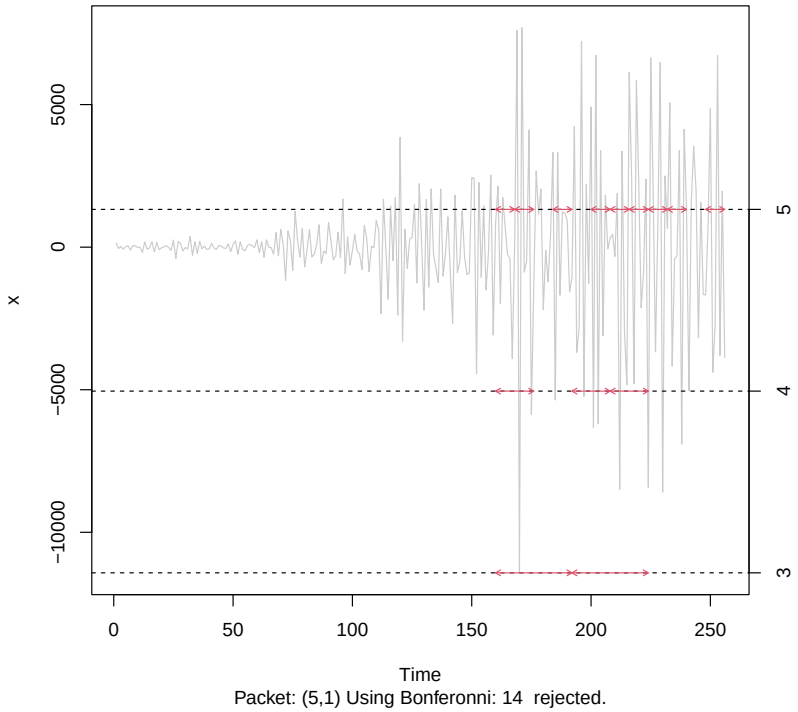

Figure 11: Plot from test of stationarity using `WPTOSpickout` function against wavelets at scale five.

```
plot(WPTOSpickout(abmld2[1:256], level=5, index=1))
```

which results in Figure 11. The selector `1:256` extracts the first 256 observations, since `WPTOSpickout()` requires the input data to have a length of a power of two. Figure 11 shows clear nonstationarities as indicated by the double-headed arrows.

3.3 Local ACVF and ACF of ABML

Since ABML appears not to be second-order stationary, we now look at the local autocovariance function. So, issue the commands

```
plot(lacf(abmld2[1:256]), lags=0:7, lcol=1:8, plotcor=FALSE)
```

which resuts in Figure 12. We choose only to display lags zero thru seven in a variety of colours. The seven was an arbitrary choice, just to cut down on the number of lags that are displayed, so the picture does not get too complicated.

The most noticeable feature of Figure 12 is that the variance of ABML (the lag zero line) and the lag one, three and five autocovariances increase/decrease over time. These increases are, no doubt, due to the increasing variance of the ABML second differences that are obvious in the time series plot.

