Package: wavethresh (via r-universe)

October 6, 2026

Type Package

Title Wavelets Statistics and Transforms

Version 4.7.3

Date 2024-08-19

Depends R (>= 2.10), MASS

Description Performs 1, 2 and 3D real and complex-valued wavelet transforms, nondecimated transforms, wavelet packet transforms, nondecimated wavelet packet transforms, multiple wavelet transforms, complex-valued wavelet transforms, wavelet shrinkage for various kinds of data, locally stationary wavelet time series, nonstationary multiscale transfer function modeling, density estimation.

License GPL (>= 2)

NeedsCompilation yes

Author Guy Nason [aut, cre], Stuart Barber [ctb], Tim Downie [ctb], Piotr Frylewicz [ctb], Arne Kovac [ctb], Todd Ogden [ctb], Bernard Silverman [ctb]

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

Repository https://guynason.r-universe.dev

Date/Publication 2024-08-20 02:55:38 UTC

RemoteUrl https://github.com/cran/wavethresh

RemoteRef HEAD

RemoteSha 3fa5edea7cf8db11845e9c8b1043df6a35685ecd

Contents

wavethresh-package 7
accessc 18
accessC 19
accessC.mwd 20
accessC.wd 21

accessC.wp . . . 23
accessC.wst . . . 24
accessD . . . 25
accessD.mwd . . . 26
accessD.wd . . . 27
accessD.wd3D . . . 29
accessD.wp . . . 31
accessD.wpst . . . 32
accessD.wst . . . 34
addpkt . . . 35
AutoBasis . . . 36
av.basis . . . 36
AvBasis . . . 37
AvBasis.wst . . . 38
AvBasis.wst2D . . . 40
BabyECG . . . 42
BabySS . . . 44
basisplot . . . 46
basisplot.BP . . . 47
basisplot.wp . . . 48
BAYES.THR . . . 49
Best1DCols . . . 51
bestm . . . 52
BMdiscr . . . 53
c2to4 . . . 54
CanUseMoreThanOneColor . . . 55
checkmyews . . . 55
Chires5 . . . 56
Chires6 . . . 57
cns . . . 58
compare.filters . . . 59
compgrot . . . 60
compress . . . 62
compress.default . . . 63
compress.imwd . . . 64
conbar . . . 65
convert . . . 67
convert.wd . . . 68
convert.wst . . . 70
ConvertMessage . . . 72
Crsswav . . . 73
cthresh . . . 74
Cthreshold . . . 76
CWavDE . . . 77
CWCV . . . 79
dclaw . . . 81
dencvwd . . . 82
denplot . . . 83

denproj . . . 84
denwd . . . 86
denwr . . . 87
DJ.EX . . . 88
dof . . . 89
doppler . . . 90
draw . . . 91
draw.default . . . 92
draw.imwd . . . 95
draw.imwdc . . . 96
draw.mwd . . . 98
draw.wd . . . 99
draw.wp . . . 100
draw.wst . . . 102
drawbox . . . 103
drawwp.default . . . 104
ewspec . . . 105
example.1 . . . 108
filter.select . . . 109
find.parameters . . . 112
first.last . . . 113
first.last.dh . . . 116
firstdot . . . 117
FullWaveletCV . . . 118
GenW . . . 119
getarrvec . . . 121
getpacket . . . 123
getpacket.wp . . . 124
getpacket.wpst . . . 126
getpacket.wst . . . 128
getpacket.wst2D . . . 130
GetRSSWST . . . 133
griddata objects . . . 134
guyrot . . . 136
HaarConcat . . . 137
HaarMA . . . 138
image.wd . . . 140
image.wst . . . 141
imwd . . . 142
imwd.object . . . 144
imwdc.object . . . 145
imwr . . . 148
imwr.imwd . . . 149
imwr.imwdc . . . 150
InvBasis . . . 152
InvBasis.wp . . . 153
InvBasis.wst . . . 154
ipd . . . 155

ipndacw . . . 156
irregwd . . . 159
irregwd.objects . . . 161
IsEarly . . . 162
IsEarly.default . . . 163
IsEarly.wd . . . 163
IsPowerOfTwo . . . 164
l2norm . . . 165
lennon . . . 166
levarr . . . 167
linfnorm . . . 168
LocalSpec . . . 169
LocalSpec.wd . . . 170
LocalSpec.wst . . . 173
logabs . . . 174
LSWsim . . . 175
lt.to.name . . . 177
madmad . . . 179
make.dwwt . . . 181
makegrid . . . 182
makewpstDO . . . 184
makewpstRO . . . 187
MaNoVe . . . 192
MaNoVe.wp . . . 193
MaNoVe.wst . . . 194
mfilter.select . . . 197
mfirst.last . . . 199
modernise . . . 201
modernise.wd . . . 202
mpostfilter . . . 202
mprefilter . . . 203
mwd . . . 204
mwd.object . . . 206
mwr . . . 208
newsure . . . 209
nlevelsWT . . . 210
nlevelsWT.default . . . 211
nullevels . . . 212
nullevels.imwd . . . 213
nullevels.wd . . . 214
nullevels.wst . . . 216
numtonv . . . 217
nv.object . . . 219
plot.imwd . . . 220
plot.irregwd . . . 221
plot.mwd . . . 223
plot.nvwp . . . 225
plot.wd . . . 226

plot.wp . . . 229
plot.wst . . . 232
plot.wst2D . . . 234
plotdenwd . . . 235
plotpkt . . . 237
print.BP . . . 238
print.imwd . . . 238
print.imwdc . . . 240
print.mwd . . . 241
print.nv . . . 243
print.nvwp . . . 244
print.w2d . . . 246
print.w2m . . . 247
print.wd . . . 248
print.wd3D . . . 249
print.wp . . . 250
print.wpst . . . 252
print.wpstCL . . . 253
print.wpstDO . . . 254
print.wpstRO . . . 256
print.wst . . . 257
print.wst2D . . . 258
PsiJ . . . 259
PsiJmat . . . 262
Psiname . . . 265
putC . . . 266
putC.mwd . . . 267
putC.wd . . . 269
putC.wp . . . 271
putC.wst . . . 272
putD . . . 274
putD.mwd . . . 275
putD.wd . . . 277
putD.wd3D . . . 279
putD.wp . . . 280
putD.wst . . . 282
putDwd3Dcheck . . . 283
putpacket . . . 284
putpacket.wp . . . 285
putpacket.wst . . . 287
putpacket.wst2D . . . 289
rcov . . . 291
rfft . . . 292
rfftinv . . . 293
rfftwt . . . 294
rm.det . . . 294
rmget . . . 295
rmname . . . 297

rotateback . . . 298
rsswav . . . 299
ScalingFunction . . . 301
Shannon.entropy . . . 302
simchirp . . . 303
ssq . . . 304
summary.imwd . . . 305
summary.imwdc . . . 306
summary.mwd . . . 307
summary.wd . . . 308
summary.wd3D . . . 309
summary.wp . . . 310
summary.wpst . . . 311
summary.wst . . . 312
summary.wst2D . . . 313
support . . . 314
sure . . . 315
teddy . . . 316
test.dataCT . . . 317
threshold . . . 318
threshold.imwd . . . 319
threshold.imwdc . . . 323
threshold.irregwd . . . 324
threshold.mwd . . . 326
threshold.wd . . . 329
threshold.wd3D . . . 334
threshold.wp . . . 337
threshold.wst . . . 340
TOgetthrda1 . . . 344
TOthreshda1 . . . 345
TOthreshda2 . . . 346
tpwd . . . 347
tpwr . . . 348
uncompress . . . 349
uncompress.default . . . 350
uncompress.imwdc . . . 351
wavegrow . . . 352
WaveletCV . . . 354
wd . . . 355
wd.dh . . . 360
wd.int . . . 361
wd.object . . . 362
wd3D . . . 364
wd3D.object . . . 365
Whistory . . . 366
Whistory.wst . . . 367
wp . . . 368
wp.object . . . 369

wpst . . . 370
wpst2discr . . . 372
wpst2m . . . 373
wpstCLASS . . . 374
wpstREGR . . . 375
wr . . . 376
wr.int . . . 377
wr.mwd . . . 378
wr.wd . . . 379
wr3D . . . 381
wst . . . 382
wst.object . . . 384
wst2D . . . 385
wst2D.object . . . 387
wstCV . . . 389
wstCVl . . . 391
WTEnv . . . 394
wvcvlrss . . . 395
wvmoments . . . 396
wvrelease . . . 397

Index **399**

`wavethresh-package` *Wavelets Statistics and Transforms*

Description

Performs 1, 2 and 3D real and complex-valued wavelet transforms, nondecimated transforms, wavelet packet transforms, nondecimated wavelet packet transforms, multiple wavelet transforms, complex-valued wavelet transforms, wavelet shrinkage for various kinds of data, locally stationary wavelet time series, nonstationary multiscale transfer function modeling, density estimation.

Details

Package:	wavethresh
Type:	Package
Title:	Wavelets Statistics and Transforms
Version:	4.7.3
Date:	2024-08-19
Authors@R:	c(person("Guy", "Nason", role=c("aut", "cre"), email="g.nason@imperial.ac.uk"), person("Stuart", "Barber", r
Depends:	R (>= 2.10), MASS
Description:	Performs 1, 2 and 3D real and complex-valued wavelet transforms, nondecimated transforms, wavelet packet tr
License:	GPL (>=2)
Author:	Guy Nason [aut, cre], Stuart Barber [ctb], Tim Downie [ctb], Piotr Frylewicz [ctb], Arne Kovac [ctb], Todd Og
Maintainer:	Guy Nason <g.nason@imperial.ac.uk>

Index of help topics:

AutoBasis	Run Coifman-Wickerhauser best basis algorithm on wavelet packet object
AvBasis	Basis averaging ("inversion")
AvBasis.wst	Perform basis averaging for (packet-ordered) non-decimated wavelet transform.
AvBasis.wst2D	Perform basis averaging for (packet-ordered) 2D non-decimated wavelet transform.
BAYES.THR	Bayesian wavelet thresholding.
BMdiscr	Subsidiary routine for makewpstDO function
BabyECG	Physiological data time series.
BabySS	Physiological data time series.
Best1DCols	Extract the best (one-dimensional) nondecimated WP packets
CWCV	C Wavelet Cross-validation
CWavDE	Simple wavelet density estimator with hard thresholding
CanUseMoreThanOneColor	Deprecated function
Chires5	Subsid routine for denproj (calcs scaling function coefs without cov)
Chires6	Subsid routine for denproj (calcs scaling function coefs with cov)
ConvertMessage	Print out a text message about an object which is from old version of WaveThresh
Crsswav	Wrapper to C code version of rsswav
Cthreshold	Calls C code to threshold wd class object.
DJ.EX	Produce Donoho and Johnstone test functions
FullWaveletCV	Perform whole wavelet cross-validation in C code
GenW	Generate (inverse) discrete wavelet transform matrix.
GetRSSWST	Computes estimate of error for function estimate.
HaarConcat	Generate a concatenated Haar MA process
HaarMA	Generate Haar MA processes.
InvBasis	Generic basis inversion for libraries
InvBasis.wp	Invert a wp library representation with a particular basis spec
InvBasis.wst	Invert a wst library representation with a basis specification
IsEarly	Generic function to detect whether object is from an early version
IsEarly.default	Detects whether object is from an earlier version of WaveThresh
IsEarly.wd	Function to detect whether a wd object is from WaveThresh2 or not
IsPowerOfTwo	Decides whether vector elements are integral

	powers of two (returns NA if not).
LSWsim	Simulate arbitrary locally stationary wavelet process.
LocalSpec	Compute Nason and Silverman smoothed wavelet periodogram.
LocalSpec.wd	Compute Nason and Silverman raw or smoothed wavelet periodogram.
LocalSpec.wst	Obsolete function (use ewspec)
MaNoVe	Make Node Vector (using Coifman-Wickerhauser best-basis type algorithm)
MaNoVe.wp	Make Node Vector (using Coifman-Wickerhauser best-basis type algorithm) on wavelet packet object
MaNoVe.wst	Make Node Vector (using Coifman-Wickerhauser best-basis type algorithm) on nondecimated wavelet transform object
PsiJ	Compute discrete autocorrelation wavelets.
PsiJmat	Compute discrete autocorrelation wavelets but return result in matrix form.
Psiname	Return a PsiJ list object style name.
ScalingFunction	Compute scaling functions on internally predefined grid
Shannon.entropy	Compute Shannon entropy
TOgetthrda1	Subsidiary routines for Ogden and Parzen's wavelet shrinkage methods
TOthreshda1	Data analytic wavelet thresholding routine
TOthreshda2	Data analytic wavelet thresholding routine
WTEnv	Environment that exists to store intermediate calculations for re-use within the same R session.
WaveletCV	Wavelet cross-validation
Whistory	Obsolete function supposedly detailed history of object
Whistory.wst	Obsolete function: as Whistory, but for wst objects
accessC	Get "detail" (mother wavelet) coefficients data from wavelet object
accessC.mwd	Get Smoothed Data from Wavelet Structure
accessC.wd	Get smoothed data from wavelet object (wd)
accessC.wp	Warning function when trying to access smooths from wavelet packet object (wp).
accessC.wst	Get smoothed data from packet ordered non-decimated wavelet object (wst)
accessD	Get "detail" (mother wavelet) coefficients data from wavelet object
accessD.mwd	Get wavelet coefficients from multiple wavelet structure (mwd).
accessD.wd	Get detail (mother wavelet) coefficients from

	wavelet object (wd).
accessD.wd3D	Get wavelet coefficients from 3D wavelet object
accessD.wp	Obtain whole resolution level of wavelet packet coefficients from a wavelet packet object (wp).
accessD.wpst	Get coefficients from a non-decimated wavelet packet object (wpst) in time order.
accessD.wst	Get mother wavelet coefficients from a packet ordered non-decimated wavelet object (wst).
accessc	Get variance information from irregularly spaced wavelet decomposition object.
addpkt	Add a wavelet packet box to an already set up time-frequency plot
av.basis	Perform basis averaging for wst class object
basisplot	Generic basis plot function
basisplot.BP	Plot time-frequency plane and basis slots associated with basis object
basisplot.wp	Function to graphically select a wavelet packet basis
bestm	Function called by makewpstRO to identify which packets are individually good for correlating with a response
c2to4	Take integer, represent in binary, then think of and return that representation in base 4
checkmyews	Check a LSW spectrum through repeated simulation and empirical averages
cns	Create new zeroed spectrum.
compare.filters	Compares two filters.
compgrot	Compute empirical shift for time ordered non-decimated transforms.
compress	Compress objects
compress.default	Do "zero" run-length encoding compression of a vector of numbers.
compress.imwd	Compress a (thresholded) imwd class object by removing zeroes.
conbar	Performs inverse DWT reconstruction step
convert	Convert one type of wavelet object into another.
convert.wd	Convert a non-decimated wd object into a wst object.
convert.wst	Convert a non-decimated wst object into a wd object.
cthresh	Estimate real signal using complex-valued wavelets
dclaw	Claw distribution
dencvwd	Calculate variances of wavlet coefficients of a p.d.f.
denplot	Calculate plotting information for a density estimate.

denproj	Calculate empirical scaling function coefficients of a p.d.f.
denwd	Wavelet decomposition of empirical scaling function coefficients of a p.d.f.
denwr	Wavelet reconstruction for density estimation.
dof	Compute number of non-zero coefficients in wd object
doppler	Evaluate the Donoho and Johnstone Doppler signal.
draw	Draw wavelets or scaling functions.
draw.default	Draw picture of a wavelet or scaling function.
draw.imwd	Draw mother wavelet associated with an imwd object.
draw.imwdc	Draw mother wavelet associated with an imwdc object.
draw.mwd	Draws a wavelet or scaling function used to compute an 'mwd' object
draw.wd	Draw mother wavelet or scaling function associated with wd object.
draw.wp	Draw wavelet packet associated with a wp object.
draw.wst	Draw mother wavelet or scaling function associated with wst object.
drawbox	Draw a shaded coloured box
drawwp.default	Subsidiary routine that actually computes wavelet packet values
ewspec	Compute evolutionary wavelet spectrum estimate.
example.1	Compute and return piecewise polynomial coordinates.
filter.select	Provide wavelet filter coefficients.
find.parameters	Find estimates of prior parameters
first.last	Build a first/last database for wavelet transforms.
first.last.dh	Build special first/last database for some wavelet density functions
firstdot	Return the location of the first period character within a character string (for a vector of strings of arbitrary length).
getarrvec	Compute and return weaving permutation for conversion from wst objects to wd class objects.
getpacket	Get a packet of coefficients from a wavelet object
getpacket.wp	Get packet of coefficients from a wavelet packet object (wp).
getpacket.wpst	Get packet of coefficients from a non-decimated wavelet packet object (wpst).
getpacket.wst	Get packet of coefficients from a packet

	ordered non-decimated wavelet object (wst).
getpacket.wst2D	Get packet of coefficients from a two-dimensional non-decimated wavelet object (wst2D).
griddata objects	Data interpolated to a grid objects.
guyrot	Cyclically rotate elements of a vector
image.wd	Produce image representation of nondecimated wavelet transform
image.wst	Produce image representation of a wst class object
imwd	Two-dimensional wavelet transform (decomposition).
imwd.object	Two-dimensional wavelet decomposition objects.
imwdc.object	Two-dimensional compressed wavelet decomposition objects.
imwr	Inverse two-dimensional wavelet transform.
imwr.imwd	Inverse two-dimensional discrete wavelet transform.
imwr.imwdc	Inverse two-dimensional discrete wavelet transform.
ipd	Inductance plethysmography data.
ipndacw	Compute inner product matrix of discrete non-decimated autocorrelation wavelets.
irregwd	Irregular wavelet transform (decomposition).
irregwd.objects	Irregular wavelet decomposition objects.
l2norm	Compute L2 distance between two vectors of numbers.
lennon	John Lennon image.
levarr	Subsidiary routine that generates a particular permutation
linfnorm	Compute L infinity distance between two vectors of numbers.
logabs	Take the logarithm of the squares of the argument
lt.to.name	Convert desired level and orientation into code used by imwd
madmad	Compute square of median absolute deviation (mad) function.
make.dwwt	Compute diagonal of the matrix WWT
makegrid	Interpolate data to a grid.
makewpstDO	Help page for a function
makewpstRO	Make a wavelet packet regression object from a dependent and independent time series variable.
mfilter.select	Provide filter coefficients for multiple wavelets.
mfirst.last	Build a first/last database for multiple wavelet transforms.
modernise	Generic function to upgrade a V2 WaveThresh

	object to V4
modernise.wd	Modernise a wd class object
mpostfilter	Multiwavelet postfilter
mprefilter	Multiwavelet prefilter
mwd	Discrete multiple wavelet transform (decomposition).
mwd.object	Multiple wavelet decomposition object (1D)
mwr	Multiple discrete wavelet transform (reconstruction).
newsure	Version of sure that acts as subsidiary for threshold.irregwd
nlevelsWT	Returns number of scale (resolution) levels.
nlevelsWT.default	Returns number of levels associated with an object
nullevels	Set whole resolution levels of coefficients equal to zero.
nullevels.imwd	Sets whole resolution levels of coefficients equal to zero in a imwd object.
nullevels.wd	Sets whole resolution levels of coefficients equal to zero in a wd object.
nullevels.wst	Sets whole resolution levels of coefficients equal to zero in a wst object.
numtonv	Convert an index number into a node vector object.
nv.object	Node vector objects.
plot.imwd	Draw a picture of the 2D wavelet coefficients using image
plot.irregwd	Plot variance factors of wavelet transform coefficients for irregularly spaced wavelet transform object
plot.mwd	Use plot on an mwd object.
plot.nvwp	Depict wavelet packet basis specfication
plot.wd	Plot wavelet transform coefficients.
plot.wp	Plot wavelet packet transform coefficients
plot.wst	Plot packet-ordered non-decimated wavelet transform coefficients.
plot.wst2D	Plot packet-ordered 2D non-decimated wavelet coefficients.
plotdenwd	Plot the wavelet coefficients of a p.d.f.
plotpkt	Sets up a high level plot ready to show the time-frequency plane and wavelet packet basis slots
print.BP	Print top best basis information for BP class object
print.imwd	Print out information about an imwd object in readable form.
print.imwdc	Print out information about an imwdc object in readable form.

