Journal of Statistical Software

October 2013, Volume 55, Issue 1. *http://www.jstatsoft.org/*

Costationarity of Locally Stationary Time Series Using costat

Alessandro Cardinali
University of Bristol

Guy Nason
University of Bristol

Abstract

This article describes the R package **costat**. This package enables a user to (i) perform a test for time series stationarity; (ii) compute and plot time-localized autocovariances, and (iii) to determine and explore any costationary relationship between two locally stationary time series. Two locally stationary time series are said to be costationary if there exists two time-varying combination functions such that the linear combination of the two series with the functions produces another time series which is stationary. Costationarity existing between two time series indicates a relationship between the series that might be usefully exploited in a number of ways. Sometimes the relationship itself is of interest, sometimes the derived stationary series is of interest and useful as a substitute for either of the original stationary series in some applications.

Keywords: local stationarity, costationary time series, local autocovariance, **costat**.

1. Summary

The **costat** package, available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=costat`, is designed for the analysis of locally stationary time series, particularly locally stationary wavelet time series. The package includes functionality for

- computing tests of stationarity, `BootTOS`;
- computing localized autocovariances, `lacv`;
- discovering costationarity between two time series, `findstysols`

Several method functions exist that print, summarize or plot the various outputs of these functions.

2. Introduction: Locally stationary time series

This article concerns the analysis of discrete time series, X_t. That is, a sequence of observations taken through time where t could be an integer or some other regular spacing. Many are familiar with the concept of a *stationary* time series: that is, loosely speaking, a series whose statistical properties do not change over time. In much of classical time series analysis stationary means second-order stationary where the mean and variance of a series, X_t, do not depend on time and the autocovariance:

$$\gamma_X(\tau) = \mathsf{E}\{(X_t - \mathsf{E}X_t)(X_{t+\tau} - \mathsf{E}X_{t+\tau})\} \tag{1}$$

only depends on the lag τ between two different variates X_t and $X_{t+\tau}$, but not the time point $t = 1, \ldots, T$. There are several excellent books dealing with (stationary) time series analysis, for example, Brockwell and Davis (1991), Chatfield (2003), Hamilton (1994), Hannan (1960), Priestley (1983) or Shumway and Stoffer (2006).

Mathematical theory demands that a stationary time series X_t possesses the following representation:

$$X_t = \int_{-\pi}^{\pi} A(\omega) e^{it\omega} dz(\omega), \tag{2}$$

where $A(\omega)$ is an amplitude function, $\{e^{it\omega}\}$ is the usual system of harmonic complex exponentials and $dz(\omega)$ is an orthonormal increments process. The amplitude function controls the second-order properties of the time series. The usual spectrum $f(\omega) = |A(\omega)|^2$ and the spectrum and autocovariance are a Fourier transform pair. The amplitude, spectrum and autocovariance here reflect the time-invariant nature of the time series X_t: none of them depend on time, t.

However, many actual time series, arising in many disciplines, are not stationary. Sometimes the nonstationarities can be detected via a simple time series plot. Another way of detecting nonstationarity is to compute a simple statistical time series statistic on different sections of the time series and look for differences. For example, using the sample autocovariance function `acf` in R, R Core Team (2013). Such exploratory methods for detecting nonstationarities are undoubtedly useful, but somewhat *ad hoc*, and more statistically reliable methods are necessary.

Over the years several extensions to the basic stationary model (2) have been proposed to deal with nonstationary time series. Important general classes of models are the *locally stationary* time series models. One locally stationary extension of (2) replaces the time-constant $A(\omega)$ amplitude function by one which explicitly depends on time, for example $A_t(\omega)$. For example, the oscillatory processes of Priestley (1983) or the locally stationary Fourier (LSF) processes introduced by Dahlhaus (1997), and more recently Dahlhaus and Polonik (2006) and Dahlhaus and Polonik (2009). A comprehensive and recent review of locally stationary processes can be found in Dahlhaus (2012).

Another locally stationary extension, developed by Nason, von Sachs, and Kroisandt (2000), replaced the exponential harmonics by nondecimated discrete wavelets as well as introducing time-locality into the expansion. Their locally stationary wavelet (LSW) model was given by

$$X_t = \sum_{j=1}^{\infty} \sum_{k=-\infty}^{\infty} w_{j,k} \psi_{j,k-t} \xi_{j,k}, \tag{3}$$

where $w_{j,k}$ is the amplitude sequence, $\{\psi_{j,\ell}\}$ are the nondecimated discrete wavelets and $\xi_{j,k}$ is a sequence of random variables satisfying $\mathsf{COV}(\xi_{j,k}, \xi_{\ell,m}) = \delta_{j,\ell}\delta_{k,m}$, where $\delta_{k,m}$ is the Kronecker delta function. The $\xi_{j,k}$ also satisfy $\mathsf{E}(\xi_{j,k}) = 0$ which means that $\mathsf{E}X_t = 0$: any real time series that does not have zero mean can be detrended, see Chatfield (2003), for example. Additionally, deterministic trend needs to be removed before modelling using LSW processes. Additional details on the LSW processes and examples of their use can be found in Nason (2008). As with stationary and LSF processes the amplitude sequence $w_{j,k}$ is linked to a spectral quantity by $w_{j,k}^2 \approx S_j(z)$, where $z = k/T$ is rescaled time, $z \in (0,1)$ and $k = 1, \ldots, T$. The quantity $\{S_j(z)\}_{j=1}^{\infty}$ is known as the evolutionary wavelet spectrum. Assuming that a time series X_t is appropriately modeled by a LSW process, a key task is to estimate its spectrum using an estimator $\{\hat{S}_j(z)\}_{j=1}^{J}$ where $2^J = T$. For a finite length T time series the spectrum can only be computed for a finite set of scales $1, \ldots, J$. The **wavethresh** package, Nason (2013a), contains a function `ewspec` which estimates the evolutionary wavelet spectrum of a time series and there are various other functions to both plot the time-varying spectrum and extract its values for further processing. Again, many more details on these kinds of practical analyses can be found in Nason (2008).

Much recent research has considered the case of nonstationary models of one kind or another, and methods of estimating the associated spectral quantities. Fewer articles consider the tricky problem of given an actual time series how does one decide whether it is a stationary or not? Some examples of stationarity tests are the early paper by Priestley and Rao (1969), tests that use wavelets by von Sachs and Neumann (2000) and Nason (2013b), Ahamada and Boutahar (2002) in economics, Stărică and Granger (2005) and Paparoditis (2009) both which measure the difference between a periodogam and a model spectrum on intervals and Dwivedi and Rao (2011).

We should mention at this point that we are considering the testing of second-order stationarity of locally stationary time series. That is, we assume that the mean is constant and there is no trend (or the series has been detrended and deseasonalized). There is a substantial literature on the topic of stationarity testing for time series exhibiting trends, see, for example, the KPSS test, Kwiatowski, Phillips, Schmidt, and Shin (1992), and the related problem of unit-root testing, see, e.g., Dickey and Fuller (1979), Elliott, Rothenberg, and Stock (1996) or Phillips and Perron (1988).

In R the package **fractal** includes an implementation of the Priestley-Subba Rao test, called `stationarity`, modified with improved spectral density function estimators based on averaging multitaper estimators, see Constantine and Percival (2007). This article describes the (new) test of stationarity introduced in Cardinali and Nason (2010) and presented in the **costat** package: this is described in Section 3.

The "flip" side to spectral quantities are covariances. In the stationary theory the autocovariance function is the inverse Fourier transform of the spectrum. For locally stationary processes attention has focussed on spectral quantities such as $f(z, \omega)$ or $S_j(z)$ although some works have concentrated on the covariance side notably Sanderson, Fryzlewicz, and Jones (2010), parts of Nason *et al.* (2000) and Nason (2013b). Section 4 provides details on how to compute both localized autocovariance (LACV) from Nason *et al.* (2000) and how to present the results in attractive and useful formats.

Finally, in contrast to the stationary or nonlinear time series world, little work has been conducted on the case of where there is more than one locally stationary time series to be

considered, in other words, multivariate time series. Again, Sanderson *et al.* (2010) is a notable exception. This article considers the implementation of the costationary methods introduced by Cardinali and Nason (2010) in Section 5. Given two *locally stationary* time series the goal of costationarity determination is to investigate whether there are two sequences α_t, β_t that can be found such that Z_t is a classically (second-order) *stationary* time series where

$$Z_t = \alpha_t X_t + \beta_t Y_t. \tag{4}$$

The concept of costationarity is inspired by the cointegration of Engle and Granger (1987) except that order-1 nonstationarity is replaced by order-2, and the costationary vectors, (α_t, β_t) are time-varying (necessarily for a non-trivial theory). In cointegration the X_t, Y_t processes are assumed to be integrated (differencing required to achieve stationarity) and a linear combination (α, β) is sought such that $Z_t = \alpha X_t + \beta Y_t$ is a stationary series. In our work X_t, Y_t have time-varying second-order statistics and the linear combination in (4) is sought to produce a second-order stationary (time invariant second-order statistics) series.

The rationale for costationary detection/determination are (i) learning of any costationary relationship itself, as this could be interesting, (ii) estimating the strength of such a relationship, (iii) using the derived stationary series Z_t, in some applications in preference to either of the original series and (iv) using the relationship to learn about X_t from data on Y_t or vice versa. Cardinali and Nason (2010) give two examples where a costationary relation is discovered. One example, from financial portfolio planning, obtains asset portfolios that are shown to be preferable to classically defined ones or others defined using classical methods augmented using conditional variance estimation techniques (essentially VECH GARCH). Their other example, investigates costationary combinations of volatile wind power series to obtain a less volatile series. Volatility is a real problem for wind power as generators prefer stable and predictable sources and wind is anything but. Usable power from wind can be obtained by aggregating wind power over geographical areas and costationarity can help discover useful combinations to enhance stability.

3. Stationarity test

3.1. The test

Given a time series one might wish to test whether it is stationary or not. The **costat** package incorporates a function `BootTOS` which performs a simple bootstrap based test of stationarity. Given a time series X_t, assumed to be modeled by a LSW process with spectrum $\{S_j(z)\}_{j=1}^J$, the null hypothesis of the **costat** test of stationarity is:

$$H_0 : S_j(z) \text{ is a constant function of } z \in (0,1) \text{ for all } j \tag{5}$$

versus the alternative $H_A : S_j(z)$ is not constant for some j. Details of the test appear in Cardinali and Nason (2010) but the test is based on the metric:

$$T(\{S_j(z)\}) = J^{-1} \sum_{j=1}^{J} \int_0^1 \{S_j(z) - \bar{S}_j\}^2 dz, \tag{6}$$

where $\bar{S}_j = \int_0^1 S_j(z) dz$. Clearly, if $S_j(z)$ is constant, for all j, then $T(\{S_j(z)\}) = 0$ and T is nonzero for nonconstant spectra. The statistical significance of the test statistic T computed

on the time series realization is assessed via a parametric bootstrap calculation. Multiple comparator simulations are produced under the null hypothesis assuming constant spectral quantities estimated from the data and then the test statistic evaluated on those simulations.

To demonstrate `BootTOS` we use the `SP500FTSElr` data distributed with **costat** which contains log-returns of both the well-known SP500 and FTSE financial index series between 1995-06-21 and 2002-10-02. For the purposes of this article we will examine a smaller segment of this data called `sret` and `fret` which are the SP500 and FTSE log-returns extracted from the `SP500FTSElr` object from index 256 to 767. The methods we describe below do work on the whole series contained in `SP500FTSElr` but can take quite a long time on single-processor machines especially when the main purpose here is pedagogical.

Should the reader wish, it is quite easy to convert the `SP500FTSElr` object into a multivariate time series objects present in R. For example, a `timeSeries` class object by loading the **timeSeries** package (Wuertz and Chalabi 2013) and typing

```
R> newts <- timeSeries(data = SP500FTSElr[, 2:3], charvec = SP500FTSElr[, 1])
```

and then this new times series, `newts`, can be plotted or analyzed using the functions present in the **timeSeries** package.

The log-returns of a sequence X_t are given by $\log(X_t/X_{t-1})$ and is a common quantity of analysis for economic and financial data. The reader will note that the extracted series are of length 512 which is a power of two. Like many wavelet-based codes the **costat** software works on data sets that are of dyadic length (i.e., 2^J for some natural number J). It should be made clear that this is not a limitation of wavelets per se, but of the computationally efficient algorithms used to compute the intended quantities. Data sets of other lengths can be handled by zero-padding or truncation. The data can be accessed by typing:

```
R> library("costat")
R> data("sret", "fret")
```

As with many data analyses it is useful to first produce a plot of the series. We can do this with the following code.

```
R> ts.plot(sret)
R> lines(fret, col = 2)
```

This code produces the plot shown in Figure 1 which indicates that both series are probably not stationary as there appears to be clear variance changes in the series. The following code performs the bootstrap test of stationarity mentioned above:

```
R> sret.TOS <- BootTOS(sret)
```

The number of bootstrap simulations is controlled by the `Bsims` argument which is set to 100 by default. The `BootTOS` test is computationally intensive. However, it can use the `mclapply` function of the **multicore** package (Urbanek 2011, now part of the base **parallel** package) to execute the bootstrap simulations in parallel. On a dual core (Intel Core 2 Duo, 2.8Ghz) iMac the bootstrap test took approximately 10.4 seconds, on a twelve core (Intel Xeon, 3.07Ghz) Linux machine the test took 1.3 seconds.

The `BootTOS` function returns an object of class `c("BootTOS", "htest")`. One can see the results of the significance test by typing the object name as follows:

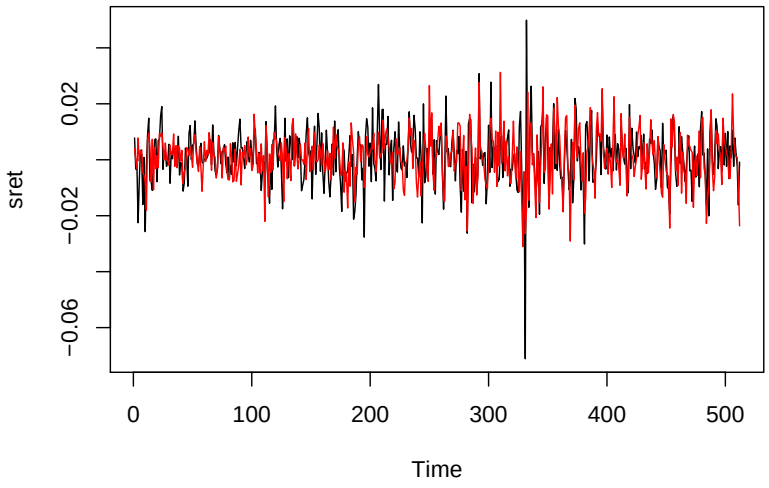

Figure 1: Time series plot of the `sret` (black) and `fret` (red) time series.

```
R> sret.TOS

        BootTOS test of stationarity

data:  sret
= 0, p-value < 2.2e-16
```

The function output indicates that the p-value of the test is less than 0.01 and hence the `sret` series is nonstationary as its spectrum is much more variable that could be obtained by a stationary equivalent (i.e., with the same marginal S_j values).

As another example, let us run the same test but on a series we know to be stationary. Let us create a new series which samples 512 observations from a standard Gaussian distribution and repeat the test of stationarity.

```
R> x <- rnorm(512)
R> x.TOS <- BootTOS(x)
R> plot(x.TOS)
R> x.TOS

        BootTOS test of stationarity

data:  x
= 0.0125, p-value = 0.98
```

Here one can see that the p-value of the test is 0.98, and hence we can feel comfortable in assessing the series `x` to be stationary (as we knew it would because we set it up that way).

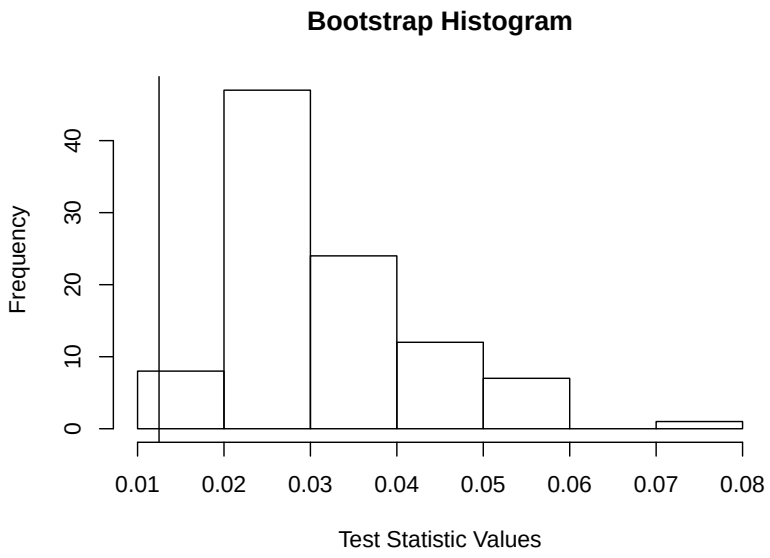

Figure 2: Histogram of bootstrap sample values resulting from `x` test of stationarity.

The plot from the `plot(x.TOS)` command is shown in Figure 2. The histogram shows all the test statistic values computed on the data *and* the bootstrap simulations. The vertical line corresponds to the test statistic computed on the original data. It can be seen that roughly 98% of the bootstrap samples are higher in value. Hence, the sample test statistic is nothing unusual compared to a bootstrap sample on comparable distributional setup and hence we have no reason to reject the null hypothesis.

3.2. Size and power: Simulation results

With any statistical test it is important to have some understanding of its size and power characteristics which we do here via simulation.

Our size and power results involve three tests of second-order stationarity. The Priestley and Rao (1969) test as implemented in the package **fractal**, the Haar wavelet test of stationarity from Nason (2013b) and `BootTOS` as described above. We shall abbreviate the names of these tests in our tables as PSR, HWTOS and `BootTOS`.

To assess size, we simulate data from a number of stationary models using Gaussian innovations and assess how often our tests of stationarity reject the null hypothesis. The models are:

S1 iid standard normal;

S2 AR(1) model with AR parameter of 0.9 with standard normal innovations;

S3 As S2 but with AR parameter of -0.9;

S4 MA(1) model with parameter of 0.8;

S5 As S4 but with parameter of -0.8.

Model	PSR	HWTOS (Bon)	`BootTOS`
S1	5.6	4.2	0.1
S2	11.8	3.2	0.0
S3	6.6	19.0	39.3
S4	5.3	2.7	0.0
S5	8.0	0.5	0.1
S6	7.0	0.4	2.2
S7	48.1	21.2	18.5

Table 1: Simulated size estimates (%) for stationary Gaussian models $T = 512$. Based on 1000 simulations with 1000 bootstrap simulations per main simulation. Note: Column PSR corresponds to simulation results from Nason (2013b) for 100 simulations.

S6 ARMA$(1, 0, 2)$ with AR parameter of -0.4, and MA parameters of $(-0.8, 0.4)$.

S7 AR(2) with AR parameters of $\alpha_1 = 1.385929$ and $\alpha_2 - 0.9604$. The roots associated with the auxiliary equation, see Chatfield (2003, page 44), are $\beta_1 = \bar{\beta}_2 = 0.98e^{i\pi/4}$. This process is stationary, but close to the 'unit root': a 'rough' stochastic process with spectral peak near $\pi/4$.

Models S1–S7 are the same as in Nason (2013b) to aid comparisons with another test of stationarity.

The empirical size values are shown in Table 1. Note: Column HWTOS (Bon) are results from the stationarity test described in Nason (2013b) soon to be uploaded to CRAN. Overall, the size characteristics of `BootTOS` are extremely good and conservative apart from S3 (where actually none of the tests meet the 5% nominal size, although `BootTOS` performs much the worst) and S7 where PSR and HWTOS do poorly and `BootTOS` better, but still not near to 5%. Note, PSR does not do very well in S2 or S6 where HWTOS and `BootTOS` do well.

Power

To explore statistical power we create processes that are nonstationary and then count the number of times each test reckons a realization is not stationary over multiple simulations. Again, we use models from Nason (2013b) as follows.

P1 Time-varying AR model $X_t = \alpha_t X_{t-1} + \epsilon_t$ with iid standard normal innovations and the AR parameter evolving linearly from 0.9 to -0.9 over the 512 observations.

P2 A LSW process based on Haar wavelets with spectrum $S_j(z) = 0$ for $j > 1$ and $S_1(z) = \frac{1}{4} - (z - \frac{1}{2})^2$ for $z \in (0, 1)$. This process is, of course, a time-varying moving average process.

P3 A LSW process based on Haar wavelets with spectrum $S_j(z) = 0$ for $j > 2$ and $S_1(z)$ as for P2 and $S_2(z) = S_1(z + \frac{1}{2})$ using periodic boundaries (for the construction of the spectrum only).

P4 A LSW process based on Haar wavelets with spectrum $S_j(z) = 0$ for $j = 2, j > 4$ and $S_1(z) = \exp\{-4(z - \frac{1}{2})^2\}$, $S_3(z) = S_1(z - \frac{1}{4})$, $S_4(z) = S_1(z + \frac{1}{4})$ again assuming periodic boundaries.

Model	PSR	HWTOS (Bon)	BootTOS
P1	39.4	99.7	100.0
P2	100.0	14.8	96.3
P3	44.3	1.9	23.8
P4	100.0	94.0	99.7

Table 2: Simulated power estimates (%) for models P1–P4 with nominal size of 5%

The spectra and single realizations for these processes are displayed in Nason (2012). All tests have excellent power for model P4. `BootTOS` has excellent power on P1 and P2 where one of PSR (for P1) or HWTOS (for P2) do not perform at all well. No test performs particularly well for P3 although PSR does best and `BootTOS` is second both detecting some nonstationarity. HWTOS for P3 performs particularly poorly.

Overall, although no test can be declared the 'winner' in these experiments the empirical performance of `BootTOS` is creditable.

4. Local autocovariance

This section introduces functions that compute and plot localized autocovariances (LACV).

4.1. LACV on a stationary series

Before analyzing a nonstationary example, we use the localized autocovariance on a stationary series, in order to assess the correctness of the answer given by comparing the result to that produced by the standard R function `acf`.

Even so, we should not expect the LACV to be as accurate as the regular `acf` function on a stationary series. This is because the `acf` function will produce its estimate by averaging over the full time period of quantities whose expectation is constant over that time period. The localized function will use local smoothing and not take advantage of our prior knowledge of the constancy of the autocorrelations in this specific case.

Computing the regular `acf`

For our stationary series we choose to generate the AR time series $X_t = 0.8X_{t-1} + Z_t$, and also print the autocovariance.

```
R> vsim <- arima.sim(model = list(ar = 0.8), n = 1024)
R> vsim.acf <- acf(vsim, plot = FALSE)
R> vsim.acf

Autocorrelations of series 'vsim', by lag

     0      1      2      3      4      5      6      7      8      9     10
 1.000  0.788  0.626  0.498  0.399  0.309  0.231  0.150  0.092  0.036  0.009
    11     12     13     14     15     16     17     18     19     20     21
-0.017 -0.049 -0.069 -0.089 -0.097 -0.098 -0.113 -0.121 -0.120 -0.121 -0.104
```

```
    22      23      24      25      26      27      28      29      30
-0.081 -0.068 -0.050 -0.064 -0.086 -0.089 -0.103 -0.077 -0.037
```

Note that the lag one autocorrelation of 0.788 estimates the AR parameter of 0.8.

Computing the localized autocovariance

We can use **costat**'s function, `lacv`, for computing the localized autocovariance, $c(z, \tau)$, which is an implementation of the formula:

$$c(z, \tau) = \sum_{j=1}^{J} S_j(z)\Psi_j(\tau), \tag{7}$$

from Nason *et al.* (2000), where $\Psi_j(\tau) = \sum_k \psi_{j,k}\psi_{j,k-\tau}$ is the autocorrelation wavelet of the discrete nondecimated wavelet $\psi_{j,k}$.

The quantity $c(z, \tau)$ in (7) is the autocovariance of X_t at lag τ and at rescaled time $z \in (0, 1)$. Rescaled time is related to real time by the formula $z = t/T$ for time points $t = 1, \ldots, T$ where T is the length of the time series.

Using **costat**'s `lacv` function yields:

```
R> vsim.lacv <- lacv(vsim, filter.number = 4, lag.max = 100)
R> vsim.lacv

Class 'lacv' : Localized Autocovariance/correlation Object:
       ~~~~  : List with 3 components with names
               lacv lacr date


summary(.):
----------
Name of originating time series:
Date produced:  Mon Oct  7 08:29:53 2013
Number of times:  1024
Number of lags:  101
```

The `lacv` function produces an object of class `lacv` which contains the localized autocovariance and autocorrelation information. The second command, where we typed the name of the object `vsim.lacv`, invokes the `print` method which informs the user of the type of object, subsequently calls the `summary` method which gives information on when the object was produced and its size.

The `vsim` data series contained $T = 1024$ data points. Without specifying the `lag.max = 100` argument the `vsim.lacv` would have contained large number of lags: 7162. We have also chosen to compute the localized autocovariance with the `filter.number = 4` argument which chooses a particular wavelet. Using different wavelets will result in different results for the autocovariance estimators but the differences will typically be small. The wavelet with four vanishing moments can be produced using the `draw.default` function in **wavethresh** and has

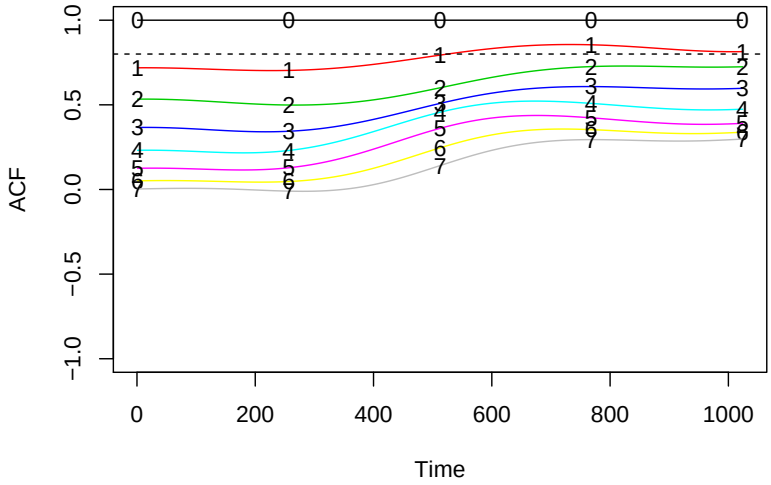

Figure 3: Localized autocorrelation plot from a realization of a stationary AR(1) process with $\alpha = 0.8$. The horizontal dashed line is plotted at height of 0.8

intermediate smoothness between the piecewise constant Haar wavelet and the very smooth wavelet with 10 vanishing moments (selected by `filter.select = 1` and `filter.select = 10` arguments respectively). More information on the different types and smoothnesses of wavelets can be found in the **wavethresh** help function on `filter.select`.

The call to `lacv` to produce `vsim.lacv` took 3.8 seconds on the iMac and 1.9 seconds on the Linux machine.

Plotting the localized autocovariance

The LACV is a function of two arguments and hence there are various ways it can be plotted to elicit information about its form.

Since the LACV is potentially a quantity that can vary over time our first plot will draw the $\hat{c}(z, \tau)$ estimates over all time $z \in (0, 1)$ for a fixed number of lags ranging from lag $\tau = 0$ to $\tau = 7$. Remember z is rescaled time and can be mapped back to the actual time points by $z = t/T$.

Execute the following commands:

```
R> plot(vsim.lacv, lags = 0:7, lcol = 1:8)
R> abline(h = 0.8, lty = 2)
```

The output from this code is shown in Figure 3. The autocorrelation at each lag $\tau = 0$ to $\tau = 7$ is plotted as a solid line in Figure 3 with the lag associated with each line plotted as integers superimposed on the line. Note, the lag $\tau = 0$ line is completely straight because autocorrelation is always one at $\tau = 0$ for regular and localized versions.

At lag $\tau = 1$ the theoretical autocorrelation value for this process is 0.8. The regular ACF estimate computed above by `acf` was 0.788. Figure 3 shows the localized autocorrelation

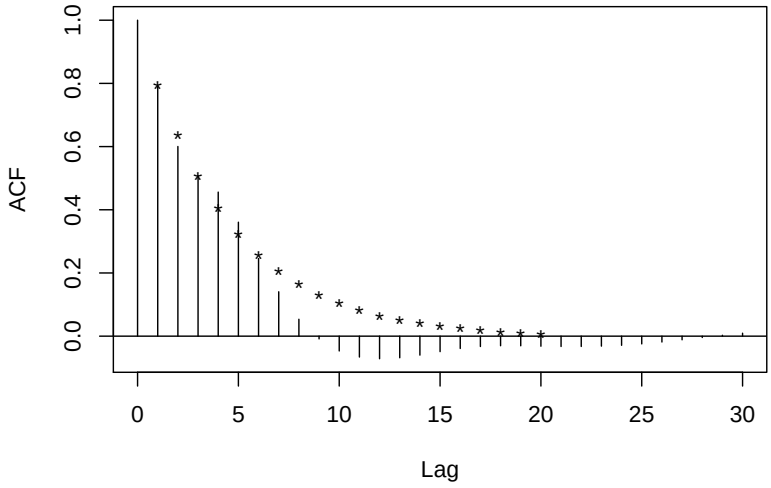

Figure 4: Localized autocorrelations plotted on an `acf` style plot at time $t = 512$ from a realization of a stationary AR(1) process with $\alpha = 0.8$. The stars are the values of the true autocorrelations for this process.

varies: on the first half of the series it is below 0.8 and for the second half it tends to be above. Although the underlying process is stationary, and all theoretical autocorrelations constant, the `lacv` function does not assume this and so all the estimated autocorrelations vary (apart from the lag zero). However, the values of $\hat{c}(z, \tau)$ are not highly variable.

Would we produce a plot such as Figure 3 in practice? Well, normally we would perform a test of stationarity first. Applying `BootTOS` to `vsim` results in a p-value of 0.82 which means that we would accept the hypothesis of stationarity for this series and, in these circumstances not compute the localized version.

The `plot.lacv` method for `lacv` objects also permits plotting of $\hat{c}(z, \tau)$ for a given fixed value of z (or t) in the style of the usual plot produced by the R function `acf`. Type:

```
R> plot(vsim.lacv, type = "acf", the.time = 512)
R> for(i in 1:20) text(i, 0.8^i, "*")
```

The first command produces the `acf`-style plot of *localized* autocorrelation values centred on the time point $t = 512$, and the second and third lines augment the plot with the true theoretical values of an AR(1) process with parameter $\alpha = 0.8$. The plot and the augmented theoretical values appears in Figure 4.

The `plot.lacv` method can also be used to produce perspective plots of the autocorrelation function using the `type = "persp"` argument.

4.2. LACV on a simulated locally stationary series

We first construct a function, `tvar1sim`, to generate realizations from a time-varying autoregressive process TVAR(1) as follows:

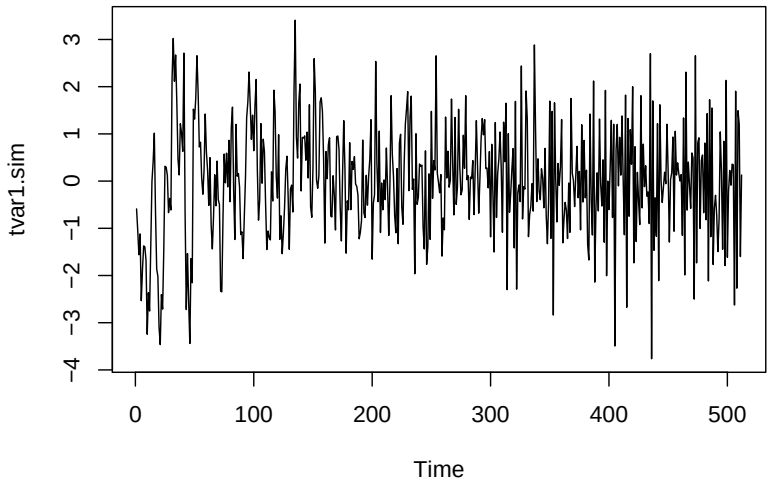

Figure 5: Realization of a TVAR(1) process with AR parameter progressing from $\alpha = 0.9$ to $\alpha = -0.9$.

```
R> tvar1sim <- function(sd = 1) {
+    n <- 512
+    arvec <- seq(from = 0.9, to = -0.9, length = 512)
+    x <- c(rnorm(1, mean = 0, sd = sd), rep(0, n - 1))
+    for(i in 2:n) x[i] <- arvec[i] * x[i - 1] + rnorm(1, mean = 0, sd = sd)
+    return(x)
+  }
```

The model behind this process is $X_t = \alpha_t X_{t-1} + \epsilon_t$, where $\{\epsilon_t\}$ is an iid white noise process with variance of σ^2. The AR parameter of this process varies from $\alpha = 0.9$ to $\alpha = -0.9$ over the length of the process.

We simulate a realization from this TVAR(1) process, and plot it using

```
R> tvar1.sim <- tvar1sim()
R> ts.plot(tvar1.sim)
```

The plot produced is shown in Figure 5. The LACV can be computed and plotted using the following commands:

```
R> tvar1.lacv <- lacv(tvar1.sim, filter.number = 4, lag.max = 50)
R> plot(tvar1.lacv, lags = 0:2, lcol = 1:3)
```

Figure 6 shows the LACV plot where we have chosen to show up to lag two for clarity. The lag one autocorrelation decays nicely from positive 0.75 to negative 0.62 across the length of the series. This decay is not perfectly linear from 0.9 to -0.9 (as specified in the function `tvar1sim()`) but the estimate is from a finite length realization and even stationary ACF

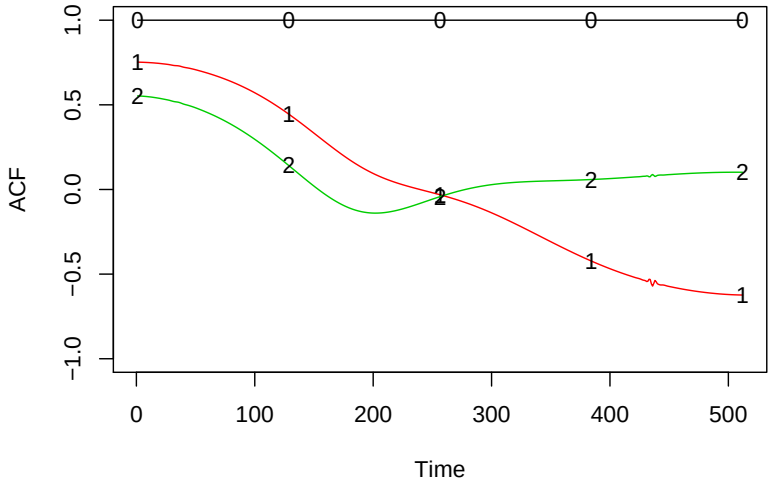

Figure 6: The LACV plot of the TVAR realization shown in Figure 5.

estimation is not perfect (For example, in the stationary AR(1) $\alpha = 0.8$ model in Section 4.1 the estimate was 0.788 and that result was achieved by averaging over 1023 pairs all with the same distribution!)

4.3. LACV on a locally stationary series

We now compute and plot the localized autocovariance function on the `sret` SP500 log-returns data set using the following commands

```
R> sret.lacv <- lacv(sret, filter.number = 4, lag.max = 50)
R> plot(sret.lacv, lags = 0:5, lcol = 1:6)
```

which produces the picture displayed in Figure 7. We have only chosen to display the first five lags so as to provide an uncluttered plot. Figure 7 shows that the autocorrelations are all reasonably small (less than 0.2 in magnitude). However, there is a spike at just prior to $t = 350$. Why do we see the spike? The answer is to look back at the actual time series and one indeed can see a negative outlier at time $t = 332$ in Figure 1. Such jump changes are permitted within the locally stationary framework and the jump is effectively localized in that the remaining autocovariances in Figure 7 will be relatively unaffected.

To see this, we remove this outlier, and a few others, using the commands:

```
R> sret.noout <- sret
R> sret.noout[abs(sret) > 0.04] <- 0
```

and then recompute and plot the localized autocovariance by

```
R> sret.noout.lacv <- lacv(sret.noout, filter.number = 4, lag.max = 50)
R> plot(sret.noout.lacv, lags = 0:5, lcol = 1:6)
```

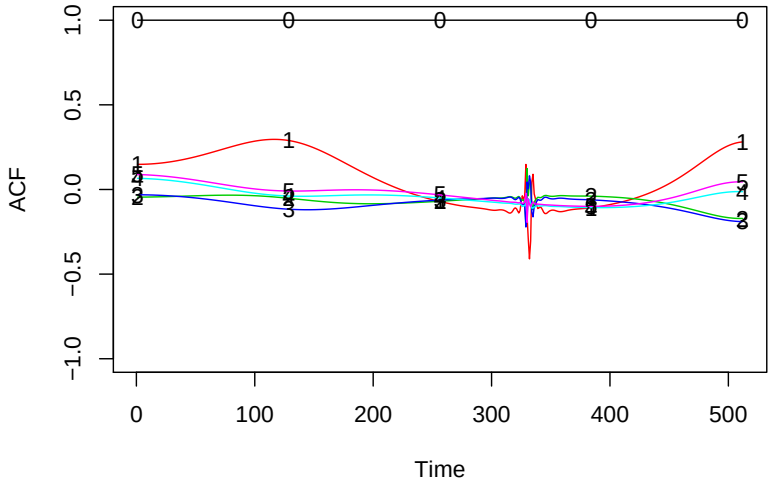

Figure 7: Localized autocorrelation values for SP500 log-returns time series `sret`.

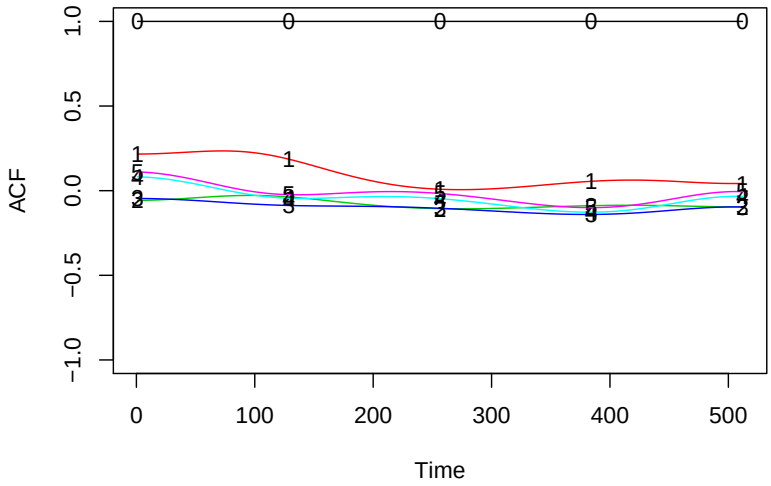

Figure 8: Localized autocorrelation values for SP500 log-returns time series with outliers removed.

The LACV plot is shown in Figure 8. Log-returns time series of this kind are often modeled with a GARCH process. One of the features of GARCH is that the empirical autocovariance function is zero for all non-zero lags and this seems to be the case here. Repeated regular applications of `acf` were applied to different portions of the series and most autocovariances on most occasions were deemed to be not significantly different from zero.

One could argue that the outlier means that the autocorrelation spike contradicts the zero autocorrelation finding. However, GARCH models often admit heavy tails and so the large negative observation would not usually be thought of as an outlier. So, although the outlier only disturbs our LACV locally it would be better, maybe, to construct a LACV estimator which is more robust to this kind of disturbance. Such a concept is desirable but beyond the scope of the current article.

5. Discovering costationarity

5.1. Commands to discover costationarity

Finally, we examine the case where we wish to discover any potential costationary solutions. That is, given time series X_t, Y_t find α_t, β_t such that we obtain (4) where Z_t is second-order stationary. The main function to discover costationarity is `findstysols`. This starts with a random choice of the vectors α_t, β_t and, on any one run, numerically optimizes the vectors to try and find the combination which results in the flattest spectrum of Z_t over time. Finally, the 'optimal' Z_t is tested using the test of stationarity mentioned above to see whether the resultant Z_t can be statistically assessed to be stationary. Since the output depends on the random input it is good practice to restart the algorithm using many random starts. The number of random starts is controlled by the `Nsims` argument. In general, you should run as many invocations with different random starts as your computational resources will allow. There is a function, `mergexy` that can merge the outputs from many runs of the `findstysols` function into a single output file. This means that parallel computation can be used to great effect by using multiple serial jobs to run from the repeated random starts. In the examples below `Nsims` is set to be 10, which is really too small. Also, `Nsims` should grow with the length of the time series. For longer series, there are typically more ways, and more flexibility, in finding stationary solutions. However, there are no hard and fast rules concerning the choice of `Nsims`.

Let us try and discover costationary solutions between the `sret` and `fret` time series. To repeat this discovery process using ten random starts we can issue the command:

```
R> sretfret.fss <- findstysols(Nsims = 10, tsx = sret, tsy = fret)
```

Note: This command takes quite some time to execute. On our dual core iMac it took 8 minutes and on the twelve core Linux machine it took 48 seconds with this using the package **multicore**/**parallel** and additionally using the `lapplyfn = "mclapply"` within the function call above.

The returned `sretfret.fss` object is of class `csFSS` and contains a lot of information concerning the solutions returned by the multiple optimizations. Printing out the results displays:

```
Class 'csFSS' : Stationary Solutions Object from costat:
       ~~~~~  : List with 13 components with names
              startpar endpar convergence minvar pvals tsx tsy tsxname
              tsyname filter.number family spec.filter.number spec.family
```

```
summary(.):
----------
Name of X time series:  sret
Name of Y time series:  fret
Length of input series:  512
There are  10  sets of solutions
Each solution vector is based on  3  coefficients
Some solutions did not converge, check convergence component for more
information. Zero indicates successful convergence, other values mean
different things and you should consult the help page for `optim' to
discover what they mean
For size level:  0.05
         0  solutions appear NOT to be stationary
         10  solutions appear to be stationary
Range of p-values: ( 0.175 , 0.99 )
```

The `summary` output contains some basic information on the names and dimensions of the input time series and solutions sought. In this case we asked for ten optimization runs to be executed and indeed ten sets of solutions were produced. However, the `summary` indicates that not all optimization runs converged. The reason why some did not converge can be gained by looking at the `convergence` component of the `csFSS` object. For example,

```
R> sretfret.fss$convergence

 [1]  0  0  0  1  0  0  0  0  0 10
```

As the text in the `summary` output indicates these numbers correspond to the status code returned by the `optim` function that was used to perform the optimizations. The meaning of the status code can be revealed by examining the help page for `optim` and looking at its `convergence` output. In particular, a code of zero indicates successful convergence, a code of one indicates that `optim` ran out of iterations before it deemed convergence had occured. The single occurence of code ten indicates degeneracy of the Nelder-Mead simplex in that method. For longer, and possibly more complex series, one often sees multiple error codes of one indicating that the optimizers require more iterations. Extra iterations can be given by increasing the `my.maxit` argument or directly passing control information to the `optim` calls by the `optim.control` argument of `findstysols`.

In any case the solution above did converge eight times and all of them appear to be stationary (in that all p-values were greater than 0.05. The value that controls this assessment for stationarity, the `size` argument in the call to `summary` can be changed, if required). So far, we have performed ten optimizations, seven have converged and look interesting. The commands described next allow us to further investigate the nature of the solutions.

5.2. Plots for obtaining an overview of all solutions

Each set of solutions consists of a set of of parameters that encode the α_t and β_t functions (actually wavelet coefficients of the functions, the number of such coefficients controlled by the `Ncoefs` argument to `findstysols`) and this is stored as a multivariate data set with the

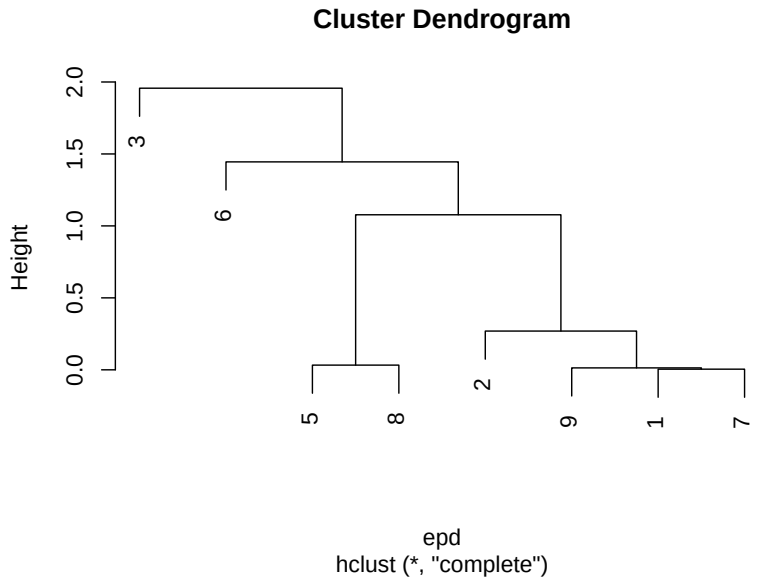

Figure 9: Dendrogram associated with the `sretfret.fss` set of solutions.

number of cases (rows) equal to the number of solutions and the number of variables equal to twice `Ncoefs` as there are coefficients associated with each of α_t and β_t.

The **costat** package provides two functions to display and interpret this information. One method uses hierarchical cluster analysis and R's `hclust` function which clusters similar solutions together. For example, for the current example, the command

```
R> plot(sretfret.fss, ALLplotscale = FALSE)
```

produces Figure 9: a dendrogram showing how solutions 1, 7, and 9 are very similar as well as solutions 5 and 8. Running the plot with the `ALLplotclust` argument set to `FALSE` causes a plot of a multidimensional scaling solution applied to a Euclidean distance calculation on the set of solutions (applying `cmdscale` to `dist` on the set of solution vectors). This latter plot would show the solution ids on a two-dimensional plot which again gives some indication of how similar different solution sets are. Suppressing both the `ALLplotclust` and `ALLplotscale` arguments causes both plots to be produced.

The idea of these 'complete set' plots is to identify which solutions are worth examining further. For example, there is little point examining both solution 1 and 7 in detail since they are very similar, so one can examine one of them and that will be representative. Typically, one would compute many more solutions. In the initial call to `findstysols` the `Nsims` argument might be a lot higher, e.g., 100. In such a case one will find that the solutions tend to group into clusters of like-valued solutions and one need only examine a single representative from each cluster, maybe using the plots described in the next section.

5.3. Plotting information about individual solutions

After the solutions are examined collectively one can then choose to investigate specific solutions. Individual solutions can be examined by using again the `plot` function with the

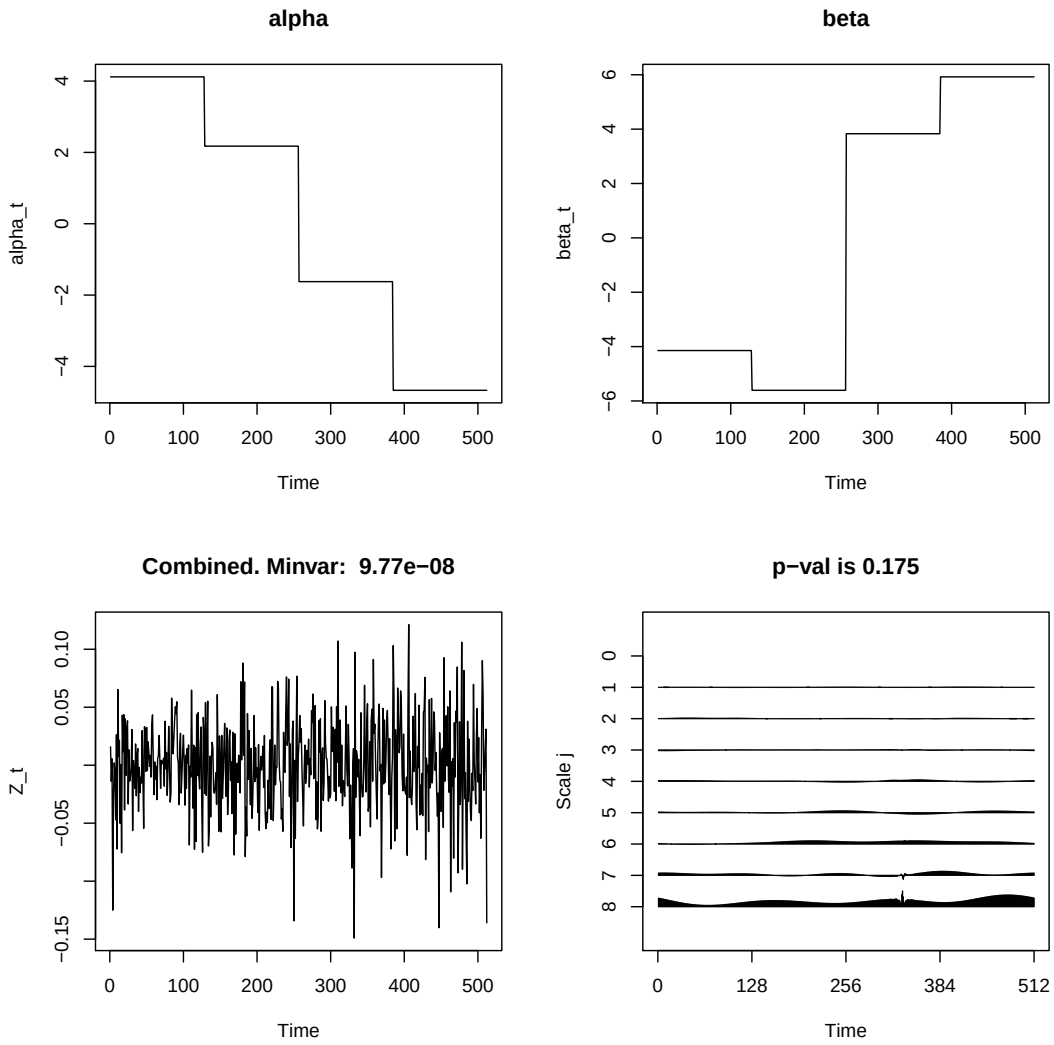

Figure 10: Top left: Optimal α_t costationary vector. Top right: Optimal β_t costationary vector. Bottom left: The combined series Z_t. Bottom right: The evolutionary wavelet spectral estimate of the series Z_t with the computed p-value for the test of stationarity for the Z_t series.

additional `solno` argument. The `solno` argument can be a vector of solution numbers and a series of four plots is produced for each solution. It can be convenient to produce all four graphics pertaining to a solution on the same plot. This can be achieved using `par(mfrow = c(2, 2))` or using `layout` such as in the following example:

```
R> def.par <- par(no.readonly = TRUE)
R> nf <- layout(matrix(1:4, 2, 2, byrow = TRUE))
R> plot(sretfret.fss, solno = 6)
R> par(def.par)
```

produces Figure 10 for solution number 6. This figure shows the costationary vectors (α_t, β_t),

that were used to produce the combined series Z_t, which is also shown along with its associated local spectral estimate. Figure 10 shows how the costationary vectors change over time. The bottom row of the plot demonstrates that the resultant Z_t series looks stationary with a fairly flat spectrum. More information can be obtained using the `plotstystat = TRUE` argument. Using this argument additionally plots the combined time series, the regular and partial autocorrelations of the combined series and a spectral estimate. These plots can be produced as the test of stationarity has deemed that this Z_t is stationary (or at least, no evidence that it is not)

References

Ahamada I, Boutahar M (2002). "Tests for Covariance Stationarity and White Noise, with an Application to Euro/US Dollar Exchange Rate: An Approach Based on the Evolutionary Spectral Density." *Economics Letters*, **77**, 177–186.

Brockwell PJ, Davis RA (1991). *Time Series: Theory and Methods*. Springer-Verlag, New York.

Cardinali A, Nason GP (2010). "Costationarity of Locally Stationary Time Series." *Journal of Time Series Economics*, **2**, Article 1.

Chatfield C (2003). *The Analysis of Time Series: An Introduction*. Chapman and Hall/CRC, London.

Constantine WLB, Percival DB (2007). ***fractal**: Insightful Fractal Time Series Modeling and Analysis*. R package version 1.0-3, URL http://CRAN.R-project.org/package=fractal.

Dahlhaus R (1997). "Fitting Time Series Models to Nonstationary Processes." *The Annals of Statistics*, **25**, 1–37.

Dahlhaus R (2012). "Locally Stationary Processes." In TS Rao, SS Rao, CR Rao (eds.), *Handbook of Statistics*, volume 30, pp. 351–413. Elsevier.

Dahlhaus R, Polonik W (2006). "Nonparametric Quasi Maximum Likelihood Estimation for Gaussian Locally Stationary Processes." *The Annals of Statistics*, **34**, 2790–2824.

Dahlhaus R, Polonik W (2009). "Empirical Spectral Processes For Locally Stationary Time Series." *Bernoulli*, **15**, 1–39.

Dickey DA, Fuller WA (1979). "Distribution of the Estimators for Autoregressive Time Series with a Unit Root." *Journal of the American Statistical Association*, **74**, 427–431.

Dwivedi Y, Rao SS (2011). "A Test for Second-Order Stationarity of a Time Series Based on the Discrete Fourier Transform." *Journal of Time Series Analysis*, **32**, 68–91.

Elliott G, Rothenberg TJ, Stock JH (1996). "Efficient Tests for an Autoregressive Unit Root." *Econometrica*, **64**, 813–836.

Engle RF, Granger CWJ (1987). "Co-Integration and Error Correction: Representation, Estimation and Testing." *Econometrica*, **55**, 251–276.

Hamilton JD (1994). *Time Series Analysis*. Princeton University Press, Princeton, New Jersey.

Hannan EJ (1960). *Time Series Analysis*. Chapman and Hall, London.

Kwiatowski D, Phillips PCB, Schmidt P, Shin Y (1992). "Testing the Null Hypothesis of Stationarity Against the Alternative of a Unit Root." *Journal of Econometrics*, **54**, 159–178.

Nason GP (2008). *Wavelet Methods in Statistics with R*. Springer-Verlag, New York.

Nason GP (2012). "Simulation Study Comparing Two Tests of Second-Order Stationarity and Confidence Intervals for Localized Autocovariance." *Technical Report 12:02*, Statistics Group, University of Bristol.

Nason GP (2013a). ***wavethresh**: Wavelets Statistics and Transforms*. R package version 4.6.5, URL http://CRAN.R-project.org/package=wavethresh.

Nason GP (2013b). "A Test for Second-Order Stationarity and Approximate Confidence Intervals for Localized Autocovariances for Locally Stationary Time Series." *Journal of the Royal Statistical Society B*, **75**. doi:10.1111/rssb.12015.

Nason GP, von Sachs R, Kroisandt G (2000). "Wavelet Processes and Adaptive Estimation of the Evolutionary Wavelet Spectrum." *Journal of the Royal Statistical Society B*, **62**, 271–292.

Paparoditis E (2009). "Testing Temporal Constancy of the Spectral Structure of a Time Series." *Bernoulli*, **15**, 1190–1221.

Phillips PCB, Perron P (1988). "Testing for a Unit Root in Time Series." *Biometrika*, **75**, 335–346.

Priestley MB (1983). *Spectral Analysis and Time Series*. Academic Press, London.

Priestley MB, Rao TS (1969). "A Test for Non-Stationarity of Time-Series." *Journal of the Royal Statistical Society B*, **31**, 140–149.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Sanderson J, Fryzlewicz PZ, Jones M (2010). "Measuring Dependence between Non-Stationary Time Series Using the Locally Stationary Wavelet Model." *Biometrika*, **97**, 435–446.

Shumway RH, Stoffer DS (2006). *Time Series Analysis and Its Applications with R Examples*. Springer-Verlag, New York.

Stărică C, Granger C (2005). "Nonstationarities in Stock Returns." *Review of Economics and Statistics*, **87**, 503–522.

Urbanek S (2011). ***multicore**: Parallel Processing of R Code on Machines with Multiple Cores or CPUs*. R package version 0.1-7, URL http://CRAN.R-project.org/package=multicore.

von Sachs R, Neumann MH (2000). "A Wavelet-Based Test for Stationarity." *Journal of Time Series Analysis*, **21**, 597–613.

Wuertz D, Chalabi Y (2013). ***timeSeries: Rmetrics*** – *Financial Time Series Objects*. R package version 3010.97, URL http://CRAN.R-project.org/package=timeSeries.

Affiliation:

Alessandro Cardinali (CTEU), Guy Nason (School of Mathematics)
University of Bristol
University Walk
BRISTOL BS8 1TW, United Kingdom
E-mail: a.cardinali@bristol.ac.uk, g.p.nason@bristol.ac.uk
URL: http://www.stats.bris.ac.uk/~maxac/, http://www.maths.bris.ac.uk/~guy/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 55, Issue 1 *Submitted:* 2011-02-22
October 2013 *Accepted:* 2013-03-19