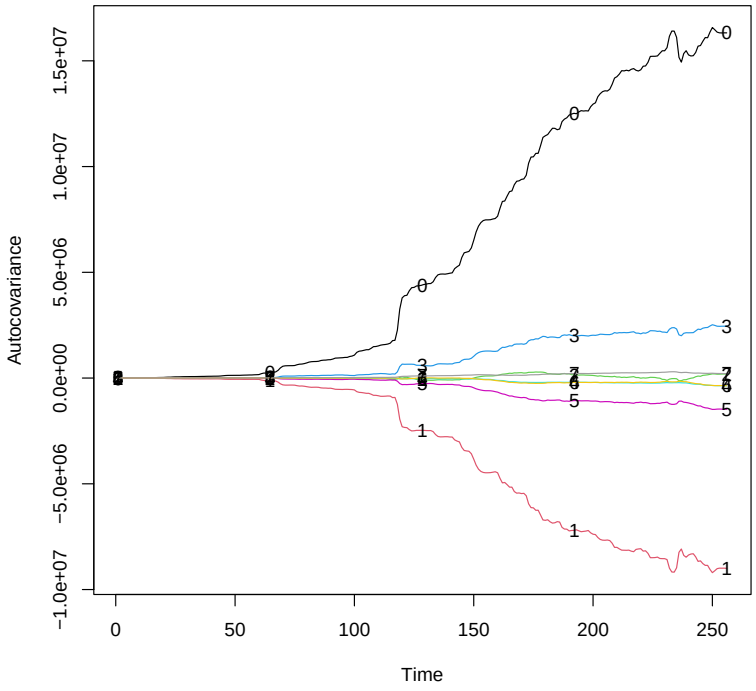

Figure 12: Local autocovariance of the ABML data set at lag zero through to seven.

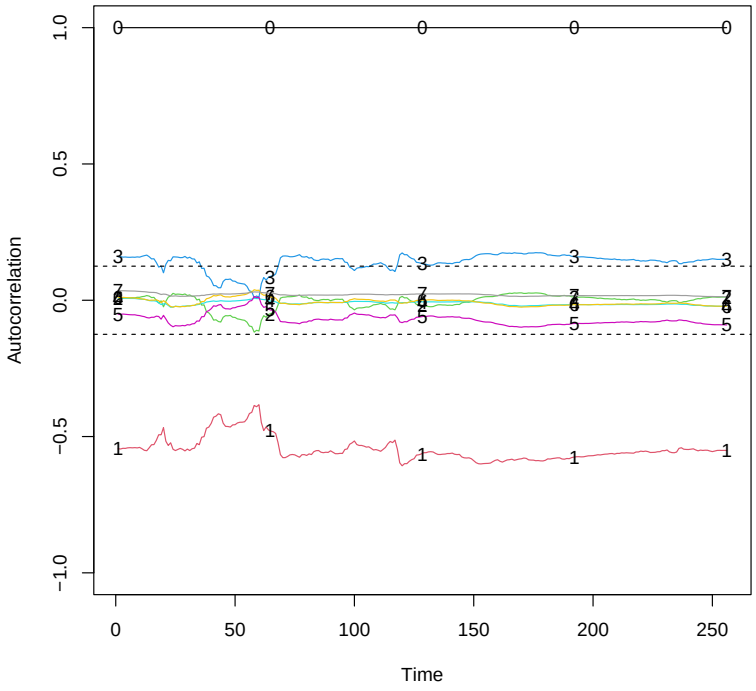

Figure 13: Local autocorrelation of the ABML data set and approximate 95% confidence intervals.

Now plot the local autocorrelation, with the commands

```
plot(lacf(abmld2[1:256]), lags=0:7, lcol=1:8)
#
# Plot approximate 95% confidence intervals
#
abline(h=c(-1, 1)*2/sqrt(256), lty=2)
```

These commands result in Figure 13. The local autocorrelation looks more constant, certainly when compared to the local autocovariance. So, might be concluded that the nonstationarity in this series is entirely due to variance changes. However, the autocorrelations, especially the lag-one autocorrelation during the first 75 time points or so does exhibit noticeable variability.

3.4 Local ACF as regular acf plot with CIs

We can plot the local autocovariance using the local acf-type plot using the commands

```
oldpar <- par(mfrow=c(1,2))
par(pty="s")
plot(Rvarlacf(abmld2[1:256], nz=50, var.lag.max=7),
    plotcor=FALSE, type="acf")
plot(Rvarlacf(abmld2[1:256], nz=200, var.lag.max=7),
    plotcor=FALSE, type="acf")
par(oldpar)
```

This results in Figure 14. The left-hand plot at time $t = 50$ shows little discernible autocovariance (probably because the values of the series are relatively tiny at the start of the plot). The right-hand plot shows significant autocovariance at lag one (and lag zero, of course).