print.mwd	Use print() on a mwd object.
print.nv	Print a node vector object, also used by several other functions to obtain packet list information
print.nvwp	Print a wavelet packet node vector object, also used by several other functions to obtain packet list information
print.w2d	Print method for printing w2d class objects
print.w2m	Print a w2m class object
print.wd	Print out information about an wd object in readable form.
print.wd3D	Print out information about an wd3D object in a readable form.
print.wp	Print out information about an wd object in readable form.
print.wpst	Prints out basic information about a wpst class object
print.wpstCL	Prints some information about a wpstCL object
print.wpstDO	Print information about a wpstDO class object
print.wpstRO	Print a wpstRO class object
print.wst	Print out information about an wst object in readable form.
print.wst2D	Print out information about an wst2d object in a readable form.
putC	Put smoothed data (father wavelet) coefficients into wavelet structure
putC.mwd	Put smoothed data into wavelet structure
putC.wd	Puts a whole resolution level of father wavelet coeffients into wd wavelet object.
putC.wp	Warning function when trying to insert father wavelet coefficients into wavelet packet object (wp).
putC.wst	Puts a whole resolution level of father wavelet coeffients into wst wavelet object.
putD	Put mother wavelet coefficients into wavelet structure
putD.mwd	Put wavelet coefficients into multiple wavelet structure
putD.wd	Puts a whole resolution level of mother wavelet coeffients into wd wavelet object.
putD.wd3D	Put wavelet coefficient array into a 3D wavelet object
putD.wp	Puts a whole resolution level of wavelet packet coeffients into wp wavelet object.
putD.wst	Puts a whole resolution level of mother wavelet coeffients into wst wavelet object.
putDwd3Dcheck	Check argument list for putD.wd3D
putpacket	Insert a packet of coefficients into a wavelet

	object.
putpacket.wp	Inserts a packet of coefficients into a wavelet packet object (wp).
putpacket.wst	Put a packet of coefficients into a packet ordered non-decimated wavelet object (wst).
putpacket.wst2D	Replace packet of coefficients in a two-dimensional non-decimated wavelet object (wst2D).
rcov	Computes robust estimate of covariance matrix
rfft	Real Fast Fourier transform
rfftinv	Inverse real FFT, inverse of rfft
rfftwt	Weight a Fourier series sequence by a set of weights
rm.det	Set coarse levels of a wavelets on the interval transform object to zero
rmget	Search for existing ipndacw matrices.
rmname	Return a ipndacw matrix style name.
rotateback	Cyclically shift a vector one place to the right
rsswav	Compute mean residual sum of squares for odd prediction of even ordinates and vice versa
simchirp	Compute and return simulated chirp function.
ssq	Compute sum of squares difference between two vectors
summary.imwd	Print out some basic information associated with an imwd object
summary.imwdc	Print out some basic information associated with an imwdc object
summary.mwd	Use summary() on a mwd object.
summary.wd	Print out some basic information associated with a wd object
summary.wd3D	Print out some basic information associated with a wd3D object
summary.wp	Print out some basic information associated with a wp object
summary.wpst	Print out some basic information associated with a wpst object
summary.wst	Print out some basic information associated with a wst object
summary.wst2D	Print out some basic information associated with a wst2D object
support	Returns support of compactly supported wavelets.
sure	Computes the minimum of the SURE thresholding function
teddy	Picture of a teddy bear's picnic.
test.dataCT	Test functions for wavelet regression and thresholding

threshold	Threshold coefficients
threshold.imwd	Threshold two-dimensional wavelet decomposition object
threshold.imwdc	Threshold two-dimensional compressed wavelet decomposition object
threshold.irregwd	hold irregularly spaced wavelet decomposition object
threshold.mwd	Use threshold on an mwd object.
threshold.wd	Threshold (DWT) wavelet decomposition object
threshold.wd3D	Threshold 3D DWT object
threshold.wp	Threshold wavelet packet decomposition object
threshold.wst	Threshold (NDWT) packet-ordered non-decimated wavelet decomposition object
tpwd	Tensor product 2D wavelet transform
tpwr	Inverse tensor product 2D wavelet transform.
uncompress	Uncompress objects
uncompress.default	Undo zero run-length encoding for a vector.
uncompress.imwdc	Uncompress an imwdc class object
wavegrow	Interactive graphical tool to grow a wavelet synthesis
wavethresh-package	Wavelets Statistics and Transforms
wd	Wavelet transform (decomposition).
wd.dh	Compute specialized wavelet transform for density estimation
wd.int	Computes "wavelets on the interval" transform
wd.object	Wavelet decomposition objects
wd3D	Three-dimensional discrete wavelet transform
wd3D.object	Three-dimensional wavelet object
wp	Wavelet packet transform.
wp.object	Wavelet Packet decomposition objects.
wpst	Non-decimated wavelet packet transform.
wpst2discr	Reshape/reformat packet coefficients into a multivariate data set
wpst2m	Converts a nondecimated wavelet packet object to a (large) matrix with packets stored as columns
wpstCLASS	Predict values using new time series values via a non-decimated wavelet packet discrimination object.
wpstREGR	Construct data frame using new time series using information from a previously constructed wpstRO object
wr	Wavelet reconstruction (inverse DWT).
wr.int	Computes inverse "wavelets on the interval" transform.
wr.mwd	Multiple wavelet reconstruction for mwd objects
wr.wd	Wavelet reconstruction for wd class objects (inverse discrete wavelet transform).

wr3D	Inverse DWT for 3D DWT object.
wst	Packet-ordered non-decimated wavelet transform.
wst.object	(Packet ordered) Nondecimated wavelet transform decomposition objects.
wst2D	(Packet-ordered) 2D non-decimated wavelet transform.
wst2D.object	(Packet ordered) Two-dimensional nondecimated wavelet transform decomposition objects.
wstCV	Performs two-fold cross-validation estimation using packet-ordered non-decimated wavelet transforms and one, global, threshold.
wstCVl	Performs two-fold cross-validation estimation using packet-ordered non-decimated wavelet transforms and a (vector) level-dependent threshold.
wvcvlrss	Computes estimate of error for function estimate.
wvmoments	Compute moments of wavelets or scaling function
wvrelease	Prints out the release number of the WaveThresh package

See book or individual help pages for main functions. For example, wd for the one-dimensional discrete wavelet transform.

Author(s)

Guy Nason [aut, cre], Stuart Barber [ctb], Tim Downie [ctb], Piotr Frylewicz [ctb], Arne Kovac [ctb], Todd Ogden [ctb], Bernard Silverman [ctb]

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

References

Nason, G.P. (2008) Wavelet methods in Statistics with R. Springer, New York. Book URL.

See Also

ewspec, imwd, threshold, wd, wst

Examples

```
#
# See examples in individual help pages
#
```

accessc *Get variance information from irregularly spaced wavelet decomposition object.*

Description

This function gets information from the c component of an irregwd.objects an irregularly spaced wavelet decomposition object.

Note that this function is *not* the same as accessC which obtains father wavelet coefficients from an wd class object.

Usage

```
accessc(irregwd.structure, level, boundary=FALSE)
```

Arguments

irregwd.structure
: Irregular wavelet decomposition object from which you wish to extract parts of the c component from.

level
: The level that you wish to extract. This value ranges from 0 to the nlevelsWT(irregwd.structure)-1.

boundary
: If this argument is T then all of the boundary correction values will be returned as well (note: the length of the returned vector may not be a power of 2). If boundary is false, then just the coefficients will be returned. If the decomposition (or reconstruction) was done with periodic boundary conditions then this option has no effect.

Details

The irregwd function produces a irregular wavelet decomposition (reconstruction) structure.

The c component is stored in a similar way to the C and D vectors which store the father and mother wavelet coefficients respectively. Hence to access the information the accessc function plays a similar role to accessC and accessD functions.

Value

A vector of the extracted data.

RELEASE

Version 3.9.4 Code Copyright Arne Kovac 1997. Help Copyright Guy Nason 2004.

Author(s)

G P Nason

See Also

irregwd, irregwd.objects, threshold.irregwd,makegrid, plot.irregwd

Examples

```
#
# Most users will not need to use this function. However, see the main
# examples for the irregular wavelet denoising in the examples for
# makegrid.
#
```

---

accessC *Get "detail" (mother wavelet) coefficients data from wavelet object*

Description

This generic function extracts detail from various types of wavelet objects. It extracts and returns a whole resolution level of coefficients. To obtain individual packets from relevant transforms use the getpacket() series of functions. This function is generic.

Particular methods exist. For objects of class:

wd use the accessC.wd method

wp use the accessC.wp method

wst use the accessC.wst method

See individual method help pages for operation and examples.

Usage

```
accessC(...)
```

Arguments

...	See individual help for details.

Value

A vector coefficients representing the detail coefficients for the requested resolution level.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

accessC.wd, accessC.wp,accessC.wst,accessD

---

accessC.mwd *Get Smoothed Data from Wavelet Structure*

Description

The smoothed and original data from a multiple wavelet decomposition structure, mwd.object ect (e.g. returned from mwd) are packed into a single matrix in that structure. TRUE his function extracts the data corresponding to a particular resolution level.

Usage

```
## S3 method for class 'mwd'
accessC(mwd, level = nlevelsWT(mwd), ...)
```

Arguments

mwd	Multiple wavelet decomposition structure from which you wish to extract the smoothed or original data if the structure is from a wavelet decomposition, or the reconstructed data if the structure is from a wavelet reconstruction.
level	The level that you wish to extract. By default, this is the level with most detail (in the case of structures from a decomposition this is the original data, in the case of structures from a reconstruction this is the top-level reconstruction).
...	any other arguments

Details

The mwd function produces a wavelet decomposition structure.

For decomposition, the top level contains the original data, and subsequent lower levels contain the successively smoothed data. So if there are mwd$filter$npsi*2^m original data points (mwd$filter$npsi is the multiplicity of wavelets), there will be m+1 levels indexed 0,1,...,m. So

accessC.mwd(Mwd, level=m)

pulls out the original data, as does

accessC.mwd(mwd)

To get hold of lower levels just specify the level that you're interested in, e.g.

accessC.mwd(mwd, level=2)

Gets hold of the second level.

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as a linear matrix of coefficients. AccessC obtains information about where the smoothed data appears from the fl.dbase component of mwd, in particular the array fl.dbase$first.last.c which gives a complete specification of index numbers and offsets for mwd$C.

Note also that this function only gets information from mwd class objects. To *put* coefficients into mwd structures you have to use the putC.mwd function.

See Downie and Silverman, 1998.

Value

A matrix with `mwd$filter$npsi` rows containing the extracted data of all the coefficients at that level.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6.)

Author(s)

G P Nason

See Also

`accessD.mwd`, `draw.mwd`, `mfirst.last`, `mfilter.select`, `mwd`, `mwd.object`, `mwr`, `plot.mwd`, `print.mwd`, `putC.mwd`, `putD.mwd`, `summary.mwd`, `threshold.mwd`, `wd`

Examples

```
#
# Get the 3rd level of smoothed data from a decomposition
#
dat <- rnorm(32)
accessC.mwd(mwd(dat), level=3)
```

`accessC.wd` *Get smoothed data from wavelet object (wd)*

Description

The smoothed and original data from a wavelet decomposition structure (returned from `wd`) are packed into a single vector in that structure. This function extracts the data corresponding to a particular resolution level.

Usage

```
## S3 method for class 'wd'
accessC(wd, level = nlevelsWT(wd), boundary=FALSE, aspect, ...)
```

Arguments

`wd`	wavelet decomposition structure from which you wish to extract the smoothed or original data if the structure is from a wavelet decomposition, or the reconstructed data if the structure is from a wavelet reconstruction.
`level`	the level that you wish to extract. By default, this is the level with most detail (in the case of structures from a decomposition this is the original data, in the case of structures from a reconstruction this is the top-level reconstruction).
`boundary`	logical; if `TRUE` then all of the boundary correction values will be returned as well (note: the length of the returned vector may not be a power of 2). If `boundary` is false, then just the coefficients will be returned. If the decomposition (or reconstruction) was done with periodic boundary conditions, this option has no effect.
`aspect`	Applies a function to the coefficients before return. Supplied as a text string which gets converted to a function. For example, "Mod" for complex-valued arguments
`...`	any other arguments

Details

The wd (`wr.wd`) function produces a wavelet decomposition (reconstruction) structure.

For decomposition, the top level contains the original data, and subsequent lower levels contain the successively smoothed data. So if there are 2^m original data points, there will be m+1 levels indexed 0,1,...,m. So

```
accessC.wd(wdobj, level=m)
```

pulls out the original data, as does

```
accessC.wd(wdobj)
```

To get hold of lower levels just specify the level that you're interested in, e.g.

```
accessC.wd(wdobj, level=2)
```

gets hold of the second level.

For reconstruction, the top level contains the ultimate step in the Mallat pyramid reconstruction algorithm, lower levels are intermediate steps.

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as a linear vector. AccessC obtains information about where the smoothed data appears from the fl.dbase component of an wd.object, in particular the array `fl.dbase$first.last.c` which gives a complete specification of index numbers and offsets for `wd.object$C`.

Note that this function is method for the generic function `accessC`. When the `wd.object` is definitely a wd class object then you only need use the generic version of this function.

Note that this function only gets information from `wd` class objects. To insert coefficients etc. into `wd` structures you have to use the `putC` function (or more precisely, the `putC.wd` method).

Value

A vector of the extracted data.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

References

Mallat, S. G. (1989) A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11**, 674–693.

Nason, G. P. and Silverman, B. W. (1994). The discrete wavelet transform in S. *Journal of Computational and Graphical Statistics,* **3**, 163–191.

See Also

wr, wd, accessD, accessD.wd, filter.select, threshold, putC.wd, putD.wd.

Examples

```
## Get the 3rd level of smoothed data from a decomposition
dat <- rnorm(64)
accessC(wd(dat), level=3)
```

---

accessC.wp — *Warning function when trying to access smooths from wavelet packet object (wp).*

Description

There are no real smooths to access in a wp wavelet packet object. This function returns an error message. To obtain coefficients from a wavelet packet object you should use the getpacket collection of functions.

Usage

```
## S3 method for class 'wp'
accessC(wp, ...)
```

Arguments

wp	Wavelet packet object.
...	any other arguments

Value

An error message!

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

getpacket

---

accessC.wst *Get smoothed data from packet ordered non-decimated wavelet object (wst)*

Description

The smoothed data from a packet ordered non-decimated wavelet object (returned from wst) are stored in a matrix. This function extracts all the coefficients corresponding to a particular resolution level.

Usage

```
## S3 method for class 'wst'
accessC(wst, level, aspect, ...)
```

Arguments

wst	Packet ordered non-decimated wavelet object from which you wish to extract the smoothed or original data (if the object is directly from a packet ordered non-decimated wavelet transform of some data).
level	The level that you wish to extract. This can range from zero (the coarsest coefficients) to nlevelsWT(wstobj) which returns the original data.
aspect	Applies function to coefficients before return. Supplied as a character string which gets converted to a function. For example "Mod" which returns the absolute values of the coefficients
...	Other arguments

Details

The wst function performs a packet-ordered non-decimated wavelet transform. This function extracts all the father wavelet coefficients at a particular resolution level specified by level.

Note that coefficients returned by this function are in *packet order*. They can be used *as is* but for many applications it might be more useful to deal with the coefficients in packets: see the function getpacket.wst for further details.

Value

A vector of the extracted data.

Author(s)

G P Nason

References

Nason, G. P. and Silverman, B. W. (1994). The discrete wavelet transform in S. *Journal of Computational and Graphical Statistics*, **3**, 163–191.

See Also

wst, wst.object, accessC, getpacket.wst

Examples

```
#
# Get the 3rd level of smoothed data from a decomposition
#
dat <- rnorm(64)
accessC(wst(dat), level=3)
```

---

accessD — *Get "detail" (mother wavelet) coefficients data from wavelet object*

Description

This generic function extracts detail from various types of wavelet objects. It extracts and returns a whole resolution level of coefficients. To obtain individual packets from relevant transforms use the getpacket() series of functions. This function is generic.

Particular methods exist. For objects of class:

wd use the accessD.wd method

wd3D use the accessD.wd3D method

wp use the accessD.wp method

wpst use the accessD.wpst method

wst use the accessD.wst method

See individual method help pages for operation and examples.

Usage

```
accessD(...)
```

Arguments

...	See individual help for details.

Value

A vector coefficients representing the detail coefficients for the requested resolution level.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

accessD.wd, accessD.wp,accessD.wst,accessC

---

accessD.mwd *Get wavelet coefficients from multiple wavelet structure (mwd).*

Description

The wavelet coefficients from a multiple wavelet decomposition structure, mwd.object, (e.g. returned from mwd) are packed into a single matrix in that structure. This function extracts the coefficients corresponding to a particular resolution level.

Usage

```
## S3 method for class 'mwd'
accessD(mwd, level, ...)
```

Arguments

mwd	Multiple wavelet decomposition structure from which you wish to extract the expansion coefficients.
level	The level that you wish to extract. If the "original" data has mwd$filter$npsi*2^m data points (mwd$filter$npsi being the multiplicity of the multiple wavelets) then there are m possible levels that you could want to access, indexed by 0,1,...,(m-1)
...	any other arguments

Details

The mwd function produces a multiple wavelet decomposition object .

The need for this function is a consequence of the pyramidal structure of Mallats algorithm and the memory efficiency gain achieved by storing the pyramid as a linear matrix. AccessD obtains information about where the coefficients appear from the fl.dbase component of mwd, in particular the array fl.dbase$first.last.d which gives a complete specification of index numbers and offsets for mwd$D.

Note that this function and accessC only work on objects of class mwd to *extract* coefficients. You have to use putD.mwd to insert wavelet coefficients into a mwd object.

See Downie and Silverman, 1998.

Value

A matrix with mwd$filter$npsi rows containing the extracted coefficients.

RELEASE

Tim Downie 1995-6

Author(s)

G P Nason

See Also

accessD.mwd, draw.mwd, mfirst.last, mfilter.select, mwd, mwd.object, plot.mwd, print.mwd, putC.mwd, putD.mwd, summary.mwd, threshold.mwd, wd, wr.mwd

Examples

```
#
# Get the 3rd level of smoothed data from a decomposition
#
data(ipd)
accessD.mwd(mwd(ipd), level=3)
```

---

accessD.wd *Get detail (mother wavelet) coefficients from wavelet object (wd).*

Description

This function extracts and returns a vector of mother wavelet coefficients, corresponding to a particular resolution level, from a wd wavelet decomposition object.

The pyramid of coefficients in a wavelet decomposition (returned from the wd function, say) are packed into a single vector in WaveThresh.

Usage

```
## S3 method for class 'wd'
accessD(wd, level, boundary=FALSE, aspect="Identity", ...)
```

Arguments

`wd`	Wavelet decomposition object from which you wish to extract the mother wavelet coefficients.
`level`	The resolution level at which you wish to extract coefficients.
`boundary`	some methods of wavelet transform computation handle the boundaries by keeping some extra bookkeeping coefficients at either end of a resolution level. If this argument is TRUE then these bookkeeping coefficients are returned when the mother wavelets are returned. Otherwise, if FALSE, these coefficients are not returned.
`aspect`	The aspect argument permits the user to supply a function to modify the returned coefficients. The function is applied to the vector of coefficients before it is returned. This can be useful, say, with the complex DWT where you could supply aspect="Mod" if you wanted to return the modulus of the coefficients at a given resolution level. The default argument, "Identity", ensures that the coefficients are not modified before returning.
`...`	any other arguments

Details

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as a linear vector. AccessD obtains information about where the smoothed data appears from the `fl.dbase` component of an `wd` object, in particular the array

`fl.dbase$first.last.d`

which gives a complete specification of index numbers and offsets for

`wd.object$D`.

Note that this function is a method for the generic function `accessD`.

Note also that this function only retrieves information from `wd` class objects. To insert coefficients into `wd` objects you have to use the `putD` function (or more precisely, the `putD.wd` method).

Value

A vector containing the mother wavelet coefficients at the required resolution level (the coefficients might have been modified depending on the value of the aspect argument).

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

References

Mallat, S. G. (1989) A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **11**, 674–693.

Nason, G. P. and Silverman, B. W. (1994). The discrete wavelet transform in S. *Journal of Computational and Graphical Statistics*, **3**, 163–191

See Also

wr, wd, accessD, filter.select, threshold

Examples

```
#
# Get the 4th resolution level of wavelet coefficients.
#
dat <- rnorm(128)
accessD(wd(dat), level=4)
```

---

accessD.wd3D *Get wavelet coefficients from 3D wavelet object*

Description

This function extracts and returns arrays of wavelet coefficients, corresponding to a particular resolution level, from a wd wavelet decomposition object.

The pyramid of coefficients in a wavelet decomposition (returned from the wd3D function, say) are packed into a single array in WaveThresh3.

Usage

```
## S3 method for class 'wd3D'
accessD(obj, level = nlevelsWT(obj)-1, block, ...)
```

Arguments

obj	3D Wavelet decomposition object from which you wish to extract the wavelet coefficients.
level	The resolution level at which you wish to extract coefficients. The minimum level you can enter is 0, the largest is one less than the number of nlevelsWT stored in the obj object.
block	if block is missing then a list containing all of the wavelet coefficient blocks GGG, GGH, GHG, GHH, HGG, HGH, HHG (and HHH, if level=0) is returned. Otherwise block should be one of the character strings GGG, GGH, GHG, GHH, HGG, HGH, HHG and then only that sub-block is returned from the resolution level specified.
...	any other arguments

Details

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as a array.

Note that this functiOn is a method for the generic function accessD.

Value

If the block is missing then a list is returned containing all the sub-blocks of coefficients for the specificed resolution level.

Otherwise the block character string specifies which sub-block of coefficients to return.

RELEASE

Version 3.9.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

link{accessD}, link{print.wd3D}, link{putD.wd3D}, link{putDwd3Dcheck}, link{summary.wd3D}, link{threshold.wd3D}, link{wd3D}, link{wd3D object}, link{wr3D}.

Examples

```
#
# Generate some test data
#
a <- array(rnorm(8*8*8), dim=c(8,8,8))
#
# Perform the 3D DWT
#
awd3D <- wd3D(a)
#
# How many levels does this object have?
#
nlevelsWT(awd3D)
# [1] 3
#
# So conceivably we could access levels 0, 1 or 2.
#
# Ok. Let's get the level 1 HGH sub-block coefficients:
#
accessD(awd3D, level=1, block="HGH")
#
#, , 1
#           [,1]      [,2]
#[1,]  0.8359289 1.3596832
#[2,] -0.1771688 0.2987303
```

```
#
#, , 2
#           [,1]       [,2]
#[1,] -1.2633313 1.00221652
#[2,] -0.3004413 0.04728019
#
# This was a 3D array of dimension size 2 (8 -> 4 -> 2, level 3, 2 and then 1)
#
#
# Let's do the same call except this time don't specify the block arg.
#
alllev1 <- accessD(awd3D, level=1)
#
# This new object should be a list containing all the subblocks at this level.
# What are the components?
#
names(alllev1)
#[1] "GHH" "HGH" "GGH" "HHG" "GHG" "HGG" "GGG"
#
# O.k. Let's look at HGH again
#
alllev1$HGH
#
#, , 1
#           [,1]      [,2]
#[1,]  0.8359289 1.3596832
#[2,] -0.1771688 0.2987303
#
#, , 2
#           [,1]       [,2]
#[1,] -1.2633313 1.00221652
#[2,] -0.3004413 0.04728019
#
# Same as before.
#
```

---

accessD.wp *Obtain whole resolution level of wavelet packet coefficients from a wavelet packet object (wp).*

Description

Get a whole resolution level's worth of coefficients from a wp wavelet packet object. To obtain packets of coefficients from a wavelet packet object you should use the getpacket collection of functions.

Usage

```
## S3 method for class 'wp'
accessD(wp, level, ...)
```

Arguments

wp	Wavelet packet object
.	
level	the resolution level that you wish to extract.
...	any other arguments

Details

The wavelet packet coefficients are actually stored in a straightforward manner in a matrix component of a wp object so it would not be too difficult to extract whole resolution levels yourself. However, this routine makes it easier to do.

Value

A vector containing the coefficients that you wanted to extract.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

accessD, getpacket

---

accessD.wpst — *Get coefficients from a non-decimated wavelet packet object (wpst) in time order.*

Description

The coefficients from a non-decimated wavelet packet object, wpst, are stored in a particular order in the wpst component of the wpstobj object. This function extracts all the coefficients corresponding to a particular wavelet packet in time order.

