Package 'locits'

July 22, 2025

Type Package

Title Test of Stationarity and Localized Autocovariance

Version 1.7.7

Date 2023-09-04

Depends R (>= 3.3), wavethresh, igraph

Description Provides test of second-order stationarity for time series (for dyadic and arbitrary-n length data). Provides localized autocovariance, with confidence intervals, for locally stationary (nonstationary) time series. See Nason, G P (2013) ``A test for second-order stationarity and approximate confidence intervals for localized autocovariance for locally stationary time series." Journal of the Royal Statistical Society, Series B, 75, 879-904. <doi:10.1111/rssb.12015>.

License GPL (>= 2)

NeedsCompilation yes

Author Guy Nason [aut, cre]

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

Repository CRAN

Date/Publication 2023-09-05 15:10:02 UTC

Contents

`locits-package` *New test of second-order stationarity and confidence intervals for localized autocovariance.*

Description

Provides functionality to perform a new test of second-order stationarity for time series. The method works by computing a wavelet periodogram and then examining its Haar wavelet coefficients for significant ones. The other main feature of the software is to compute the localized autocovariance and pointwise confidence intervals.

Details

For the test of stationarity there are two main functions. The original is the `hwtos2` function and this returns a `tos` object. The `hwtos2` function works on data sets whose length is a power of two. Version 1.5 introduced a new function, `hwtos` which carries out the test on arbitrary length data. The `summary.tos` function performs a Bonferroni and FDR statistical analysis to detect which Haar wavelet coefficients are significant. The function `plot.tos` provides a plot of the original time series with any non-stationarities clearly indicated on the plot (actually locations and scales of the Haar wavelet coefficients).

For the localized autocovariance the main function is `Rvarlacf`. This computes the localized autocovariance values and approximate pointwise condifence intervals. The function `plot.lacfCI` can then plot the localized autocovariance and its confidence intervals in a number of forms.

Author(s)

Guy Nason

Maintainer: Guy Nason

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`Rvarlacf`, `hwtos2`

Examples

```
#
# Here's a simple simulated example.
#
# A series which is a concatenation of two iid Gaussian series with
# different variances.
#
x <- c(rnorm(256, sd=1), rnorm(256, sd=2))
#
# Let's do a test of stationarity
#
st.test <- hwtos2(x)
#8    7    6    5    4    3
#
# Ok, that's the computation gone, let's look at the results.
#
st.test

#Class 'tos' : Stationarity Object :
#~~~~  : List with 9 components with names
# nreject rejpval spvals sTS AllTS AllPVal alpha x xSD
#
```

```
#
#summary(.):
#----------
#There are  186  hypothesis tests altogether
#There were  4  FDR rejects
#The rejection p-value was  0.0001376564
#Using Bonferroni rejection p-value is  0.0002688172
#And there would be  4  rejections.
#Listing FDR rejects... (thanks Y&Y!)
#P: 5 HWTlev:  0  indices on next line...[1] 1
#P: 6 HWTlev:  0  indices on next line...[1] 1
#P: 7 HWTlev:  0  indices on next line...[1] 1
#P: 8 HWTlev:  0  indices on next line...[1] 1
#
# In the lines above if there are any rejects then the series is
# deemed to be nonstationary, and note that there were 4 in both
# the lines above (sometimes FDR rejects a few more).
#
# You can also plot the object and it shows you where it thinks the
# nonstationarities are
#
## Not run: plot(st.test)
#
# See the help page for the hwtos2 function, where there is an example
# with a stationary series.
#
# For the localized autocovariance...
#
# Let's use the function tvar1sim which generates a time-varying AR model
# with AR(1) paramter varying over the extent of the series from 0.9
# to -0.9 (that is, near the start of the series it behaves like an
# AR(1) with parameter 0.9, and near the end like an AR(1) with parameter
# -0.9, and in between the parameter is somewhere between 0.9 and -0.9
# figured linearly between the two.
#
x <- tvar1sim()
#
# Plot it, so you know what the series looks like, should always do this.
#
## Not run: ts.plot(x)
#
# Now, let's compute the localized autocovariance and also confidence intervals
# For the variance, let's look at the first 20 lags
#
# Do it at t=50 and t=450, ie what is the localized autocovariance at these
# two times.
#
x.lacf.50 <- Rvarlacf(x=x, nz=50, var.lag.max=20)
x.lacf.450 <- Rvarlacf(x=x, nz=450, var.lag.max=20)
#
# Now plot the answers, you may want to do this on two different plots
# so that you can compare the answers
#
```

```
#
## Not run: plot(x.lacf.50, plotcor=FALSE, type="acf")
## Not run: plot(x.lacf.450, plotcor=FALSE, type="acf")
#
# Note that the plotcor argument is set so covariances and not correlations
# are plotted. Also, the type is set to "acf" to make the plot *look* like
# the regular acf plot. But DON'T be fooled, it is not the regular acf
# that is plotted, but a time localized plot. The two plots should look
# very different, both like AR(1) but with different parameters (from the
# same time series).
#
# You could also plot the regular acf and see how it gets it wrong!
#
```

`AutoBestBW` *Choose a good bandwidth for running mean smoothing of a EWS spectral estimator.*

Description

Computes running mean estimator closest to wavelet estimator of evolutionary wavelet spectrum. The idea is to obtain a good linear bandwidth.

Usage

```
AutoBestBW(x, filter.number = 1, family = "DaubExPhase",
    smooth.dev = var, AutoReflect = TRUE, tol = 0.01, maxits = 200,
    plot.it = FALSE, verbose = 0, ReturnAll = FALSE)
```

Arguments

`x`	Time series you want to analyze.
`filter.number`	The wavelet filter used to carry out smoothing operations.
`family`	The wavelet family used to carry out smoothing operations.
`smooth.dev`	The deviance estimate used for the smoothing (see ewspec help)
`AutoReflect`	Mitigate periodic boundary conditions of wavelet transforms by reflecting time series about RHS end before taking transforms (and is undone before returning the answer).
`tol`	Tolerance for golden section search for the best bandwidth
`maxits`	Maximum number of iterations for the golden section search
`plot.it`	Plot the values of the bandwidth and its closeness of the linear smooth to the wavelet smooth, if TRUE.
`verbose`	If nonzero prints out informative messages about the progress of the golden section search. Higher integers produce more messages.
`ReturnAll`	If TRUE then return the best bandwidth (in the ans component), the wavelet smooth (in EWS.wavelet) and the closest linear smooth (EWS.linear). If FALSE then just the bandwidth is returned.

Details

Tries to find the best running mean fit to an estimated spectrum obtained via wavelet shrinkage. The goal is to try and find a reasonable linear bandwidth.

Value

If ReturnAll argument is FALSE then the best bandwidth is returned.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

Rvarlacf

Examples

```
#
# Generate synthetic data
#
x <- rnorm(256)
#
# Compute best linear bandwidth
#
tmp <- AutoBestBW(x=x)
#
# Printing it out in my example gives:
# tmp
# [1] 168
```

---

covI *Compute the covariance between two wavelet periodogram ordinates at the same scale, but different time locations.*

Description

Computes $cov(I_{\ell,m}, I_{\ell,n})$ using the formula given in Nason (2012) in Theorem 1. Note: one usually should use the covIwrap function for efficiency.

Usage

```
covI(II, m, n, ll, ThePsiJ)
```

Arguments

II	Actually the *spectral* estimate S, not the periodogram values. This is for an assumed stationary series, so this is just a vector of length J, one for each scale of S.
m	Time location m
n	Time location n
ll	Scale of the raw wavelet periodogram
ThePsiJ	Autocorrelation wavelet corresponding to the wavelet that computed the raw peridogram (also assumed to underlie the time series

Value

The covariance is returned.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

covIwrap

Examples

```
P1 <- PsiJ(-5, filter.number=1, family="DaubExPhase")
#
# Compute the covariance
#
covI(II=c(1/2, 1/4, 1/8, 1/16, 1/32), m=1, n=3, ll=5, ThePsiJ=P1)
#
# [1] 0.8430809
```

---

covIwrap *A wrapper for the covI function.*

Description

Computation of the covI function is intensive. This function permits values of covI to be stored in an object, and then if these values are requested again the values can be obtained from a store rather than being computed from scratch.

Usage

```
covIwrap(S, m, n, ll, storewrap, P)
```

Arguments

S	Same argument as for covI, a spectral estimate (for a stationary series).
m	The same argument as in covI.
n	The same argument as in covI.
ll	The same argument as in covI.
storewrap	A list. On first call to this function the user should supply storewrap=NULL. This causes the function to initialize the storage. On every return from this function the storewrap component should be extracted from the list and then this storewrap component should be resupplied to any future calls to this function. In this way the function has access to previously computed values.
P	Same argument as in covI. An autocorrelation wavelet computed using the PsiJ function in wavethresh.

Details

Note: covIwrap could be removed from the function tree altogether. I.e. varip2 could call covI directly. However, covIwrap considerably improves the efficiency of the algorithm as it stores intermediate calculations that can be reused rather than being computed repeatedly.

Value

A list containing the following components:

ans	The appropriate covariance
storewrap	A list containing information about all previously computed covariances. This list should be supplied as the storewrap argument to any future calls of this function, so if the same covariance is requested it can be returned from storewrap and not computed again.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

varip2, covI

Examples

```
P1 <- PsiJ(-5, filter.number=1, family="DaubExPhase")
#
# First call to covIwrap
#
ans <- covIwrap(S=c(1/2, 1/4, 1/8, 1/16, 1/32), m=1, n=3, ll=5,
    storewrap=NULL, P=P1)
#
# Make sure you keep the storewrap component.
#
my.storewrap <- ans$storewrap
#
# What is the answer?
#
ans$ans
#[1] 0.8430809
#
# Issue next call to covIwrap: but storewrap argument is now the one we stored.
#
ans <- covIwrap(S=c(1/2, 1/4, 1/8, 1/16, 1/32), m=1, n=3, ll=5,
    storewrap=my.storewrap, P=P1)
#
# This call will reuse the stored value. However, if you change any of the
# arguments then the store won't be used.
```

Cvarip2 *Computes variance of Haar wavelet coefficients of wavelet periodogram using C code.*

Description

Performs precisely the same role as varip2 except it is implemented internally using C code and hence is much faster.

Usage

```
Cvarip2(i, p, ll, S, Pmat, PsiJL)
```

Arguments

i	Scale parameter of Haar wavelet analyzing periodogram. Scale 1 is the finest scale.
p	Location parameter of Haar wavelet analyzing periodogram
ll	Scale of the raw wavelet periodogram being analyzed.
S	Estimate of the spectrum, under the assumption of stationarity. So, this is just a vector of (possibly) J scales (which is often the usual spectral estimate averaged over time). Note: that the main calling function, hwtos2, actually passes maxD levels.

Pmat	Matrix version of autocorrelation wavelet computed using the PsiJmat function in wavethresh
PsiJL	True length of the autocorrelation wavelets in the Pmat matrix. This can be obtained simply by using the list version of the ac wavelet (computed by PsiJ) and applying sapply.

Value

The list returned from the .C calling function. The only object of real interest is the ans component which contains the variance.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwtos2, varip2

Examples

```
#
# See example from varip2
#
#
my.Pmat <- PsiJmat(-5, filter.number=1, family="DaubExPhase")
my.PsiJ <- PsiJ(-5, filter.number=1, family="DaubExPhase")
my.PsiJL <- sapply(my.PsiJ, "length")
Cvarip2(i=1, p=10, ll=2, S=c(1/2,1/4,1/8,1/16,1/32),
    Pmat=my.Pmat, PsiJL=my.PsiJL)
#
# Gives answer 1.865244, which is the same as given in the example for varip2
```

EstBetaCov *Compute estimate of wavelet periodogram and the estimate's covariance matrix.*

Description

An estimate of the wavelet periodogram at a location nz is generated. This is obtained by first computing the empirical raw wavelet periodogram by squaring the results of the nondecimated wavelet transform of the time series. Then a running mean smooth is applied.

Usage

```
EstBetaCov(x, nz, filter.number = 1, family = "DaubExPhase", smooth.dev = var,
    AutoReflect = TRUE, WPsmooth.type = "RM", binwidth = 0, mkcoefOBJ,
    ThePsiJ, Cverbose = 0, verbose = 0, OPLENGTH = 10^5, ABB.tol = 0.1,
    ABB.plot.it = FALSE, ABB.verbose = 0, ABB.maxits = 10, do.init = TRUE,
    truedenom=FALSE, ...)
```

Arguments

x	The time series for which you wish to have the estimate for.
nz	The time point at which you want the estimate computed at. This is an integer ranging from one up to the length of the time series.
filter.number	The analysis wavelet (the wavelet periodogram is computed using this to form the nondecimated wavelet coefficients)
family	The family of the analysis wavelet.
smooth.dev	The deviance function used in smoothing via the internal call to the ewspec3 function.
AutoReflect	Whether better smoothing is to be obtained by AutoReflection to mitigate the effects of using periodic transforms on non-periodic data. See ewspec3
WPsmooth.type	The type of wavelet periodogram smoothing. For here leave the option at "RM" otherwise unpredictable results can occur
binwidth	The running mean length. If zero then a good bandwidth will be chosen using the AutoBestBW function.
mkcoefOBJ	If this argument is missing then it is computed internally using the mkcoef function which computes discrete wavelets. If this function is going to be repeatedly called then it is more efficient to supply this function with a precomputed version.
ThePsiJ	As for mkcoefOBJ argument but for the autocorrelation wavelet and the function PsiJ.
Cverbose	This function called the C routine CstarIcov if you set Cverbose to true then the routine instructs the C code to produce debugging messages.
verbose	If TRUE then debugging messages from the R code are produced.
OPLENGTH	Subsidiary parameters for potential call to PsiJ function
ABB.tol	Tolerance to be passed to AutoBestBW function.
ABB.plot.it	Argument to be passed to AutoBestBW plot.it argument.
ABB.verbose	Argument to be passed to AutoBestBW verbose argument.
ABB.maxits	Argument to be passed to AutoBestBW maxits argument.
do.init	Initialize stored statistics, for cache hit rate info.
truedenom	If TRUE use the actual number of terms in the sum as the denominator in the formula for the calculation of the covariance of the smoothed periodogram. If FALSE use the (2s+1)
...	Other arguments that are passed to the ewspec3 function.

Details

First optionally computes a good bandwidth using the AutoBestBW function. Then computes raw wavelet periodogram using ewspec3 using running mean smoothing with the binwidth bandwith (which might be automatically chosen). This computes the estimate of the wavelet periodogram at time nz. The covariance matrix of this estimate is then computed in C using the CstarIcov function and this is returned.

Value

A list with two components:

betahat	A vector of length J (the number of scales in the wavelet periodogram, which is $\log_2$ of the number of observations T
Sigma	A matrix of dimensions $J \times J$ which is the covariance of $\hat{\beta}_j$ with $\hat{\beta}_\ell$.

Author(s)

Guy Nason

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

AutoBestBW, ewspec3

Examples

```
#
# Small example, not too computationally demanding on white noise
#
myb <- EstBetaCov(rnorm(64), nz=32)
#
# Let's see the results (of my run)
#
## Not run: myb$betahat
#[1] 0.8323344 0.7926963 0.7272328 1.3459313 2.1873395 0.8364632
#
# For white noise, these values should be 1 (they're estimates)
## Not run: myb$Sigma
#            [,1]        [,2]        [,3]       [,4]        [,5]        [,6]
#[1,] 0.039355673 0.022886994 0.008980497 0.01146325 0.003211176 0.001064377
#[2,] 0.022886994 0.054363333 0.035228164 0.06519112 0.017146883 0.006079162
#[3,] 0.008980497 0.035228164 0.161340373 0.38326812 0.111068916 0.040068318
#[4,] 0.011463247 0.065191118 0.383268115 1.31229598 0.632725858 0.228574601
#[5,] 0.003211176 0.017146883 0.111068916 0.63272586 1.587765187 0.919247252
#[6,] 0.001064377 0.006079162 0.040068318 0.22857460 0.919247252 2.767615374
#
```

```
# Here's an example for T (length of series) bigger, T=1024
#
## Not run: myb <- EstBetaCov(rnorm(1024), nz=512)
#
# Let's look at results
#
## Not run: myb$betahat
# [1] 1.0276157 1.0626069 0.9138419 1.1275545 1.4161028 0.9147333 1.1935089
# [8] 0.6598547 1.1355896 2.3374615
#
# These values (especially for finer scales) are closer to 1
#
```

---

ewspec3 *Compute evolutionary wavelet spectrum of a time series.*

Description

This function is a development of the ewspec function from wavethresh but with more features. The two new features are: the addition of running mean smoothing and autoreflection which mitigates the problems caused in ewspec which performed periodic transforms on data (time series) which were generally not periodic.

Usage

```
ewspec3(x, filter.number = 10, family = "DaubLeAsymm",
    UseLocalSpec = TRUE, DoSWT = TRUE, WPsmooth = TRUE,
    WPsmooth.type = "RM", binwidth = 5, verbose = FALSE,
    smooth.filter.number = 10, smooth.family = "DaubLeAsymm",
    smooth.levels = 3:WPwst$nlevels - 1, smooth.dev = madmad,
    smooth.policy = "LSuniversal", smooth.value = 0,
    smooth.by.level = FALSE, smooth.type = "soft",
    smooth.verbose = FALSE, smooth.cvtol = 0.01,
    smooth.cvnorm = l2norm, smooth.transform = I,
    smooth.inverse = I, AutoReflect = TRUE)
```

Arguments

x	The time series you want to compute the evolutionary wavelet spectrum for.
filter.number	Wavelet filter number underlying the analysis of the spectrum (see filter.select or wd for more details).
family	Wavelet family. Again, see filter.select or wd for more details.
UseLocalSpec	As ewspec, should usually leave as is.
DoSWT	As ewspec, should usually leave as is.
WPsmooth	If TRUE then smoothing is applied to the wavelet periodogram (and hence spectrum).

`WPsmooth.type` The type of periodogram smoothing. If this argument is `"RM"` then running mean linear smoothing is used. Otherwise, wavelet shrinkage as in `ewspec` is used.

`binwidth` If the periodogram smoothing is `"RM"` then the this argument supplies the `binwidth` or number of consecutive observations used in the running mean smooth.

`verbose` If `TRUE` then messages are produced. If `FALSE` then they are not.

`smooth.filter.number`
If wavelet smoothing of the wavelet periodogram is used then this specifies the index number of wavelet to use, exactly as `ewspec`.

`smooth.family` If wavelet smoothing of the wavelet periodogram is used then this specifies the family of wavelet to use, exactly as `ewspec`.

`smooth.levels` If wavelet smoothing of the wavelet periodogram is used then this specifies the levels to smooth, exactly as `ewspec`.

`smooth.dev` If wavelet smoothing of the wavelet periodogram is used then this specifies deviance used to compute smoothing thresholds, exactly as `ewspec`.

`smooth.policy` If wavelet smoothing of the wavelet periodogram is used then this specifies the policy of wavelet shrinkage to use, exactly as `ewspec`.

`smooth.value` If wavelet smoothing of the wavelet periodogram is used then this specifies the value of the smoothing parameter for some policies, exactly as `ewspec`.

`smooth.by.level`
If wavelet smoothing of the wavelet periodogram is used then this specifies whether level-by-level thresholding is applied, or one threshold is applied to all levels, exactly as `ewspec`.

`smooth.type` If wavelet smoothing of the wavelet periodogram is used then this specifies the type of thresholding, "hard" or "soft", exactly as `ewspec`.

`smooth.verbose` If wavelet smoothing of the wavelet periodogram is used then this specifies whether or not verbose messages are produced during the smoothing, exactly as `ewspec`.

`smooth.cvtol` If wavelet smoothing of the wavelet periodogram is used then this specifies a tolerance for the cross-validation algorithm if it is specified in the `smooth.policy`, exactly as `ewspec`.

`smooth.cvnorm` Ditto to the previous argument, but this one supplies the norm used by the cross-validation.

`smooth.transform`
If wavelet smoothing of the wavelet periodogram is used then this specifies whether a transform is used to transform the periodogram before smoothing, exactly as `ewspec`.

`smooth.inverse` Should be the mathematical inverse of the `smooth.transform` argument.

`AutoReflect` Whether the series is internally reflected before application of the wavelet transforms. So, `x` becomes `c(x, rev(x))` which is a periodic sequence. After estimation of the spectrum the second-half of the spectral estimate is junked (because it is a reflection of the first half). However, the estimate is better. This argument improves over `ewspec` where poor estimates near boundaries were obtained because the transforms assume periodicity but most time series are not (and X_1 and X_T are very different, etc).

Value

Precisely the same kind of output as ewspec.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

AutoBestBW, lacf

Examples

```
#
# Generate time series
#
x <- tvar1sim()
#
# Compute its evolutionary wavelet spectrum, with linear running mean smooth
#
x.ewspec3 <- ewspec3(x)
#
# Plot the answer, probably its a bit variable, because the default bandwidth
# is 5, which is probably inappropriate for many series
#
## Not run: plot(x.ewspec3$S)
#
# Try a larger bandwidth
#
x.ewspec3 <- ewspec3(x, binwidth=100)
#
# Plot the answer, should look a lot smoother
#
# Note, a lot of high frequency power on the right hand side of the plot,
# which is expected as process looks like AR(1) with param of -0.9
#
## Not run: plot(x.ewspec3$S)
#
# Do smoothing like ewspec (but additionally AutoReflect)
#
x.ewspec3 <- ewspec3(x, WPsmooth.type="wavelet")
#
# Plot the results
#
## Not run: plot(x.ewspec3$S)
#
```

```
# Another possibility is to use AutoBestBW which tries to find the best
# linear smooth closest to a wavelet smooth. This makes use of ewspec3
#
```

---

ewspecHaarNonPer *Compute evolutionary wavelet spectrum (EWS) estimate based on the Haar wavelet transform.*

Description

This function uses the special HwdS function to compute the Haar wavelet transform with out boundary conditions (neither periodic, interval, mirror reflection). This is so all coefficients are genuine Haar coefficients without involving extra/repeated data.

Usage

```
ewspecHaarNonPer(x, filter.number = 1, family = "DaubExPhase",
    UseLocalSpec = TRUE, DoSWT = TRUE, WPsmooth = TRUE,
    verbose = FALSE, smooth.filter.number = 10,
    smooth.family = "DaubLeAsymm",
    smooth.levels = 3:WPwst$nlevels - 1, smooth.dev = madmad,
    smooth.policy = "LSuniversal", smooth.value = 0,
    smooth.by.level = FALSE, smooth.type = "soft",
    smooth.verbose = FALSE, smooth.cvtol = 0.01,
    smooth.cvnorm = l2norm, smooth.transform = I,
    smooth.inverse = I)
```

Arguments

x	A vector of dyadic length that contains the time series you want to form the EWS of.
filter.number	Should always be 1 (for Haar)
family	Should always be "DaubExPhase", for Haar.
UseLocalSpec	Should always be TRUE.
DoSWT	Should always be TRUE
WPsmooth	Should alway be TRUE to do smoothing. If FALSE then not smoothed.
verbose	If TRUE informative messages are printed during the progress of the algorithm.
smooth.filter.number	Wavelet filter number for doing the wavelet smoothing of the EWS estimate.
smooth.family	Wavelet family for doing the wavelet smoothing of the EWS estimate.
smooth.levels	Which levels of the EWS estimate to apply smoothing to.
smooth.dev	What kind of deviance to use. The default is madmad, an alternative might be var.
smooth.policy	What kind of smoothing to use. See help page for ewspec

`smooth.value` If a manual value has to be supplied according to the `smooth.policy` then this is it.

`smooth.by.level` If `TRUE` then all levels are smoothed independently with different smoothing, otherwise all levels are smoothed together (eg one threshold for all levels).

`smooth.type` The type of wavelet smoothing "hard" or "soft"

`smooth.verbose` If `TRUE` then informative messages about the smoothing are printed.

`smooth.cvtol` If cross-validation smoothing is used, this is the tolerance

`smooth.cvnorm` If cross-validation smoothing used, this is the norm that's used

`smooth.transform` A transform is applied before smoothing

`smooth.inverse` The inverse transform is applied after smoothing

Details

This function is very similar to `ewspec` from wavethresh, and many arguments here perform the same function as there.

Value

The same value as for the `ewspec` function.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`hwtos2`, `HwdS`

Examples

```
#
# Requires wavethresh, so not run directly in installation of package
#
ewspecHaarNonPer(rnorm(512))
```

getridofendNA *Replaces all NAs in vector by 0*

Description

Replaces all NAs in vector by 0

Usage

```
getridofendNA(x)
```

Arguments

x	Vector that might contain NAs

Details

Originally, this function did something more complex, but now it merely replaces NAs by 0

Value

The same vector as x but with NAs replaced by 0

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

HwdS

Examples

```
#
#
#
x <- c(3, 4, 6, NA, 3)
getridofendNA(x)
#[1] 3 4 6 0 3
```

HwdS *Compute the non-decimated Haar wavelet transform without using periodic boundary conditions.*

Description

Function uses the `filter` function to achieve its aims.

Usage

```
HwdS(x)
```

Arguments

x	A vector of dyadic length that you wish to transform.

Details

The regular `wd` function that can compute the non-decimated transform uses different kinds of boundary conditions, which can result in coefficients being used multiply for consideration in a test of stationarity, and distort results. This function only computes Haar coefficients on the data it can, without wraparound.

Value

An object of class `wd` which contains the nondecimated Haar transform of the input series, `x` without periodic boundary conditions (nor interval, nor reflection).

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`ewspecHaarNonPer`, `getridofendNA`

Examples

```
#
# Apply Haar transform to Gaussian data
#
HwdS(rnorm(32))
#Class 'wd' : Discrete Wavelet Transform Object:
#       ~~  : List with 8 components with names
#               C D nlevels fl.dbase filter type bc date
#
#$C and $D are LONG coefficient vectors
#
#Created on : Tue Jul 17 15:14:59 2012
#Type of decomposition:  station
#
#summary(.):
#----------
#Levels:  5
#Length of original:  32
#Filter was:  Haar wavelet
#Boundary handling:  periodic
#Transform type:  station
#Date:  Tue Jul 17 15:14:59 2012
```

---

hwt *Compute a Haar wavelet transform for data of arbitrary n length*

Description

Function computes Haar wavelet and scaling function coefficients for data set of any length. Algorithm computes every possible coefficient that it can for both decimated and nondecimated versions of the transform.

Usage

```
hwt(x, type = c("wavelet", "station"), reindex = FALSE)
```

Arguments

x	A vector of length n, where n is a positive integer. This is the data that you wish to compute the Haar wavelet transform for.
type	The type of transform, either the decimated or nondecimated algorithm.
reindex	If TRUE then the routine attempts to match scales with the usual dyadic transform, wd. If FALSE then the coefficients that are returned are "as is"

Details

Essentially, this algorithm attempts to compute every possible Haar wavelet coefficient. For example, if the length of the input series was 6 then this means that three coefficients at the finest scale can be computed using the first, second and third pair of input data points using the weights `c(1, -1)/sqrt(2)`. However, from the three coefficients that result from this, there is only one pair, so only one "next coarser" coefficient can be computed.

The `reindex` option is subtle. Essentially, it tries to ensure that the returned coefficients end up at the same scales as if a data set of the next highest dyadic length was analyzed by the wd function. E.g. if the length of the series was 10 then with `reindex=FALSE` (default) only three levels are returned for each of the wavelet and scaling coefficients. If `reindex=TRUE` then the number of levels returned would be as if `wd` analysed a data set of length 16 (the smallest dyadic number larger than 10). The wd levels would be zero to three and this is what would be returned in this function if `reindex=TRUE`. However, note, in this case, the coarsest level coefficient happens to be NULL (or not computable). One can view the algorithm as computing a partial transform of 10 of the 16 elements and substituting NA for anything it can't compute.

Value

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

`c`	The scaling function coefficients. This is a list of length `nlevels` which contains the scaling function coefficients. The coarsest scale coefficients are to be found in the lowest-indexed slots of the list (e.g. c[[1]]) and increasing slot index corresponds to finer scales. So, `c[[length(c)]]` corresponds to the finest coefficients. Note, an entry in the slot can also be NULL. This indicates that no coefficients could be calculated at this scale, usually the coarsest.
`d`	A for `c` but for wavelet coefficients.
`nlevels`	The number of scale levels in the Haar wavelet decomposition. if `reindex=TRUE` then this number will be the log to base 2 of the smallest power of two larger than the length of the input vector `x`.
`type`	Whether a decimated wavelet transform has been computed (`"wavelet"`) or a nondecimated transform (`"station"`). Note, the name of the argument `"station"` has been chosen to coincide with the `type` in the regular wavelet transform computed by `wd`.
`reindex`	Either `TRUE` or `FALSE`. If `TRUE` then the scale levels correspond directly to those computed by `wd`, the regular wavelet transform. If `FALSE` then the levels returned in `c` and `d` are just indexed from the first non-null level.

Author(s)

G. P. Nason

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

Priestley, M.B. and Subba Rao (1969) A test for non-stationarity of time series. *J. R. Statist. Soc. B*, **31**, 140-149.

von Sachs, R. and Neumann, M.H. (2000) A wavelet-based test for stationarity. *J. Time Ser. Anal.*, **21**, 597-613.

See Also

hwtos, plot.hwtANYN, print.hwtANYN, summary.hwtANYN

Examples

```
#
# Generate test data set of length 5 (note, NOT a power of two)
#
v2 <- rnorm(5)
#
# Compute its Haar transform
#
v2hwt <- hwt(v2)
#
# How many levels does it have?
#
nlevelsWT(v2hwt)
#
# What are the coarsest scale wavelet coefficients
#
v2hwt$d[[1]]
#
# What are the finest scale scaling function coefficients
#
v2hwt$c[[nlevels(v2hwt$c)-1]]
```

---

hwtos *Haar wavelet test for (second-order) stationarity for arbitrary length time series.*

Description

NOTE: CURRENTLY THIS FUNCTION IS NOT INCLUDED IN THE PACKAGE. USE hwtos2. This function computes the raw wavelet periodogram of the arbitrary time series vector x. The periodogram is then subject to a hypothesis test to see if its expectation over time, for different scales, is constant. The constancy test is carried out using tests on its Haar wavelet coefficients. The overall test is for second-order stationarity (e.g. constant variance, constant acf function, mean is assumed zero).

Usage

```
hwtos(x, alpha = 0.05, lowlev = 1, WTscale = NULL, maxSD = NULL,
    verbose = FALSE, silent = FALSE, UseCForVarip2 = TRUE, OPLENGTH = 1e+05,
    mc.method = p.adjust.methods)
```

Arguments

x	The time series you wish to test for second-order stationarity. Minimum length series that this function will operate for is 20. However, for short series the power of the test might not be good and could be investigated via simulation that reflect your particular circumstances. This should be a stochastic series. The function will report an error if x is a constant function. The function might not work properly if it contains a significant trend or patches of non-stochastic observations.
alpha	The (nominal) size of the hypothesis test.
lowlev	Controls the lowest scale of the wavelet periodogram that gets analyzed. Generally, leave this parameter alone.
WTscale	Controls the finest scale of the Haar wavelet transform of a particular wavelet periodogram scale. Generally, we have to stay away from the finest Haar wavelet transform scales of the periodogram as the test relies on a central limit theorem effect which only "kicks in" when the Haar wavelet scale is medium-to-coarse. Generally, leave this argument alone.
maxSD	Parameter which controls which scales go towards overall variance calculation. Generally, leave alone.
verbose	If TRUE then informative messages are printed. If FALSE only limited informational messages are printed unless silent=TRUE.
silent	If TRUE then no messages are printed during the operation of the function at all.
UseCForVarip2	If TRUE then fast C code is used for the variance calculation, otherwise slower R code is used.
OPLENGTH	Some of the internal functions require workspace to perform their calculations. In exceptional circumstances more static workspace might be required and so this argument might need to be higher than the default. However, the code will tell you how high this number will need to be. The code can, with default arguments, handle series that are up to 30000 in length. However, at 35000 the OPLENGTH parameter will need to be increased.
mc.method	Method to control overall size for test taking into account multiple comparisons. The default argument is p.adjust.methods which is the same as the default argument to the p.adjust function in R. This includes a number of the popular methods such as "Holm", "Bonferonni" and "FDR", for example.

Details

This function computes all possible Haar wavelet coefficients of the time series x. Then, squares those to obtain the raw wavelet periodogram. Then the test of stationarity works by taking each level of the raw wavelet periodogram and subjecting it to another (decimated) Haar wavelet transform and then assessing whether any of those coefficients is significantly different to zero. It does this by using a Gaussian approximation first introduced by Neumann and von Sachs (2000). This is a multiple testing problem: many individual wavelet coefficients need to be assessed simultaneously and the user can choose the type of assessment using the mc.method argument.

Value

An object of class `tosANYN`. This is a list containing the following components.

`nreject`	The number of wavelet coefficients that reject the null hypothesis of being zero.
`mc.method`	The multiple comparison method used.
`AllTS`	All the t-statistics. This is a list containing J levels, where J is the number of periodogram levels. Each slot in the `AllTS` list itself contains a Haar wavelet transform object (`hwtANYN`) which are the t-statistics associated with each Haar wavelet coefficient of the Haar raw wavelet periodogram.
`AllPVal`	As `AllTS` but for p-values
`alpha`	The size of the test
`x`	The time series that was analyzed
`xSD`	The estimated mean spectrum value for each level of the spectrum, mean over time that is.
`allTS`	A vector containing all of the test statistics. So, the information in `AllTS` but arranged as a single vector
`allpvals`	As `allTS` but for p-values. These values have been adjusted to take account of the multiple comparisons. See the vaector `allpvals.unadjust` for an unadjusted set.
`allbigscale`	The wavelet periodogram scale associated with each t-statistic in `allTS`.
`alllitscale`	As for `allbigscale` but for the wavelet transform of the wavelet periodogram.
`allindex`	As for `allbigscale` but the wavelet coefficient index in the Haar wavelet transform of the wavelet periodogram
`alllv`	The maximum number of wavelet coefficients in a particular Haar wavelet scale of a particular scale of the wavelet periodogram. Note, this information is useful because the wavelet transforms are computed on arbitrary length objects and so keeping track of the number of coefficients per scale is useful later, e.g. for plotting purposes. This information is not required in the dyadic case because the coefficient vector lengths are completely predictable.
`allpvals.unadjust`	A vector of p-values that has not been adjusted by a multiple hypothesis test technique.

Author(s)

G. P. Nason

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

Priestley, M.B. and Subba Rao (1969) A test for non-stationarity of time series. *J. R. Statist. Soc. B*, **31**, 140-149.

von Sachs, R. and Neumann, M.H. (2000) A wavelet-based test for stationarity. *J. Time Ser. Anal.*, **21**, 597-613.

See Also

link{hwt}, hwtos2, plot.tosANYN, print.tosANYN, summary.tosANYN

Examples

```
#
# Generate test data set of non-dyadic length
#
v3 <- rnorm(300)
#
# Run the test of stationarity
#
## Not run: v3.TOS <- hwtos(v3)
#
#Scales get printed
#8    7    6    5    4    3    2
#
## Not run: print(v3.TOS)
#Class 'tosANYN' : Stationarity Object for Arbitrary Length Data :
# ~~~~~~~  : List with 14 components with names
# nreject mc.method AllTS AllPVal alpha x xSD allTS
# allpvals allbigscale alllitscale allindex alllv
# allpvals.unadjust
#
#
#summary(.):
#----------
#There are  54  hypothesis tests altogether
#There were  0  reject(s)
#P-val adjustment method was:  holm
#
# Note, nothing got rejected. So accept the H_0 null hypothesis of stationarity.
# This is precisely what you'd expect operating on iid Gaussians.
#
# Let's construct obvious example of non-stationarity.
#
v4 <- c(rnorm(150), rnorm(150,sd=3))
#
# I.e. v4 has sharp variance change halfway along
# Now compute test of stationarity
#
## Not run: v4.TOS <- hwtos(v4)
#
#  Print out results
#
## Not run: print(v4.TOS)
#
#Class 'tosANYN' : Stationarity Object for Arbitrary Length Data :
#       ~~~~~~~  : List with 14 components with names
#       nreject mc.method AllTS AllPVal alpha x xSD allTS
#       allpvals allbigscale alllitscale allindex alllv
#       allpvals.unadjust
```

```
#
#
#summary(.):
#----------
#There are  54  hypothesis tests altogether
#There were  5  reject(s)
#P-val adjustment method was:  holm
#Listing rejects...
#P:  7  HWTlev:  2  Max Poss Ix:  2  Indices:  2
#P:  7  HWTlev:  1  Max Poss Ix:  1  Indices:  1
#P:  6  HWTlev:  1  Max Poss Ix:  1  Indices:  1
#P:  5  HWTlev:  1  Max Poss Ix:  1  Indices:  1
#P:  4  HWTlev:  1  Max Poss Ix:  1  Indices:  1
```

`hwtos2` *Test of second-order stationarity using wavelets.*

Description

The main function to perform a test of second-order stationarity as outlined in Nason (2012). Essentially, this routine computes an evolutionary wavelet spectral estimate and then computes the Haar wavelet coefficients of each scale of the spectral estimate. Any large Haar coefficients are indicative of nonstationarity. A multiple hypothesis test assesses whether any of the Haar coefficients are large enough to reject the null hypothesis of stationarity.

Usage

```
hwtos2(x, alpha = 0.05, filter.number = 1, family = "DaubExPhase",
    lowlev = 3, WTscale = NULL, maxSD = NULL, verbose = FALSE,
    silent = FALSE, UseCForVarip2 = TRUE, OPLENGTH = 1e+05)
```

Arguments

`x`	The time series you want to test for second order stationarity. This should be a stochastic series. The function will report an error if `x` is a constant function. The function might not work properly if it contains a significant trend or patches of non-stochastic observations.
`alpha`	The overall (nominal) size of the test.
`filter.number`	The index number of the wavelet used to compute the evolutionary spectral estimate with.
`family`	The family of wavelet used to compute the evolutionary spectral estimate.
`lowlev`	Do not compute Haar wavelet coefficients on evolutionary wavelet spectra at level lower than `lowlev`.
`WTscale`	The theory of the test shows that the Haar wavelet coefficients of the raw wavelet periodogram are asymptotically normal as long as the scale of the Haar wavelet is 'coarse' enough. Roughly, speaking `WTscale` is internally coded to be the log of the square root of T, the length of the series (J/2), but you can set another value.

maxSD	As part of its execution, this function computes an evolutionary wavelet spectral estimate from the time series. Since the test is based on the assumption of stationarity, the EWS is averaged over time. There will be $J = \log_2 T$ scale levels and, if maxSD = NULL then all of the J levels get used for later functions, such as computing the variance of Haar wavelet coefficients. This argument permits you to restrict the number of coarse scales going into further calculations (e.g. removes the coarser scales from further examination). Mostly, the default will be fine and maximises the use of the available information.
verbose	If TRUE then informative error messages are printed. If FALSE they are not.
silent	If TRUE then no informative messages are printed. If FALSE then a limited amout of informative is printed.
UseCForVarip2	If TRUE then fast C code is use to compute wavelet coefficients' variance. If FALSE then R code is used wholly throughout, but the execution will be much slower.
OPLENGTH	The PsiJ and PsiJmat routines both used preallocated storage. This argument can be provided to increase the amount of storage. Note, you should not need to change this unless the routine as whole stops and tells you to rerun it with increased storage.

Details

This function looks at the Haar wavelet coefficients of an evolutionary wavelet spectrum. This is a modification of the principle of von Sachs and Neumann (2000) which worked with the Haar wavelet coefficients of a local Fourier spectrum.

See also, the stationarity test which implements the Priestley-Subba Rao (1969) test. This function is contained in the fractal package.

Value

An object of class tos, a list containing the following components:

nreject	The number of FDR rejections
rejpval	The p-value associated with FDR rejections
spvals	A vector of p-values from all of the tests, sorted in ascending order.
sTS	A vector of sorted test statistics from all of the tests, sorted into the same order as spvals
AllTS	A list containing all of the test statistics. The first entry contains test statistics corresponding to the coarsest scale, the last entry corresponds to the finest scale. Each component in the list is either empty (because the scale was omitted because it was less than lowlev) or contains a wd class object. The wd class object contains the test statistics for each Haar wavelet coefficient (not the coefficients). Hence, the value of the test statistic for any scale/location or level of the wavelet periodogram can easily be extracted.
AllPVal	As AllTS except the values stored are the p-values, not the test statistics.
alpha	The nominal size of the overall hypothesis test.
x	The original time series that was analyzed
xSD	A vector containing J levels, which is the EWS estimate averaged across time.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

Priestley, M.B. and Subba Rao (1969) A test for non-stationarity of time series. *J. R. Statist. Soc. B*, **31**, 140-149.

von Sachs, R. and Neumann, M.H. (2000) A wavelet-based test for stationarity. *J. Time Ser. Anal.*, **21**, 597-613.

See Also

varip2, stationarity

Examples

```
#
# First, test a set of iid Gaussians: should be stationary!
#
hwtos2(rnorm(256))
#  8    7    6    5    4    3
#Class 'tos' : Stationarity Object :
#       ~~~~  : List with 9 components with names
#               nreject rejpval spvals sTS AllTS AllPVal alpha x xSD
#
#
#summary(.):
#----------
#There are  186  hypothesis tests altogether
#There were  0  FDR rejects
#No p-values were smaller than the FDR val of:
#Using Bonferroni rejection p-value is  0.0002688172
#And there would be  0  rejections.
#
#  NOTE: the summary indicates that nothing was rejected: hence stationary!
#
# Second, example. Concatenated Gaussians with different variances
#
hwtos2(c(rnorm(256), rnorm(256,sd=2)))
#  9    8    7    6    5    4    3
#Class 'tos' : Stationarity Object :
#       ~~~~  : List with 9 components with names
#               nreject rejpval spvals sTS AllTS AllPVal alpha x xSD
#
#
#summary(.):
#----------
#There are  441  hypothesis tests altogether
```

```
#There were  5  FDR rejects
#The rejection p-value was  3.311237e-06
#Using Bonferroni rejection p-value is  0.0001133787
#And there would be  5  rejections.
#Listing FDR rejects... (thanks Y&Y!)
#P: 5 HWTlev:  0  indices on next line...[1] 1
#P: 6 HWTlev:  0  indices on next line...[1] 1
#P: 7 HWTlev:  0  indices on next line...[1] 1
#P: 8 HWTlev:  0  indices on next line...[1] 1
#P: 9 HWTlev:  0  indices on next line...[1] 1
#
# NOTE: This time 5 Haar wavelet coefficients got rejected: hence series
# is not stationary.
```

`idlastzero` *Return the index of the last zero in a vector*

Description

Return the index of the last zero in a vector, otherwise stop and return errror message. A helper routine for `mkcoef`.

Usage

```
idlastzero(v)
```

Arguments

v	Vector you wish to investigate

Value

The index within v of the last (right-most or one with the largest index) zero.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`mkcoef`

Examples

```
idlastzero(c(3,4,5,0,9))
#[1] 4
```

`lacf` *Compute localized autocovariance.*

Description

Compute localized autocovariance function for nonstationary time series. Note: this function is borrowed from the `costat` package, and modified to have linear smoothing, and when that package is complete, it will be removed from this package.

Usage

```
lacf(x, filter.number = 10, family = "DaubLeAsymm", smooth.dev = var,
    AutoReflect = TRUE, lag.max = NULL, WPsmooth.type = "RM",
    binwidth, tol=0.1, maxits=5, ABBverbose=0, verbose=FALSE, ...)
```

Arguments

`x`	The time series you wish to analyze
`filter.number`	Wavelet filter number you wish to use to analyse the time series (to form the wavelet periodogram, etc) See `filter.select` for more details.
`family`	Wavelet family to use, see `filter.select` for more details.
`smooth.dev`	Change variance estimate for smoothing. Note: `var` is good for this purpose.
`AutoReflect`	If `TRUE` then an internal reflection method is used to repackage the time series so that it can be analyzed by the periodic-assuming wavelet transforms.
`lag.max`	The maximum lag of acf required. If NULL then the same default as in the regular `acf` function is used.
`WPsmooth.type`	The type of smoothing used to produce the estimate. See `ewspec3` for more advice on this.
`binwidth`	If necessary, the `binwidth` for the spectral smoothing, see `ewspec3` for more info. If `WTsmooth.type=="RM"` then this argument specifies the binwidth of the kernel smoother applied to the wavelet periodogram. If the argument is missing or zero then an automatic bandwidth is calculated by `AutoBestBW`.
`tol`	Tolerance argument for `AutoBestBW`
`maxits`	Maximum iterations argument for `AutoBestBW`
`ABBverbose`	Verbosity of execution of `AutoBestBW`
`verbose`	If `TRUE` then informative message is printed
`...`	Other arguments for `ewspec3`.

Details

In essence, this routine is fairly simple. First, the EWS of the time series is computed. Then formula (14) from Nason, von Sachs and Kroisandr (2000) is applied to obtain the time-localized autocovariance from the spectral estimate.

Value

An object of class `lacf` which contains the autocovariance. This object can be handled by functions from the `costat` package. The idea in this package is that the function gets used internally and much of the same functionality can be achieved by running `Rvarlacf` and `plot.lacfCI`. However, running `lacf` on its own is much faster than `Rvarlacf` as the CI computation is intenstive.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

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

See Also

`Rvarlacf`

Examples

```
#
# With wavethresh attached, note binwidth is fabricated here,
# just to make the example work. The lacf implementation in
# the costat package performs wavelet (ie maybe better) smoothing automatically
#
v <- lacf(rnorm(256), binwidth=40)
#
# With costat attached also
#
## Not run: plot(v)
```

littlevar *Subsidiary helper function for hwtos2*

Description

Computes a variance estimate for hwtos2 Merely takes a wavelet periodogram (actually wd class object), and a level argument. Then extracts the wavelet periodogram coefficients at that level and returns twice the mean of their squares.

Usage

```
littlevar(WP, ll)
```

Arguments

WP	The wavelet periodogram that you wish to analyze (actually a wd class object, type="station"
ll	A valid level for the periodogram

Value

Twice the mean of the square of the coefficients at the level extracted.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwtos2

Examples

```
#
# Not intended for direct user use
#
```

mkcoef *Compute discrete wavelets.*

Description

For a given wavelet computes a list with each entry of the list containing that discrete wavelet at a different scale. The first entry corresponds to the finest wavelet, the next entry to the next finest, and so on.

Usage

```
mkcoef(J, filter.number = 10, family = "DaubLeAsymm")
```

Arguments

J	A NEGATIVE integer. -J is the maximum number of levels to compute.
filter.number	The filter number (number of vanishing moments) of the underlying wavelet to use.
family	The family of the wavelet. See wd help for further info.

Value

A list of length J. The first entry contains the discrete wavelet at the finest scale, the 2nd entry contains the next most finest wavelet, and so on.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

Rvarlacf, whichlevel

Examples

```
#
# E.g. compute discrete Haar wavelets on scales 1, 2, 3.
#
mkcoef(-3, 1, "DaubExPhase")
#[[1]]
#[1]  0.7071068 -0.7071068
#
```

```
#[[2]]
#[1]  0.5  0.5 -0.5 -0.5
#
#[[3]]
#[1]  0.3535534  0.3535534  0.3535534  0.3535534 -0.3535534 -0.3535534 -0.3535534
#[8] -0.3535534
```

---

plot.hwtANYN *Plots the transform contained in an* hwtANYN *object.*

Description

An hwtANYN object contains the results of a Haar wavelet transform computed on an object of non-dyadic length. It is the equivalent of the wd object for non-dyadic vectors for Haar wavelets. Note, the plot can only be carried out where the reindex slot of the object is TRUE.

Usage

```
## S3 method for class 'hwtANYN'
plot(x, xlabvals, xlabchars, ylabchars, first.level = 1,
    main = "Haar Wavelet Coefficients", scaling = c("global", "by.level"),
    rhlab = FALSE, sub, NotPlotVal = 0.005, xlab = "Translate",
    ylab = "wd-equivalent Resolution Level", miss.coef.col = 2,
    miss.coef.cex = 0.5, miss.coef.pch = 2, ...)
```

Arguments

x	The hwtANYN object containing the Haar wavelet transform coefficients you wish to plot.
xlabvals	Coordinates of x-axis labels you wish to add.
xlabchars	Labels to be printed at the x-axis labels specified.
ylabchars	Y-axis labels
first.level	Specifies the coarsest level to be plotted.
main	Specify a different main title for the plot.
scaling	How coefficients will be scaled on the plot. This can be two arguments "global" where all coefficients are plotted to the same scale and "by.level" where all coefficients on the same resolution level are plotted to the same scale, but coefficients on different resolution levels might be of different scales.
rhlab	If TRUE then the scale factor used for each level is shown.
sub	Specify a different subtitle for the plot.
NotPlotVal	Coefficients will not be plotted if their scaled height is less than NotPlotVal in absolute value. This is a useful way to completely suppress very small coefficient values.
xlab	Specify the x-axis label.

ylab	Specify the y-axis label.
miss.coef.col	What color to plot "missing coefficients" in.
miss.coef.cex	How big to plot the "missing coefficients" symbol.
miss.coef.pch	The type of plotting character used to plot the "missing coefficients".
...	Other arguments to plot.

Details

A plot of the different wavelet coefficients at the scales ranging from first.level to the finest scale. Note, in this plot the coefficients are NOT aligned with time at different scales in the same way as in the wd type plot - except the finest scale.

The Haar wavelet transform objects that this function plots are obtained originally from vectors of non-dyadic length. One can think of such a vector as a sub-vector of a longer vector of dyadic length. E.g. if your vector is of length 35 then it is a sub-vector of a vector of 64 (the next highest power of two). So, you can think of the Haar wavelet transform being of a vector of length 64 where 64-35=29 of the observations are missing. These missing observations "contribute" to wavelet (and scaling function) coefficients that are missing. This function has the ability to plot the "missing" coefficients, by default as small red triangles. The user can control the colour, size and plotting character of the missing observations.

Value

A single vector of length the number of levels plotted containing the value of the maximum absolute coefficient value.

Author(s)

G. P. Nason

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

Priestley, M.B. and Subba Rao (1969) A test for non-stationarity of time series. *J. R. Statist. Soc. B*, **31**, 140-149.

von Sachs, R. and Neumann, M.H. (2000) A wavelet-based test for stationarity. *J. Time Ser. Anal.*, **21**, 597-613.

See Also

hwt, print.hwtANYN

Examples

```
#
# Generate test data of length 82
#
v3 <- rnorm(82)
#
# Compute Haar wavelet transform, note reindex has to be true for subsequent
# plot.
#
v3.hwt <- hwt(v3, reindex=TRUE)
#
#
## Not run: plot(v3.hwt)
```

`plot.lacf` *Plot localized autocovariance (lacf) object.*

Description

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

Usage

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

Arguments

`x`	The localized autocovariance object you want to plot (lacf)
`plotcor`	If TRUE then plot autocorrelations, otherwise plot autocovariances.
`type`	The lacf 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 lacf 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.
plot.it	If TRUE the plot is produced and displayed. If FALSE then no plot is produced but the autocovariance or autocorrelation values that would have been produced are returned as numerical values instead. This means that this function is an extractor function for the lacf class object.
xlab	X-axis label, constructed internally if not supplied
ylab	Y-axis label, constructed internally if not supplied
...	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, G.P. (2012) Costationarity of Locally Stationary Time Series using costat.

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. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

lacf

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.
#
vlacf <- lacf(v, lag.max=30)
#
# Now let's do some plotting of the localized autocovariance
#
## Not run: plot(vlacf, 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(vlacf, 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(vlacf, type="acf", the.time=200)
#
# Now plot the autocovariance, rather than the autocorrelation.
#
## Not run: plot(vlacf, 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.
```

`plot.lacfCI` *Plot confidence intervals for localized autocovariance for locally stationary time series.*

Description

Plot the localized autocovariance and approximate confidence intervals.

Usage

```
## S3 method for class 'lacfCI'
plot(x, plotcor = TRUE, type = "line",
    lags = 0:as.integer(10 * log10(nrow(x$lacf))), tcex = 1,
    lcol = 1, llty = 1, ylim = NULL, segwid = 1,
    segandcross = TRUE, conf.level = 0.95, plot.it = TRUE,
    xlab, ylab, sub, ...)
```

Arguments

x	The lacfCI object you wish to plot, e.g. produced by the Rvarlacf function.
plotcor	If TRUE then autocorrelations are plotted, if FALSE then autocovariances are. Note: not all combinations of types of plot and plotcor are valid, but many are.
type	This can be one of three values "line", "persp" or "acf". The value "acf" produces a plot like the regular acf function, but note, the values plotted are from a localized autocovariance function centred at the time location contained in the object object (and that time appears in the subtitle). This is the only plot that also plots the confidence intervals. The "line" plot plots autocorrelations (only) for the specified lags and does this over all time for the whole extent of the series. This plot is useful to see if the autocorrelations are changing over time. The final option, "persp" produces a perspective plot of the autocovariance or autocorrelations. Arguments can be supplied (theta, phi) to rotate the perspective plot, as it can be sometimes hard to visualize the plot.
lags	The lags that you wish to display. This should be a list of non-negative integers, but not necessarily consecutive.
tcex	On the "line" plot this argument controls the expansion of the font for the labels on the lines. So, setting tcex=2, for example, will double the size of these. These labels visually indicate which line corresponds to which lag.
lcol	On the "line" plot, this argument controls the colour of the lines that are used to show the acfs.
llty	As lcol but for line types.
ylim	The vertical limits of the plot.
segwid	On the "acf" plot, this argument controls the widths of the little acf segments that connect the x-axis with the acf values.
segandcross	If TRUE then a small diamond is plotted at the location of the acf, to make it clearer.
conf.level	The confidence level of the confidence intervals.
plot.it	If FALSE then no plot is produced. This can be used if you merely want to extract the relevant acf values (which are returned).
xlab	X-axis label, constructed internally if not supplied
ylab	Y-axis label, constructed internally if not supplied
sub	A subtitle for the plot
...	Other arguments to the main plot command.

Details

This function can plot the localized autocovariance in three ways. Like a regular acf plot (but obviously a slice out of a time-varying autocovariance, not the regular acf), a line plot which shows the acfs over time and a perspective plot which can plot the estimate of $c(z, \tau)$ as a 2D function. Currently, the confidence intervals can only be displayed on the "acf" type plot.

Value

A vector of the extracted acfvals invisibly returned. Note: what is returned depends on the arguments, what is returned is what would have been plotted if plot.it were TRUE

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

Rvarlacf

Examples

```
#
# Simulate a TVAR(1) process
#
x <- tvar1sim()
#
# Computes its time-localized autocovariance and confidence intervals
# Note: smoothing is done automatically!
#
x.lacf <- Rvarlacf(x=x, nz=50, var.lag.max=20)
#
# Now plot this, plot covariances as an acf plot, with the CIs
#
## Not run: plot(x.lacf, type="acf", plotcor=FALSE)
#
# Now plot it as a line plot, as correlations and can't do CIs
#
## Not run: plot(x.lacf)
```

plot.tos *Produces a graphical representation of the results of a test of stationarity contained in a* tos *object.*

Description

After a test of stationarity for dyadic data (e.g. hwtos2) is applied to a time series it generates a results object of class tos. This function takes objects of that class and produces a graphical representation of the test.

Usage

```
## S3 method for class 'tos'
plot(x, mctype = "FDR", sub = NULL, xlab = "Time",
    arrow.length = 0.05, verbose = FALSE, ...)
```

Arguments

x	The tos class object, the results of the test of stationarity that you wish to plot.
mctype	Whether you wish to see rejections (if they exist) according to a Bonferroni assessment ("BON") or according to FDR ("FDR")
sub	An argument to change the subtitle.
xlab	An argument to change the x-axis label.
arrow.length	The length of the edges of the arrow head (in inches). Note that this is the argument that is supplied as the length argument of the arrow function that is called by this routine to draw the arrows.
verbose	If TRUE then some meaningless debugging information is printed.
...	Other arguments to the main ts.plot routine that does the plotting.

Details

The following things are usually plotted. 1. The time series that was investigated. The left-hand axes is that for the time series. The horizontal axis is time (but just integers indexing). If the series was deemed stationary by the test then that's it except that the subtitle indicates that no Haar wavelet coefficients were rejected as being nonzero.

If the test indicated that the series was nonstationary then the subtitle indicates this by stating the number of rejections (this might be according to FDR or Bonferroni depending on the setting of the mctype argument. Then graphical representations of any significant Haar wavelet coefficients are plotted as double-headed red horizontal arrows on the plot. The horizontal extent corresponds to the support of the underlying wavelet. The vertical position of the arrows gives an indication of the wavelet periodogram scale where the significant coefficient was found. The wavelet periodogram scales are indexed by the right hand axis, and beware, the numbers might not be consecutive, but they will be ordered (so e.g. if no signficant coefficients were discovered at wavelet periodogram scale level 6, then that scale/axis label will not appear). The scale within the Haar wavelet transform is indicated by the vertical position WITHIN ticks between wavelet periodogram scales (ie, there are

TWO scales: the wavelet periodogram scale that is currently being analyzed, and the Haar wavelet transform scale within the periodogram scale). So, if two right hand axis labels are, e.g., 4 and 5, and horizontal arrows appear between these two they actually correspond to different Haar wavelet transform scales AT wavelet periodogram level 4. It is not usually possible to tell precisely which Haar wavelet transform scale the coefficients can come from, but the information can be extracted from the summary.tos function which lists this.

Value

None.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwtos2, summary.tos

Examples

```
#
# Produces an interesting plot with high probability
#
#
# Note that the input time series is two concatenated white noise
# sequences with very different variances.
#
answer <- hwtos2(c(rnorm(256), rnorm(256, sd=5)))
## Not run: plot(answer)
```

`plot.tosANYN` *Produces a graphical representation of the results of a test of stationarity from a* `tosANYN` *object.*

Description

After a test of stationarity (e.g. hwtos) is applied to a time series it generates a results object of class `tosANYN`. This function takes objects of that class and produces a graphical representation of the test.

Usage

```
## S3 method for class 'tosANYN'
plot(x, sub = NULL, xlab = "Time",
    arrow.length = 0.05, verbose = FALSE, ...)
```

Arguments

x	The tosANYN class object, the results of the test of stationarity that you wish to plot.
sub	An argument to change the subtitle.
xlab	An argument to change the x-axis label.
arrow.length	The length of the edges of the arrow head (in inches). Note that this is the argument that is supplied as the length argument of the arrow function that is called by this routine to draw the arrows.
verbose	If TRUE then some meaningless debugging information is printed.
...	Other arguments to the main ts.plot routine that does the plotting.

Details

The following things are usually plotted. 1. The time series that was investigated. The left-hand axes is that for the time series. The horizontal axis is time (but just integers indexing). If the series was deemed stationary by the test then that's it except that the subtitle indicates that no Haar wavelet coefficients were rejected as being nonzero.

If the test indicated that the series was nonstationary then the subtitle indicates this by stating the number of rejections. Then graphical representations of any significant Haar wavelet coefficients are plotted as double-headed red horizontal arrows on the plot. The horizontal extent corresponds to the support of the underlying wavelet. The vertical position of the arrows gives an indication of the wavelet periodogram scale where the significant coefficient was found. The wavelet periodogram scales are indexed by the right hand axis, and beware, the numbers might not be consecutive, but the will be ordered (so e.g. if no signficant coefficients were discovered at wavelet periodogram scale level 6, then that scale/axis label will not appear). The scale within the Haar wavelet transform is indicated by the vertical position WITHIN ticks between wavelet periodogram scales (ie, there are TWO scales: the wavelet periodogram scale that is currently being analyzed, and the Haar wavelet transform scale within the periodogram scale). So, if two right hand axis labels are, e.g., 4 and 5, and horizontal arrows appear between these two they actually correspond to different Haar wavelet transform scales AT wavelet periodogram level 4. It is not usually possible to tell precisely which Haar wavelet transform scale the coefficients can come from, but the information can be extracted from the summary.tosANYN function which lists this.

Value

None.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwtos, summary.tosANYN

Examples

```
#
# Produces an interesting plot with high probability
#
#
# Note that the input time series is two concatenated white noise
# sequences with very different variances.
#
## Not run: answer <- hwtos(c(rnorm(256), rnorm(256, sd=5)))
## Not run: plot(answer)
```

---

print.hwtANYN *Print out a* hwtANYN *class object, eg from the* link{hwt} *function.*

Description

Prints out very basic information on an object that represents a Haar wavelet transform of a data set of non-dyadic length.

Usage

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

Arguments

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

Value

This function calls the summary.hwtANYN function as its last action. So, the return from this function is the return from summary.hwtANYN

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwt, summary.hwtANYN

Examples

```
#
# Generate test vector of length 5
#
v2 <- rnorm(5)
#
# Compute Haar wavelet transform
#
v2.hwt <- hwt(v2)
#
# Print out the answer
#
print(v2.hwt)
#Class 'hwtANYN' : Haar Wavelet for Arbitrary Length Data object:
#       ~~~~~~~  : List with 5 components with names
#    c d nlevels type reindex
#
#
#summary(.):
#----------
#Levels:  2
#Filter was: Haar
#Transform type:  wavelet
#Object was reindex to match wd:  FALSE
```

---

print.lacf *Print lacf class object*

Description

Prints information about lacf class object.

Usage

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

Arguments

x	The lacf class object you want to print
...	Other arguments

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, G.P. (2012) Costationarity of Locally Stationary Time Series using `costat`.

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. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`lacf`, `plot.lacf`, `summary.lacf`

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.
#
vlacf <- lacf(v, lag.max=30)
#
# Now let's print the lacf object
#
print(vlacf)
#Class 'lacf' : Localized Autocovariance/correlation Object:
#       ~~~~  : List with 3 components with names
#             lacf 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
```

`print.lacfCI` *Print basic information about a* `lacfCI` *object.*

Description

Prints basic information about a `lacfCI` object, which contains information on confidence intervals for localized autocovariance.

Usage

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

Arguments

`x`	The `lacfCI` object.
`...`	Other arguments

Value

The last action of this function is to compute `summary.tos` so the return code is whatever that function returns.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`summary.lacfCI`, `Rvarlacf`

Examples

```
#
# See example on Rvarlacf help page
```

print.tos *Print out a* tos *class object, eg from the* link{hwtos2} *function.*

Description

Prints out very basic information on an object that represents the output from a test of stationarity.

Usage

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

Arguments

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

Value

This function calls the summary.tos function as its last action. So, the return from this function is the return from summary.tos

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwtos2

Examples

```
#
# See example at end of help for hwtos2
#
```

print.tosANYN *Print out a* tosANYN *class object, eg from the* link{hwtos} *function.*

Description

Prints out very basic information on an object that represents the output from a test of stationarity.

Usage

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

Arguments

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

Value

This function calls the summary.tosANYN function as its last action. So, the return from this function is the return from summary.tosANYN

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwtos,summary.tosANYN

Examples

```
#
# See example at end of help for hwtos
#
```

`runmean` *Compute a running mean of a vector*

Description

This function essentially uses the `running.mean` function from the `igraph` package. However, adjustments are made to ensure that the output is always the same length as the input (by fiddling at the boundaries).

Usage

```
runmean(x, binwidth)
```

Arguments

`x`	Vector that you wish to smooth using a running mean.
`binwidth`	Number of ordinates over which you wish to average

Details

For example, if `binwidth=2` and `x=1:6` then the function averages each pair to get 1.5, 2.5, 3.5, 4.5, 5.5. However, this is only 5 numbers and the input had 6. So, in this case the function arranges for the output to be extended (in this case 1 gets padded onto the front. For vectors of length > 3 the padding depends on whether the vector is even or odd.

Value

The running mean of the input at the given bandwidth.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`ewspec3`

Examples

```
runmean(1:6, 2)
#
# [1] 1.0 1.5 2.5 3.5 4.5 5.5
#
runmean(1:14, 4)
#
#  [1]  1.75  2.50  3.50  4.50  5.50  6.50  7.50  8.50  9.50 10.50 11.50 12.5
# [13] 13.25 13.50
#
```

`Rvarlacf` *Compute confidence intervals for localized autocovariance for locally stationary time series.*

Description

Compute a localized autocovariance and associated confidence intervals for a locally stationary time series. The underlying theory assumes a locally stationary wavelet time series, but will work well for other time series that are not too far away.

Usage

```
Rvarlacf(x, nz, filter.number = 1, family = "DaubExPhase",
    smooth.dev = var, AutoReflect = TRUE, lag.max = NULL,
    WPsmooth.type = "RM", binwidth = 0, mkcoefOBJ, ThePsiJ,
    Cverbose = 0, verbose = 0, OPLENGTH = 10^5, var.lag.max = 3,
    ABB.tol = 0.1, ABB.plot.it = FALSE, ABB.verbose = 0,
    ABB.maxits = 10, truedenom=FALSE, ...)
```

Arguments

`x`	The time series you wish to analyze
`nz`	The time point at which you wish to compute the localized autocovariance for.
`filter.number`	The analysis wavelet for many things, including smoothing. See `wd` for information on the various types.
`family`	The analysis wavelet family. See `wd` again.
`smooth.dev`	The deviance function used to perform smoothing of the evolutionary wavelet spectrum.
`AutoReflect`	The internal wavelet transforms assume periodic boundary conditions. However, most time series are not periodic (in terms of their support, e.g. the series at time 1 is not normally anywhere near the value of the series at time T). This argument, if `TRUE` mitigates this by reflecting the whole series by the right-hand end, computing the transform (for which periodic transforms are now valid) and then junking the second half of the estimate. Although this is slightly more computationally intensive, the results are better.

lag.max	The maximum number of lags to compute the localized autocovariance for. The default is the same as in the regular acf function.
WPsmooth.type	The type of smoothing of the evolutionary wavelet spectrum and the localized autocovariance. See the arguments to lacf.
binwidth	The smoothing bandwidth associated with the smoothing controlled by WPsmooth.type. If this value is zero then the binwidth is computed automatically by the routine. And if verbose>0 the value is also printed.
mkcoefOBJ	Optionally, the appropriate discrete wavelet transform object can be supplied. If it is not supplied then the routine automatically computes it. There is a small saving in providing it, so for everyday use probably not worth it.
ThePsiJ	As for mkcoefOBJ but the autocorrelation wavelet object.
Cverbose	If positive integer then the called C code produces verbose messages. Useful for debugging.
verbose	If positive integer >0 then useful messages are printed. Higher values give more information.
OPLENGTH	Parameter that controls storage allocated to the PsiJ routine. It is possible, for large time series, you might be asked to increase this value.
var.lag.max	Number of lags that you want to compute confidence intervals for. Usually, it is quick to compute for more lags, so this could usually be set to be the value of lag.max above.
ABB.tol	The routine selects the automatic bandwidth via a golden section search. This argument controls the optimization tolerance.
ABB.plot.it	Whether or not to plot the iterations of the automatic bandwidth golden section search. (TRUE/FALSE)
ABB.verbose	Positive integer controlling the amount of detail from the automatic bandwidth golden section search algorithm. If zero nothing is produced.
ABB.maxits	The maximum number of iterations in the automatic bandwidth golden section search.
truedenom	If TRUE use the actual number of terms in the sum as the denominator in the formula for the calculation of the covariance of the smoothed periodogram. If FALSE use the eqn(2s+1)^-2 (this was the default in versions prior to 1.7.4)
...	Other arguments

Details

1. If binwidth=0 the function first computes the 'best' linear running mean binwidth (bandwidth) for the smooth of the localized autocovariance. 2. The function computes the localized autocovariance smoothed with a running mean with the selected binwidth. Then, the variance of $\hat{c}(z, \tau)$ is computed for the selected value of time z=nz and for the lags specified (in var.lag.max). The results are returned in an object of class lacfCI.

Note, this function computes and plots localized autocovariances for a particular and fixed time location. Various other plots, including perspective plots or the localized autocovariance function over all time can be found in the costat package. (Indeed, this function returns a lacfCI object that contains a lacf object, and interesting plots can be plotted using the plot.lacf function within costast.

Value

An object of class `lacfCI`. This is a list with the following components.

`lag`	The lags for which the localized autocovariance variance is computed
`cvar`	The variances associated with each localized autocovariance
`the.lacf`	The `lacf` class object that contains the localized autocovariances themselves. This object can be handled/plotted/etc using the functions in the `costat` package although `plot.lacfCI` contains much of the functionality of `plot.lacf`.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`plot.lacfCI`, `print.lacfCI`, `summary.lacfCI`

Examples

```
#
# Do localized autocovariance on a iid Gaussian sequence
#
tmp <- Rvarlacf(rnorm(256), nz=125)
#
# Plot the localized autocovariances over time (default plot, doesn't
# produce CIs)
#
## Not run: plot(tmp)
#
# You should get a plot where the lag 0 acs are all near 1 and all the
# others are near zero, the acfs over time.
#
## Not run: plot(tmp, plotcor=FALSE, type="acf")
#
# This plots the autocovariances (note: \code{plotcor=FALSE}) and the
# type of plot is \code{"acf"} which means like a regular ACF plot, except
# this is c(125, tau), ie the acf localized to time=125 The confidence
# intervals are also plotted.
# The plot subtitle indicates that it is c(125, tau) that is being plotted
#
```

`StoreStatistics` *Interogates calculation store to see how well we are reusing previous calculations (debugging)*

Description

The computation of the variance of the lacf estimator is intensive and we try to speed that up by reusing calculations. These calculations are stored in internal C arrays. This function interrogates those arrays and can provide details on how well the storage is working and provide hints if more storage needs to be allocated. For very large time series it is possible that values need to be calculated that can be stored and this function can monitor this.

Usage

```
StoreStatistics()
```

Details

The function prints out the state of the storage. Three numbers are reported on. 1. The number of values that were calculated but not stored "outside framework". Ideally you want this number to be low, if it gets persistently high then more storage needs to be allocated in the C code (notably MAXELL, MAXJ, MAXK, MAXD for the ThmStore and ValExists arrays).

The other two numbers are "Number stored" and "Number found". The first number corresponds to the number of values calculated once and then stored. The second number contains the number of times the software interogated the store and found a value that it did not have to then calculate. So, ideally, you'd like the latter number to be a high percentage of the former number, as this means the store is working efficiently.

Note, this function is definitely not intended for casual users. However, for users of very large series, who have the computational resources, these storage parameters might need to be increased.

The values will be zero if `Rvarlacf` has not yet been called, and only refer to the last call to that function (as the function zeroes the store on invocation).

Value

None.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

Rvarlacf

Examples

```
#
# Simulate some data
#
x <- tvar1sim()
#
# Calculate lacf and confidence intervals
#
x.lacf <- Rvarlacf(x=x, nz=50, var.lag.max=20)
#
# Find out how the store did
#
StoreStatistics()
#Number calculated outside framework:  0
#Number calculated then stored:  154440
#Number found in store:  14980680
#
#Overall % calculated:  1.020408
#% outside framework:  0
```

---

summary.hwtANYN *Summarize the results of a Haar wavelet transform object computed from an arbitrary length vector.*

Description

This function summarizes the results of a hwtANYN object that contains the results of a Haar wavelet transform that had been carried out on an original vector of (potentially) non-dyadic length.

Usage

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

Arguments

object	The object that you want a summary of. The object might be the results from, e.g., hwt function.
...	Other arguments

Value

None.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`hwt`, `print.hwtANYN`

Examples

```
#
# See help for print.hwtANYN
```

`summary.lacf` *Summarizes a lacf object*

Description

Summarizes a lacf object

Usage

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

Arguments

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

Value

None

Author(s)

Guy Nason

References

Cardinali, A. and Nason, G.P. (2012) Costationarity of Locally Stationary Time Series using `costat`.

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. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`lacf`, `plot.lacf`, `print.lacf`

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.
#
vlacf <- lacf(v, lag.max=20)
#
# Now let's summarize the lacf object
#
summary(vlacf)
#Name of originating time series:
#Date produced:  Thu Oct 25 12:11:29 2012
#Number of times:  256
#Number of lags:  20
```

`summary.lacfCI` *Produce a brief summary of the contents of a* `lacfCI` *object*

Description

Produces brief summary of the contents of a `lacfCI` object.

Usage

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

Arguments

object	The lacfCI object that you wish to glean info on
...	Other arguments.

Value

No value

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

print.lacfCI,Rvarlacf

Examples

```
#
# See example in the Rvarlacf function
```

---

summary.tos *Summarize the results of a test of stationarity contained i a* tos *object.*

Description

This function summarizes the results of a tos object that contains information about a test of stationarity. The summary function prints out information about how many individual hypothesis tests there were, how many were rejected and what the (equivalent) rejection p-value was (in the last cases both for FDR and Bonferroni). If the hypothesis of stationarity is rejected then the routine also prints out a list of the Haar wavelet coefficients (and their scales, locations and scale of the wavelet periodogram) that were significant. The function also returns a lot of this information (invisibly).

Usage

```
## S3 method for class 'tos'
summary(object, size = 0.05, quiet = FALSE, mctype = "FDR", ...)
```

Arguments

object	The output from a function that carries out a test of stationarity, e.g. hwtos2.
size	The nominal overall significance level of the test.
quiet	If this argument is TRUE then nothing is printed to the screen, although the information is still returned as an object. If FALSE then this function prints out the information about the hypothesis test.
mctype	This argument can be "FDR" for false discovery rate or "BON" for Bonferroni. This argument does not effect the basis printout. However, it does control whether FDR or Bonferroni rejects are listed, and it does control the type of information returned by the function (whether FDR or Bonferroni info).
...	Other arguments

Value

The function returns a list which contain a list of the rejected coefficients. Each list item contains the index of a particular rejected coefficient, which is a vector of at least three elements. The first element corresponds to the scale of the wavelet periodogram, the second is the level of the Haar wavelet transform, and all remaining values are the index of the significant wavelet coefficients at that Haar wavelet transform scale. The list also contains the total number of Haar wavelet coefficients rejected and the mctype argument also.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwtos2, print.tos

Examples

```
#
# See example for hwtos2, this contains two examples where
# summary.tos (as summary(.) is used in the output.
```

summary.tosANYN *Summarize the results of a test of stationarity contained in an* tosANYN *class object.*

Description

This function summarizes the results of a tosANYN object that contains information about a test of stationarity. The summary function prints out information about how many individual hypothesis tests there were, how many were rejected. If the hypothesis of stationarity is rejected then the routine also prints out a list of the Haar wavelet coefficients (and their scales, locations and scale of the wavelet periodogram) that were significant. The function also returns a lot of this information (invisibly).

Usage

```
## S3 method for class 'tosANYN'
summary(object, quiet = FALSE, ...)
```

Arguments

object	The output from a function that carries out a test of stationarity, e.g. hwtos.
quiet	If this argument is TRUE then nothing is printed to the screen, although the information is still returned as an object. If FALSE then this function prints out the information about the hypothesis test.
...	Other arguments

Value

The function returns a list which contain a list of the rejected coefficients. Each list item contains the index of a particular rejected coefficient, which is a vector of at least three elements. The first element corresponds to the scale of the wavelet periodogram, the second is the level of the Haar wavelet transform, and all remaining values are the index of the significant wavelet coefficients at that Haar wavelet transform scale. The list also contains the total number of Haar wavelet coefficients rejected and the mctype argument also.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

hwtos, print.tosANYN

Examples

```
#
# See example for hwtos, this contains two examples where
# summary.tosANYN (as summary(.) is used in the output.
```

`tvar1sim` *Simulate a realization from a particular TVAR(1) model.*

Description

Simulates a realization from a TVAR(1) model where the AR(1) parameter moves from 0.9 to -0.9 in equal steps over 512 time points. The realization is also of length 512. The innovations are normally distributed with mean zero and standard deviation of `sd`.

Usage

```
tvar1sim(sd = 1)
```

Arguments

`sd`	This is the standard deviation of the Gaussian innovation.

Details

This function is easily converted into one that does the same thing but for a different sample size.

Value

A realization of the aforementioned TVAR(1) process.

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

`Rvarlacf`

Examples

```
#
# Generate realization from the TVAR(1) process
#
x <- tvar1sim()
#
# Maybe plot it
#
## Not run: ts.plot(x)
```

`varip2` *Direct computation of estimate of variance of v_ip, the Haar wavelet coefficients of the periodogram.*

Description

Performs a direct computation of an estimate of the variance of the Haar wavelet coefficients of the raw wavelet periodogram of a time series.

Usage

```
varip2(i, p, ll, S, P)
```

Arguments

`i`	Scale parameter of Haar wavelet analyzing periodogram. Scale 1 is the finest scale.
`p`	Location parameter of Haar wavelet analyzing periodogram
`ll`	Scale of the raw wavelet periodogram being analyzed
`S`	Estimate of the spectrum, under the assumption of stationarity. So, this is just a vector of (possibly) J scales (which is often the usual spectral estimate averaged over time). Note: that the main calling function, `hwtos2`, actually passes `maxD` levels.
`P`	Is an autocorrelation wavelet object, returned by the `PsiJ` function. The wavelet concerned is the analyzing one underlying the raw wavelet periodogram of the series.

Details

Computes the variance of the Haar wavelet coefficients of the raw wavelet periodogram. Note, that this is merely an estimate of the variances.

Value

A list with the following components:

covAA	A component of the variance
covAB	A component of the variance
covBB	A component of the variance
ans	The actual variance

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

Cvarip2,hwtos2, covIwrap

Examples

```
#
# Generate autocorrelation wavelets
#
P1 <- PsiJ(-5, filter.number=1, family="DaubExPhase")
#
#
# Now compute varip2: this is the variance of the Haar wavelet coefficient
# at the finest scale, location 10 and P1 autocorrelation wavelet.
# Note, I've used S to be the exact coefficients which would be for white noise.
# In practice, S would be an *estimate* calculated from the data.
#
varip2(i=1, p=10, ll=2, S=c(1/2, 1/4, 1/8, 1/16, 1/32), P=P1)
#
# Ans component is 1.865244
```

---

whichlevel *Helper routine for* mkcoef

Description

Helps mkcoef by finding out how many more levels are required to compute a set of discrete wavelets to a given (other) level.

Usage

```
whichlevel(J, filter.number = 10, family = "DaubLeAsymm")
```

Arguments

J	The level that mkcoef wants to compute to.
filter.number	The wavelet number (see wd)
family	The wavelet family (see wd)

Details

When computing the discrete wavelets up to a given scale we use the inverse wavelet transform to do this. However, to generate a wavelet within the range of a wavelet decomposition you have to use more scales in the inverse wavelet transform than first requested. This is because wavelet coefficients at the coarsest scales are associated with wavelets whose support is greater than the whole extent of the series. Hence, you have to have a larger wavelet transform, with more levels, insert a coefficient mid-level to generate a discrete wavelet whose support lies entirely within the extent of the series. This function figures out what the extra number of levels should be.

Value

Simply returns the required number of levels

Author(s)

Guy Nason.

References

Nason, G.P. (2013) A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *J. R. Statist. Soc.* B, **75**, 879-904. doi:10.1111/rssb.12015

See Also

mkcoef

Examples

```
whichlevel(6)
# [1] 11
#
# E.g. mkcoef wanted to generate 6 levels of discrete wavelets and
# whichlevel tells it that it needs to generate a wavelet transform
# of at least 11 levels.
```

zeropad *Intersperse zeroes in a vector.*

Description

Take a vector of any length and then intersperse zeroes between existing elements (and add a further zero to the end).

Usage

```
zeropad(x)
```

Arguments

x	Vector that you want to intersperse zeros into.

Details

Title says it all.

Value

A vector whose odd elements are just x and whose even elements are zeroes.

Author(s)

G.P. Nason

See Also

hwt

Examples

```
#
# Operate on a test set
#
v <- 1:3
zeropad(v)
#[1] 1 0 2 0 3 0
```

Index