The confidence intervals shown in the autocovariance plots in Figure 14 are more tailored and probably more accurate than the 'global' single horizontal dashed lines shown in Figure 13. The horizontal dashed confidence intervals are just the standard ones that relate not only to a stationary series, but specifically to a set of independent and identically distributed random variables. These are just a guide and probably not that useful in the nonstationary time series world.

3.5 Local partial autocorrelation of ABML

From Figures 12 and 13 it certainly seems like some low-order autocovariances and autocorrelations are significant.

To shine some more light on this data let us now look at the local partial autocorrelation of ABML. We can produce this using the following commands using the `lpacf` package:

```
plot(lpacf(abmld2[1:256], allpoints=TRUE), lags=1:5, lcol=1:5)
abline(h=c(-1, 1)*2/sqrt(256), lty=2)
```

which results in Figure 15. Note, the `lpacf` function can deal with any length time series, but for now, we stick to the first 256 observations for comparability with earlier plots. The local partial autocorrelations are extremely noisy on the left-hand side of the plot. This is because the series is small here and the partial autocorrelation involves ratios of small quantities, which render it unstable. The situation is better on the right of the plot.

Figure 15 indicates that the partial autocorrelations at lags one thru five are all likely significant at different times, probably lag four is not significant for most of the time. So, it looks like a low-order time-varying AR process might be a suitable model here.

3.6 Which wavelet to use?

For any given real time series we do not know whether a time series is an actual locally stationary wavelet process (unlikely — "all models are wrong, but some are useful", Box) and, even if it was, what is the underlying wavelet?

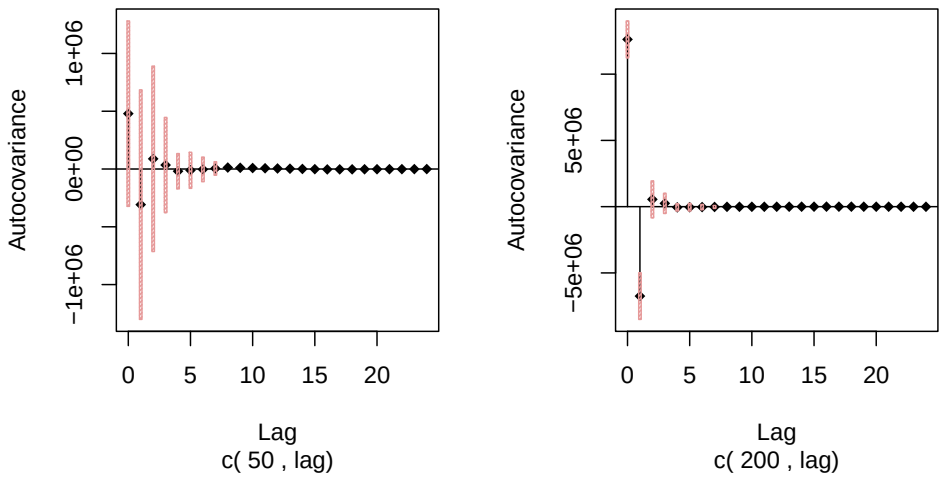

Figure 14: Local autocorrelation of the ABML data set as an acf type plot. Left: centred on $t = 50$; right: centred on $t = 200$ with (red) 95% confidence intervals.

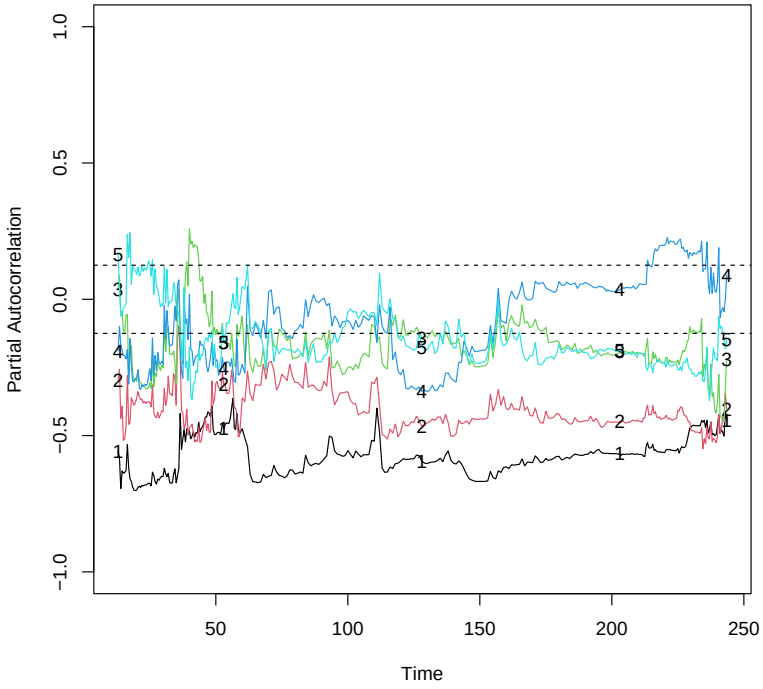

Figure 15: Local partial autocorrelation of the ABML data with approximate 95% confidence intervals.

We can get some information on which analysis wavelets might be useful for our local autocorrelation, autocovariance, spectrum and forecasting by using the `which.wavelet.best` function. This function works using predictive inference: it uses different wavelet bases to do some forecasting and compares the results to actual data. The wavelets that give the smallest mean-squared prediction error (MSPE) values are indicated as the 'best'.

We can run

```
which.wavelet.best(abmld2[1:256])
```

This results in output:

```
    filter.number      family     rmse    min.mse
1               1 DaubExPhase 1655.083
2               2 DaubExPhase 1634.235 <- Min MSE
3               3 DaubExPhase 1787.172
4               4 DaubExPhase 2313.819
5               5 DaubExPhase 2690.474
6               6 DaubExPhase 2900.942
...
```

which indicates we should use Daubechies' Extremal Phrase wavelets with two vanishing moments. So, in commands below we will add `filter.number=2` and `family="DaubExPhase"` to select these wavelets.

If your machine has multiple CPU cores, then a parallel version of `which.wavelet.best` exists. So, for example, issuing

```
library("parallel")
#
# Number of CPU cores reserved for R parallel processing
options(mc.cores=5)
which.wavelet.best(abmld2[1:256], lapplyfn=mclapply)
```

runs the function over five cores (my machine has six cores, and I usually reserve one for the operation of the machine itself).

You might wish to retry the local autocovariance and autocorrelation commands above, but add in `filter.number=2, family="DaubExPhase"` to the argument list to switch the analysis wavelet to see what difference it makes.

Exercise

Try applying the `which.wavelet.best` function to your time series. You might try recomputing the `lacf` local autocorrelation/covariance or the `lpacf` local partial autocorrelation using the "best" wavelet to see whether it makes much difference.

3.7 Forecasting ABML second differences: mid-pandemic!

We use the `forecast.lpacf` function from the `forecastLSW` package, which is intended to forecast locally stationary (wavelet) time series. First, we forecast the first 256 second difference series, i.e. the forecaster is trained with observations up to Quarter 2 2019. We can do this with these commands:

```
plot(forecast.lpacf(abmld2[1:256], h=4, forecast.type="recursive",
    filter.number=2, family="DaubExPhase"), zoom=TRUE)
points(257:260, abmld2[257:260], col=3, pch=19)
```

Some explanation of these functions and their arguments follows. For the `forecast.lpacf` function, the arguments are:

abmld2 the second difference series, and we use the first 256 observations to train the forecaster.

h=4 this causes the function to forecast four steps ahead

forecast.type="recursive" this is the recommended type of forecasting.

filter.number and family set to the 'best' type as judged by the `which.wavelet.best` function.

zoom this has been set to true to focus on the end of the time series for the plot, to enable the last values of the series and forecast to be viewed.