Usage

```
## S3 method for class 'wpst'
accessD(wpst, level, index, ...)
```

Arguments

`wpst`	Non-decimated wavelet packet object from which you wish to extract time-ordered coefficients.
`level`	The resolution level that you wish to extract. This can range from zero (the coarsest coefficients) to nlevelsWT-1(wstobj) which are the finest scale coefficients.
`index`	The wavelet packet index that you require (sequency ordering). This can range from 0 (father wavelet coeffcients) to `2^(nlevelsWT - level) - 1`, i.e. the maximum is dependent on the resolution level.
`...`	any other arguments

Details

The `wpst` function performs a non-decimated wavelet packet transform. This function extracts the coefficients at a particular resolution level specified by level in time order.

It is possible to extract the individual packets (before interweaving, i.e. the direct result of multiple applications of the packet operators) by using the `getpacket.wpst` function.

Author(s)

G P Nason

References

Nason, G.P., Sapatinas, T. and Sawczenko, A. Statistical modelling using undecimated wavelet transforms.

See Also

`wpst`, `wpst.object`, `accessD`, `getpacket.wpst`

Examples

```
#
# Get the 4th level of coefficients from a decomposition
#
dat <- rnorm(128)
accessD(wpst(dat), level=4, index=3)
```

`accessD.wst` *Get mother wavelet coefficients from a packet ordered non-decimated wavelet object (wst).*

Description

The mother wavelet coefficients from a packet ordered non-decimated wavelet object, `wst`, are stored in a matrix. This function extracts all the coefficients corresponding to a particular resolution level.

Usage

```
## S3 method for class 'wst'
accessD(wst, level, aspect = "Identity", ...)
```

Arguments

`wst`	Packet ordered non-decimated wavelet object from which you wish to extract the mother wavelet coefficients.
`level`	The level that you wish to extract. This can range from zero (the coarsest coefficients) to nlevelsWT(wstobj) which returns the original data.
`aspect`	Function to apply to coefficient before return. Supplied as a character argument which gets converted to a function. For example, "Mod" which returns the absolute value of complex-valued coefficients.
`...`	Other arguments

Details

The `wst` function performs a packet-ordered non-decimated wavelet transform. This function extracts all the mother wavelet coefficients at a particular resolution level specified by `level`.

Note that coefficients returned by this function are in *packet order*. They can be used *as is* but for many applications it might be more useful to deal with the coefficients in packets: see the function `getpacket.wst` for further details.

Note that all the coefficients here are those of mother wavelets. The non-decimated transform efficiently computes all possible shifts of the discrete wavelet transform computed by `wd`.

Value

A vector of the extracted coefficients.

Author(s)

G P Nason

References

Nason, G.P. and Silverman, B.W. The stationary wavelet transform and some statistical applications.

See Also

wst, wst.object, accessD, getpacket.wst

Examples

```
#
# Get the 4th level of mother wavelet coefficients from a decomposition
#
dat <- rnorm(128)
accessD(wst(dat), level=4)
```

---

addpkt — *Add a wavelet packet box to an already set up time-frequency plot*

Description

This function assumes that a high-level plot has already been set up using plotpkt. Given that this function plots a wavelet packet box at a given level, packet index and with particular shading and color and optionally plotting a sequence of coefficients at that location rather than a shaded box.

Usage

```
addpkt(level, index, density, col, yvals)
```

Arguments

level	The level at which the box or yvals are plotted
index	The packet index at which the box of yvals are plotted
density	The density of the shading of the box
col	The color of the box
yvals	If this argument is missing then a shaded coloured box is drawn, otherwise a time series of yvals is plotted where the box would have been.

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

basisplot,basisplot.BP, basisplot.wp, plotpkt, plot.nvwp

AutoBasis — *Run Coifman-Wickerhauser best basis algorithm on wavelet packet object*

Description

Runs the Coifman-Wickerhauser best basis algorithm on a wavelet packet object. Packets not in the basis are replaced by vectors of NAs. Superceded by the MaNoVe functions.

Details

Superceded by the MaNoVe functions (which run in C code).

Value

A wp class object which contains the select basis. All packets that are not in the basis get replaced by vectors of NAs.

Author(s)

G P Nason

See Also

MaNoVe

av.basis — *Perform basis averaging for wst class object*

Description

Note: that this function is not for direct user use. This function is a helper routine for the AvBasis.wst function which is the one that should be used by users.

This function works by recursion, essentially it merges the current levels C coefficients from one packet shift with its associated D coefficients, does the same for the other packet shift and then averages the two reconstructions to provide the C coefficients for the next level up.

Usage

```
av.basis(wst, level, ix1, ix2, filter)
```

Arguments

wst	The wst.object that you wish to basis average
level	The resolution level the function is currently operating at
ix1	Which "left" packet in the level you are accessing
ix2	Which "right" packet
filter	The wavelet filter details, see filter.select

Details

Description says all, see help page for AvBasis.wst.

Value

Returns the average basis reconstruction of a wst.object.

Author(s)

G P Nason

See Also

AvBasis, AvBasis.wst, conbar, rotateback, getpacket

AvBasis *Basis averaging ("inversion")*

Description

Average of whole collection of basis functions.

This function is generic.

Particular methods exist. For the wst class object this generic function uses AvBasis.wst. In the future we hope to add methods for wp and wpst class objects.

Usage

```
AvBasis(...)
```

Arguments

...	See individual help pages for details

Details

See individual method help pages for operation and examples.

Value

A vector containing the average of the representation over all bases.

RELEASE

Version 3.6.0 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

wst, wst.object, AvBasis.wst

AvBasis.wst *Perform basis averaging for (packet-ordered) non-decimated wavelet transform.*

Description

Perform basis averaging for (packet-ordered) non-decimated wavelet transform.

Usage

```
## S3 method for class 'wst'
AvBasis(wst, Ccode=TRUE, ...)
```

Arguments

wst	An object of class wst that contains coefficients of a packet ordered non-decimated wavelet transform (e.g. produced by the wst function.
Ccode	If TRUE then fast compiled C code is used to perform the transform. If FALSE then S code is used. Almost always use the default TRUE option. (It is conceivable that some implementation can not use the C code and so this option permits use of the slower S code).
...	any other arguments

Details

The packet-ordered non-decimated wavelet transform computed by wst computes the coefficients of an input vector with respect to a library of all shifts of wavelet basis functions at all scales. Here "all shifts" means all integral shifts with respect to the finest scale coefficients, and "all scales" means all dyadic scales from 0 (the coarsest) to J-1 (the finest) where 2^J = n where n is the number of data points of the input vector. As such the packet-ordered non-decimated wavelet transform contains a library of all possible shifted wavelet bases.

`Basis selection` It is possible to select a particular basis and invert that particular representation. In WaveThresh a basis is selected by creating a `nv` (node.vector) class object which identifies the basis. Then the function `InvBasis` takes the wavelet representation and the node.vector and inverts the representation with respect to the selected basis. The two functions `MaNoVe` and `numtonv` create a node.vector: the first by using a `Coifman-Wickerhauser` minimum entropy best-basis algorithm and the second by basis index.

Basis averaging. Rather than select a basis it is often useful to preserve information from all of the bases. For examples, in curve estimation, after thresholding a wavelet representation the coefficients are coefficients of an estimate of the truth with respect to all of the shifted basis functions. Rather than select *one* of them we can average over all estimates. This sometimes gives a better curve estimate and can, for examples, get rid of Gibbs effects. See Coifman and Donoho (1995) for more information on how to do curve estimation using the packet ordered non-decimated wavelet transform, thresholding and basis averaging.

Further it might seem that inverting each wavelet transform and averaging might be a computationally expensive operation: since each wavelet inversion costs order n operations and there are n different bases and so you might think that the overall order is n^2. It turns out that since many of the coarser scale basis functions are duplicated between bases there is redundancy in the non-decimated transform. Coifman and Donoho's TI-denoising algorithm makes use of this redundancy which results in an algorithm which only takes order $n \log n$ operations.

For an examples of denoising using the packet-ordered non-decimated wavelet transform and basis averaging see Johnstone and Silverman, 1997. The WaveThresh implementation of the basis averaging algorithm is to be found in Nason and Silverman, 1995

Value

A vector containing the average of the wavelet representation over all the basis functions. The length of the vector is `2^nlev` where `nlev` is the number of levels in the input `wst` object.

RELEASE

Version 3.6.0 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

`av.basis`, `wst`, `wst.object`, `MaNoVe`, `numtonv`, `InvBasis`, `wavegrow`

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
#
# Now take the packet-ordered non-decimated wavelet transform
#
```

```
tdwst <- wst(test.data)
#
# Now "invert" it using basis averaging
#
tdwstAB <- AvBasis(tdwst)
#
# Let's compare it to the original
#
sum( (tdwstAB - test.data)^2)
#
# [1] 9.819351e-17
#
# Very small. They're essentially same.
#
# See the threshold.wst help page for an
# an examples of using basis averaging in curve estimation.
```

`AvBasis.wst2D` *Perform basis averaging for (packet-ordered) 2D non-decimated wavelet transform.*

Description

Perform basis averaging for (packet-ordered) 2D non-decimated wavelet transform.

Usage

```
## S3 method for class 'wst2D'
AvBasis(wst2D, ...)
```

Arguments

`wst2D`	An object of class `wst2D` that contains coefficients of a packet ordered 2D non-decimated wavelet transform (e.g. produced by the `wst2D` function.
`...`	any other arguments

Details

The packet-ordered 2D non-decimated wavelet transform computed by `wst2D` computes the coefficients of an input matrix with respect to a library of all shifts of wavelet basis functions at all scales. Here "all shifts" means all integral shifts with respect to the finest scale coefficients with shifts in both the horizontal and vertical directions, and "all scales" means all dyadic scales from 0 (the coarsest) to J-1 (the finest) where `2^J = n` where `n` is the dimension of the input matrix. As such the packet-ordered 2D non-decimated wavelet transform contains a library of all possible shifted wavelet bases.

Basis averaging. Rather than select *a* basis it is often useful to preserve information from all of the bases. For examples, in curve estimation, after thresholding, the coefficients are coefficients of an estimate of the truth with respect to all of the shifted basis functions. Rather than select one of

them we can average over all estimates. This sometimes gives a better curve estimate and can, for examples, get rid of Gibbs effects. See Coifman and Donoho (1995) for more information on how to do curve estimation using the packet ordered non-decimated wavelet transform, thresholding and basis averaging. See Lang et al. (1995) for further details of surface/image estimation using the 2D non-decimated DWT.

Value

A square matrix of dimension $2^nlevelsWT$ containing the average-basis "reconstruction" of the `wst2D` object.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

`wst2D`, `wst2D.object`

Examples

```
#
# Generate some test data
#
#test.data <- matrix(rnorm(16), 4,4)
#
# Now take the 2D packet ordered DWT
#
#tdwst2D <- wst2D(test.data)
#
# Now "invert" it using basis averaging
#
#tdwstAB <- AvBasis(tdwst2D)
#
# Let's compare it to the original
#
#sum( (tdwstAB - test.data)^2)
#
# [1] 1.61215e-17
#
# Very small. They're essentially same.
#
```

BabyECG *Physiological data time series.*

Description

Two linked medical time series containing 2048 observations sampled every 16 seconds recorded from 21:17:59 to 06:27:18. Both these time series were recorded from the same 66 day old infant by Prof. Peter Fleming, Dr Andrew Sawczenko and Jeanine Young of the Institute of Child Health, Royal Hospital for Sick Children, Bristol. BabyECG, is a record of the infant's heart rate (in beats per minute). BabySS is a record of the infant's sleep state on a scale of 1 to 4 as determined by a trained expert monitoring EEG (brain) and EOG (eye-movement). The sleep state codes are 1=quiet sleep, 2=between quiet and active sleep, 3=active sleep, 4=awake.

Format

The BabyECG time series is a nice examples of a non-stationary time series whose spectral (time-scale) properties vary over time. The function ewspec can be used to anaylse this time series to inspect the variation in the power of the series over time and scales.

The BabySS time series is a useful independent time series that can be associated with changing power in the BabyECG series. See the discussion in Nason, von Sachs and Kroisandt.

RELEASE

Version 3.9 Copyright Guy Nason 1998

SEE ALSO

ewspec

Author(s)

G P Nason

Source

Institute of Child Health, Royal Hospital for Sick Children, Bristol.

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

Examples

```
data(BabyECG)
data(BabySS)
#
# Plot the BabyECG data with BabySS overlaid
#

# Note the following code does some clever scaling to get the two

# time series overlaid.

#

myhrs <- c(22, 23, 24, 25, 26, 27, 28, 29, 30)

mylab <- c("22", "23", "00", "01", "02", "03", "04", "05", "06")

initsecs <- 59 + 60 * (17 + 60 * 21)

mysecs <- (myhrs * 3600)

secsat <- (mysecs - initsecs)/16

mxy <- max(BabyECG)

mny <- min(BabyECG)

ro <- range(BabySS)

no <- ((mxy - mny) * (BabySS - ro[1]))/(ro[2] - ro[1]) + mny

rc <- 0:4

nc <- ((mxy - mny) * (rc - ro[1]))/(ro[2] - ro[1]) + mny

## Not run: plot(1:length(BabyECG), BabyECG, xaxt = "n", type = "l", xlab =
 "Time (hours)", ylab = "Heart rate (beats per minute)")
## End(Not run)

## Not run: lines(1:length(BabyECG), no, lty = 3)

## Not run: axis(1, at = secsat, labels = mylab)

## Not run: axis(4, at = nc, labels = as.character(rc))

#

# Sleep state is the right hand axis

#
#
```

BabySS *Physiological data time series.*

Description

Two linked medical time series containing 2048 observations sampled every 16 seconds recorded from 21:17:59 to 06:27:18. Both these time series were recorded from the same 66 day old infant by Prof. Peter Fleming, Dr Andrew Sawczenko and Jeanine Young of the Institute of Child Health, Royal Hospital for Sick Children, Bristol. BabyECG, is a record of the infant's heart rate (in beats per minute). BabySS is a record of the infant's sleep state on a scale of 1 to 4 as determined by a trained expert monitoring EEG (brain) and EOG (eye-movement). The sleep state codes are 1=quiet sleep, 2=between quiet and active sleep, 3=active sleep, 4=awake.

Format

The BabyECG time series is a nice examples of a non-stationary time series whose spectral (time-scale) properties vary over time. The function ewspec can be used to anaylse this time series to inspect the variation in the power of the series over time and scales.

The BabySS time series is a useful independent time series that can be associated with changing power in the BabyECG series. See the discussion in Nason, von Sachs and Kroisandt.

RELEASE

Version 3.9 Copyright Guy Nason 1998

SEE ALSO

ewspec

Author(s)

G P Nason

Source

Institute of Child Health, Royal Hospital for Sick Children, Bristol.

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

Examples

```
data(BabyECG)
data(BabySS)
#

# Plot the BabyECG data with BabySS overlaid

#

# Note the following code does some clever scaling to get the two

# time series overlaid.

#

myhrs <- c(22, 23, 24, 25, 26, 27, 28, 29, 30)

mylab <- c("22", "23", "00", "01", "02", "03", "04", "05", "06")

initsecs <- 59 + 60 * (17 + 60 * 21)

mysecs <- (myhrs * 3600)

secsat <- (mysecs - initsecs)/16

mxy <- max(BabyECG)

mny <- min(BabyECG)

ro <- range(BabySS)

no <- ((mxy - mny) * (BabySS - ro[1]))/(ro[2] - ro[1]) + mny

rc <- 0:4

nc <- ((mxy - mny) * (rc - ro[1]))/(ro[2] - ro[1]) + mny

## Not run: plot(1:length(BabyECG), BabyECG, xaxt = "n", type = "l", xlab =
 "Time (hours)", ylab = "Heart rate (beats per minute)")
## End(Not run)

## Not run: lines(1:length(BabyECG), no, lty = 3)

## Not run: axis(1, at = secsat, labels = mylab)

## Not run: axis(4, at = nc, labels = as.character(rc))

#

# Sleep state is the right hand axis

#
```

```
#
```

`basisplot` *Generic basis plot function*

Description

Plots a representation of a time-frequency plane and then plots the locations, and sometimes time series representations of coefficients, for the packets in the basis.

Usage

```
basisplot(x, ...)
```

Arguments

`x`	basis to plot
`...`	various arguments to methods

Details

Description says all

Value

Nothing, usually

Author(s)

G P Nason

See Also

`basisplot.BP`, `basisplot.wp`

basisplot.BP *Plot time-frequency plane and basis slots associated with basis object*

Description

The x objects store basis information obtained through the makewpstDO object. This function plots where the basis packets are on the time frequency plane.

Usage

```
## S3 method for class 'BP'
basisplot(x, num=min(10, length(BP$level)), ...)
```

Arguments

x	The BP class object, possibly coming from the BP component of the object returned by makewpstDO that you wish to plot
num	The number of packets that you wish to add to the plot
...	Other arguments

Details

Description says all

Value

Nothing of note

Author(s)

G P Nason

See Also

makewpstDO,Best1DCols

Examples

```
#
# See example in help for \code{\link{makewpstDO}}
#
```

basisplot.wp *Function to graphically select a wavelet packet basis*

Description

Note, one or two (depending on the state of `draw.mode`) graphics windows with mouse-clickable interfaces have to open to use this function.

Graphically select a wavelet packet basis associated with a wavelet packet object. Left-click selects packets, right click exits the routine.

Usage

```
## S3 method for class 'wp'
basisplot(x, draw.mode=FALSE, ...)
```

Arguments

`x`	The `wp.object` for which you wish to select a basis graphically for.
`draw.mode`	If TRUE then TWO graphics windows have to be open. Every time a packet is selected in the packet selection window, a representation of the wavelet packet basis function is drawn in the other window
`...`	Other arguments

Details

A wavelet packet basis described in WaveThresh using the node vector object (class from `MaNoVe.wp`) which for wavelet packets is `nvwp`. This function takes a `wp.object` object and graphically depicts all possible basis function locations. The user is then invited to click on different packets, these change colour. When finished, the user right clicks on the graphic and the selected basis is returned.

Note that the routine does not check to see whether the basis is legal. You have to do this. A legal basis can select packets from different levels, however you can't select packets that both cover the same packet index, however every packet index has to be covered.

A better function *would* check basis legality!

Value

An object of class `nvwp` which contains the specification for the basis.

Author(s)

G P Nason

See Also

`addpkt`, `InvBasis`, `MaNoVe.wp`, `plotpkt`, `wp`

BAYES.THR *Bayesian wavelet thresholding.*

Description

This function carries out Bayesian wavelet thresholding of noisy data, using the BayesThresh method of Abramovich, Sapatinas, & Silverman (1998).

Usage

```
BAYES.THR(data, alpha = 0.5, beta = 1, filter.number = 8, family =
"DaubLeAsymm", bc = "periodic", dev = var, j0 = 5, plotfn = FALSE)
```

Arguments

`data` A vector of length a power of two, containing noisy data to be thresholded.

`alpha, beta` Hyperparameters which determine the priors placed on the wavelet coefficients. Both alpha and beta take positive values; see Abramovich, Sapatinas, & Silverman (1998) or Chipman & Wolfson (1999) for more details on selecting `alpha` and `beta`.

`filter.number` This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.

For the "wavelets on the interval" (`bc="interval"`) transform the filter number ranges from 1 to 8. See the table of filter coefficients indexed after the reference to Cohen, Daubechies and Vial, (1993).

`family` Specifies the family of wavelets that you want to use. Two popular options are "DaubExPhase" and "DaubLeAsymm" but see the help for filter.select for more possibilities.

This argument is ignored for the "wavelets on the interval" transform (`bc="interval"`).

`bc` Specifies the boundary handling. If `bc="periodic"` the default, then the function you decompose is assumed to be periodic on it's interval of definition, if `bc="symmetric"` then the function beyond its boundaries is assumed to be a symmetric reflection of the function in the boundary. The symmetric option was the implicit default in releases prior to 2.2. If`bc=="interval"` then the "wavelets on the interval algorithm" due to Cohen, Daubechies and Vial is used. (The `WaveThresh` implementation of the "wavelets on the interval transform" was coded by Piotr Fryzlewicz, Department of Mathematics, Wroclaw University of Technology, Poland; this code was largely based on code written by Markus Monnerjahn, RHRK, Universitat Kaiserslautern; integration into `WaveThresh` by `GPN`).

`dev` This argument supplies the function to be used to compute the spread of the absolute values coefficients. The function supplied must return a value of spread on the variance scale (i.e. not standard deviation) such as the `var()` function. A popular, useful and robust alternative is the `madmad` function.

`j0`	The primary resolution level. While BayesThresh thresholds at all resolution levels, j0 is used in assessing the universal threshold which is used in the empirical Bayes estimation of hyperparameters.
`plotfn`	If TRUE, BAYES.THR draws the noisy data and the thresholded function estimate.

Details

A mixture prior consisting of a zero-mean normal distribution and a point mass at zero is placed on each wavelet coefficient. The empirical coefficients are then calculated and the priors updated to give posterior distributions for each coefficient. The thresholded value of each coefficient is the median of that coefficient's posterior distribution. See Abramovich, Sapatinas, & Silverman (1998) for more details of the procedure; the help page for `threshold.wd` has more information about wavelet thresholding in general.

The function `wave.band` uses the same priors to compute posterior credible intervals for the regression function, using the method described by Barber, Nason, & Silverman (2001).

Value

A vector containing the thresholded estimate of the function from which the data was drawn.

RELEASE

3.9.5 Code by Fanis Sapatinas/Felix Abramovich Documentation by Stuart Barber

Author(s)

G P Nason

See Also

`threshold.wd`, `wd`

Examples

```
#
# Generate some noisy test data and plot it.
#
blocks.data <- DJ.EX(n=512, noisy=TRUE)$blocks
#
# Now try BAYES.THR with the default parameters.
#
blocks.thr <- BAYES.THR(blocks.data, plotfn=TRUE)
#
# The default wavelet is Daubechies' least asymmetric wavelet
# with 8 vanishing moments; quite a smooth wavelet. Since the
# flat sections are still rather noisy, try Haar wavelets:
#
blocks.thr <- BAYES.THR(blocks.data, plotfn=TRUE, filter.number=1,
        family = "DaubExPhase")
```

```
#
# To show the importance of a sensible prior, consider alpha = 4,
# beta = 1 (which implies a smoother prior than the default).
#
blocks.thr <- BAYES.THR(blocks.data, plotfn=TRUE, filter.number=1,
        family = "DaubExPhase", alpha=4, beta=1)
#
# Here, the extreme values of the function are being smoothed towards zero.
#
```

`Best1DCols` *Extract the best (one-dimensional) nondecimated WP packets*

Description

This function takes the whole set of nondecimated wavelet packets and selects those packets that correlate best with the "response" groups. The idea is to reduce the large dimensionality (number of packets) into something more manageable which can then be fed into a proper discriminator.

Usage

```
Best1DCols(w2d, mincor=    0.69999999999999996)
```

Arguments

`w2d`	An object that gets returned from a call to the `wpst2discr` function which turns a wpst class object into a regular multivariate matrix
`mincor`	The threshold above which variables (packets) get included into the final mix if their correlation with the groups variable is higher than this value.

Details

This function is not intended for direct user use. In this function, the w2d object contains a matrix where each column contains the coefficients of a single packet from a non-decimated wavelet packet transform. The number of rows of the matrix is the same as the original time series and hence each column can be correlated with a separate groups variable that contains the group membership of a separate variable which changes over time. Those packet columns that have correlation greater than the `mincor` value are extracted and returned in the `BasisMatrix` item of the returned list.

Value

A list with the following components:

`nlevelsWT`	The number of levels of the nondecimated wavelet packet encapsulator, w2d
`BasisMatrix`	The highest correlating packets, sorted according to decreasing correlation
`level`	The levels corresponding to the selected packets
`pkt`	The packet indices corresponding to the selected packets
`basiscoef`	The sorted correlations
`groups`	The groups time series

Author(s)

G P Nason

See Also

makewpstDO,wpst2discr

---

bestm — *Function called by makewpstRO to identify which packets are individually good for correlating with a response*

Description

This function is used when you have a huge number of packets where you want to identify which ones are, individually, candidates for the good prediction of a response

Usage

```
bestm(w2mobj, y, percentage = 50)
```

Arguments

w2mobj	The w2m object that contains the packets you wish to preselect
y	The response time series
percentage	The percentage of the w2m packets that you wish to select

Details

This function naively addresses a very common problem. The object w2mobj contains a huge number of variables which might shed some light on the response object y. The problem is that the dimensionality of w2mobj is larger than that of the length of the series y.

The solution here is to choose a large, but not huge, subset of the variables that might be potentially useful in correlating with y, discard the rest, and return the "best" or preselected variables. Then the dimensionality is reduced and more sophisticated methods can be used to perform better quality modelling of the response y on the packets in w2mobj.

Value

A list of class w2m with the following components:

m	A matrix containing the select packets (as columns), reordered so that the best packets come first
ixvec	A vector which indexes the best packets into the original supplied matrix
pktix	The original wavelet packet indices corresponding to each packet
level	As pktix but for the wavelet packet levels
nlevelsWT	The number of resolution levels in the original wavelet packet object
cv	The ordered correlations

Author(s)

G P Nason

See Also

makewpstRO

BMdiscr *Subsidiary routine for makewpstDO function*

Description

Function actually performs discrimination on reduced variable set supplied to it from Best1DCols function.

Usage

```
BMdiscr(BP)
```

Arguments

BP	An list of the same format as returned by Best1DCols

Details

Not intended for direct user use

Value

Returns a list of objects: essentially the input argument BP and the return value from a call to the lda function which performs the discrimination operation.

Author(s)

G P Nason

See Also

Best1DCols,makewpstDO

`c2to4` *Take integer, represent in binary, then think of and return that representation in base 4*

Description

Not designed, or really useful, for casual user use!

For example: take the integer 5. In binary this is 101. Then, this representation in base 4 is 16+1 =17.

This function is used by `accessD.wpst` to help it access coefficients.

Usage

```
c2to4(index)
```

Arguments

`index`	The integer you wish to convert

Details

Description says all

Value

The converted number

Author(s)

G P Nason

See Also

`accessD.wpst`

Examples

```
c2to4(5)
```

CanUseMoreThanOneColor

Deprecated function

Description

Not used any more. This function used to interrogate the display device to see whether more than one color could be used. The function is set to return true whether of not the display device actually has this capability. It is used in the `plot.wp` function.

Usage

```
CanUseMoreThanOneColor()
```

Details

Description says it all.

Value

This function always returns TRUE

Author(s)

G P Nason

See Also

`plot.wp`

checkmyews

Check a LSW spectrum through repeated simulation and empirical averages

Description

Given a LSW spectrum this function simulates `nsim` realizations, estimates the spectrum, and then averages the results. The large sample averages should converge to the original spectrum.

Usage

```
checkmyews(spec, nsim=10)
```

Arguments

`spec`	The LSW spectrum
`nsim`	The number of realizations

Value

A LSW spectrum obtained as the average of `nsim` simulations from the `spec` spectrum.

Author(s)

G P Nason

See Also

`cns`,`LSWsim`, `ewspec`

`Chires5` *Subsid routine for denproj (calcs scaling function coefs without cov)*

Description

A subsidiary routine for `denproj`. Not intended for direct user use.

Usage

```
Chires5(x, tau=1, J, filter.number=10, family="DaubLeAsymm", nT=20)
```

Arguments

`x`	The data (random sample for density estimation)
`tau`	Fine tuning parameter
`J`	Resolution level
`filter.number`	The smoothness of the wavelet, see `filter.select`
`family`	The family of the wavelet, see `family`
`nT`	The number of iterations in the Daubechies-Lagarias algorithm

Details

As description

Value

A list with the following components:

`coef`	The scaling function coefficients
`klim`	The integer translates of the scaling functions used
`p`	The primary resolution, calculated in code as tau*2^J
`filter`	The usual filter information, see `filter.select`
`n`	The length of the data `x`
`res`	A list containing components: `p`, as above, `tau` as input and `J` as above. This summarizes the resolution information

Author(s)

David Herrick

See Also

Chires6,denproj

Chires6 *Subsid routine for denproj (calcs scaling function coefs with cov)*

Description

Function is essentially the same as Chires5 but also returns covariances between coefficients. A subsidiary routine for denproj. Not intended for direct user use.

Usage

```
Chires6(x, tau=1, J, filter.number=10, family="DaubLeAsymm", nT=20)
```

Arguments

x	The data (random sample for density estimation)
tau	Fine tuning parameter
J	Resolution level
filter.number	The smoothness of the wavelet, see filter.select
family	The family of the wavelet, see family
nT	The number of iterations in the Daubechies-Lagarias algorithm

Details

As description

Value

A list with the following components:

coef	The scaling function coefficients
covar	The coefficients' covariance matrix
klim	The integer translates of the scaling functions used
p	The primary resolution, calculated in code as tau*2^J
filter	The usual filter information, see filter.select
n	The length of the data x
res	A list containing components: p, as above, tau as input and J as above. This summarizes the resolution information

Author(s)

David Herrick

See Also

Chires6,denproj

---

cns *Create new zeroed spectrum.*

Description

Part of a two-stage function suite designed to simulate locally stationary wavelet processes in conjunction with the LSWsim function.

Usage

```
cns(n, filter.number=1, family="DaubExPhase")
```

Arguments

n	The length of the simulated process that you want to produce. Must be a power of two (for this software).
filter.number	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
family	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".

Details

This simple routine merely computes the time-ordered non-decimated wavelet transform of a zero vector of the same length as the eventual simulated series that you wish to produce.

If you look at this routine you will see that it is extremely simple. First, it checks to see whether the n that you supplied is a power of two. If it is then it creates a zero vector of that length. This is then non-decimated wavelet transformed with the appropriate wavelet.

The output can then be processed and then finally supplied to LSWsim for process simulation.

Value

An object of class: wd, and, in fact, of the non-decimated variety. All wavelet coefficients of this are zero.

Author(s)

G P Nason

See Also

LSWsim, ewspec

---

compare.filters *Compares two filters.*

Description

Compares two filters (such as those returned from filter.select). This function returns TRUE is they are the same otherwise returns FALSE.

Usage

```
compare.filters(f1,f2)
```

Arguments

f1	Filter, such as that returned by filter.select
f2	Filter, such as that returned by filter.select

Details

A very simple function. It only needs to check that the family and filter.number components of the filter are the same.

Value

If f1 and f2 are the same the function returns TRUE, otherwise it returns FALSE.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

filter.select.

Examples

```
#
# Create three filters!
#
filt1 <- filter.select(4, family="DaubExPhase")
filt2 <- filter.select(3, family="DaubExPhase")
filt3 <- filter.select(4, family="DaubLeAsymm")
#
# Now let us see if they are the same...
#
compare.filters(filt1, filt2)
# [1] FALSE
compare.filters(filt1, filt3)
# [1] FALSE
compare.filters(filt2, filt3)
# [1] FALSE
#
# Nope, (what a surprise) they weren't. How about
#
compare.filters(filt1, filt1)
# [1] TRUE
#
# Yes, they were the same!
```

`compgrot` *Compute empirical shift for time ordered non-decimated transforms.*

Description

Computes the empirical shift required for time-ordered non-decimated transform coefficients to bring them into time order.

Usage

```
compgrot(J, filter.number, family)
```

Arguments

`J`	The `number of levels` in the non-decimated transform where coefficients are to be time-aligned.
`filter.number`	The wavelet filter number to be used, see `filter.select`
`family`	The wavelet family, see `filter.select`

Details

Time-ordered non-decimated transform coefficients when raw are not in exact time alignment due to the phase of the underlying wavelet. This function returns the shifts that are necessary to apply to each resolution level in the transform to bring back each set of time-ordered coefficients into

time alignment. Note that the shifts returned are approximate shifts which work for any Daubechies wavelet. More accurate shifts can be computed using detailed knowledge of the particular wavelet used.

Each shift is "to the left". I.e. higher indexed coefficients should take the place of lower-indexed coefficients. Periodic boundaries are assumed.

This realignment is mentioned in Walden and Contreras Cristan, (1997) and Nason, Sapatinas and Sawczenko, (1998).

Value

A vector containing the shifts that need to be applied to each scale level to return them to the correct time alignment.

There are `J` entries in the vector. The first entry corresponds to the shift required for the finest level coefficients (i.e. level `J-1`) and the last entry corresponds to the coarsest level (i.e. level 0). Entry `j` corresponds to the shift required for scale level `J-j`.

RELEASE

Version 3.6 Copyright Guy Nason 1997

Note

GROT was the shop started by Reginald Perrin. Unfortunately, GROT stands for "Guy ROTation".

Author(s)

G P Nason

See Also

`wst`, `wst.object`, `wpst`, `wpst.object`.

Examples

```
#
# Let's see how the resolution levels have to be shifted
#
compgrot(4, filter.number=10, family="DaubExPhase")
#[1]  2  6 15 31
#
# In other words. Scale level 3 needs to be shifted two units.
# Scale level 2 needs to be shifted 6 units
# Scale level 1 needs to be shifted 15 units
# Scale level 0 needs to be shifted 31 units.
```

`compress` *Compress objects*

Description

Compress objects.

This function is generic.

Particular methods exist. For the `imwd` class object this generic function uses `compress.imwd`. There is a default compression method: `compress.default` that works on vectors.

Usage

```
compress(...)
```

Arguments

...	See individual help pages for details.

Details

See individual method help pages for operation and examples

Value

A compressed version of the input.

RELEASE

Version 2.0 Copyright Guy Nason 1993

Author(s)

G P Nason

See Also

`compress.default`, `compress.imwd`, `imwd`, `imwd.object`, `imwdc.object`, `threshold.imwd`

`compress.default` *Do "zero" run-length encoding compression of a vector of numbers.*

Description

Efficiently compress a vector containing many zeroes.

Usage

```
## Default S3 method:
compress(v, verbose=FALSE,...)
```

Arguments

`v`	The vector that you wish to compress. This compression function is efficient at compressing vectors with many zeroes, but is not a *general* compression routine.
`verbose`	If`TRUE` then this routine prints out the degree of compression achieved.
`...`	any other arguments

Details

Images are large objects. Thresholded 2d wavelet objects (`imwd`) are also large, but many of their elements are zero. compress.default takes a vector, decides whether compression is necessary and if it is makes an object of class `compressed` containing the nonzero elements and their position in the original vector.

The decision whether to compress the vector or not depends on two things, first the number of non-zero elements in the vector (r, say), and second the length of the vector (n, say). Since the position and value of the non-zero elements is stored we will need to store 2r values for the non-zero elements. So compression takes place if `2r < n`.

This function is the default method for the generic function `compress`. It can be invoked by calling compress for an object of the appropriate class, or directly by calling compress.default regardless of the class of the object.

Value

An object of class compressed if `compression` took place, otherwise a an object of class `uncompressed`.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Note

Sometimes the compressed object can be larger than the original. This usually only happens for small objects, so it doesn't really matter.

Author(s)

G P Nason

See Also

compress, imwd, threshold.imwd, uncompress

Examples

```
#
# Compress a vector with lots of zeroes
#
compress(c(rep(0,100),99))
#$position:
#[1] 101
#
#$values:
#[1] 99
#
#$original.length:
#[1] 101
#
#attr(, "class"):
#[1] "compressed"
#
# Try to compress a vector with not many zeroes
#
compress(1:10)
#$vector:
#[1]  1 2  3  4  5  6  7  8  9 10
#
#attr(, "class"):
#[1] "uncompressed"
#
#
```

---

compress.imwd — *Compress a (thresholded) imwd class object by removing zeroes.*

Description

Compress a (thresholded) imwd class object by removing zeroes.

Usage

```
## S3 method for class 'imwd'
compress(x, verbose=FALSE, ...)
```

Arguments

x	Object to compress. Compression only does anything on thresholded imwd.object.
verbose	If this is true then report on compression activity.
...	any other arguments

Details

Thresholded imwd objects are usually very large and contain many zero elements. This function compresses these objects into smaller imwd objects by using the compress.default function which removing the zeroes. This function is a method for the generic function compress for class imwd objects. It can be invoked by calling compress for an object of the appropriate class, or directly by calling compress.imwd regardless of the class of the object

Value

An object of type "imwdc" representing the compressed imwd object.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

compress, compress.default, imwd, imwd.object, imwdc.object, threshold.imwd.

Examples

```
#
# The user shouldn't need to use this function directly as the
# \code{\link{threshold.imwd}} function calls it
# automatically.
#
```

---

conbar *Performs inverse DWT reconstruction step*

Description

Wrapper to the C function conbar which is the main function in WaveThresh to do filter convolution/reconstruction with data. Although users use the wr function to perform a complete inverse discrete wavelet transform (DWT) this function repeatedly uses the conbar routine, once for each level to reconstruct the next finest level. The C conbar routine is possibly the most frequently utilized by WaveThresh.

Usage

```
conbar(c.in, d.in, filter)
```

Arguments

c.in	The father wavelet coefficients that you wish to reconstruct in this level's convolution.
d.in	The mother wavelet coefficients that you wish to reconstruct in this level's convolution.
filter	A given filter that you wish to use in the level reconstruction. This should be the output from the filter.select function.

Details

The wr function performs the inverse wavelet transform on an wd.object class object.

Internally, the wr function uses the C conbar function. Other functions also make use of conbar and some R functions also would benefit from using the fast C code of the conbar reconstruction hence this WaveThresh function. Some of the other functions that use conbar are listed in the SEE ALSO section. Many other functions call C code that then uses the C version of conbar.

Value

A vector containing the reconstructed coefficients.

Author(s)

G P Nason

See Also

av.basis InvBasis.wp wr

Examples

```
#
# Let's generate some test data, just some 32 normal variates.
#
v <- rnorm(32)
#
# Now take the wavelet transform with default filter arguments (which
# are filter.number=10, family="DaubLeAsymm")
#
vwd <- wd(v)
#
# Now, let's take an arbitrary level, say 2, and reconstruct level 3
# scaling function coefficients
#
c.in <- accessC(vwd, lev=2)
d.in <- accessD(vwd, lev=2)
#
```

```
conbar(c.in, d.in, filter.select(filter.number=10, family="DaubLeAsymm"))
#[1] -0.50368115  0.04738620 -0.90331807  1.08497622  0.90490528  0.06252717
#[7]  2.55894899 -1.26067508
#
# Ok, this was the pure reconstruction from using only level 2 information.
#
# Let's check this against the "original" level 3 coefficients (which get
# stored on the decomposition step in wd)
#
accessC(vwd, lev=3)
#[1] -0.50368115  0.04738620 -0.90331807  1.08497622  0.90490528  0.06252717
#[7]  2.55894899 -1.26067508
#
# Yep, the same numbers!
#
```

`convert` *Convert one type of wavelet object into another.*

Description

Convert one type of wavelet object into another.

This function is generic.

Particular methods exist:

`convert.wd` is used to convert non-decimated `wd` objects into `wst` objects. `convert.wst` is used to convert `wst` objects into non-decimated `wd` objects.

Usage

```
convert(...)
```

Arguments

`...`	See individual help pages for details.

Details

See individual method help pages for operation and examples.

Value

An object containing the converted representation.

RELEASE

Version 3.6.0 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

convert.wd, convert.wst, wst, wst.object, wst, wst.object.

---

convert.wd *Convert a non-decimated wd object into a wst object.*

Description

Convert a time-ordered non-decimated wavelet transform object into a packet-ordered non-decimated wavelet transform object.

Usage

```
## S3 method for class 'wd'
convert(wd, ...)
```

Arguments

wd	The wd class object that you wish to convert.
...	any other arguments

Details

In WaveThresh3 a non-decimated wavelet transform can be ordered in two different ways: as a time-ordered or packet-ordered representation. The coefficients in the two objects are *exactly the same* it is just their internal representation and ordering which is different. The two different representations are useful in different situations. The packet-ordering is useful for curve estimation applications and the time-ordering is useful for time series applications.

See Nason, Sapatinas and Sawczenko, 1998 for further details on ordering and weaving.

Note that the input object must be of the non-decimated type. In other words the type component of the input object must BE "station". Once the input object has been converted the output can be used with any of the functions suitable for the wst.object.

The getarrvec function actually computes the permutation to weave coefficients from one ordering to another.

Value

An object of class wst containing exactly the same information as the input object but ordered differently as a packet-ordered object.

RELEASE

Version 3.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

convert, getarrvec, levarr, wd, wd.object, wst, wst.object.

Examples

```
#
# Generate a sequence of 32 random normals (say) and take their
# \code{time-ordered non-decimated wavelet transform}
#
myrand <- wd(rnorm(32), type="station")
#
# Print out the result (to verify the class and type of the object)
#
#myrand
#Class 'wd' : Discrete Wavelet Transform Object:
#       ~~  : List with 8 components with names
#              C D nlevelsWT fl.dbase filter type bc date
#
#$ C and $ D are LONG coefficient vectors !
#
#Created on : Tue Sep 29 12:17:53 1998
#Type of decomposition:  station
#
#summary(.):
#----------
#Levels:  5
#Length of original:  32
#Filter was:  Daub cmpct on least asymm N=10
#Boundary handling:  periodic
#Transform type:  station
#Date:  Tue Sep 29 12:17:53 1998
#
# Yep, the myrand object is of class: \code{\link{wd.object}}.
#
# Now let's convert it to class \code{\link{wst}}. The object
# gets returned and, as usual in S, is printed.
#
convert(myrand)
#Class 'wst' : Stationary Wavelet Transform Object:
#       ~~~  : List with 5 components with names
#              wp Carray nlevelsWT filter date
#
#$wp and $Carray are the coefficient matrices
#
#Created on : Tue Sep 29 12:17:53 1998
#
#summary(.):
#----------
```

```
#Levels:  5
#Length of original:  32
#Filter was:  Daub cmpct on least asymm N=10
#Date:  Tue Sep 29 12:17:53 1998
#
# Yes. The returned object is of class \code{\link{wst.object}}.
# I.e. it has been converted successfully.
```

`convert.wst` *Convert a non-decimated wst object into a wd object.*

Description

Convert a packed-ordered non-decimated wavelet transform object into a time-ordered non-decimated wavelet transform object.

Usage

```
## S3 method for class 'wst'
convert(wst, ...)
```

Arguments

`wst`	The `wst` class object that you wish to convert.
`...`	any other arguments

Details

In WaveThresh3 a non-decimated wavelet transform can be ordered in two different ways: as a time-ordered or packet-ordered representation. The coefficients in the two objects are *exactly the same* it is just their internal representation and ordering which is different. The two different representations are useful in different situations. The packet-ordering is useful for curve estimation applications and the time-ordering is useful for time series applications.

See Nason, Sapatinas and Sawczenko, 1998 for further details on ordering and weaving.

Note that the input object must be of the non-decimated type. In other words the type component of the input object must be "`station`". Once the input object has been converted the output can be used with any of the functions suitable for the `wd.object`.

The actual weaving permutation for shuffling coefficients from one representation to another is achieved by the `getarrvec` function.

Value

An object of class `wd` containing exactly the same information as the input object but ordered differently as a packet-ordered object.

RELEASE

Version 3.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

convert, getarrvec, levarr, wd, wd.object, wst, wst.object.

Examples

```
#
# Generate a sequence of 32 random normals (say) and take their
# \code{packed-ordered non-decimated wavelet transform}
#
myrand <- wst(rnorm(32))
#
# Print out the result (to verify the class and type of the object)
#
#myrand
#Class 'wst' : Stationary Wavelet Transform Object:
#       ~~~  : List with 8 components with names
#              wp Carray nlevelsWT filter date
#
#$WP and $Carray are the coefficient matrices

#
#Created on : Tue Sep 29 12:29:45 1998
#
#summary(.):
#----------
#Levels:  5
#Length of original:  32
#Filter was:  Daub cmpct on least asymm N=10
#Boundary handling:  periodic

#Date:  Tue Sep 29 12:29:45 1998
#
# Yep, the myrand object is of class: \code{\link{wst.object}}.
#
# Now let's convert it to class \code{\link{wd}}. The object
# gets returned and, as usual in S, is printed.
#
convert(myrand)
#Class 'wd' : Discrete Wavelet Transform Object:
#       ~~  : List with 8 components with names
#              C D nlevelsWT fl.dbase filter type bc date
#
#$ C and $ D are LONG coefficient vectors !
#
#Created on : Tue Sep 29 12:29:45 1998
#Type of decomposition:  station
#
#summary(.):
```

```
#----------
#Levels:  5
#Length of original:  32
#Filter was:  Daub cmpct on least asymm N=10
#Boundary handling:  periodic
#Transform type:  station
#Date:  Tue Sep 29 12:29:45 1998
#
# The returned object is of class \code{\link{wd}} with a
# type of "station".
# I.e. it has been converted successfully.
```

`ConvertMessage` *Print out a text message about an object which is from old version of WaveThresh*

Description

Print out text message about an object being from an old version of WaveThresh.

Usage

```
ConvertMessage()
```

Arguments

None

Details

Description says all!

Value

None

Author(s)

G P Nason

See Also

`IsEarly.default`,`IsEarly`

Crsswav *Wrapper to C code version of rsswav*

Description

Crsswav is called by WaveletCV which is itself called by threshold.wd to carry out its cross-validation policy.

Usage

```
Crsswav(noisy, value = 1, filter.number = 10, family = "DaubLeAsymm",
    thresh.type = "hard", ll = 3)
```

Arguments

noisy	A vector of dyadic (power of two) length that contains the noisy data that you wish to compute the averaged RSS for.
value	The specified threshold.
filter.number	This selects the smoothness of wavelet that you want to perform wavelet shrinkage by cross-validation.
family	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
thresh.type	this option specifies the thresholding type which can be "hard" or "soft".
ll	The primary resolution that you wish to assume. No wavelet coefficients that are on coarser scales than ll will be thresholded.

Details

Description says all

Value

Same value as for rsswav

Author(s)

G P Nason

See Also

rsswav, WaveletCV

cthresh *Estimate real signal using complex-valued wavelets*

Description

Implements the multiwavelet style and empirical Bayes shrinkage procedures described in Barber & Nason (2004)

Usage

```
cthresh(data, j0 = 3, dwwt, dev = madmad, rule = "hard",
    filter.number = 3.1, family = "LinaMayrand", plotfn = FALSE,
    TI = FALSE, details = FALSE, policy = "mws", code = "NAG", tol = 0.01)
```

Arguments

`data`	The data to be analysed. This should be real-valued and of length a power of two.
`j0`	Primary resolution level; no thresholding is done below this level.
`dwwt`	description to come
`dev`	A function to be used to estimate the noise level of the data. The function supplied must return a value of spread on the variance scale (i.e. not standard deviation) such as the var() function. A popular, useful and robust alternative is the madmad function.
`rule`	The type of thresholding done. If policy = "mws", available rules are "hard" or "soft"; if policy = "ebayes", then rule can be "hard", "soft" or "mean".
`filter.number`, `family`	These parameters specify the wavelet used. See `filter.select` for details. Also, if filter.number = 5, estimation is done with all the complex-valued wavelets with 5 vanishing moments and the results averaged. If filter.number = 0, then he averaging is over all available complex-valued wavelets.
`plotfn`	If `plotfn = true`, then a plot of the noisy data and estimated signal are produced.
`TI`	If TI = T, then the non-decimated transform is used. See the help pages for wd and wst for more on the non-decimated transform.
`details`	If `details = FALSE` (the default), only the estimate of the underlying signal is returned. If `details = TRUE`, many other details are also returned.
`policy`	Controls the type of thresholding done. Available policies are multiwavelet style (policy = "mws") and empirical Bayes (policy = "ebayes").
`code`	Tells cthresh whether external C or NAG code is available to help with the calculations.
`tol`	A tolerance parameter used in searching for prior parameters if the empirical Bayes policy is used.

Details

If a real-valued signal is decomposed using a complex-valued wavelet (like the Lina-Mayrand wavelets supplied by filter.select), then the wavelet coefficients are also complex-valued. Wavelet shrinkage can still be used to estimate the signal, by asking the question "which coefficients are small (and represent noise) and which are large (and represent signal)?" Two methods of determining which coefficients are small and which are large are proposed by Barber & Nason (2004). One is "multiwavelet style" thresholding (similar to that in Downie & Silverman (1998) where the coefficients are treated like the coefficients of a multiwavelet). Here, the "size" of the wavelet coefficient is determined as modulus of a standardised version of the coefficient. The standardisation is by the square root of the covariance matrix of the coefficient. A Bayesian method is to place a mixture prior on each coefficient. The prior has two components: a bivariate normal and a point mass at (0,0). The parameters are determined by an empirical Bayes argument and then the prior is updated by the data.

Value

Either a vector containing the estimated signal (if details = FALSE), or a list with the following components:

`data`	The original data as supplied to cthresh.
`data.wd`	The wavelet decomposition of the data.
`thr.wd`	The thresholded version of data.wd.
`estimate`	The estimate of the underlying signal.
`Sigma`	The covariance matrices induced by the wavelet transform. See `make.dwwt` for more details.
`sigsq`	The estimate of the variance of the noise which corrupted the data.
`rule`	Which thresholding rule was used
`EBpars`	The empirical Bayes parameters found by the function find.parameters. Only present if the "ebayes" policy was used.
`wavelet`	A list with components filter.number and family which, when supplied to `link{filter.select}`, determine the wavelet used to decompose the data.

RELEASE

Part of the CThresh addon to WaveThresh. Copyright Stuart Barber and Guy Nason 2004.

Note

The estimates returned by cthresh have an imaginary component. In practice, this component is usually negligible.

Author(s)

Stuart Barber

See Also

filter.select, find.parameters, make.dwwt, test.dataCT, and the undocumented functions in CThresh.

Examples

```
#
# Make up some noisy data
#
y <- example.1()$y
ynoise <- y + rnorm(512, sd=0.1)
#
# Do complex-valued wavelet shrinkage with decimated wavelets
#
est1 <- cthresh(ynoise, TI=FALSE)
#
# Do complex-valued wavelet shrinkage with nondecimated wavelets
#
est2 <- cthresh(ynoise, TI=TRUE)
#
#
#
plot(1:512, y, lty=2, type="l")
lines(1:512, est1, col=2)
lines(1:512, est2, col=3)
```

Cthreshold *Calls C code to threshold wd class object.*

Description

A routine that calls a C code function to do thresholding. This is really a test routine to call a C thresholding function (Cthreshold) and the user is advised to use the R based generic thresholding function threshold and/or its methods as they contain a wider range of thresholding options.

Usage

```
Cthreshold(wd, thresh.type = "soft", value = 0, levels = 3:(nlevelsWT(wd) - 1))
```

Arguments

wd	The wavelet object that you wish to threshold.
thresh.type	The type of thresholding. This can be "soft" or "hard". See threshold and methods for further details.
value	The threshold value that you want to be used (e.g. for hard thresholding wavelet coefficients whose absolute value is less than
levels	The resolution levels that you wish to compute the threshold on and apply the threshold to.

Details

For general use it is recommended to use the threshold functions as they have a wider variety of options and also work for more complex varieties of wavelet transforms (i.e. non-decimated, complex-valued, etc). However, in the right, limited, situation this function can be useful. This function directly calls the C thresholding function Cthreshold(). The C function is used by routines that operate on behalf of the function that carries out two-fold cross validation in C (CWCV) which is also accessible using the policy="cv" option too threshold.wd

This function can be used by the user. It might be a bit faster than threshold.wd but mostly because it is simpler and does less checking than threshold.wd.

Value

A wd.object class object, but containing thresholded coefficients.

Author(s)

G P Nason

See Also

threshold

Examples

```
#
# See copious examples in the help to threshold.wd
#
```

CWavDE *Simple wavelet density estimator with hard thresholding*

Description

This function implements the density estimator with hard thresholding described by Hall, P. and Patil, P. (1995) Formulae for mean integrated squared error of nonlinear wavelet-based density estimators, *Ann. Statist.*, **23**, 905-928.

Usage

```
CWavDE(x, Jmax, threshold=0, nout=100, primary.resolution=1, filter.number=10,
 family="DaubLeAsymm", verbose=0, SF=NULL, WV=NULL)
```

Arguments

x	Vector of real numbers. This is the data for which you want a density estimate for
Jmax	The maximum resolution of wavelets
threshold	The hard threshold value for the wavelet coefficients
nout	The number of ordinates in the density estimate
primary.resolution	The usual wavelet density estimator primary resolution
filter.number	The wavelet filter number, see filter.select
family	The wavelet family, see filter.select
verbose	The level of reporting performed by the function, legit values are 0, 1 or 2, with 2 being more reports
SF	Scaling function values in format as returned by draw.default
WV	Wavelet function values in format as returned by draw.default

Details

As the description.

Value

A list containing the following components:

x	A vector of length nout that covers the range of the input data x, plus some more depending on the support of the wavelet and the primary resolution.
y	A vector of length nout that contains the output wavelet density estimate
sfix	The integer values of the translates of the scaling functions used in the estimate
wvixmin	As for sfix, but a vector of length Jmax which contains the minimum integer wavelet translates
wvixmax	As for wvixmin, but with the maxima

Author(s)

G P Nason

Examples

```
#
# Let's generate a bi-modal artificial set of data.
#
x <- c( rnorm(100), rnorm(100, 10))
#
# Now perform simple wavelet density estimate
#
wde <- CWavDE(x, Jmax=10, threshold=1)
#
```

```
# Plot results
#
## Not run: plot(wde$x, wde$y, type="l")
```

CWCV *C Wavelet Cross-validation*

Description

Two-fold wavelet shrinkage cross-validation (in C)

Usage

```
CWCV(ynoise, ll, x = 1:length(ynoise), filter.number = 10, family =
 "DaubLeAsymm", thresh.type = "soft", tol = 0.01,
 maxits=500, verbose = 0,
 plot.it = TRUE, interptype = "noise")
```

Arguments

`ynoise`	A vector of dyadic (power of two) length that contains the noisy data that you wish to apply wavelet shrinkage by cross-validation to.
`ll`	The primary resolution that you wish to assume. No wavelet coefficients that are on coarser scales than ll will be thresholded.
`x`	This function is capable of producing informative plots. It can be useful to supply the x values corresponding to the ynoise values. Further this argument is returned by this function which can be useful for later processors.
`filter.number`	This selects the smoothness of wavelet that you want to perform wavelet shrinkage by cross-validation.
`family`	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
`thresh.type`	this option specifies the thresholding type which can be "hard" or "soft".
`tol`	this specifies the convergence tolerance for the cross-validation optimization routine (a golden section search).
`maxits`	maximum number of iterations for the cross-validation optimization routine (a golden section search).
`verbose`	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default
`plot.it`	If this is TRUE then plots of the universal threshold (used to obtain an upper bound on the cross-validation threshold) reconstruction and the resulting cross-validation estimate are produced.
`interptype`	Can take two values noise or normal. This option controls how cross-validation compares the estimate formed by leaving out the data with the "left-out" data. If interptype="noise" then two noisy values are averaged to compare with the estimated curve in between, otherwise if interptype="normal" then the curve estimate is averaged either side of a noisy left-out point.

Details

Compute the two-fold cross-validated wavelet shrunk estimate given the noisy data ynoise according to the description given in Nason, 1996.

You must specify a primary resolution given by `ll`. This must be specified individually on each data set and can itself be estimated using cross-validation (although I haven't written the code to do this).

Note. The two-fold cross-validation method performs very badly if the input data is correlated. In this case I would advise using the methods proposed in Donoho and Johnstone, 1995 or Johnstone and Silverman, 1997 which can be carried out in WaveThresh using the `threshold` function using the `policy="sure"` option.

Value

A list with the following components

`x`	This is just the x that was input. It gets passed through more or less for convenience for the user.
`ynoise`	A copy of the input ynoise noisy data.
`xvwr`	The cross-validated wavelet shrunk estimate.
`yuvtwr`	The universal thresholded version (note this is merely a starting point for the cross-validation algorithm. It should not be ta ken seriously as an estimate. In particular its estimate of variance is likely to be inflated.)
`xvthresh`	The cross-validated threshold
`xvdof`	The number of non-zero coefficients in the cross-validated shrunk wavelet object (which is not returned).
`uvdof`	The number of non-zero coefficients in the universal threshold shrunk wavelet object (which also is not returned)
`xkeep`	always returns NULL!
`fkeep`	always returns NULL!

RELEASE

Version 3.0 Copyright Guy Nason 1994

Note

Plots of the universal and cross-validated shrunk estimates might be plotted if `plot.it=TRUE`.

Author(s)

G P Nason

See Also

`threshold`. `threshold.wd`.

Examples

```
#
# This function is best used via the policy="cv" option in
# the threshold.wd function.
# See examples there.
#
```

---

dclaw *Claw distribution*

Description

Random generation, density and cumulative probability for the claw distribution.

Usage

```
rclaw(n)
dclaw(x)
pclaw(q)
```

Arguments

n	Number of draws from rclaw distribution
x	Vector of ordinates
q	Vector of quantiles

Details

The claw distribution is a normal mixture distribution, introduced in Marron & Wand (1992). Marron, J.S. & Wand, M.P. (1992). Exact Mean Integrated Squared Error. *Ann. Stat.*, **20**, 712–736.

Value

Random samples (rclaw), density (dclaw) or probability (pclaw) of the claw distribution.

Author(s)

David Herrick

Examples

```
# Plot the claw density on the interval [-3,3]
x <- seq(from=-3, to=3, length=500)
## Not run: plot(x, dclaw(x), type="l")
```

dencvwd *Calculate variances of wavlet coefficients of a p.d.f.*

Description

Calculates the variances of the empirical wavelet coefficients by performing a 2D wavelet decomposition on the covariance matrix of the empirical scaling function coefficients of the probability density function.

Usage

```
dencvwd(hrproj, filter.number=hrproj$filter$filter.number,
    family=hrproj$filter$family, type="wavelet", bc="zero",
    firstk=hrproj$klim, RetFather=TRUE, verbose=FALSE)
```

Arguments

hrproj	Output from denproj with covar=T argument.
filter.number	The filter number of the wavelet basis to be used. This argument should not be altered from the default, as it is tied to the hrproj argument
family	The family of wavelets to use. This argument should not be altered.
type	The type of decomposition to be performed. This argument should not be altered.
bc	The type of boundary conditions to be used. For density estimation this should always be zero.
firstk	The bounds on the translation index of the empirical scaling function coefficients.
RetFather	Ignore this.
verbose	If TRUE the function will be chatty. Note that comments are only availble for part of the algorithm, so might not be very enlightening.

Details

This function is basically imwd adapted to handle zero boundary conditions, except that only the variances are returned, i.e. the diagonals of the covariance matrices produced. Note that this code is not very efficient. The full covariance matrices of all levels of coefficients are calculated, and then the diagonals are extracted.

Value

An object of class wd.object, but the contents are not a standard wavelet transform, ie the object is used to hold other information which organisationally is arranged like a wavelet tranform, ie variances of coefficients.

Author(s)

David Herrick

See Also

denproj,imwd

Examples

```
# Simulate data from the claw density, find the
# empirical scaling function coefficients and covariances and then decompose
# both to give wavelet coefficients and their variances.

data <- rclaw(100)
datahr <- denproj(data, J=8, filter.number=2,family="DaubExPhase", covar=TRUE)
data.wd <- denwd(datahr)
## Not run: plotdenwd(data.wd, top.level=(datahr$res$J-1))

datavar <- dencvwd(datahr)
## Not run: plotdenwd(datavar, top.level=(datahr$res$J-1))
```

---

denplot — *Calculate plotting information for a density estimate.*

Description

Calculates plotting information for a wavelet density estimate from high level scaling function coefficients.

Usage

```
denplot(wr, coef, nT=20, lims, n=50)
```

Arguments

wr	Scaling function coefficients, usually at some high level and usually smoothed (thresholded).
coef	The output from denproj for this analysis, i.e. the object containing the empirical scaling function coefficients. This is required because of the information it contains about the wavelet filter used, the resolution of the projection, and the bounds on the translation index of the scaling function coefficients.
lims	Vector containing the minimum and maximum x values required on the plot.
nT	The number of iterations to be performed in the Daubechies-Lagarias algorithm, which is used to evaluate the scaling functions of the specified wavelet basis at the plotting points.
n	The number of points at which the density estimate is to be evaluated.

Details

The density estimate is evaluated at `n` points between the values in `lims`. This function can be used to plot the empirical scaling function density estimate by entering `wr=coef$coef`, but since the empirical coefficients are usually found at some very high resolution, such a plot will be very noisy and not very informative. This function will be of much more use as and when thresholding function are included in this density estimation package.

Value

A list with components:

`x`	The points at which the density estimate is evaluated.
`y`	The values of the density estimate at the points in `x`.

Author(s)

David Herrick

See Also

`denproj`,`rclaw`

Examples

```
# Simulate data from the claw density and find the
# empirical scaling function coefficients at a lowish resolution and plot
# the resulting density estimate
data <- rclaw(100)
datahr <- denproj(data, J=3, filter.number=2,family="DaubExPhase")
datapl <- denplot(datahr$coef, datahr, lims=c(-3,3), n=1000)
## Not run: plot(datapl, type="l")
```

---

denproj — *Calculate empirical scaling function coefficients of a p.d.f.*

Description

Calculates empirical scaling function coefficients of the probability density function from sample of data from that density, usually at some "high" resoloution.

Usage

```
denproj(x, tau=1, J, filter.number=10, family="DaubLeAsymm", covar=FALSE, nT=20)
```

Arguments

`x`	Vector containing the data. This can be of any length.
`J`	The resolution level at which the empirical scaling function coefficients are to be calculated.
`tau`	This parameter allows non-dyadic resolutions to be used, since the resolution is specified as `tau * 2J`.
`filter.number`	The filter number of the wavelet basis to be used.
`family`	The family of wavelets to use, can be "DaubExPhase" or "DaubLeAsymm".
`covar`	Logical variable. If TRUE then covariances of the empirical scaling function coefficients are also calculated.
`nT`	The number of iterations to be performed in the Daubechies-Lagarias algorithm, which is used to evaluate the scaling functions of the specified wavelet basis at the data points.

Details

This projection of data onto a high resolution wavelet space is described in detail in Chapter 3 of Herrick (2000). The maximum and minimum values of `k` for which the empirical scaling function coefficient is non-zero are determined and the coefficients calculated for all k between these limits as `sum(phiJk(xi))/n`. The scaling functions are evaluated at the data points efficiently, using the Daubechies-Lagarias algorithm (Daubechies & Lagarias (1992)). Coded kindly by Brani Vidakovic.

Herrick, D.R.M. (2000) Wavelet Methods for Curve and Surface Estimation. PhD Thesis, University of Bristol.

Daubechies, I. & Lagarias, J.C. (1992). Two-Scale Difference Equations II. Local Regularity, Infinite Products of Matrices and Fractals. SIAM Journal on Mathematical Analysis, 24(4), 1031–1079.

Value

A list with components:

`coef`	A vector containing the empirical scaling function coefficients. This starts with the first non-zero coefficient, ends with the last non-zero coefficient and contains all coefficients, including zeros, in between.
`covar`	Matrix containing the covariances, if requested.
`klim`	The maximum and minimum values of k for which the empirical scaling function coefficients cJk are non-zero.
`p`	The primary resolution `tau * 2J`.
`filter`	A list containing the filter.number and family specified inthe function call.
`n`	The length of the data vector `x`.
`res`	A list containing the values of `p`, `tau` and `J`.

Author(s)

David Herrick

See Also

Chires5, Chires6, denwd, denwr

Examples

```
# Simulate data from the claw density and find the
# empirical scaling function coefficients
data <- rclaw(100)
datahr <- denproj(data, J=8, filter.number=4,family="DaubLeAsymm")
```

---

denwd *Wavelet decomposition of empirical scaling function coefficients of a p.d.f.*

Description

Performs wavelet decomposition on the empirical scaling function coefficients of the probability density function.

Usage

```
denwd(coef)
```

Arguments

coef	Output from denproj

Details

The empirical scaling function coefficients are decomposed using the DWT with zero boundary conditions.

Value

An object of class wd.object

Author(s)

David Herrick

See Also

denproj,plotdenwd,wd, denwr

Examples

```
# Simulate data from the claw density, find the empirical
# scaling function coefficients, decompose them and plot
# the resulting wavelet coefficients

data <- rclaw(100)
datahr <- denproj(data, J=8, filter.number=2,family="DaubExPhase")
data.wd <- denwd(datahr)
## Not run: plotdenwd(data.wd, top.level=(datahr$res$J-1))
```

---

denwr *Wavelet reconstruction for density estimation.*

Description

Performs wavelet reconstruction for density estimation.

Usage

```
denwr(wd, start.level=0, verbose=FALSE, bc=wd$bc,
    return.object=FALSE, filter.number=wd$filter$filter.number,
    family=wd$filter$family)
```

Arguments

wd	Wavelet decomposition object to reconstruct
start.level	The level you wish to start the reconstruction at. This is usually the first level (level 0). Note that this option assumes the coarsest level is labelled 0, so it is best to think of this argument as "the number of levels up from the coarsest level to start the reconstruction".
verbose	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.
bc	The boundary conditions used. These should be determined by those used to create the supplied wd.object object. In the case of density estimation they are "zero".
filter.number	The filter number of the wavelet used to do the reconstruction. Again, as for bc, you should probably leave this argument alone.
family	The type of wavelet used to do the reconstruction. You can change this argument from the default but it is probably NOT wise.
return.object	If this is FALSE then the top level of the reconstruction is returned (this is the reconstructed function at the highest resolution). Otherwise, if it is TRUE, the whole wd reconstructed object is returned.

Details

This is the same as wr.wd, except that it can handle zero boundary conditions.

Value

Either a vector containing the top level reconstruction or an object of class `wd.object` containing the results of the reconstruction.

Author(s)

David Herrick

`DJ.EX` *Produce Donoho and Johnstone test functions*

Description

Function to produce the blocks, bumps, Doppler and heavisine functions described by Donoho and Johnstone (1994).

Usage

```
DJ.EX(n=1024, signal=7, rsnr=7, noisy=FALSE, plotfn=FALSE)
```

Arguments

`n`	Number of samples of the function required.
`signal`	A factor that multiples the function values.
`rsnr`	If Gaussian noise is to be added to the functions then this argument specifies the root signal to noise ratio.
`noisy`	If TRUE then Gaussian noise is added to the signal so that the root signal to noise ratio is `rsnr`. If FALSE then just the signals are returned.
`plotfn`	If TRUE then a plot is produced. If FALSE no plot is produced.

Details

The Donoho and Johnstone test functions were designed to reproduce various features to be found in real world signals such as jump discontinuities (blocks), spikes (the NMR-like bumps), varying frequency behaviour (Doppler) and jumps/spikes in smooth signals (heavisine). These functions are most often used for testing wavelet shrinkage methods and comparing them to other nonparametric regression techniques. (Donoho, D.L. and Johnstone, I.M. (1994), Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, **81**, 425–455).

Another version of the Doppler function can be found in the standalone `doppler` function.

Another function for this purpose is the Piecewise Polynomial created in Nason and Silverman (1994) an encapsulated in WaveThresh by `example.1` (Nason, G.P. and Silverman, B.W. (1994) The discrete wavelet transform in S, *J. Comput. Graph. Statist.*, **3**, 163–191.

NOTE: This function might not give exactly the same function values as the equivalent function in WaveLab

Value

A list with four components: blocks, bumps, heavi and doppler containing the sampled signal values for the four types of Donoho and Johnstone test functions. Each of these are deemed to be sampled on an equally spaced grid from 0 to 1.

Author(s)

Theofanis Sapatinas

See Also

doppler,example.1, threshold, wd

Examples

```
#
# Show a picture of the four test functions with the default args
#
## Not run: DJ.EX(plotfn=TRUE)
```

---

dof — *Compute number of non-zero coefficients in wd object*

Description

Compute number of non-zero coefficients in wd object

Usage

```
dof(wd)
```

Arguments

wd	A wavelet decomposition object (such as that returned by the wd function).

Details

Very simple function that counts the number of non-zero coefficients in a wd class object.

Value

An integer that represents the number of non-zero coefficients in the input wd object.

RELEASE

Version 3.0 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wd, wd.object, threshold, threshold.wd.

Examples

```
#
# Let's generate some purely random numbers!!
#
myrandom <- rnorm(512)
#
# Take the discrete wavelet transform
#
myrandomWD <- wd(myrandom)
#
# How many coefficients are non-zero?
#
dof(myrandomWD)
# [1] 512
#
# All of them were nonzero!
#
# Threshold it
#
myrandomWDT <- threshold(myrandomWD, policy="universal")
#
# Now lets see how many are nonzero
#
dof(myrandomWDT)
# [1] 8
#
# Wow so 504 of the coefficients were set to zero! Spooky!
#
```

---

doppler — *Evaluate the Donoho and Johnstone Doppler signal.*

Description

This function evaluates and returns the Doppler signal from Donoho and Johnstone, (1994).

Usage

```
doppler(t)
```

Arguments

t	The domain of the Doppler function (where you wish to evaluate this Doppler function

Details

This function evaluates and returns the Doppler signal from Donoho and Johnstone, (1994). (Donoho, D.L. and Johnstone, I.M. (1994), Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, **81**, 425–455).

Another version of this function can be found in DJ.EX.

Value

A vector of the same length as the input vector containing the Doppler signal at t

Author(s)

Arne Kovac

See Also

DJ.EX

Examples

```
#
# Evalute the Doppler signal at 100 arbitrarily spaced points.
#
tt <- sort(runif(100))
dopp <- doppler(tt)
## Not run: plot(tt, dopp, type="l")
```

---

draw *Draw wavelets or scaling functions.*

Description

Draws the mother wavelet or scaling function associated with an object.

This function is generic.

Particular methods exist. The following functions are used for the following objects:

imwd.object the draw.imwd function is used.

imwdc.object the draw.imwdc function is used.

wd.object the draw.wd function is used.

wp.object the draw.wp function is used.

wst.object the draw.wst function is used.

All of the above method functions use the draw.default function which is the function which actually does the drawing.

Usage

```
draw(...)
```

Arguments

...	methods may have additional arguments

Details

See individual method help pages for operation and examples.

Value

If the `plot.it` argument is supplied then the draw functions tend to return the coordinates of what they were meant to draw and don't actually draw anything.

RELEASE

Version 2 Copyright Guy Nason 1993

Author(s)

G P Nason

See Also

`draw.default`, `draw.imwd`, `draw.imwdc`, `draw.wd`, `draw.wp`, `draw.wst`, `imwd.object`, `imwdc.object`, `wd.object`, `wp.object`, `wst.object`.

`draw.default` *Draw picture of a wavelet or scaling function.*

Description

This function can produce pictures of one- or two-dimensional wavelets or scaling functions at various levels of resolution.

Usage

```
## Default S3 method:
draw(filter.number = 10, family = "DaubLeAsymm", resolution = 8192,
    verbose = FALSE, plot.it = TRUE, main = "Wavelet Picture", sub = zwd$
    filter$name, xlab = "x", ylab = "psi", dimension = 1, twodplot
     = persp, enhance = TRUE, efactor = 0.05, scaling.function = FALSE,
 type="l",
    ...)
```

Arguments

`filter.number` This selects the index number of the wavelet or scaling function you want to draw (from within the wavelet family).

`family` specifies the family of wavelets that you want to draw. The options are "DaubExPhase" and "DaubLeAsymm".

`resolution` specifies the resolution that the wavelet or scaling function is computed to. It does not necessarily mean that you see all of these points as if the enhance option is TRUE then some function points are omitted.

`verbose` Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.

`plot.it` If TRUE then this function attempts to plot the function (i.e. draw it on a graphics device which should be active). If FALSE then this function returns the coordinates of the object that would have been plotted.

`main` a main title for the plot

`sub` a subtitle for the plot.

`xlab` a label string for the x-axis

`ylab` a label string for the y-axis

`dimension` whether to make a picture of the one-dimensional wavelet or the two-dimensional wavelet.

`twodplot` which function to use to produce the two-dimensional plot if dimension=2. The function you supply should accept data just like the contour or persp functions supplied with S-Plus.

`enhance` If this argument is TRUE then the plot is enhanced in the following way. Many of Daubechies' compactly supported wavelets are near to zero on a reasonable proportion of their support. So if such a wavelet is plotted quite a lot of it looks to be near zero and the interesting detail seems quite small. This function chooses a nice range on which to plot the central parts of the function and the function is plotted on this range.

`efactor` Variable which controls the range of plotting associated with the enhance option above. Any observations smaller than efactor times the range of the absolute function values are deemed to be too small. Then the largest range of "non-small" values is selected to be plotted.

`scaling.function` If this argument is TRUE the scaling function is plotted otherwise the mother wavelet is plotted.

`type` The `type` argument supplied to the plot command

`...` other arguments you can supply to the plot routine embedded within this function.

Details

The algorithm underlying this function produces a picture of the wavelet or scaling function by first building a `wavelet decomposition` object of the correct size (i.e. `correct resolution`) and setting all entries equal to zero. Then one coefficient at a carefully selected resolution level is set to one and the decomposition is inverted to obtain a picture of the wavelet.

Value

If `plot.it=FALSE` then usually a list containing coordinates of the object that *would* have been plotted is returned. This can be useful if you don't want S-Plus to do the plotting or you wish to use the coordinates of the wavelets for other purposes.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Note

A plot is produced of the wavelet or scaling function if `plot.it=TRUE`.

Author(s)

G P Nason

See Also

`filter.select`, `ScalingFunction`,`wd`, `wd.object`, `wr`, `wr.wd`.

Examples

```
#
# First make sure that your favourite graphics device is open
# otherwise S-Plus will complain.
#
# Let's draw a one-dimensional Daubechies least-asymmetric wavelet
# N=10
#
## Not run: draw.default(filter.number=10, family="DaubLeAsymm")
#
# Wow. What a great picture!
#
# Now how about a one-dimensional Daubechies extremal-phase scaling function
# with N=2
#
## Not run: draw.default(filter.number=2, family="DaubExPhase")
#
# Excellent! Now how about a two-dimensional Daubechies least-asymmetric
# N=6 wavelet
#
# N.b. we'll also reduce the resolution down a bit. If we used the default
# resolution of 8192 this would be probably too much for most computers.
#
## Not run: draw.default(filter.number=6, family="DaubLeAsymm", dimension=2, res=256)
#
# What a pretty picture!
```

`draw.imwd` *Draw mother wavelet associated with an imwd object.*

Description

This function draws the mother wavelet associated with an `imwd.object` — a two-dimensional wavelet decomposition.

Usage

```
## S3 method for class 'imwd'
draw(wd, resolution=128, ...)
```

Arguments

`wd`	The `imwd` class object whose associated wavelet you wish to draw.
`resolution`	The resolution at which the computation is done to compute the wavelet picture. Generally the resolution should be lower for two-dimensional wavelets since the number of computations is proportional to the square of the resolution (the DWT is still O(n) though).
`...`	Additional arguments to pass to the `draw.default` function which does the drawing.

Details

This function extracts the `filter` component from the `imwd` object (which is constructed using the `filter.select` function) to decide which wavelet to draw. Once decided the `draw.default` function is used to actually do the drawing.

Value

If the plot.it argument is set to `TRUE` then nothing is returned. Otherwise, as with `draw.default`, the coordinates of what would have been plotted are returned.

RELEASE

Version 2 Copyright Guy Nason 1993

Note

If the `plot.it` argument is`TRUE` (which it is by default) a plot of the mother wavelet or scaling function is plotted on the active graphics device.

Author(s)

G P Nason

See Also

filter.select, imwd.object, draw.default.

Examples

```
#
# Let's use the lennon test image
#
data(lennon)
## Not run: image(lennon)
#
# Now let's do the 2D discrete wavelet transform using Daubechies'
# least-asymmetric wavelet N=6
#
lwd <- imwd(lennon, filter.number=6)
#
# And now draw the wavelet that did this transform
#
## Not run: draw(lwd)
#
# A nice little two-dimensional wavelet!
#
```

---

draw.imwdc *Draw mother wavelet associated with an imwdc object.*

Description

This function draws the mother wavelet associated with an imwdc.object — a compressed two-dimensional wavelet decomposition.

Usage

```
## S3 method for class 'imwdc'
draw(wd, resolution=128, ...)
```

Arguments

wd	The imwd class object whose associated wavelet you wish to draw. (I know its called wd, sorry).
resolution	The resolution at which the computation is done to compute the wavelet picture. Generally the resolution should be lower for two-dimensional wavelets since the number of computations is proportional to the square of the resolution (the DWT is still O(n) though).
...	Additional arguments to pass to the draw.default function which does the drawing.

Details

This function extracts the `filter` component from the `imwd` object (which is constructed using the `filter.select` function) to decide which wavelet to draw. Once decided the `draw.default` function is used to actually do the drawing.

Value

If the `plot.it` argument is set to `TRUE` then nothing is returned. Otherwise, as with `draw.default`, the coordinates of what would have been plotted are returned.

RELEASE

Version 2 Copyright Guy Nason 1993

Note

If the `plot.it` argument is `TRUE` (which it is by default) a plot of the mother wavelet or scaling function is plotted on the active graphics device.

Author(s)

G P Nason

See Also

`filter.select`, `imwdc.object`, `draw.default`.

Examples

```
#
# Let's use the lennon test image
#
data(lennon)
## Not run: image(lennon)
#
# Now let's do the 2D discrete wavelet transform using Daubechies'
# least-asymmetric wavelet N=6
#
lwd <- imwd(lennon, filter.number=6)
#
# Now let's threshold the 2D DWT
# The resultant class of object is imwdc object.
#
lwdT <- threshold(lwd)
#
# And now draw the wavelet that did this transform
#
## Not run: draw(lwdT)
#
# A nice little two-dimensional wavelet!
#
```

draw.mwd *Draws a wavelet or scaling function used to compute an 'mwd' object*

Description

Draws picture of one wavelet or scaling function associated with the multiple wavelet decomposition object. mwd.object.

Usage

```
## S3 method for class 'mwd'
draw(mwd, phi = 0, psi = 0, return.funct = FALSE, ...)
```

Arguments

mwd	The mwd class object whose associated wavelet or scaling function you wish to draw.
phi	description not yet available
psi	If phi is non-zero then the 'phi'-th scaling function of the wavelet family used for mwd will be plotted. phi must be between 0 and mwd$filter$nphi.
return.funct	If true then the vector used as phi/psi in the plot command is returned.
...	Additional arguments to pass to the plot function

Details

It is usual to specify just one of phi and psi. IF neither phi nor psi are specified then phi=1 is the default. An error is generated if both phi=0 and psi=0 or if both are nonzero.

Value

If the return.funct argument is set to TRUE then the function values in the plot are returned otherwise NULL is returned.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6).

Note

If the return.funct argument is FALSE a plot of the mother wavelet or scaling function is plotted on the active graphics device.

Author(s)

G P Nason

See Also

accessC.mwd, accessD.mwd, mfirst.last, mfilter.select, mwd, mwd.object, mwr, plot.mwd, print.mwd, putC.mwd, putD.mwd, summary.mwd, threshold.mwd, wd, wr.mwd.

Examples

```
#
# Do a multiple wavelet decomposition on vector: ynoise
#
ynoise <- rnorm(512, sd = 0.1)
ymwd <- mwd(ynoise,filter.type="Geronimo")
#
# Draw a picture of the second Geronimo wavelet.
#
## Not run: draw(ymwd,psi=2)
#
#
```

---

draw.wd *Draw mother wavelet or scaling function associated with wd object.*

Description

This function draws the mother wavelet or scaling function associated with a wd.object.

Usage

```
## S3 method for class 'wd'
draw(wd, ...)
```

Arguments

wd	The wd class object whose associated wavelet or scaling function you wish to draw.
...	Additional arguments to pass to the draw.default function which does the drawing. In particular, arguments can be set to choose between drawing the mother wavelet and scaling function, to set the resolution of the plot, to choose between drawing one and two dimensional pictures.

Details

This function extracts the filter component from the wd object (which is constructed using the filter.select function) to decide which wavelet to draw. Once decided the draw.default function is used to actually do the drawing.

Value

If the plot.it argument is set to TRUE then nothing is returned. Otherwise, as with draw.default, the coordinates of what would have been plotted are returned.

RELEASE

Version 2 Copyright Guy Nason 1993

Note

If the `plot.it` argument is `TRUE` (which it is by default) a plot of the mother wavelet or scaling function is plotted on the active graphics device.

Author(s)

G P Nason

See Also

`filter.select`, `wd.object`, `draw.default`.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
# Now do the discrete wavelet transform of the data using the Daubechies
# least-asymmetric wavelet N=10 (the default arguments in
# wd).
#
tdwd <- wd(test.data)
#
# What happens if we try to draw this new tdwd object?
#
## Not run: draw(tdwd)
#
# We get a picture of the wavelet that did the transform
#
```

`draw.wp` *Draw wavelet packet associated with a wp object.*

Description

This function draws a wavelet packet associated with a `wp.object`.

Usage

```
## S3 method for class 'wp'
draw(wp, level, index, plot.it=TRUE, main, sub, xlab, ylab, ...)
```

Arguments

wp	The wp class object whose associated wavelet packet you wish to draw.
level	The resolution level of wavelet packet in the wavelet packet decomposition that you wish to draw (corresponds to scale).
index	The packet index of the wavelet packet in the wavelet packet decomposition that you wish to draw (corresponds to number of oscillations).
plot.it	If TRUE then the wavelet packet is plotted on the active graphics device. If FALSE then the y-coordinates of the packet are returned. Note that x-coordinates are not returned (the packet is periodic on its range anyway).
main	The main argument for the plot
sub	The subtitle for the plot
xlab	The labels for the x axis
ylab	The labels for the y axis
...	Additional arguments to pass to the drawwp.default function which does the drawing. In particular, arguments can be set to choose between drawing the mother wavelet and scaling function, to set the resolution of the plot, to choose between drawing one and two dimensional pictures.

Details

This function extracts the filter component from the wp object (which is constructed using the filter.select function) to decide which wavelet packet family to draw. Once decided the drawwp.default function is used to actually do the drawing.

Value

If the plot.it argument is set to TRUE then nothing is returned. Otherwise, if plot.it is set to FALSE the coordinates of what would have been plotted are returned.

RELEASE

Version 3.9.6 Copyright Guy Nason 1998

Note

If the plot.it argument is TRUE (which it is by default) a plot of the appropriate wavelet packet is plotted on the active graphics device.

Author(s)

G P Nason

See Also

filter.select, wp, wp.object, drawwp.default.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
# Now do the wavelet packet transform of the data using the Daubechies
# least-asymmetric wavelet N=10 (the default arguments in
# wp).
#
tdwp <- wp(test.data)
#
# What happens if we try to draw this new tdwp object?
#
## Not run: draw(tdwd, level=4, index=12)
```

`draw.wst` *Draw mother wavelet or scaling function associated with wst object.*

Description

This function draws the mother wavelet or scaling function associated with a `wst.object`.

Usage

```
## S3 method for class 'wst'
draw(wst, ...)
```

Arguments

`wst`	The `wst` class object whose associated wavelet or scaling function you wish to draw.
`...`	Additional arguments to pass to the `draw.default` function which does the drawing. In particular, arguments can be set to choose between drawing the mother wavelet and scaling function, to set the resolution of the plot, to choose between drawing one and two dimensional pictures.

Details

This function extracts the `filter` component from the `wst` object (which is constructed using the `filter.select` function) to decide which wavelet packet family to draw. Once decided the `draw.default` function is used to actually do the drawing.

Value

If the `plot.it` argument is set to `TRUE` then nothing is returned. Otherwise, Otherwise, as with `draw.default`, the coordinates of what would have been plotted are returned.

RELEASE

Version 3.6 Copyright Guy Nason 1995

Note

If the `plot.it` argument is `TRUE` (which it is by default) a plot of the appropriate wavelet packet is plotted on the active graphics device.

Author(s)

G P Nason

See Also

`filter.select`, `wst.object`, `draw.default`.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
# Now do the \code{packet-ordered non-decimated DWT}  of the data using the Daubechies
# least-asymmetric wavelet N=10 (the default arguments in \code{\link{wst}}).
#
tdwst <- wst(test.data)
#
# What happens if we try to draw this new tdwst object?
#
## Not run: draw(tdwst)
#
# We get a picture of the wavelet that did the transform
#
```

`drawbox` *Draw a shaded coloured box*

Description

Simply draws a box with bottom left corner at (x,y), or width w and height h with shading of density and colour of col.

Usage

```
drawbox(x,y,w,h,density,col)
```

Arguments

x	The bottom left x coordinate of the box
y	The bottom left y coordinate of the box
w	The width of the box
h	The height of the box
density	The shading density of the box
col	The colour of the box

Details

Description says all

Value

None

Author(s)

G P Nason

See Also

addpkt

---

drawwp.default *Subsidiary routine that actually computes wavelet packet values*

Description

Function computes the values of a given wavelet packet on a discrete grid.

Usage

```
drawwp.default(level, index, filter.number = 10, family = "DaubLeAsymm",
    resolution = 64 * 2^level)
```

Arguments

level	The resolution level of the packet you want
index	The packet index of the packet you want
filter.number	The type of wavelet you want, see filter.select
family	The family of wavelet you want, see filter.select
resolution	The number of ordinates at which you want the wavelet packet

Details

Function works by computing a wavelet packet transform of a zero vector. Then inserting a single one somewhere in the desired packet, and then inverts the transform.

Value

A vector containing the "y" values of the required wavelet packet.

Author(s)

G P Nason

See Also

draw.wp,InvBasis, nlevelsWT, putpacket, wp

---

ewspec — *Compute evolutionary wavelet spectrum estimate.*

Description

This function computes the evolutionary wavelet spectrum (EWS) estimate from a time series (or non-decimated wavelet transform of a time series). The estimate is computed by taking the non-decimated wavelet transform of the time series data, taking its modulus; smoothing using TI-wavelet shrinkage and then correction for the redundancy caused by use of the non-decimated wavelet transform. Options below beginning with smooth. are passed directly to the TI-wavelet shrinkage routines.

Usage

```
ewspec(x, filter.number = 10, family = "DaubLeAsymm",
    UseLocalSpec = TRUE, DoSWT = TRUE, WPsmooth = TRUE,
    verbose = FALSE, smooth.filter.number = 10,
    smooth.family = "DaubLeAsymm",
    smooth.levels = 3:(nlevelsWT(WPwst) - 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	The time series that you want to analyze. (See DETAILS below on how to supply preprocessed versions of the time series which bypass early parts of the ewspec function).

`filter.number` This selects the index of the wavelet used in the analysis of the time series (i.e. the wavelet basis functions used to model the time series). For Daubechies compactly supported wavelets the filter number is the number of vanishing moments.

`family` This selects the wavelet family to use in the analysis of the time series (i.e. which wavelet family to use to model the time series). Only use the Daubechies compactly supported wavelets `DaubExPhase` and `DaubLeAsymm`.

`UseLocalSpec` If you input a time series for `x` then this argument should always be `T`. (However, you can precompute the modulus of the non-decimated wavelet transform yourself and supply it as `x` in which case the `LocalSpec` call within this function is not necessary and you can set UseLocalSpec equal to `F`).

`DoSWT` If you input a time series for `x` then this argument should always be `T`. (However, you can precompute the non-decimated wavelet transform yourself and supply it as `x` in which case the `wd` call within the function will not be necessary and you can set DoSWT equal to `F`).

`WPsmooth` Normally a wavelet periodogram is smoothed before it is corrected. Use `WPsmooth=F` is you do not want any wavelet periodogram smoothing (correction is still done).

`verbose` If this option is `T` then informative messages are printed as the function progresses.

`smooth.filter.number`
This selects the index number of the wavelet that smooths each scale of the wavelet periodogram. See `filter.select` for further details on which wavelets you can use. Generally speaking it is a good idea to use a smoother wavelet for smoothing than the one you used for analysis (above) but since one still wants local smoothing it is best not to use a wavelet that is much smoother.

`smooth.family` This selects the wavelet family that smooths each scale of the wavelet periodogram. See `filter.select` for further details on which wavelets you can use. There is no need to use the same family as you used to analyse the time series.

`smooth.levels` The levels to smooth when performing the TI-wavelet shrinkage smoothing.

`smooth.dev` The method for estimating the variance of the empirical wavelet coefficients for smoothing purposes.

`smooth.policy` The recipe for smoothing: determines how the threshold is chosen. See `threshold` for TI-smoothing and choice of potential policies. For EWS estimation `LSuniversal` is recommended for thi Chi-squared nature of the periodogram coefficients. However, if the coefficients are transformed (using `smooth.transform` and `smooth.inverse`) then other, more standard, policies may be appropriate.

`smooth.value` When a manual policy is being used this argument is used to supply a threshold value. See `threshold` for more information.

`smooth.by.level`
If `TRUE` then the wavelet shrinkage is performed by computing and applying a separate threshold to each level in the non-decimated wavelet transform of each scale. Note that each scale in the EWS is smoothed separately and independently: and each smooth consists of taking the (second-stage) non-decimated wavelet transform and applying a threshold to each level of a wavelet transformed scale.

If FALSE then the same threshold is applied to the non-decimated wavelet transform of a scale. Different thresholds may be computed for different scales (in the time series model) but the threshold will be the same for each level arising from the non-decimated transform of a scale.

Note: a scale refers to a set of coefficients coming from a particular scale of the non-decimated wavelet transform of the time series data that models the time series. A level refers to the levels of wavelet coefficients obtained from taking the non-decimated wavelet transform of a particular scale.

smooth.type — The type of shrinkage: either "hard" or "soft".

smooth.verbose — If T then informative messages concerning the TI-transform wavelet shrinkage are printed.

smooth.cvtol — If cross-validated wavelet shrinkage (smooth.policy="cv") is used then this argument supplies the cross-validation tolerance.

smooth.cvnorm — no description for object

smooth.transform — The transform function to use to transform the wavelet periodogram estimate. The wavelet periodogram coefficients are typically chi-squared in nature, a log transform can pull the coefficients towards normality so that a smooth.policy for Gaussian data could be used (e.g. universal).

smooth.inverse — the inverse transform of smooth.transform.

Details

This function computes an estimate of the evolutionary wavelet spectrum of a time series according to the paper by Nason, von Sachs and Kroisandt. The function works as follows:

1. The non-decimated wavelet transform of the series is computed.
2. The squared modulus of the non-decimated wavelet transform is computed (this is the raw wavelet periodogram, which is returned).
3. The squared modulus is smoothed using TI-wavelet shrinkage.
4. The smoothed coefficients are corrected using the inverse of the inner product matrix of the discrete non-decimated autocorrelation wavelets (produced using the ipndacw function).

To display the EWS use the plotfunction on the S component, see the examples below.

It is possible to supply the non-decimated wavelet transform of the time series and set DoSWT=F or to supply the squared modulus of the non-decimated wavelet transform using LocalSpec and setting UseLocalSpec=F. This facility saves time because the function is then only used for smoothing and correction.

Value

A list with the following components:

S — The evolutionary wavelet spectral estimate of the input x. This object is of class wd and so can be plotted, printed in the usual way.

WavPer	The raw wavelet periodogram of the input x. The EWS estimate (above) is the smoothed corrected version of the wavelet periodgram. The wavelet periodogram is of class wd and so can be plotted, printed in the usual way.
rm	This is the matrix A from the paper by Nason, von Sachs and Kroisandt. Its inverse is used to correct the raw wavelet periodogram. This matrix is computed using the ipndacw function.
irm	The inverse of the matrix A from the paper by Nason, von Sachs and Kroisandt. It is used to correct the raw wavelet periodogram.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

See Also

Baby Data, filter.select, ipndacw, LocalSpec, threshold wd wd.object

Examples

```
#
# Apply the EWS estimate function to the baby data
#
```

---

example.1 *Compute and return piecewise polynomial coordinates.*

Description

This function computes and returns the coordinates of the piecewise polynomial described by Nason and Silverman, 1994. This function is a useful test function for evaluating wavelet shrinkage methodology as it contains smooth parts, a discontinuity and it is periodic.

(Nason, G.P. and Silverman, B.W. (1994) The discrete wavelet transform in S, *J. Comput. Graph. Statist.*, **3**, 163–191.)

Usage

```
example.1()
```

Arguments

None

Details

This function computes and returns the x and y coordinates of the piecewise polynomial function described in Nason and Silverman, 1994. The formula for the piecewise polynomial (which is piecewise cubic) is given in Nason and Silverman, 1994.

The piecewise polynomial returned is a discrete sample on 512 equally spaced points between 0 and 1 (including 0 but excluding 1).

The Donoho and Johnstone test functions can be generated using the DJ.EX function.

Value

A list with two components:

x	a vector of length 512 containing the ordered x ordinates of the piecewise polynomial.
y	a vector of length 512 containing the corresponding y ordinates of the piecewise polynomial.

Author(s)

G P Nason

See Also

DJ.EX

Examples

```
#
# Generate the piecewise polynomial
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
```

---

filter.select *Provide wavelet filter coefficients.*

Description

This function stores the filter coefficients necessary for doing a discrete wavelet transform (and its inverse), including complex-valued compactly supported wavelets.

Usage

```
filter.select(filter.number, family="DaubLeAsymm", constant=1)
```

Arguments

`filter.number` This selects the desired filter, an integer that takes a value dependent upon the family that you select. For the complex-valued wavelets in the Lina-Mayrand family, the filter number takes the form x.y where x is the number of vanishing moments (3, 4, or 5) and y is the solution number (1 for x = 3 or 4 vanishing moments; 1, 2, 3, or 4 for x = 5 vanishing moments). Note: this argument has a different meaning for Littlewood-Paley wavelets, see the note below in the Details section.

`family` This selects the basic family that the wavelet comes from. The choices are **DaubExPhase** for Daubechies' extremal phase wavelets, **DaubLeAsymm** for Daubechies' "least-asymmetric" wavelets, **Coiflets** for Coiflets, **Lawton** for Lawton's complex-valued wavelets (equivalent to Lina-Mayrand 3.1 wavelets), **LittlewoodPaley** for a approximation to Littlewood-Paley wavelets, or **LinaMayrand** for the Lina-Mayrand family of complex-valued Daubechies' wavelets.

`constant` This constant is applied as a multiplier to all the coefficients. It can be a vector, and so you can adapt the filter coefficients to be whatever you want. (This is feature of negative utility, or "there is less to this than meets the eye" as my old PhD supervisor would say [GPN]).

Details

This function contains at least three types of filter. Two types can be selected with family set to DaubExPhase, these wavelets are the Haar wavelet (selected by filter.number=1 within this family) and Daubechies "extremal phase" wavelets selected by filter.numbers ranging from 2 to 10). Setting family to DaubLeAsymm gives you Daubechies least asymmetric wavelets, but here the filter number ranges from 4 to 10. For Daubechies wavelets, filter.number corresponds to the N of that paper, the wavelets become more regular as the filter.number increases, but they are all of compact support.

With family equal to "Coiflets" the function supports filter numbers ranging from 1 to 5. Coiflets are wavelets where the scaling function also has vanishing moments.

With family equal to "LinaMayrand", the function returns complex-valued Daubechies wavelets. For odd numbers of vanishing moments, there are symmetric complex-valued wavelets i this family, and for five or more vanishing moments there are multiple distinct complex-valued wavelets, distinguished by their (arbitrary) solution number. At present, Lina-Mayrand wavelets 3.1, 4.1, 5.1, 5.2, 5.3, and 5.4 are available in WaveThresh.

Setting family equal to "Lawton" chooses complex-valued wavelets. The only wavelet available is the one with "filter.number" equal to 3.

With family equal to "LittlewoodPaley" the Littlewood-Paley wavelet is used. The scaling function is also the same as (or at least proportional to, depending on your normalization) that of the Shannon scaling function, so its an approximation to the Shannon wavelet transform. The "filter.number" argument has a special meaning for the Littlewood-Paley wavelets: it does not represent vanishing moments here. Instead, it controls the number of filter taps in the quadrature mirror filter: typically longer values are better, up to the length of the series. Increasing it higher than the length of the series does not usually have much effect. Note: extreme caution should be taken with the Littlewood-Paley wavelet. This implementation is pure time-domain and as such can only be thought of as an approximation to a complete Shannon/LP implementation. For example, in actuality the wavelets

are NOT finite impluse response filters like with Daubechies wavelets. This means that it is possible for an infinite number of Littlewood Paley wavelet coefficients to be nonzero. However, computers can not store an infinite number of coefficients and some will be lost. This is most noticeable with functions with discontinuities and other homogeneities but it can also happen with some smooth functions. A way to check how "bad" is can be is to transform your desired function followed immediately by the inverse transform and compare the original with the resultant sequence.

The function `compare.filters` can be used to compare two filters.

Value

Alist is returned with four components describing the filter:

`H`	A vector containing the filter coefficients.
`G`	A vector containing filter coefficients (if Lawton or Lina-Mayrand wavelets are selected, otherwise this is NULL).
`name`	A character string containing the name of the filter.
`family`	A character string containing the family of the filter.
`filter.number`	The filter number used to select the filter from within a family.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994, This version originally part of the cthresh release which was merged into wavethresh in Oct 2012. Original cthresh version due to Stuart Barber

Note

The (Daubechies) filter coefficients should always sum to sqrt(2). This is a useful check on their validity.

Author(s)

Stuart Barber and G P Nason

See Also

`wd`, `wr`, `wr.wd`, `accessC`, `accessD`, `compare.filters`, `imwd`, `imwr`, `threshold`, `draw`.

Examples

```
#This function is usually called by others.
#However, on occasion you may wish to look at the coefficients themselves.

#
# look at the filter coefficients for N=4 (by default Daubechies'
# least-asymmetric wavelets.)
#
filter.select(4)
#$H:
#[1] -0.07576571 -0.02963553  0.49761867  0.80373875  0.29785780
```

```
#[6] -0.09921954 -0.01260397  0.03222310
#
#$G:
#NULL
#
#$name:
#[1] "Daub cmpct on least asymm N=4"
#
#$family:
#[1] "DaubLeAsymm"
#
#$filter.number:
#[1] 4
```

`find.parameters` *Find estimates of prior parameters*

Description

Estimate the prior parameters for the complex empirical Bayes shrinkage procedure.

Usage

```
find.parameters(data.wd, dwwt, j0, code, tol, Sigma)
```

Arguments

`data.wd`	Wavelet decomposition of the data being analysed.
`dwwt`	The diagonal elements of the matrix Wt(W). See `make.dwwt` for details.
`j0`	Primary resolution level, as discussed in the help for threshold.wd
`code`	Tells the function whether to use NAG code for the search (code="NAG"), R/S-plus for the search with C code to evaluate the likelihood (code="C"), or R/S-plus code for all calculations (code="R" or code="S"). Setting code="NAG" is strongly recommended.
`tol`	A tolerance parameter which bounds the mixing weight away from zero and one and the correlation between real and imaginary parts of the prior away from plus or minus one.
`Sigma`	The covariance matrix of the wavelet coefficients of white noise.

Details

The complex empirical Bayes (CEB) shrinkage procedure described by Barber & Nason (2004) places independent mixture priors on each complex-valued wavelet coefficient. This routine finds marginal maximum likelihood estimates of the prior parameters. If the NAG library is available, routine E04JYF is used otherwise the search is done using optimize (in R) or nlminb (in S-plus). In the latter case, the likelihood values should be computed externally using the C code supplied as part of the CThresh package - although a pure R / S-plus version is available, it is very slow. This function will not usually be called directly by the user, but is called from within cthresh.

Value

A list with the following components:

pars	Estimates of the prior parameters. Each row of this matrix contains the following parameter estimates for one level of the transform: mixing weight; variance of the real part of the wavelet coefficients; covariance between the real and imaginary parts; variance of the imaginary part of the wavelet coefficients. Note that for levels below the primary resolution, this search is not done and the matrix is full of zeros.
Sigma	The covariance matrix as supplied to the function.

RELEASE

Part of the CThresh addon to WaveThresh. Copyright Stuart Barber and Guy Nason 2004.

Note

There may be warning messages from the NAG routine E04JYF. If the indicator variable IFAIL is equal to 5, 6, 7, or 8, then a solution has been found but there is doubt over the convergence. For IFAIL = 5, it is likely that the correct solution has been found, while IFAIL = 8 means that you should have little confidence in the parameter estimates. For more details, see the NAG software documentation available online at `http://www.nag.co.uk/numeric/fl/manual19/pdf/E04/e04jyf_fl19.pdf`

Author(s)

Stuart Barber

See Also

`cthresh`

---

first.last — *Build a first/last database for wavelet transforms.*

Description

This function is not intended for user use, but is used by various functions involved in computing and displaying wavelet transforms. It basically constructs "bookeeping" vectors that `WaveThresh` uses for working out where coefficient vectors begin and end.

Usage

```
first.last(LengthH, DataLength, type, bc="periodic", current.scale=0)
```

Arguments

`LengthH`	Length of the filter used to produce a wavelet decomposition.
`DataLength`	Length of the data before transforming. This must be a power of 2, say 2^m.
`type`	The type of wavelet transform. Can be "wavelet" or "periodic"
`bc`	This character string argument determines how the boundaries of the the function are to be handled. The permitted values are `periodic` or `symmetric`.
`current.scale`	Can handle a different initial scale, but usually left at the default

Details

Suppose you begin with $2^m = 2048$ coefficients. At the next level you would expect 1024 smoothed data coefficients, and 1024 wavelet coefficients, and if `bc="periodic"` this is indeed what happens. However, if `bc="symmetric"` you actually need more than 1024 (as the wavelets extend over the edges). The first last database keeps track of where all these "extras" appear and also where they are located in the packed vectors C and D of pyramidal coefficients within wavelet structures.

For examples, given a `first.last.c` `row` of

$$-2320$$

The actual coefficients would be

$$c_{-2}, c_{-1}, c_0, c_1, c_2, c_3$$

In other words, there are 6 coefficients, starting at -2 and ending at 3, and the first of these (c_{-2}) appears at an offset of 20 from the beginning of the `$C` component vector of the wavelet structure.

You can "do" `first.last` in your head for `periodic` boundary handling but for more general boundary treatments (e.g. `symmetric`) `first.last` is indispensable.

Value

A first/last database structure, a list containing the following information:

`first.last.c`	A (m+1)x3 matrix. The first column specifies the real index of the first coefficient of the smoothed data at a level, the 2nd column is the real index of the last coefficient, the last column specifies the offset of the first smoothed datum at that level. The offset is used by the C code to work out where the beginning of the sequence is within a packed vector of the pyramid structure. The first and 2nd columns can be used to work out how many numbers there are at a level. If `bc="periodic"` then the pyramid is a true power of 2 pyramid, that is it starts with a power of 2, and the next level is half of the previous. If `bc="symmetric"` then the pyramid is nearly exactly a power of 2, but not quite, see the Details section for why this is so.
`ntotal`	The total number of smoothed data/original data points.
`first.last.d`	A mx3 matrix. As for `first.last.c` but for the wavelet coefficients packed as the D component of a wavelet structure.
`ntotal.d`	The total number of wavelet coefficients.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

References

Nason, G.P. and Silverman, B.W. (1994). The discrete wavelet transform in S.

See Also

wd, wr, wr.wd, accessC, accessD, filter.select. imwd.

Examples

```
#
#If you're twisted then you may just want to look at one of these.
#
first.last(length(filter.select(2)), 64)
#$first.last.c:
#First Last Offset
#[1,]     0    0    126
#[2,]     0    1    124
#[3,]     0    3    120
#[4,]     0    7    112
#[5,]     0   15     96
#[6,]     0   31     64
#[7,]     0   63      0
#
#$ntotal:
#[1] 127
#
#$first.last.d:
#First Last Offset
#[1,]     0    0     62
#[2,]     0    1     60
#[3,]     0    3     56
#[4,]     0    7     48
#[5,]     0   15     32
#[6,]     0   31      0
#
#$ntotal.d:
#[1] 63
#
#
```

`first.last.dh` *Build special first/last database for some wavelet density functions*

Description

This function builds a special first/last database for some of the wavelet density estimation functions written by David Herrick and described in his PhD thesis.

See `first.last` to see what this kind of function does.

Usage

```
first.last.dh(LengthH, DataLength, type = "wavelet", bc = "periodic",
    firstk = c(0, DataLength - 1))
```

Arguments

`LengthH`	The length of the smoothing (C) filter
`DataLength`	The length of the data that you wish to transform
`type`	The type of wavelet transform, `wavelet` or `station` for decimated and nondecimated transforms respectively.
`bc`	Boundary conditions, `periodic` or `symmetric`
`firstk`	The first k index, leave as default

Details

Description says all.

Value

A list with several components in exactly the same format as for `first.last`.

Author(s)

David Herrick

See Also

`dencvwd`,`first.last`,`wd.dh`

`firstdot` *Return the location of the first period character within a character string (for a vector of strings of arbitrary length).*

Description

Returns the index of the location of the first period character within a character string for a series of strings in a vector of character string of arbitrary length).

This is a subsidiary routine for `rmget` and not really intended for user use.

Usage

```
firstdot(s)
```

Arguments

`s`	Vector of character strings.

Details

A very simple function. It searches through a character string for the first period character and the returns the position of that period character. It performs this search for each of the character strings in the input vector.

Value

A vector of integers of the same length as the input vector. Each integer in the output vector is the index position of the first period character in the corresponding character string in the input vector. If a character string does not contain a period character then the corresponding output integer is zero.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

See Also

`rmget`

Examples

```
#
# Let's find the first dot in the following strings...
#
firstdot("mary.had.a.little.lamb")
#[1] 5
#
# I.e. the first period was after "mary" -- the fifth character
#
# This following string doesn't have any periods in it.
#
firstdot("StellaArtois")
#[1] 0
#
# The function works on vectors of character strings
#
TopCricketAve <- c("Don.Bradman", "Graeme.Pollock", "George.Headley",
 "Herbert.Sutcliffe", "Vinod.Kambli", "Javed.Miandad")
firstdot(TopCricketAve)
#[1] 4 7 7 8 6 6
```

`FullWaveletCV` *Perform whole wavelet cross-validation in C code*

Description

Perform whole wavelet cross-validation in C code. This routine equivalent to `CWCV` except that more preparatory material is passed to C code for speed.

The major difference is that **only** the cross-validated wavelet threshold is returned.

Usage

```
FullWaveletCV(noisy, ll = 3, type = "soft", filter.number = 10, family =
    "DaubLeAsymm", tol = 0.01, verbose = 0)
```

Arguments

`noisy`	A vector of dyadic (power of two) length that contains the noisy data that you wish to apply wavelet shrinkage by cross-validation to.
`ll`	The primary resolution that you wish to assume. No wavelet coefficients that are on coarser scales than ll will be thresholded.
`type`	this option specifies the thresholding type which can be "hard" or "soft".
`filter.number`	This selects the smoothness of wavelet that you want to perform wavelet shrinkage by cross-validation.
`family`	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".

`tol`	this specifies the convergence tolerance for the cross-validation optimization routine (a golden section search).
`verbose`	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.

Details

Description says all

Value

The cross-validated wavelet threshold.

Author(s)

G P Nason

See Also

`CWCV`

`GenW` *Generate (inverse) discrete wavelet transform matrix.*

Description

This function generates a matrix that can perform the discrete wavelet transform (useful for understanding the DWT but use the fast algorithm coded in `wd` for general use). The function returns the matrix for the inverse transform. Since the matrix is orthogonal transpose the matrix to obtain the forward transform matrix.

Usage

```
GenW(n=8, filter.number=10, family="DaubLeAsymm", bc="periodic")
```

Arguments

`n`	The order of the DWT matrix will be n times n. n should be a power of two.
`filter.number`	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
`family`	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
`bc`	boundary conditions to use. This can be `periodic` or `symmetric` depending on whether you want the returned matrix to assume periodic or symmetric end-reflection boundary conditions.

Details

The discrete wavelet transform is usually computed using the fast pyramid algorithm of Mallat. However, the transform can be written in a matrix form and this is useful for understanding what the fast transform does. One wouldn't normally use the matrix for performing the transform but use the fast transform function wd instead.

The matrix returned by this function represents the inverse DWT. Since the matrix (and transform) is orthogonal one can obtain the matrix representation of the forward transform simply by transposing the matrix using the t function in S-Plus.

The returned matrix is organised as follows. The first column always corresponds to the linear combination corresponding to the scaling function coefficient (so the column is constant. The next n/2 columns correspond to the finest scale wavelet coefficients; the next n/4 columns to the next finest scale and so on until the last column which corresponds to the coarsest scale wavelet coefficients.

The matrix is computed by performing successive fast DWTs on unit vectors.

Value

A matrix of order n that contains the inverse discrete wavelet transform.

RELEASE

Version 3.2 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

wd, wr.

Examples

```
#
# Generate the wavelet transform matrix corresponding to the Haar wavelet
# transform of order 8
#
haarmat <- GenW(8, filter.number=1, family="DaubExPhase")
#
# Let's look at this matrix
#
#haarmat
#          [,1]       [,2]       [,3]       [,4]       [,5] [,6] [,7]       [,8]
#[1,] 0.3535534  0.7071068  0.0000000  0.0000000  0.0000000  0.5  0.0  0.3535534
#[2,] 0.3535534 -0.7071068  0.0000000  0.0000000  0.0000000  0.5  0.0  0.3535534
#[3,] 0.3535534  0.0000000  0.7071068  0.0000000  0.0000000 -0.5  0.0  0.3535534
#[4,] 0.3535534  0.0000000 -0.7071068  0.0000000  0.0000000 -0.5  0.0  0.3535534
#[5,] 0.3535534  0.0000000  0.0000000  0.7071068  0.0000000  0.0  0.5 -0.3535534
#[6,] 0.3535534  0.0000000  0.0000000 -0.7071068  0.0000000  0.0  0.5 -0.3535534
#[7,] 0.3535534  0.0000000  0.0000000  0.0000000  0.7071068  0.0 -0.5 -0.3535534
#[8,] 0.3535534  0.0000000  0.0000000  0.0000000 -0.7071068  0.0 -0.5 -0.3535534
```

```
#
# As noted above the first column is the l.c. corresponding to the scaling
# function coefficient and then the l.c.s corresponding to the wavelet
# coefficients from the finest to the coarsest.
#
# The above matrix represented the inverse DWT. Let's compute the forward
# transform matrix representation:
#
#t(haarmat)
#         [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]       [,8]
#[1,] 0.3535534  0.3535534  0.3535534  0.3535534  0.3535534  0.3535534  0.3535534  0.3535534
#[2,] 0.7071068 -0.7071068  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
#[3,] 0.0000000  0.0000000  0.7071068 -0.7071068  0.0000000  0.0000000  0.0000000  0.0000000
#[4,] 0.0000000  0.0000000  0.0000000  0.0000000  0.7071068 -0.7071068  0.0000000  0.0000000
#[5,] 0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.7071068 -0.7071068
#[6,] 0.5000000  0.5000000 -0.5000000 -0.5000000  0.0000000  0.0000000  0.0000000  0.0000000
#[7,] 0.0000000  0.0000000  0.0000000  0.0000000  0.5000000  0.5000000 -0.5000000 -0.5000000
#[8,] 0.3535534  0.3535534  0.3535534  0.3535534 -0.3535534 -0.3535534 -0.3535534 -0.3535534
#
#
```

`getarrvec` *Compute and return weaving permutation for conversion from wst objects to wd class objects.*

Description

Computes weaving permutation for conversion from `wst` objects to `wd`

Usage

```
getarrvec(nlevels, sort=TRUE)
```

Arguments

`nlevels`	The number of `levels` in the non-decimated transform for which the permutation is to be computed.
`sort`	If `TRUE` then compute permutation for indexing a `wst` object. If `FALSE` then compute permutation for indexing a `wd` object.

Details

Conversion of `wst` objects into `wd` objects and vice versa can be carried out using the `convert.wst` and `convert.wd` functions. These latter functions depend on this getarrvec function to compute the permutation which maps coefficients from one ordering to the other.

This function returns a matrix which gives the necessary permutations for scale levels 1 to `nlevels-1`. If you want to get the permutation for the level 0 coefficients of the `wst` object you will have to call the `levarr` function directly.

This permutation is described in Nason, Sapatinas and Sawczenko, 1998.

The function that actually computes the permutations is levarr. This function just combines the results from levarr.

Value

A matrix with nlevels-1 columns. Column 1 corresponds to scale level nlevels-1 in the wst object, and column nlevels-1 corresponds to scale level 1 in the wst object. Replace wst by wd if sort=FALSE.

RELEASE

Version 3.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

convert, convert.wd, convert.wst, levarr, wst, wst.object, wpst.

Examples

```
#
# What would the permutation be for a wst
# object with 4 levels?
#
arrvec <- getarrvec(4)
#arrvec
#      [,1] [,2] [,3]
# [1,]    1    1    1
# [2,]    9    9    9
# [3,]    2    5    5
# [4,]   10   13   13
# [5,]    3    2    3
# [6,]   11   10   11
# [7,]    4    6    7
# [8,]   12   14   15
# [9,]    5    3    2
#[10,]   13   11   10
#[11,]    6    7    6
#[12,]   14   15   14
#[13,]    7    4    4
#[14,]   15   12   12
#[15,]    8    8    8
#[16,]   16   16   16
#
# The permutation for level 3 is in column 1
# The permutation for level 2 is in column 2
# The permutation for level 1 is in column 3.
#
```

```
# The following shows that the above is the right permutation (for level 2
# at least.
#
# Start off with some random normal data!
#
myrand <- rnorm(1:16)
#
# Now take both the time ordered non-decimated wavelet
# transform and the packet ordered non-decimated wavelet
# transform.
#
myrwdS <- wd(myrand, type="station")
myrwst <- wst(myrand)
#
# Let's look at the level 2 coefficients of myrwdS
#
accessD(myrwdS, level=2)
# [1] -0.73280829 -0.97892279  1.33305777  1.46320165 -0.94790098
# [6] -1.39276215  0.40023757  0.82517249 -0.56317955 -0.89408713
#[11]  0.77166463  1.56204870 -0.34342230 -1.64133182  0.08235115
#[16]  1.05668106
#
# Let's look at the level 2 coefficients of myrwst
#
accessD(myrwst, level=2)
# [1] -0.73280829 -0.94790098 -0.56317955 -0.34342230  1.33305777
# [6]  0.40023757  0.77166463  0.08235115 -0.97892279 -1.39276215
#[11] -0.89408713 -1.64133182  1.46320165  0.82517249  1.56204870
#[16]  1.05668106
#
# O.k. So the coefficients are the same, but they are not in the
# same order as in myrwdS. So let's use the permutation in the
# second column of arrvec to reorder the myrwst coefficients
# to have the same order as the myrwdS ones
#
accessD(myrwst, level=2)[arrvec[,2]]
# [1] -0.73280829 -0.97892279  1.33305777  1.46320165 -0.94790098
# [6] -1.39276215  0.40023757  0.82517249 -0.56317955 -0.89408713
#[11]  0.77166463  1.56204870 -0.34342230 -1.64133182  0.08235115
#[16]  1.05668106
#
# These coefficients have the correct ordering.
```

`getpacket` *Get a packet of coefficients from a wavelet object*

Description

This generic function extracts packets of coefficients from various types of wavelet objects.

This function is generic.

Particular methods exist. For objects of class:

wp use the `getpacket.wp` method.

wst use the `getpacket.wst` method.

wpst use the `getpacket.wpst` method.

See individual method help pages for operation and examples.

Use the `accessC` and `accessD` function to extract whole resolution levels of coefficients simultaneously.

Usage

```
getpacket(...)
```

Arguments

...	See individual help pages for details.

Value

The packet of coefficients requested.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

`getpacket.wp`, `getpacket.wst`, `getpacket.wpst`, `accessD`, `accessC`.

`getpacket.wp` *Get packet of coefficients from a wavelet packet object (wp).*

Description

This function extracts and returns a packet of coefficients from a wavelet packet (`wp`) object.

Usage

```
## S3 method for class 'wp'
getpacket(wp, level, index, ... )
```

Arguments

`wp`	Wavelet packet object from which you wish to extract the packet from.
`level`	The resolution level of the coefficients that you wish to extract.
`index`	The index number within the resolution level of the packet of coefficients that you wish to extract.
`...`	any other arguments

Details

The `wp` produces a wavelet packet object. The coefficients in this structure can be organised into a binary tree with each node in the tree containing a packet of coefficients.

Each packet of coefficients is obtained by chaining together the effect of the *two packet operators* DG and DH: these are the high and low pass quadrature mirror filters of the Mallat pyramid algorithm scheme followed by decimation (see Mallat~(1989b)).

Starting with data c^J at resolution level J containing 2^J data points the wavelet packet algorithm operates as follows. First DG and DH are applied to c^J producing d^{J-1} and c^{J-1} respectively. Each of these sets of coefficients is of length one half of the original data: i.e. 2^{J-1}. Each of these sets of coefficients is a set of *wavelet packet coefficients*. The algorithm then applies both DG and DH to both d^{J-1} and c^{J-1} to form a four sets of coefficients at level J-2. Both operators are used again on the four sets to produce 8 sets, then again on the 8 sets to form 16 sets and so on. At level j=J,...,0 there are 2^{J-j} packets of coefficients each containing 2^j coefficients.

This function enables whole packets of coefficients to be extracted at any resolution level. The index argument chooses a particular packet within each level and thus ranges from 0 (which always refer to the father wavelet coefficients), 1 (which always refer to the mother wavelet coefficients) up to 2^{J-j}.

Value

A vector containing the packet of wavelet packet coefficients that you wished to extract.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

`wp`, `putpacket.wp`, `basisplot.wp`, `draw.wp`, `InvBasis.wp`, `MaNoVe.wp`, `nlevelsWT.wp`, `plot.wp`. `threshold.wp`.

Examples

```
#
# Take the wavelet packet transform of some random data
#
MyWP <- wp(rnorm(1:512))
#
# The above data set was 2^9 in length. Therefore there are
# coefficients at resolution levels 0, 1, 2, ..., and 8.
#
# The high resolution coefficients are at level 8.
# There should be 256 DG coefficients and 256 DH coefficients
#
length(getpacket(MyWP, level=8, index=0))
#[1] 256
length(getpacket(MyWP, level=8, index=1))
#[1] 256
#
# The next command shows that there are only two packets at level 8
#
## Not run: getpacket(MyWP, level=8, index=2)
#Index was too high, maximum for this level is  1
#Error in getpacket.wp(MyWP, level = 8, index = 2): Error occured
#Dumped
#
# There should be 4 coefficients at resolution level 2
#
# The father wavelet coefficients are (index=0)
getpacket(MyWP, level=2, index=0)
#[1] -0.9736576  0.5579501  0.3100629 -0.3834068
#
# The mother wavelet coefficients are (index=1)
#
#[1]  0.72871405  0.04356728 -0.43175307  1.77291483
#
# There will be 127 packets at this level.
#
```

`getpacket.wpst` *Get packet of coefficients from a non-decimated wavelet packet object (wpst).*

Description

This function extracts and returns a packet of coefficients from a non-decimated wavelet packet (`wpst`) object.

Usage

```
## S3 method for class 'wpst'
getpacket(wpst, level, index, ... )
```

Arguments

`wpst`	Non-decimated wavelet packet object from which you wish to extract the packet from.
`level`	The resolution level of the coefficients that you wish to extract. Can range from 0 to `nlevelsWT`(wpst). The coefficients at level `nlevels` are the data the created the `wpst.object`.
`index`	The index number within the resolution level of the packet of coefficients that you wish to extract. Index ranges from 0 to $(4^r) - 1$ where `r = nlevelsWT - level`.
`...`	any other arguments

Details

The `wpst` transform produces a non-decimated wavelet packet object. This is a "cross" between a `wavelet packet` object and a `non-decimated wavelet` object. In other words the transform produces *wavelet packet* coefficients at every possible integer shift (unlike the ordinary wavelet packet transform which is aligned to a dyadic grid).

Each packet of coefficients is obtained by chaining together the effect of the two *packet operators* DG and DH: these are the high and low pass quadrature mirror filters of the Mallat pyramid algorithm scheme followed by both even *and* odd decimation. For a full description of this algorithm and how coefficients are stored within see Nason, Sapatinas and Sawczenko, 1998.

Note that this function extracts *packets*. If you want to obtain the wavelet packet coefficients for each shift you need to use the `accessD.wpst`function. This function extracts particular wavelet packet coefficients for a particular shift. In particular, this function returns a number of coefficients dependent on the scale level requested whereas `accessD.wpst` always returns a vector of coefficients of length equal to the input data that created the `wpst.object` initially.

Value

A vector containing the packet of non-decimated wavelet packet coefficients that you wished to extract.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

`accessD.wpst`, `wpst`,

Examples

```
#
# Create some random data
#
myrand <- rnorm(16)
#myrand
# [1]  0.19268626 -0.41737181 -0.30806613  0.07435407  0.99871757
# [6] -0.58935121 -1.38049759 -0.13346631  1.55555403 -1.60581265
#[11]  0.14353621  1.21277774  1.13762337 -1.08577934 -0.29745609
#[16]  0.50977512
#
# Do the non-decimated wavelet packet transform
#
myrwpst <- wpst(myrand)
#
# Let's access what is a level nlevelsWT(myrwpst)
#
getpacket(myrwpst, nlevelsWT(myrwpst), index=0)
# [1]  0.19268626 -0.41737181 -0.30806613  0.07435407  0.99871757
# [6] -0.58935121 -1.38049759 -0.13346631  1.55555403 -1.60581265
#[11]  0.14353621  1.21277774  1.13762337 -1.08577934 -0.29745609
#[16]  0.50977512
#
# I.e. the data that created the object.
#
# How about extracting the 3rd (last) packet at level 3?
#
getpacket(myrwpst, 3, index=3)
#[1] -2.660657144  0.688415755 -1.764060698  0.717267105 -0.206916242
#[6] -0.659983747  0.005836952 -0.196874007
#
# Of course, there are only 8 coefficients at this level.
```

`getpacket.wst` *Get packet of coefficients from a packet ordered non-decimated wavelet object (wst).*

Description

This function extracts and returns a packet of coefficients from a packet-ordered non-decimated wavelet object (`wst`) object. The `wst` objects are computed by the `wst` function amongst others.

Usage

```
## S3 method for class 'wst'
getpacket(wst, level, index, type="D", aspect, ...)
```

Arguments

`wst`	Packet-ordered non-decimated wavelet object from which you wish to extract the packet from.
`level`	The resolution level of the coefficients that you wish to extract.
`index`	The index number within the resolution level of the packet of coefficients that you wish to extract.
`type`	This argument must be either `"C"` or `"D"`. If the argument is `"C"` then non-decimated father wavelet coefficients corresponding to the packet that you want are returned. If the argument is `"D"` then non-decimated mother wavelet coefficients are returned.
`aspect`	Function applied to the coefficients before return. This is suppled as a character string which gets converted to a function to apply. For example, "Mod" for complex-valued coefficients returns the absolute values.
`...`	Other arguments

Details

The `wst` function produces a packet-ordered non-decimated wavelet object: `wst`. The coefficients in this structure can be organised into a binary tree with each node in the tree containing a packet of coefficients.

Each packet is obtained by repeated application of the usual DG quadrature mirror filter with both even and odd dyadic decimation. See the detailed description given in Nason and Silverman, 1995.

This function enables whole packets of coefficients to be extracted at any resolution level. The index argument chooses a particular packet within each level and thus ranges from 0 to 2^{J-j} for j=0,..., J-1. Each packet corresponds to the wavelet coefficients with respect to different origins.

Note that both mother and father wavelet coefficient at different shifts are available by using the type argument.

Value

A vector containing the packet of packet-ordered non-decimated wavelet coefficients that you wished to extract.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

`wst`, `wst.object`,

Examples

```
#
# Take the packet-ordered non-decimated transform of some random data
#
MyWST <- wst(rnorm(1:512))
#
# The above data set was 2^9 in length. Therefore there are
# coefficients at resolution levels 0, 1, 2, ..., and 8.
#
# The high resolution coefficients are at level 8.
# There should be 256 coefficients at level 8 in index location 0 and 1.
#
length(getpacket(MyWST, level=8, index=0))
#[1] 256
length(getpacket(MyWST, level=8, index=1))
#[1] 256
#
# There are also 256 FATHER wavelet coefficients at each of these two indices
# (origins)
#
length(getpacket(MyWST, level=8, index=0, type="C"))
#[1] 256
length(getpacket(MyWST, level=8, index=1, type="C"))
#[1] 256
#
# There should be 4 coefficients at resolution level 2
#
getpacket(MyWST, level=2, index=0)
#[1] -0.92103095  0.70125471  0.07361174 -0.43467375
#
# Here are the equivalent father wavelet coefficients
#
getpacket(MyWST, level=2, index=0, type="C")
#[1] -1.8233506 -0.2550734  1.9613138  1.2391913
```

`getpacket.wst2D` *Get packet of coefficients from a two-dimensional non-decimated wavelet object (wst2D).*

Description

This function extracts and returns a packet of coefficients from a two-dimensional non-decimated wavelet (`wst2D`) object.

Usage

```
## S3 method for class 'wst2D'
getpacket(wst2D, level, index, type="S", Ccode=TRUE, ...)
```

Arguments

`wst2D`	2D non-decimated wavelet object from which you wish to extract a packet from.
`level`	The resolution level of the coefficients that you wish to extract. Can range from 0 to `nlevelsWT`(wpst)-1.
`index`	The index number within the resolution level of the packet of coefficients that you wish to extract. Index is a base-4 number which is r digits long. Each digit can be 0, 1, 2 or 3 corresponding to no shifts, horizontal shift, vertical shift or horizontal and vertical shifts. The number r indicates the depth of the resolution level from the data resolution i.e. where `r = nlevelsWT - level`. Where there is a string of more than one digit the left most digits correspond to finest scale shift selection, the right most digits to the coarser scales (I think).
`type`	This is a one letter character string: one of "S", "H", "V" or "D" for the smooth coefficients, horizontal, vertical or diagonal detail.
`Ccode`	If `T` then fast C code is used to obtain the packet, otherwise slow SPlus code is used. Unless you have some special reason always use the C code (and leave the argument at its default).
`...`	any other arguments

Details

The `wst2D` function creates a `wst2D` class object. Starting with a smooth the operators H, G, GS and HS (where G, H are the usual Mallat operators and S is the shift-by-one operator) are operated first on the rows and then the columns: i.e. so each of the operators HH, HG, GH, GG, HSH, HSG, GSH, GSG HHS, GHS, HGS, GGS HSHS, HSGS, GSHS and GSGS are applied. Then the same collection of operators is applied to all the derived smooths, i.e. HH, HSH, HHS and HSHS.

So the next level is obtained from the previous level with basically HH, HG, GH and GG but with extra shifts in the horizontal, vertical and horizontal and vertical directions. The index provides a way to enumerate the paths through this tree where each smooth has 4 children and indexed by a number between 0 and 3.

Each of the 4 children has 4 components: a smooth, horizontal, vertical and diagonal detail, much in the same way as for the Mallat 2D wavelet transform implemented in the WaveThresh function `imwd`.

Value

A matrix containing the packet of the 2D non-decimated wavelet coefficients that you require.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

putpacket.wst2D, wst2D, wst2D.object.

Examples

```
#
# Create a random image.
#
myrand <- matrix(rnorm(16), nrow=4, ncol=4)
#myrand
#            [,1]       [,2]        [,3]       [,4]
#[1,]  0.01692807  0.1400891 -0.38225727  0.3372708
#[2,] -0.79799841 -0.3306080  1.59789958 -1.0606204
#[3,]  0.29151629 -0.2028172 -0.02346776  0.5833292
#[4,] -2.21505532 -0.3591296 -0.39354119  0.6147043
#
# Do the 2D non-decimated wavelet transform
#
myrwst2D <- wst2D(myrand)
#
# Let's access the finest scale detail, not shifted in the vertical
# direction.
#
getpacket(myrwst2D, nlevelsWT(myrwst2D)-1, index=0, type="V")
#           [,1]       [,2]
#[1,] -0.1626819 -1.3244064
#
# Compare this to the ordinary 2D DWT for the vertical detail at this
# resolution level
imwd(myrand)[[lt.to.name( 1, "DC")]]
#[1] -0.1626819 -1.3244064  1.4113247 -0.7383336
#
# The same numbers but they're not in matrix format because
# imwd returns vectors not matrices.
#
# Now back to the wst2D object. Let's
# extract vertical detail again at level 1 but this time the horizontally
# shifted data.
#
getpacket(myrwst2D, level=1, index=1, type="V")
#           [,1]      [,2]
#[1,] -0.5984427 0.2599445
#[2,] -0.6502002 1.8027955
#
# So, yes, different data. Now how about at a deeper resolution level.
# Lets have a horizontal shift, as before, for the level 1 but follow it
# with a diagonal shift and this time extract the smooth component:
#
getpacket(myrwst2D, level=0, index=13, type="S")
#           [,1]
#[1,] -0.5459394
#
```

```
# Of course, only one number because this is at level 0
```

GetRSSWST *Computes estimate of error for function estimate.*

Description

Computes estimate of error for function estimate. Given noisy data and a threshold value this function uses Nason's 1996 two-fold cross-validation algorithm, but using packet ordered non-decimated wavelet transforms to compute two estimates of an underlying "true" function and uses them to compute an estimate of the error in estimating the truth.

Usage

```
GetRSSWST(ndata, threshold, levels, family = "DaubLeAsymm",
 filter.number = 10, type = "soft", norm = l2norm, verbose = 0,
 InverseType = "average")
```

Arguments

ndata	the noisy data. This is a vector containing the signal plus noise. The length of this vector should be a power of two.
threshold	the value of the threshold that you wish to compute the error of the estimate at
levels	the levels over which you wish the threshold value to be computed (the threshold that is used in computing the estimate and error in the estimate). See the explanation for this argument in the threshold.wst function.
family	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
filter.number	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
type	whether to use hard or soft thresholding. See the explanation for this argument in the threshold.wst function.
norm	which measure of distance to judge the dissimilarity between the estimates. The functions l2norm and linfnorm are suitable examples.
verbose	If TRUE then informative messages are printed during the progression of the function, otherwise they are not.
InverseType	The possible options are "average" or "minent". The former uses basis averaging to form estimates of the unknown function. The "minent" function selects a basis using the Coifman and Wickerhauser, 1992 algorithm to select a basis to invert.

Details

This function implements the component of the cross-validation method detailed by Nason, 1996 for computing an estimate of the error between an estimate and the "truth". The difference here is that it uses the packet ordered non-decimated wavelet transform rather than the standard Mallat wd discrete wavelet transform. As such it is an examples of the translation-invariant denoising of Coifman and Donoho, 1995 but uses cross-validation to choose the threshold rather than SUREshrink.

Note that the procedure outlined above can use AvBasis basis averaging or basis selection and inversion using the Coifman and Wickerhauser, 1992 best-basis algorithm

Value

A real number which is estimate of the error between estimate and truth at the given threshold.

RELEASE

Version 3.6 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

linfnorm, linfnorm, wstCV, wstCVl.

Examples

```
#
# This function performs the error estimation step for the
# \code{\link{wstCV}} function and so is not intended for
# user use.
#
```

---

griddata objects *Data interpolated to a grid objects.*

Description

These are objects of classes

griddata

These objects store the results of interpolating a 1-D regression data set to a grid which is a power of two in length

Details

The help page for makegrid and Kovac, (1997), p.81 give further details about how a griddata object is constructed.

Value

The following components must be included in a legitimate griddata object.

gridt	a vector containing the values of the grid on the "x" axis.
gridy	a vector containing the values of the grid on the "y" axis. This vector has to be the same length as gridt. Typically the values in (gridt, gridy) are the results of interpolating arbitrary data (x,y) onto (gridt, gridy).
G	Codes the value of the linear interpolant matrix for the corresponding entry in gridt. The value at each point corresponds to the proportion of the original data point pointed to by Gindex that contributes to the new value at the corresponding gridt value. See Kovac, (1997), page 81 for further information.
Gindex	Each entry in Gindex refers to one of the pairs in (x,y) which is contributing to the (gridt, gridy) interpolant. See previous help for G.

GENERATION

This class of objects is returned from the makegrid function to represent the results of interpolating a 1-D regression data set to a grid.

METHODS

The griddata class of objects really on has one function that uses it: irregwd.

RELEASE

Version 3.9.6 Copyright Arne Kovac 1997 Copyright Guy Nason (help pages) 1999.

SEE ALSO

makegrid, irregwd

Author(s)

Arne Kovac

guyrot *Cyclically rotate elements of a vector*

Description

This function shifts (or rotates) the elements of the input vector in a cyclic fashion (end periodicity is used).

Usage

```
guyrot(v, n)
```

Arguments

v	Vector whose elements you wish to rotate
n	Integer determining the amount to rotate, can be negative

Details

A very simple function which cyclically shifts the elements of a vector. Not necessarily intended as a top level user function but it is a useful little function.

Value

A vector containing the shifted or rotated coefficients.

Author(s)

G P Nason

See Also

wpst2discr, wpstCLASS

Examples

```
#
# Start off with an example vector
#
v <- c(1,2,3,4,5,6)
#
# Rotate it one element to the right, rightmost element gets rotated round
# to be first element.
#
guyrot(v,1)
# [1] 6 1 2 3 4 5
#
# