Package: costat (via r-universe)

October 10, 2024

Type Package

Title Time Series Costationarity Determination

Version 2.4.1

Date 2023-09-06

Depends R (>= 2.14), wavethresh (>= 4.6.1)

Suggests parallel

Description Contains functions that can determine whether a time series is second-order stationary or not (and hence evidence for locally stationarity). Given two non-stationary series (i.e. locally stationary series) this package can then discover time-varying linear combinations that are second-order stationary. Cardinali, A. and Nason, G.P. (2013) <doi:10.18637/jss.v055.i01>.

License GPL (>= 2)

NeedsCompilation no

Author Guy Nason [aut, cre], Alessandro Cardinali [aut, ctb]

Maintainer Guy Nason <g.nason@imperial.ac.uk>

Repository CRAN

Date/Publication 2023-09-06 21:32:33 UTC

Contents

`costat-package` *Computes localized autocovariance and searches for costationary solutions to bivariate time series.*

Description

Computes a time-varying autocovariance and associated plots for plotting this. Also can search for costationary solutions between two time series.

Details

See `findstysols` for help page for main function.

Author(s)

Guy Nason, <g.nason@imperial.ac.uk>

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols, lacv

Examples

```
#
# Compute localized acv
#
x <- c(rnorm(128, sd=1), rnorm(128, sd=3))
xlacv <- lacv(x, lag.max=30)
#
# Plot the time-varying autocovariance at time t=100
#
## Not run: plot(xlacv, type="acf", the.time=100, plotcor=FALSE)
#
# Plot the time-varying autocovariance at time t=400
#
## Not run: plot(xlacv, type="acf", the.time=400, plotcor=FALSE)
#
# See examples for findstysols for other examples
#
```

AntiAR *Undo autoreflection action for an EWS object (wd stationary)*

Description

The BootTOS function has the ability to deal with boundary conditions by augmenting the right-hand end of a time series by a reflected version of that series. So, the series doubles in length and the new vector has periodic boundary conditions. One can then compute a local spectrum on this data which returns an EWS in a wd object, usually with a type attribute of "station". This function can take this wd object and properly can return the first half of it, which corresponds to the boundary-correct spectrum of the original series.

Usage

```
AntiAR(S)
```

Arguments

S	A wd class object of type "station". This corresponds to a EWS estimate on a reflected time series.

Details

This function arises because using spectral estimation functions, like ewspec from the wavethresh package doesn't always work that well at the boundaries. This is because the wavelet functions in wavethresh usually assume periodic boundary conditions and this is not appropriate for a discrete

time series where time 1 and time T are usually very different (and cannot be assumed to be the same).

Hence, a previous function could generate a new time series by taking the original, e.g. x, reflecting it with rev(x) and then sticking the reflected onto the right-hand end of the original. Spectral estimation, (e.g. using ewspec) can then be applied to this new reflected/augmented series and the boundaries are now roughly correct as the start and end of the series correspond to time 1.

The spectral estimate so obtained though is double the size of the the one that is needed, and contains the spectrum of the reflected series. Hence, this function obtains the first half of the estimate and returns it.

Not usually intended for the casual user

Value

A wd class object containing the boundary-corrected estimate of the spectrum for the original series.

Author(s)

G. P. Nason.

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

BootTOS

Examples

```
#
# Generate example, temporary series
#
x <- rnorm(128)
#
# Reflect it about its end point
#
x2 <- c(x, rev(x))
#
# Compute EWS estimate
#
x2ews <- ewspec(x2)
#
# Now get bit corresponding to x into object
#
xews <- AntiAR(x2ews$S)
```

`BootTOS` *Perform bootstrap stationarity test for time series*

Description

Given a time series this function runs a bootstrap hypothesis test to see whether it is stationary. The null hypothesis is that the series is stationary, the alternative is that it is not - and hence possesses a time-varying evolutionary wavelet spectrum if deemed non-stationary.

Usage

```
BootTOS(x, Bsims = 100, WPsmooth = TRUE, verbose = FALSE, plot.avspec = FALSE,
        plot.avsim = FALSE, theTS = TOSts, AutoReflect=TRUE, lapplyfn=lapply)
```

Arguments

`x`	Time series to test. Must have a power of two length
`Bsims`	Number of bootstrap simulations to carry out
`WPsmooth`	Whether or not to carry out wavelet periodogram smoothing
`verbose`	If `TRUE` informative messages are printed
`plot.avspec`	If `TRUE` then the 'average' evolutionary wavelet spectrum (EWS) is plotted. This is called $\bar{S}_j$ in the Cardinali and Nason paper.
`plot.avsim`	If `TRUE` for each bootstrap simulation plot the time series of the simulated time series from the average EWS (the one that might be plotted by `plot.avspec=TRUE`
`theTS`	Specifies the particular test statistic to be used
`AutoReflect`	If TRUE then the series is reflected and augmented by its end point on the RH-side, and the spectral quantities are evaluated on that. Everything returned though applies only to the original series, the reflection is merely to ensure that the periodic wavelet algorithms can be used on non-periodic data
`lapplyfn`	List processing function. Parallel processing of the bootstrap simulations can be achieved by using the `multicore` package and the `mclapply` function. Sequential processing can be achieved using the standard `lapply` function. So, if you can't run multicore then you should use `lapply`, otherwise try and use `mclapply` for faster execution times.

Details

The details of our testing methodology are set out in the Cardinali and Nason paper referenced below.

Essentially, the testing process works as follows. First, one has to define a test statistic. Given a time series this has return a statistic that measures 'degree of nonstationarity'. For example, estimating the EWS, and then computing the sum of the sample variances of each scale is such as measure (and known as the T_{vS} statistic). This statistic is zero for a constant spectrum and positive for non-constant spectrum (and generally larger for larger variations of the spectrum).

Once a test statistic T is selected then a parametric Monte Carlo test can be used. First, T is computed on the series itself. Then, for statistical assessment of the 'significance' of the test statistic the following procedure is carried out. Assuming, for a moment that the time series is stationary, we estimate its evolutionary wavelet spectrum (EWS) and then average this over time ($\bar{S}_j$). Then we use the function `LSWsim` to simulate a time series whose EWS is the constant, stationary, spectral estimate. Then we compute our test statistic, T_b, on this simulated series.

Then we calculate T_b for `Bsim-1` simulations. The function then returns `BSim` numbers. The first is the test statistic computed on the actual data. The remaining ones are the test statistic computed on the simulated stationary series.

The idea being that if the time series is really stationary then the first value will be comparable to the ones obtained by simulation. If the time series is not stationary then the first test statistic will be much larger than the ones obtained by simulation (since the actual data T will have been computed on a time series with varying spectrum, whereas the simulated ones are all computed on constant spectra, and their variation is only due to sampling variation).

The test statistic supplied to this function (as argument `theTS`) should take an EWS object as an argument. For example, the WaveThresh function `ewspec` produces a suitable spectral estimate in its `$S` argument (both objects are actually examples of a non-decimated wavelet transform object, class `wd`).

The function `plotBS` can be used the present the results of this function in an interpretable form and calculate the p-value of the test, although you should use the generic `plot` function to call this.

Value

A vector of length `Bsim`. The first entry is the value of the test statistic computed on the data. The remaining entries are boostrap values computed on the 'averaged' EWS estimate with constant spectrum.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`TOSts`, `plotBS`

Examples

```
#
# Calculate test of stationarity on example we know to be stationary,
# a series of iid values
#
```

```
plot(BootTOS(rnorm(64), Bsims=10), plot=FALSE)
#
# The following text is what gets printed
#
#Realized Bootstrap is  0.04543729
#p-value is  0.93
#Series was stationary
#[1] 0.93
#
# The realized bootstrap value is the value of the test statistic on the
# actual data (0.0454 here).
#
# The p-value is also printed (this is just the number of simulated series
# test statistic values less than the actual test statistic) and returned.
#
# The text "Series is stationary" just means that the empirical p-value
# was greater than the nominal test size (alpha=0.05, by default).
#
# Let's now try another example with the series sret: note that if you
# have a slow single core machine, this can take a long time, so we don't
# run it in the examples. However, on a fastish machine it is quick, on
# a fast multicore machine it is really quick!
#
## Not run: plot(BootTOS(sret))
#
#Realized Bootstrap is  2.662611e-09
#p-value is  0
#Series was NOT stationary
#[1] 0
#
# In contrast to the previous example, the p-value is 0, hence indicative
# of non-stationarity.
#
```

`COEFbothscale` *Produces plots from output of findstysol that attempt to group different solutions.*

Description

Uses hierarchical clustering and multidimensional scaling to produce a plot of all the convergence stationary solutions. These plots are designed to aid the user in identifying 'unique' sets of stationary solutions.

Usage

```
COEFbothscale(l, plotclustonly = FALSE, StyPval=0.05, ...)
```

Arguments

l	An object returned by findstysols, of class csFSS, which contains the results of an optimization to find solutions that correspond to stationary series which are the time-varying linear combination of two locally stationary time series.
plotclustonly	If TRUE then only produce the hierarchical clustering plot.
StyPval	The p-value by which solutions are deemed to be stationary or not for inclusion into plots. If the p-value for a particular solution is greater than StyPval then the solution is deemed stationary and included.
...	Additional arguments to the hierarchical clustering plot.

Details

The function findstysols uses numerical optimization to try and discover time-varying linear combinations of two time series to find a combination which is stationary. Like many numerical optimizations the optimizer is supplied with starting coordinates and proceeds through an optimization routine to end coordinates which are located at the minimum (in this case). So, the user has a choice over where to start each optimization.

A priori there is no recipe for knowing where to start the optimizer, so such situations are usually handled by running the optimizer many time each time starting in a different position. The solution here is to start from a set of different randomly chosen starting points. After the optimizer is run from these different starting positions it ends up in the same number of potentially different ending positions.

However, some of the ending solutions might be identical, some might be very close, some might be reflections (e.g. the if the coefficients (a,b) result in a stationary solution then so does (-a, -b)). Morally, though, all of these cases would reference the same solution.

Hence, we require some method for identifying the set of unique solutions. We can be considerably aided in this task by multidimensional scaling (which uses inter-solution distances to produce a map of how close solution sets really are) or hierarchical clustering (which can produce a nice picture to indicate how the solutions might be related).

In other words, the solution vectors can be viewed as a multivariate data set where the cases correspond to the results of different optimization runs and the variables correspond to the coefficients of the time-varying linear combinations.

Both multidimensional scaling (cmdscale) and hiearchical clustering (hclust) are used to determine possible clusterings of solutions. Then, representative members from these clusters can be further investigated with a function such as LCTSres

Value

An object of class csFSSgr is returned containing the following components: the results of the multidimensional scaling and hierarchical clustering are returned as list with two components epscale and epclust respectively, and the input l object is returned as component x and the StyPval object is returned as a component.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols, LCTSres

Examples

```
#
# See example in findstysols
#
```

`coeftofn` *Convert wavelet coefficients for two time-varying functions into two functions with respect to time.*

Description

In much of the costationarity code the combination functions are represented in terms of wavelet coefficients. At certain points the actual combination functions themselves are required (in the time domain) for purposes such as actually forming the linear combination. This function turns the coefficients, for the two combination functions, into their time domain functional representation.

Usage

```
coeftofn(alpha, beta, n = 256, filter.number = 1,
 family = c("DaubExPhase", "DaubLeAsymm"))
```

Arguments

`alpha`	One set of coefficients for one of the combination functions
`beta`	The other set of coefficients
`n`	The length of resulting function that you require
`filter.number`	The type of wavelet (the number of vanishing moments)
`family`	The type of wavelet (the wavelet family)

Details

A degree of efficiency is built into the code. Typically, for forming stationary linear combinations then only a few (or at least a medium number) of coarser scale coefficients need to be manipulated (eg modified in the optimizer). However, the actual length of the function (time series length) is typically much longer (e.g. n=256, n=512, or higher). So, this function pads out the small number of coarse coefficients with zeros before forming the combination functions which end up at the correct length, n.

Value

An object of class `csBiFunction` which is list containing two components:

`alpha`	A vector, of length n, containing one of the time-varying combination functions
`beta`	Same as `alpha`, but contains the other combination function.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`LCTS`, `LCTSres`

Examples

```
#
# Very artifical example
#
tmp.a <- c(1, -1)
tmp.b <- c(0.5, 0.5)
#
#
#
ans <- coeftofn(tmp.a, tmp.b)
#
# Print it out
#
ans
#Class 'csBiFunction' : Contains two sampled functions:
#       ~~~~  : List with 2 components with names
#               alpha beta
#
#
#summary(.):
#----------
#Length of functions is:  256
```

EWSsmoothRM *Perform running mean smoothing of an EWS object*

Description

Performs running mean smoothing of bandwidth s of an EWS, such as that returned by the ewspec function of wavethresh.

Usage

```
EWSsmoothRM(S, s)
```

Arguments

S	The spectrum to smooth
s	The bandwidth (or number of ordinates to include in the running mean)

Details

Each level of the EWS is subject to a running mean smooth. After smoothing a level the resultant smooth is shorter than the original level (due to the mean not being able to overlap the boundaries). This deficit is made up by augmenting the start of the smooth with a right number of smoothed values taken from the first smoothed value.

Value

A EWS object contained in a wd object of type "station" which contains the smoothed spectrum.

Author(s)

G.P. Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

lacv

Examples

```
#
# Make dummy time series
#
x <- rnorm(128)
#
# Compute spectrum, but don't do smoothing
#
xews <- ewspec(x, WPsmooth=FALSE)$S
#
# Now smooth the spectrum using running mean smoothing with bandwidth of 5
#
ans <- EWSsmoothRM(xews, s=5)
```

`extractCS` *Extractor function for* `csFSS` *object.*

Description

Get much information from the slots of a `csFSS`. Each slot can carry information from multiple solutions per slot. This function permits an arbitrary selection of solutions for information from a slot.

Usage

```
extractCS(object, slot=c("startpar", "endpar", "convergence",
 "minvar", "pvals", "lcts"), coeftype=c("all", "alpha", "beta",
 "alphafunc", "betafunc"), solno, ...)
```

Arguments

`object`	The `csFSS` object that you want to extract information from.
`slot`	The slot that you want to get information on. These are `startpar`: the starting parameters for the optimization for each solution; `endpar`: the final parameters calculated by the optimization for each solution; `convergence`: the status codes returned by the optimization for each solution; `minvar`: the minimum variance of the spectral estimate at the optimial solution, one for each solution; `pvals`: the p-values for the test of stationarity for the final optimal parameter set; `lcts`: the (time-varying) linear combination of the time series, one for each solution. These are the Z_t time series, the combined series which are meant to be stationary. The `startpar`, `endpar` and `lcts` slots return result in one vector for each solution requested, organized as a matrix. Each row of the matrix corresponds to one of the solutions requested. The remaining slots return numbers, one number for each solution organized as a vector.

`coeftype` For the slots that return coefficients, these can be returned in various ways. Each coefficient vector (one per solution) actually stores two sets of coefficients: one associated with the alpha_t linear combination and the other with the beta_t linear combination. Setting `coeftype` to the following causes the following to happen: `all`: the complete vector of coefficients is returned (these are actually wavelet coefficients corresponding to the wavelet specification in the `csFSS` object); `alpha`: only the alpha_t coefficients are returned; `beta`: only the beta_t coefficients are returned; `alphafunc`: the alpha_t function (in the time domain) is returned, ie as a function in time rather than a set of transform coefficients; `betafunc`: as for `alphafunc` but for the beta_t function.

`solno` The indices of which solutions you want the information on

`...` Other arguments to `coeftofn`. For example, by default the length of the functional representations of alpha_t and beta_t is 256 caused by the default `n=256` of the `coeftofn`

argument.

Details

Extracts slot information from `csFSS` objects.

Value

Information from the relevant slot, as a number, vector or matrix depending on what it is that is requested as described in the various arguments above.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`findstysols`, `coeftofn`

Examples

```
#
# Create dummy data
#
x1 <- rnorm(32)
y1 <- rnorm(32)
#
# Find stationary combinations
```

```
# Note: we don't run this example in installation/package formation as
# it takes a long time. However, this precise command IS run in
# the help to findstysols
#
## Not run: ans <- findstysols(Nsims=100, tsx=x1, tsy=y1)
#
# Get the optimal (endpar) alphas for the first 10 solutions
#
## Not run: extractCS(ans, slot="endpar", coeftype="alpha", solno=1:10)
#
# Plot the beta_t associate with the optimal solution for solution 29
#
## Not run: ts.plot(extractCS(ans, slot="endpar", coeftype="betafunc",
 solno=29))
## End(Not run)
#
# Get the p-value associated with solution 29
#
## Not run: extractCS(ans, slot="pvals", solno=29)
```

`findstysols` *Given two time series find some time-varying linear combinations that are stationary.*

Description

Find some time-varying linear combinations of two time series that are stationary. The complexity of the time-varying combinations is restricted by the `Ncoefs` argument.

Usage

```
findstysols(Nsims = 100, Ncoefs = 3, tsx, tsy, sf=100, plot.it = FALSE,
        print.it=FALSE, verbose = FALSE, lctsfn=LCTS, prodcomb.fn=prodcomb,
        filter.number=1, family=c("DaubExPhase", "DaubLeAsymm"),
 my.maxit=500, spec.filter.number=1,
 spec.family=c("DaubExPhase","DaubLeAsymm"),
        optim.control=list(maxit=my.maxit, reltol=1e-6),
        irng=rnorm, lapplyfn=lapply, Bsims=200, ...)
```

Arguments

`Nsims`	Number of searches attempted
`Ncoefs`	Number of Haar wavelet coefficients to use. Must be >= 1. Should only increase in steps of powers of two. E.g. can only supply the values 1, 3, 7, 15, etc. So, "1" means only one coarse scale coefficient (corresponds to piecewise constant with one centrally located jump), "3" means one coarse, and two next coarse scale coefficients (corresponds to piecewise constant with 4 equally sized piece with jumps at 1/4, 1/2 and 3/4), "7" means one coarse, two next coarse, four next coarse, and so on.

tsx	One of the time series
tsy	The other time series, values at the same time locations as tsx
sf	A scale factor to multiply both time series by (not really of much use)
plot.it	If TRUE then the plot.it argument passed to LCTS via optim is made TRUE. This has the effect of plotting the results of every trial in the optimation (what actually is plotted is described in the help to LCTS
print.it	Not currently used in this function, reserved for future use
verbose	If TRUE then helpful messages get printed.
lctsfn	The function to compute the 'linear combination test of stationarity'. I.e. it is the function that combines the two series and returns the value of the test statistic on the combination.
prodcomb.fn	The function that can produce the linear combination of the two time series and return the combination, and optionally vectors containing the combination functions.
filter.number	Gets passed to lctsfn and prodcomb.fn
family	Gets passed to lctsfn and prodcomb.fn
my.maxit	Maximum number of iterations in the optimization. May need to be increased to, e.g. 1000 or 2000 for longer time series (e.g. T=2048)
spec.filter.number	Wavelet filter number. This argument gets passed to the lctsfn and is used for the wavelet for all spectral smoothing.
spec.family	Same as spec.filter.number but for the wavelet family.
optim.control	Argument passed to the optim optimizer as its control argument. optim performs optimization. See help page for optim.
irng	Random number generator used to generate coefficients for starting parameters for the linear combination of time series (actually wavelet coefficients of the combination functions)
lapplyfn	Function to use to process lists. If this argument is mclapply then the multicore library function mclapply is used to parallel process the lists. If you don't have multicore then the lapply function can be used to process things sequentially.
Bsims	The number of bootstrap simulations for the (single) test of stationarity BootTOS.
...	Other arguments, passed to the optim call.

Details

The function searches for time-varying linear combinations of two time series, tsa and tsy, such that the combination is stationary (according to the TOSts test statistic).

Each linear combination is parametrised by a coarse scale Haar wavelet decomposition (controlled by Ncoefs). Initially, the Haar wavelet coefficients (up to a fixed finite scale, controlled by Ncoefs) are randomly chosen. These coefficients are converted to functions α_t, β_t by the coeftofn function and then a linear combination with the time series is formed out of those and the time series, i.e. $Z_t = \alpha_t x_t + \beta_t y_t$ The non-stationarity of Z_t is measured using the TOSts test statistic and this value is minimized over the coarse scale Haar wavelet coefficients.

This optimization procedure is repeated Nsims times. If the lapplyfn is set to mclapply then this function from the multicore package is used to process the lists in parallel.

This function can be called multiple times (e.g. on different processors in a multiprocessor environment. The result sets from different runs can be combined using the mergexy function.

The variance Ncoefs is very important, it controls the complexity of the linear combinations. If it is too big the linear combinations themselves can be extremely oscillatory and stationarity is easy to obtain. Small values of Ncoefs results in piecewise constant functions with fewer jumps.

The Ncoefs value must take the value of $2^k - 1$. If this is the case the k is the number of scale levels present in the Haar representation of the combining function α_t, β_t (excluding the scaling function coefficient, just the wavelet coefficients from the coarsest scale).

The functions to compute the linear combination and also the test statistic on that combination, and just to compute the combination and return also (optionally) the combination vectors are supplied in lctsfn and prodcomb.fn. By default, these are just the LCTS and prodcomb functions. However, it is possible to recode these to look at operating on combinations that operate on portfolios. I.e. rather than look at linear combinations of log-returns (which if tsx and tsy were) one can look at linear combinations of actual series (ie portfolios) and then look for stationarity of log-returns of the portfolios. These functions will be made available in a later package.

Value

An object of class csFSS which is a list with the following components.

startpar	A matrix with Nsims rows and 2*Ncoefs columns containing the initial random coefficients of the linear combination functions, one row for each optimization run. The first Ncoefs numbers on each row correspond to the α_t coefficients, the second Ncoefs numbers correspond to the β_t coefficients.
endpar	Same dimension as startpar except containing the final coefficients obtained after running the optimizer. If, for a particuar run, the optimizer converged and the p-value is less than 0.05 then one can say that this solution represents a valid time-varying linear combination where the combination is stationary (coefficient storage format as for startpar).
convergence	A vector of length Nsims. Reports the convergence code from optim for each optimization run. A value of 0 indicates successful convergence.
minvar	A vector of length Nsims. Contains the minimum variance achieved on each run.
pvals	A vector of length Nsims. Contains the p-values achieved on each run.
tsx	The tsx time series that was supplied to this function
tsy	The tsy time series that was supplied to this function
tsxname	The name of the tsx object that was supplied
tsyname	The name of the tsy object that was supplied
filter.number	The filter number that was used
family	The wavelet family that was used
spec.filter.number	The filter number that was used
spec.family	The wavelet family that was used

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

LCTS, BootTOS,plotBS, prodcomb, COEFbothscale, LCTSres, print.csFSS, summary.csFSS, plot.csFSS

Examples

```
#
# Find some stationary solutions with \code{Ncoefs=3}.
#
# Note: this is a toy example
#
tsx1 <- rnorm(32) # A x time series
tsy1 <- rnorm(32) # A y time series
#
# Find costationary solutions, but only from 2 random starts
#
# Typically, the length of tsx and tsy would be bigger (eg sret, fret are
# other examples you might use). Also, Nsims would be bigger, you need
# to use many random starts to ensure good coverage of the solution
# space, e.g. Nsims=100
#
# Note: the following examples are not run so as to adhere to CRAN
# requirements for package execution timings
#
## Not run: ans <- findstysols(Nsims=3, tsx=tsx1, tsy=tsy1)
#
# Print out a summary of the results
#
## Not run: ans
#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:  tsx1
#Name of Y time series:  tsy1
#Length of input series:  32
#There are  3  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
#  3  solutions appear to be stationary
#Range of p-values: ( 0.93 , 0.995 )
#
#Wavelet filter for combinations:  1   DaubExPhase
#Wavelet filter for spectrum:  1   DaubExPhase
#
#_______________
#
# Ok. The printout above suggests that some solutions did not converge.
# Which ones?
#
## Not run: ans$convergence
# [1] 0 1 0
#
# The second one did not converge, the others did. Good. The printout
# above also indicates that all the resultant solutions were stationary
# (this is not surprising for this example, as the inputs tsx1 and tsy1
# are stationary, and indeed iid).
#
# Let's see how the solutions compare. For example, let's plot the
# hierarchical cluster analysis of the final solutions (those that
# converged and are stationary)
#
## Not run: plot(ans, ALLplotscale=FALSE)
#
# My cluster shows that solution 1 and 3 are similar. Let's
# view solution 3.
#
## Not run: oldpar <- par(mfrow=c(2,2))
## Not run: plot(ans, solno=3)
## Not run: par(oldpar)
```

`fret` *Particular section of FTSE log-return series.*

Description

Observations 256:767 from the SP500 log-returns series stored in `SP500FTSElr` dataset.

Usage

```
data(fret)
```

Format

A vector of 512 observations of the FTSE100 log-returns series

Details

Its just more convenient to refer to `fret` than to `SP500FTSElr[256:767,3]`.

Source

Yahoo! Finance

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

Examples

```
## Not run: ts.plot(fret)
```

---

getpvals *Form a particular linear combination of two time series and assess the combination's stationarity p-value*

Description

Given two time series, a set of combination coefficients, a function to combine them, this function makes the combination, tests the combination for stationarity, and returns the pvalue. Effectively, returns "how stationary" the combination is.

Usage

```
getpvals(par, prodcomb.fn, tsx, tsy, filter.number,
 family=c("DaubExPhase", "DaubLeAsymm"),
 verbose, tos = BootTOS, Bsims = 100, lapplyfn = lapply)
```

Arguments

`par`	The coefficients used to make the combination via the `prodcomb.fn` function.
`prodcomb.fn`	The function which computes the combination given the two time series and the combination parameters.
`tsx`	One of the time series.
`tsy`	The other time series.
`filter.number`	Wavelet smoothness to be used in the time series combination.

family	Wavelet family to be used in the time series combination.
verbose	Supplied directly to the call to plotBS function.
tos	The function the computes a test of stationarity
Bsims	Number of bootstrap simulations the test uses (if it does)
lapplyfn	The function used to process lists. Can be the regular lapply. If you have multicore package then can be the mclapply parallel processing to process the bootstraps in parallel.

Value

A single number between zero and one indicating the p-value from the hypothesis test of stationarity of the combination.

Author(s)

G. P. Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols

Examples

```
#
# Generate two toy time series data sets
#
x1 <- rnorm(32)
y1 <- rnorm(32)
#
# Generate two toy sets of parameters (for combination)
#
tmp.a <- c(1,-1)
tmp.b <- c(0.5, 0.5)
#
# Call the function and find out the degree of stationarity of this
# combination
#
## Not run: ans <- getpvals(c(tmp.a, tmp.b), prodcomb.fn=prodcomb, tsx=x1, tsy=y1,
        filter.number=1, family="DaubExPhase")
## End(Not run)
#
# What is the p-value?
#
```

```
## Not run: ans
# [1] 0.53
```

`lacv` *Computes localized (wavelet) autocovariance function*

Description

Compute the LACV function for a locally stationary wavelet process.

Usage

```
lacv(x, filter.number = 10,
 family = c("DaubExPhase", "DaubLeAsymm"), smooth.dev=var,
 AutoReflect=TRUE, lag.max=NULL, smooth.RM=0, ...)
```

Arguments

`x`	The time series you want to compute the LACV for
`filter.number`	The wavelet that you wish to compute the LACV with respect to
`family`	The wavelet family
`smooth.dev`	The deviance used in smoothing if running mean smoothing is not used, ie in the call to `ewspec`.
`AutoReflect`	If TRUE then the spectrum is computed on a boundary-corrected series, overcoming the lack of periodicity in the time series.
`lag.max`	The maximum lag that the function computes. If this option is `NULL` then the largest possible will be computed and used
`smooth.RM`	If this is zero then regular wavelet smoothing of the periodogram will be used. If not zero then running mean smoothing of the periodogram will be used with a bandwidth given by this argument.
`...`	Additional arguments to the spectrum computation contained within

Details

A locally stationary wavelet process is a particular kind of non-stationary time series constructed out of wavelet atoms, with a time-varying spectrum (slowly varying). This kind of model is useful for time series whose spectral properties change over time.

The time-varying spectrum can be computed from within the WaveThresh library by the `ewspec` function. However, just as in the classical stationary case, where the spectrum and autocovariance are a Fourier transform pair, the paper Nason, von Sachs, Kroisandt (2000) [NvSK2000] shows that the evolutionary wavelet spectrum is paired to a localized autocovariance function using a wavelet-like transform. This is expressed in formula (14) of the NvSK2000 paper.

This function computes the localized autocovariance by first computing the estimate of the evolutionary spectrum, and then directly transforming it using formula (14) via the autocorrelation wavelet transform.

Value

An object of class `lacv`. This is a list with the following components: `lacv` which is a matrix that contains the localized autocovariance. If the original time series was of length T, then the number of rows of the returned matrix is also T, one row for each time point. The columns of the array correspond to the lag. The number of columns, 2K+1, depends both on the length of the time series and also the order of the wavelet (smoother wavelets return lacv matrices with larger number of lags). Lag 0 is always the centre column, with negative lags from -K to -1 are the leftmost columns, lags from 1 to K are the rightmost columns; `lacr`: a matrix, with the same dimensions as `lacv` but containing the local autocorrelations; `date`: the date this function was executed.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

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

See Also

`ewspec`, `print.lacv`, `plot.lacv`, `summary.lacv`

Examples

```
#
# Generate an AR(1) time series
#
vsim <- arima.sim(model=list(ar=0.8), n=1024)
#
# Compute the ACF of this stationary series
#
vsim.acf <- acf(vsim, plot=FALSE)
#
# Compute the localized autocovariance. We'll use
# a reasonably smooth wavelet.
#
vsim.lacv <- lacv(vsim, filter.number=4, lag.max=30)
#
# Now plot the time-varying autocorrelations, only the first 5 lags
#
## Not run: plot(vsim.lacv, lags=0:5)
#
# Now plot the localized autocorrelation at time t=100, a plot similar
# to the usual R acf plot.
#
```

```
## Not run: plot(vsim.lacv, type="acf", the.time=100)
```

LCTS *Computes a Linear Combination Test Statistics*

Description

Given a particular linear combination, specified in terms of coefficients, cfs, this functions forms the linear combination of two time series, tsx, tsy and returns the result of a stationarity test statistic on the combination.

Usage

```
LCTS(cfs, tsx, tsy, filter.number = 1,
 family = c("DaubExPhase", "DaubLeAsymm"), plot.it = FALSE,
        spec.filter.number = 1,
 spec.family = c("DaubExPhase", "DaubLeAsymm"))
```

Arguments

`cfs`	Coefficients describing the linear combination vectors. The first half correspond to the first vector (alpha) the second half to the beta vector. Hence this vector must have an even length, and each half has a length a power of two minus one.
`tsx`	The x time series
`tsy`	The y time series
`filter.number`	This function turns the coefficients into a linear combination function (e.g. alpha). This argument specifies the filter.number of the inverse wavelet transform that turns coefficients into a lc function.
`family`	Same as filter.number but for the wavelet family
`plot.it`	If TRUE then various things are plotted: both of the linear combination vectors/time series, the combined time series and its EWS estimate
`spec.filter.number`	The wavelet filter used to compute the EWS estimate
`spec.family`	The wavelet family used to compute the EWS estimate

Details

This function forms a time-varying linear combination of two times series to form a third time series. Then a 'stationarity test' test statistic is applied to the third time series to compute how stationary (or non-stationary it is). This function is called by `findstysols` and actually does the work of forming the lc of two time series and gauging the stationarity

Value

A single number which is the value of the test of stationarity for the combined time series. This is the result of `TOSts` but normalized for the squared coefficient norm

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols, TOSts, coeftofn

Examples

```
#
# Apply this function to random combination coefficients.
#
# The combination coefficients: comprised of two vectors each of length 3
# Note that 3 = 2^2 - 1, vectors need to be of length a power two minus 1
#
# sret, fret are two time series in the package
#
data(sret)
data(fret)
LCTS( c(rnorm(3), rnorm(3)), sret, fret)
#[1] 1.571728e-13
#
# The value of the test statistic is 1.57e-13
```

LCTSres *Plots solutions that are identified by findstysols*

Description

Plots lots of useful information concerning solutions identified using findstysols. It only plots those where the optimizer converged. Can additionally return the time-varying linear combination associated with any solution if plots are turned off.

Usage

```
LCTSres(res, tsx, tsy, inc = 0, solno = 1:nrow(res$endpar), filter.number = 1,
  family = c("DaubExPhase", "DaubLeAsymm"), plot.it = FALSE,
  spec.filter.number = 1,
  spec.family = c("DaubExPhase", "DaubLeAsymm"), plotcoef = FALSE,
  sameplot = TRUE, norm = FALSE, plotstystat = FALSE,
  plotsolinfo = TRUE, onlyacfs = FALSE,
  acfdatatrans = I, xlab = "Time", ...)
```

Arguments

res	Solution set returned by findstysols
tsx	The x time series
tsy	The y time series
inc	Adds an increment to the x-axis values.
solno	Which solution number to look at. This can be a vector of solution numbers. The default is to look at all solutions (which can be a lot, depending on how many you've got)
filter.number	The wavelet filter number to use in reconstructing the linear combination function
family	The wavelet family to use in reconstructing the linear combination function.
plot.it	Currently unused in this function
spec.filter.number	This function computes the linear combination time series and also then computes its EWS. The wavelet (spec.filter.number is the filter number of this wavelet) used to compute the EWS can be different to the one used to compute the linear combination, as the latter is only a means to an end - e.g. in principle, other basis functions could be use in the linear combination. Also the spectrum computed is only used to assess its constancy, so could be a locally stationary Fourier one.
spec.family	The family of the wavelet used to compute the spectrum
plotcoef	If TRUE then only the linear combination functions are plotted. If FALSE then a (set of potentially multiple) composite plot(s) are produced. These composite plots are what are usually most useful.
sameplot	If TRUE then the linear combination functions are plotted on the same plot.
norm	If TRUE then the linear combination functions are normalized before plotting if sameplot is TRUE. This is so as to be able to compare the patterns in each function without regard to their overall size.
plotstystat	If TRUE (and if plotcoef=FALSE) this option causes the function to plot statistics associated with the stationary solution, Z_t. The acf and partial acf are always plotted. The time series plot of Z_t and its spectrum are optionally plotted too if onlyacfs=FALSE.
plotsolinfo	If TRUE (and if plotsolinfo=FALSE) this option plots the α_t linear combination function, the β_t one (ie both of them), the stationary linear combination Z_t, and an estimate of the EWS of Z_t computed using the spec.filter.number and spec.family wavelet. The variance associated with Z_t (the minimizing variance from the optimizer in findstysols and the p-value associated with the solution are displayed as plot titles.
onlyacfs	Only plot the two acfs if plotstystat=TRUE
acfdatatrans	A function (e.g. log) to transform the series before taking and displaying the acf functions.
xlab	An x label for the time series plots, and spectral plots
...	Extra arguments for the acf plots.

Details

The function findstysols takes two time series and attempts to find time-varying linear combinations of the two that are stationary. If one is found, we call it Z_t. However, findstysols works by numerical optimization, typically from random starts, and, generally, there is no unique stationary solution.

This function takes the results obtained by findstysols in an object called res and then for a set of solutions already identifed by the user, and supplied to this function via solno, this function takes each identified solution in turn and produces a set of plots.

Determining which solutions are interesting is another problem. The COEFbothscale is a useful function which can analyze all solution sets simultaneously and, usually, arrange them into groups which are mutually similar. Then representative members from each group can be further analyzed by LCTSres.

Probably the most useful set of options is plotcoef=FALSE and to issue a par(mfrow=c(2,2)) command prior to running LCTSres. This produces the plots, four to a page, and enables interesting features to be compared from plot to plot.

The plotcoef=FALSE option causes four plots to be produced (on the same page if mfrow is set as the previous paragraph suggests). The first two are the (potentially) time-varying linear combination functions, the next is the stationary linear combination, Z_t, itself and the final plot is an estimate of the Z_t's evolutionary wavelet spectrum. The titles of the latter two plots display the process variance of Z_t (the global unconditional variance, because Z_t is assumed to be stationary) and the p-value associated the the hypothesis test of stationarity of Z_t. The spectral estimate show exhibit near constancy because of the stationarity (as assessed by hypothesis test) of Z_t.

If plotstystat=TRUE then further plots are produced of the results of various classical time series analyses of Z_t. If onlyacfs=TRUE then only the acf and partial acf of Z_t are plotted, otherwise Z_t and its classical spectrum are also plotted (remember, Z_t, has tested to be stationary and so these classical methods are valid).

If more than one solution is to be plotted, then the scan() function is employed to pause the plots between plots.

Value

The stationary solution, Z_t, associated with the last solution to be plotted is returned. Of course, if there is only one solution to be plotted then it is the only possibility. Hence, if all the plot arguments are FALSE then no plots are produced and the stationary linear combination of the (last) solution number is returned.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols

Examples

```
#
# See examples in findstysols (the plot method for the results of
# findstysols make use of LCTSres)
```

---

localvar *Compute the time-localized (unconditional) variance for a time series*

Description

Compute the time localized variance from an evolutionary wavelet spectrum of a time series

Usage

```
localvar(spec)
```

Arguments

spec	An evolutionary wavelet spectrum, such as that computed by ewspec in WaveThresh.

Details

One can compute the local variance of a time series by first computing its evolutionary wavelet spectrum, e.g., by using ewspec, and then applying localvar on the S component of that returned by ewspec.

Value

A vector representing the local variance estimate at successive times.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

ewspec

Examples

```
#
# Let's look at a iid standard normal sequence, variance should be 1, always
# for all times.
#
zsim <- rnorm(64)
#
# Note, in the following I use var as the method of deviance estimation,
# as described in the help there it can be more accurate when transformations
# are not used.
#
z.ews <- ewspec(zsim, smooth.dev=var)$S
#
# Compute the local variance
#
z.lv <- localvar(z.ews)
#
# Plot the local variance against time
#
## Not run: ts.plot(z.lv)
#
# Should be around 1. Note, the vertical scale of the plot might be
# deceptive, as R plots expand the function to the maximum available
# space. If you look again it should be quite close to 1 (e.g. on the
# example I am looking at now the variance is within +/- 0.15 of 1.
#
# However, it might not be close to 1 because the sample size is quite small,
# only 64, so repeat the above analysis with a larger sample size, e.g. 1024.
#
```

`mergexy` *Concatenate a set of solution results into one set*

Description

Merges several sets of optimization results from multiple calls to `findstysols` into a single object for further analysis

Usage

```
mergexy(...)
```

Arguments

`...`	An unspecified number of arguments of class `csFSS`. (usually a set of objects containing a set of optimization solutions, such as that returned by `findstysols`)

Details

The return object from an invocation of the findstysols is a list containing a number of interesting components containing information about the starting parameters, the (hopefully optimal) ending parameters, convergence status, minimum variance achieved and p-value associated with the final test of stationarity after an optimization.

It is possible to ask findstysols to execute multiple optimization runs in the same function, by choice of the Nsims parameter. However, for truly large runs, it can be convenient to run multiple copies of findstysols, for example on multiple processors simultaneously (a coarse grained parallelism).

In particular, for large time series, it can be useful to run findstysols for **one** optimization run (as running more than one for a very large series can cause the software to fail as R can run out of memory. Actually, for very very large series even one optmization run can fail for memory reasons).

In this way multiple optimization runs can be executed with each one producing its own set of results. This function (mergexy) takes a list of object names of all of the results, and merges the results into one object as if a single call to findstysols had been executed. Such a single set of results can then be passed on to further analysis routines, such as COEFbothscale or LCTSres.

Value

A set of optimization solutions in the same format as those returned by findstysols

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols, LCTSres, COEFbothscale

Examples

```
#
# Generate two dummy time series
#
x1 <- rnorm(32)
y1 <- rnorm(32)
#
# Run two optimizations
#
## Not run: solnset1 <- findstysols(Nsims=1, tsx=x1, tsy=y1)
## Not run: solnset2 <- findstysols(Nsims=1, tsx=x1, tsy=y1)
#
```

```
# Merge them
#
## Not run: solnset <- mergexy(solnset1, solnset2)
```

`plot.BootTOS` *Plots results of a Bootstrap Test of Stationarity*

Description

Produces Bootstrap simulation result as a histogram with a vertical line indicating the test statistic computed on the actual data.

Usage

```
## S3 method for class 'BootTOS'
plot(x, ...)
```

Arguments

`x`	The object you wish to get a plot on.
`...`	Other arguments to plot.

Details

Produces a histogram of all the bootstrap statistics and the test statistic computed on the true data. Also produces a vertical line indicating the position of the true statistic.

Value

None.

Author(s)

G.P. Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1. Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`BootTOS`

Examples

```
#
v <- rnorm(512)
## Not run: v.BootTOS <- BootTOS(v)
## Not run: plot(v.BootTOS)
```

`plot.csBiFunction` *Plot a* `csBiFunction` *object*

Description

A `csBiFunction` object contains representations of two functions. This function plots the two functions superimposed.

Usage

```
## S3 method for class 'csBiFunction'
plot(x, ...)
```

Arguments

`x`	An object of class `csBiFunction`
`...`	Other arguments to plot call

Value

None

Author(s)

G.P. Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`coeftofn`, `print.csBiFunction`, `summary.csBiFunction`

Examples

```
## Not run: plot(coeftofn(c(1,-1), c(0.5, 0.5)))
```

`plot.csFSS` *Plot a* `csFSS` *object.*

Description

Produces two types of plot from the information in a `csFSS` object, such as that returned by `findstysols`.

Usage

```
## S3 method for class 'csFSS'
plot(x, solno = NULL, ALLplotclust = TRUE, ALLplotscale = TRUE, sollabels=TRUE,
    SNinc = 0, ...)
```

Arguments

`x`	The `csFSS` object you wish to produce plots for.
`solno`	If missing then the plot produces plots that show information on all solutions at once, first in a scatter plot, then in a dendrogram. If provided then the plot produces information on that specific solution.
`ALLplotclust`	If TRUE then the dendrogram is plotted, if FALSE it is not.
`ALLplotscale`	If TRUE then the two-dimensional scaling solution is plotted. If FALSE, it is not.
`sollabels`	If TRUE then solution numbers are plotted on the scaling plot, if produced.
`SNinc`	An argument passed to the `LCTSres` function if called. When plotting add an increment on where to start looking at the time series/solutions from.
`...`	Other arguments passed to plot.

Details

This function can produce either a scatterplot, which indicates the two-dimensional scaling picture of the optimization solution sets, or a dendrogram showing putative clustering of solutions. In both cases it is a plot considering ALL solutions at once. These plots are delegated to the `plot.csFSSgr` function.

If the argument `solno` is provided then plots are produced which show information on a single solution. This plot is delegated to the `LCTSres` function.

Value

None.

Author(s)

G.P.Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols, LCTSres, plot.csFSSgr, print.csFSS, summary.csFSS

Examples

```
#
# Create dummy data
#
x1 <- rnorm(32)
y1 <- rnorm(32)
#
# Find stationary combinations
# Note: we don't run this example in installation/package formation as
# it takes a long time. However, this precise command IS run in
# the help to findstysols
#
## Not run: ans <- findstysols(Nsims=100, tsx=x1, tsy=y1)
#
# Produce dendrogram
#
## Not run: plot(ans)
#
# Produce four pictures relating to solution 3 (can also do
# par(mfrow=c(2,2)) to make a nice 4 plot on one page.)
#
## Not run: plot(ans, solno=3)
#solno is  3
#3
#1:
```

---

plot.csFSSgr *Produce plots from a* csFSSgr *object.*

Description

A csFSS object contains a set of solutions obtained from a series of optimizations. Each solution corresponds to a time-varying linear combination of two time series (or rather the wavelet coefficients of such combinations) where the combination has found to be stationary and the optimizer that got there converged. Often one wishes to interrogate the results, such as seeing how the solutions cluster, or what their low-dimensional scaling solution projection looks like, such analyses are produced by the COEFbothscale function and the whole plot is marshalled by the plot.csFSS function.

Usage

```
## S3 method for class 'csFSSgr'
plot(x, plotclust = TRUE, plotscale = TRUE, sollabels=FALSE, ...)
```

Arguments

x	The csFSSgr object to be plotted.
plotclust	If TRUE then the dendrogram clustering is plotted, if FALSE it is not.
plotscale	If TRUE then the scaling solution picture is plotted, if FALSE it is not.
sollabels	If TRUE then solution numbers are plotted on the scaling plot, if produced.
...	Other arguments to plot.

Value

None.

Author(s)

G.P. Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

plot.csFSS

Examples

```
#
# This function is a helper function for plot.csFSS so see the example there.
#
```

plot.lacv *Plot localized autocovariance (lacv) object.*

Description

Produces various ways of looking at a localized autocovariance (lacv) object.

Usage

```
## S3 method for class 'lacv'
plot(x, plotcor = TRUE, type = "line",
        lags = 0:min(as.integer(10 * log10(nrow(x$lacv))), ncol(x$lacv) - 1),
        tcex = 1, lcol = 1, llty = 1, the.time = NULL, ...)
```

Arguments

x	The localized autocovariance object you want to plot (lacv)
plotcor	If TRUE then plot autocorrelations, otherwise plot autocovariances.
type	The lacv objects are fairly complex and so there are different ways you can plot them. The types are line, persp or acf, see the details for description. Note that the line plot only works with correlations currently.
lags	The lags that you wish included in the plot. The default is all the lags from 0 up to the maximum that is used in the R acf plot
tcex	In the line plot lines are plotted that indicate the time-varying correlation. Each lag gets a different line and the lines are differentiated by the lag id being placed at intervals along the line. This argument changes the size of those ids (numbers).
lcol	Controls the colours of the lines in the line plot.
llty	Controls the line types of the lines in the line plot.
the.time	If the acf plot is chosen then you have to specify a time point about which to plot the acf. I.e. in general this funcion's lacv argument is a 2D function: $c(t, \tau)$, the acf plot produces a plot like the regular acf function and so you have to turn the 2D $c(t, \tau)$ into a 1D function $c(t_0, \tau)$ by specifying a fixed time point t_0.
...	Other arguments to plot.

Details

This function produces pictures of the two-dimensional time-varying autocovariance or autocorrelation, $c(t, \tau)$, of a locally stationary time series. There are three types of plot depending on the argument to the type argument.

The line plot draws the autocorrelations as a series of lines, one for each lag, as lines over time. E.g. a sequence #of lines $c(t, \tau_i)$ is drawn, one for each τ_i. The zeroth lag line is the autocorrelation at lag 0 which is always 1. By default all the lags are drawn which can result in a confusing picture. Often, one is only interested in the low level lags, so only these can be plotted by changing the lags

argument and any selection of lags can be plotted. The colour and line type of the plotted lines can be changed with the `lcol` and the `llty` arguments.

The `acf` plot produces pictures similar to the standard R `acf()` function plot. However, the regular acf is a 1D function, since it is defined to be constant over all time. The time-varying acf supplied to this function is not constant over all time (except for stationary processes, theoretically). So, this type of plot requires the user to specify a fixed time at which to produce the plot, and this is supplied by the `the.time` argument.

The `persp` plot plots the 2D function $c(t, \tau)$ as a perspective plot.

Value

For the `acf` type plot the acf values are returned invisibly. For the other types nothing is returned.

Author(s)

G.P. Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

lacv

Examples

```
#
# Make some dummy data, e.g. white noise
#
v <- rnorm(256)
#
# Compute the localized autocovariance (ok, the input is stationary
# but this is just an example. More interesting things could be achieved
# by putting the results of simulating from a LSW process, or piecewise
# stationary by concatenating different stationary realizations, etc.
#
vlacv <- lacv(v, lag.max=30)
#
# Now let's do some plotting of the localized autocovariance
#
## Not run: plot(vlacv, lags=0:6)
#
# Should get a plot where lag 0 is all up at value 1, and all other
# autocorrelations are near zero (since its white noise).
#
#
```

```
# How about just looking at lags 0, 2 and 4, and some different colours.
#
## Not run: plot(vlacv, lags=c(0,2,4), lcol=c(1,2,3))
#
# O.k. Let's concentrate on time t=200, let's look at a standard acf
# plot near there.
#
## Not run: plot(vlacv, type="acf", the.time=200)
#
# Now plot the autocovariance, rather than the autocorrelation.
#
## Not run: plot(vlacv, type="acf", the.time=200, plotcor=FALSE)
#
# Actually, the plot doesn't look a lot different as the series is white
# noise, but it is different if you look closely.
```

`plotBS` *Compute p-value for parametric Monte Carlo test and optionally plot test statistic values*

Description

Computes and returns a p-value for the result of a parametric Monte Carlo test. Optionally, plots a histogram of the test statistics (on the original data, and using test statistics resulting from simulations from the null hypothesis distribution).

Usage

```
plotBS(BS, alpha = 0.05, plot = TRUE, verbose = FALSE, main = "Bootstrap Histogram",
        xlab = "Test Statistic Values", ylab = "Frequency")
```

Arguments

`BS`	The results from a Monte Carlo test. This should be a vector of arbitrary length. The first value must be the value of the test statistic computed on the data. The remaining values are the test statistics computed on simulations constructed under the null hypothesis.
`alpha`	A nominal size for the test. This only effects the reporting. If the computed p-value is less than `alpha` then the function prints out that the series is not stationary.
`plot`	If `TRUE` then a histogram of all the test statistics is produced, with a vertical line showing the position of the test statistic computed on the actual data. If the vertical line is much larger than all the histogram values then this is indicative of stationarity. If the vertical line is well within the histogram values then this is indicative of no evidence against stationarity.
`verbose`	If `TRUE` then the p-value is printed and a sentence declaring "stationary" or "not stationary" is printed (relative to the nominal p-value)

main	A main label for the plot, if produced
xlab	An xlab x axis label for the plot, if produced
ylab	An ylab y axis label for the plot, if produced

Value

The p-value computed from the Monte Carlo test results is returned

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

getpvals,BootTOS

Examples

```
#
# See example in \code{\link{BootTOS}}.
#
```

---

print.csBiFunction *Print a* csBiFunction *object.*

Description

A csBiFunction object contains representations of two functions. This function prints information about the object

Usage

```
## S3 method for class 'csBiFunction'
print(x, ...)
```

Arguments

x	The object you want printed.
...	Other arguments

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`plot.csBiFunction`, `summary.csBiFunction`

Examples

```
print(coeftofn(c(1,-1), c(0.5, 0.5)))
#Class 'csBiFunction' : Contains two sampled functions:
#       ~~~~  : List with 2 components with names
#               alpha beta
#
#
#summary(.):
#----------
#Length of functions is:  256
```

`print.csFSS` *Print a* `csFSS` *object.*

Description

Print information about a `csFSS` object.

Usage

```
## S3 method for class 'csFSS'
print(x, ...)
```

Arguments

`x`	The `csFSS` object you want printed.
`...`	Other arguments.

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols, plot.csFSS, summary.csFSS

Examples

```
#
# Create dummy data
#
x1 <- rnorm(32)
y1 <- rnorm(32)
#
# Find stationary combinations. Note: normally Nsims would be much bigger
#
## Not run: ans <- findstysols(Nsims=100, tsx=x1, tsy=y1)
#
# Print this csFSS object
#
## Not run: print(ans)
#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:  x1
#Name of Y time series:  y1
#Length of input series:  32
#There are  100  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
```

```
#        97  solutions appear to be stationary
#Range of p-values: ( 0.885 , 0.975 )
#
#Wavelet filter for combinations:  1   DaubExPhase
#Wavelet filter for spectrum:  1   DaubExPhase
```

`print.csFSSgr` *Print* `csFSSgr` *object.*

Description

Prints out information on a `csFSSgr` object.

Usage

```
## S3 method for class 'csFSSgr'
print(x, ...)
```

Arguments

`x`	The `csFSSgr` object you wish to print.
`...`	Other arguments.

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`plot.csFSSgr`, `summary.csFSSgr`

Examples

```
#
# The user should normally never need to use this function as the
# csFSSgr object is only ever internally produced and used.
#
```

`print.lacv` *Print lacv class object*

Description

Prints information about lacv class object.

Usage

```
## S3 method for class 'lacv'
print(x, ...)
```

Arguments

`x`	The lacv class object you want to print
`...`	Other arguments

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`lacv`, `plot.lacv`, `summary.lacv`

Examples

```
#
# Make some dummy data, e.g. white noise
#
v <- rnorm(256)
#
# Compute the localized autocovariance (ok, the input is stationary
# but this is just an example. More interesting things could be achieved
# by putting the results of simulating from a LSW process, or piecewise
# stationary by concatenating different stationary realizations, etc.
#
```

```
vlacv <- lacv(v, lag.max=30)
#
# Now let's print the lacv object
#
print(vlacv)
#Class 'lacv' : Localized Autocovariance/correlation Object:
#       ~~~~  : List with 3 components with names
#             lacv lacr date
#
#
#summary(.):
#----------
#Name of originating time series:
#Date produced:  Thu Oct 25 12:11:29 2012
#Number of times:  256
#Number of lags:  30
```

---

prodcomb *Combine two time series using a time-varying linear combination.*

Description

This function takes the cfs vector and splits it into two halves. The first half contains the wavelet coefficients for the alpha linear combination function, and the second half for the beta one. Then the functions themselves are generated by using the coeftofn function. Then, the coefficient functions are multiplied by the respective time series (tsx by alpha and tsy by beta) and the result returned.

Usage

```
prodcomb(cfs, tsx, tsy, filter.number = 1,
 family = c("DaubExPhase", "DaubLeAsymm"), all = FALSE)
```

Arguments

cfs	Wavelet coefficients of the two combination functions. The coefficients for alpha/beta combination functions are stored in the first/last half of the vector.
tsx	The x time series to combine
tsy	The y time series to combine
filter.number	The wavelet filter to use to obtain functions from coefficients
family	The wavelet family to do the same.
all	If TRUE then a list containing the combined series in the component lcts and the combination functions in components alpha and beta. Although the combined series is the thing that is usually later tested for stationarity, it is often useful to see, at some stage, what the combination functions are, as these provide interpretation as to what the combination might mean. If FALSE then just the combined series is returned.

Details

This function is called by findstysols and makes use of coeftofn to turn coefficients into a function used in the combination.

Value

If all=TRUE then a list with the following components:

lcts	The combined series, $\alpha_t X_t + \beta_t Y_t$
alpha	The α_t combination function.
beta	The β_t combination function.

If all=FALSE then only lcts is returned.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols, coeftofn

Examples

```
#
# Toy example
#
tmp.a <- c(1, -1)
tmp.b <- c(0.5, 0.5)
#
# Generate toy time series
#
xxx <- rnorm(256)
yyy <- rnorm(256)
#
# Combine xxx and yyy using the functions produced by inverse wavelet
# transform of tmp.a and tmp.b
#
## Not run: tmp <- prodcomb(c(tmp.a, tmp.b), tsx=xxx, tsy=yyy)
#
# E.g. plot combination
#
## Not run: ts.plot(tmp)
```

```
#
# Potentially test its stationarity.... etc
#
```

SP500FTSElr *Log-returns time series of the SP500 and FTSE100 indices*

Description

Log-returns of the SP500 and FTSE indices between 21th June 1995 until 2nd October 2002. Only trading days where both indices were recorded are stored. There are 2048 observations.

Usage

```
data(SP500FTSElr)
```

Format

A data frame with 2048 observations on the following 3 variables.

`Date` The trading day that the index was recorded.

`SP500lr` The log-return for SP500

`FTSElr` The log-return for FTSE100

Source

Downloaded from Yahoo! Finance

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

Examples

```
#
# Plot the log-returns for the SP500
#
## Not run: ts.plot(SP500FTSElr[,2])
```

sret *Particular section of SP500 log-returns series.*

Description

Observations 256:767 from the SP500 log-return series stored in SP500FTSElr dataset.

Usage

```
data(sret)
```

Format

A vector of 512 observations of the SP500 log-returns series.

Details

Its just more convenient to refer to sret than to SP500FTSElr[256:767,2].

Source

Yahoo! Finance

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

Examples

```
## Not run: ts.plot(sret)
```

summary.csBiFunction *Summarize a* csBiFunction *object.*

Description

Summarize a csBiFunction object.

Usage

```
## S3 method for class 'csBiFunction'
summary(object, ...)
```

Arguments

object	The object to summarize
...	Other arguments

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

plot.csBiFunction, print.csBiFunction

Examples

```
#
# See example to print.csBiFunction, as this calls summary(.)
#
```

---

summary.csFSS *Summarize a* csFSS *object.*

Description

Summarizes a csFSS object.

Usage

```
## S3 method for class 'csFSS'
summary(object, size = 0.05, ...)
```

Arguments

object	Object you wish to summarize.
size	A hypothesis test size. The csFSS object contains a number of p-values, this argument controls what is considered significant (but not corrected for multiple tests)
...	Other arguments

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

findstysols,plot.csFSS, print.csFSS

Examples

```
#
# See example to print.csFSS which calls summary(.)
#
```

---

summary.csFSSgr *Summarize a* csFSSgr *object.*

Description

Summarizes a csFSSgr object.

Usage

```
## S3 method for class 'csFSSgr'
summary(object, ...)
```

Arguments

object	The csFSSgr object you wish to summarize.
...	Other arguments

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

`plot.csFSSgr`, `print.csFSSgr`

Examples

```
#
# See example for print.csFSSgr which calls summary(.)
```

`summary.lacv` *Summarizes a lacv object*

Description

Summarizes a lacv object

Usage

```
## S3 method for class 'lacv'
summary(object, ...)
```

Arguments

`object`	The lacv object you wish summarized.
`...`	Other arguments

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

lacv, plot.lacv, print.lacv

Examples

```
#
# Make some dummy data, e.g. white noise
#
v <- rnorm(256)
#
# Compute the localized autocovariance (ok, the input is stationary
# but this is just an example. More interesting things could be achieved
# by putting the results of simulating from a LSW process, or piecewise
# stationary by concatenating different stationary realizations, etc.
#
vlacv <- lacv(v, lag.max=20)
#
# Now let's summarize the lacv object
#
summary(vlacv)
#Name of originating time series:
#Date produced:  Thu Oct 25 12:11:29 2012
#Number of times:  256
#Number of lags:  20
```

TOSts *A test statistic for stationarity*

Description

The T_{VS} test statistic from the Cardinali and Nason article. Measures the degree of non-stationarity using the estimated evolutionary wavelet spectrum (EWS)

Usage

```
TOSts(spec)
```

Arguments

spec	An EWS estimate, e.g. from the $S component from ewspec

Details

Given an EWS estimate. This computes the sample variance of the estimate for each scale level and then returns the sum of these variances.

Value

A single number which is the sum of the sample variances of each scale level from an EWS estimate. If the EWS estimate is constant for each scale then the return value is zero.

Author(s)

Guy Nason

References

Cardinali, A. and Nason, Guy P. (2013) Costationarity of Locally Stationary Time Series Using costat. *Journal of Statistical Software*, **55**, Issue 1.

Cardinali, A. and Nason, G.P. (2010) Costationarity of locally stationary time series. *J. Time Series Econometrics*, **2**, Issue 2, Article 1.

See Also

BootTOS

Examples

```
#
# Compute a spectral estimate on an sample time series (just use iid data)
#
xsim <- rnorm(128)
xews <- ewspec(xsim, smooth.dev=var)$S
#
# You could plot this spectral estimate if you liked
#
## Not run: plot(xews)
#
# Compute test statistic
#
TOSts(xews)
#[1] 0.1199351
#
# Although the time series x here is a realization from a stationary process
# the test statistic is not zero (this is because of the estimation error
# inherent in this small sample).
#
# This is why the bootstrap test, \code{\link{BootTOS}} is required to
# assess the significance of the test statistic value.
```

Index