The results of the forecasting and plotting are shown in Figure 16. Unfortunately, all of the green outcome values are outside of the prediction intervals! This is perhaps not surprising as this period of time was mid-COVID19 pandemic and the forecaster was trained mostly prior to the pandemic.

3.8 Forecasting ABML second differences from end 2010

We now repeat the exercise above, but now train the forecaster up to and including Quarter 4 2010 — this is observation $t = 226$. The commands are

```
end<- 226

# Forecast vector indices
fvec <- (end+1):(end+4)

# Compute and plot locally stationary forecast
plot(forecast.lpacf(abmld2[1:end], h=4, forecast.type="recursive",
    filter.number=2, family="DaubExPhase"), zoom=TRUE)

# Plot the actual values (Q1 to Q4 2011)
points(fvec, abmld2[fvec], col=3, pch=19)

# Compute the forecast package forecast
stat.for <- forecast(abmld2[1:end], h=4)

# Augment plot with forecast and its prediction ints
points(fvec, stat.for$mean, col=4, pch=19)
points(fvec, stat.for$upper[,2], col=4, pch=3)
points(fvec, stat.for$lower[,2], col=4, pch=3)
```

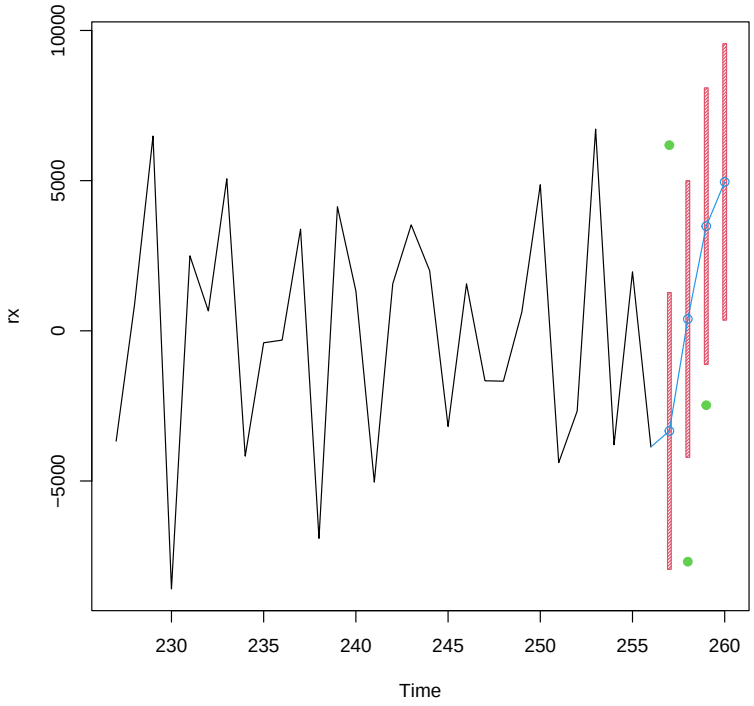

Figure 16: Forecast of ABML data. The forecaster is trained on all observations up to and including $t = 256$, which corresponds to Quarter 2 2019. The forecasted values for 2019.Q3, 2019.Q4, 2020.Q1 and 2020.Q2 are shown as blue circles joined by a blue line, with accompanying 95% prediction intervals. The true (outcome) values are shown in green.

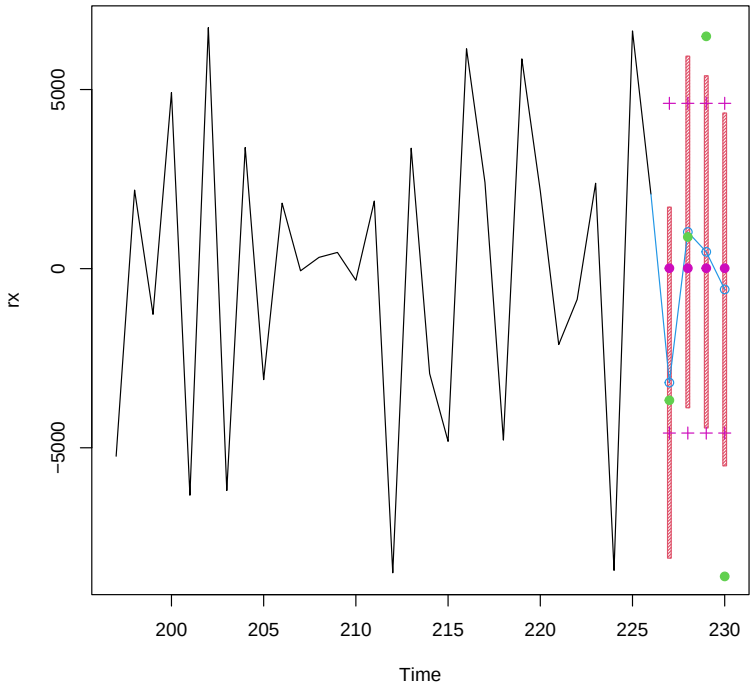

Figure 17: Forecast of ABML data. The forecaster is trained on all observations up to and including $t = 226$, which corresponds to Quarter 4 2010. The forecasted values for 2011.Q1, 2011.Q2, 2011.Q3 and 2020.Q4 are shown as blue open circles joined by a blue line, with accompanying 95% prediction intervals. The true (outcome) values are shown in green. Forecasts and 95% prediction intervals from the `forecast` function are shown in magenta.

We also compute a forecast using the default arguments in the `forecast` function from the `forecast` package and plot its prediction intervals in blue. The results these commands are shown in Figure 17. The first two locally stationary forecasts (for 2011.Q1 and 2011.Q2) are very close to their actual out-turns. The locally stationary forecasts for 2011.Q3 and 2011.Q4 are not so near, but they are nearer to the true out-turns than the stationary forecaster. Simulation results in Killick et al. (2023) show that the locally stationary forecaster performs pretty well in practice.

Exercise

Try forecasting with your time series. Use the `forecast.lpacf` function and plot the results, play around with the forecast horizon, `h`. You could use the "best" wavelet identified earlier using the `which.wavelet.best` function and/or try different wavelets. You can compare these forecasts with those produced by the `forecast` function.

4 More Time?

If you have more time, then explore the following packages:

LS2W two-dimensional locally stationary wavelet processes.

mvLSWimpute multivariate time series imputation

BootWPTOS wavelet packet tests of stationarity

costat costationarity determination.

A Useful Commands Summary

`BootWPTOS::BootWPTOS` Carry out test of stationarity with respect to multiple wavelet packet functions.

`BootWPTOS::WPTOSpickout` Carry out test of stationarity with respect to a single wavelet packet function (such as wavelets).

`forecastLSW::forecast.lpacf` performs locally stationary wavelet forecasting

`forecastLSW::which.wavelet.best` tries to work out a good choice of wavelet for forecasting and other time series methods.

`locits::hwtos2` Carry out test of second-order stationarity on a time series which has a dyadic length.

`locits::lacf` Computes the local autocovariance and/or autocorrelation of a locally stationary series.

`lpacf::lpacf` Computes the local partial autocorrelation of a locally stationary series.

References

A. Cardinali and G. P. Nason. Practical powerful wavelet packet tests for second-order stationarity. *App. Comp. Harm. Anal.*, 44:558–583, 2018.

R. Killick, M. I. Knight, G. P. Nason, M. A. Nunes, and I. A. Eckley. Automatic locally stationary time series forecasting with application to predicting U.K. Gross Value Added time series under sudden shocks caused by the COVID pandemic, 2023. arXiv:2303.07772.

G. P. Nason. A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, 75:879–904, 2013.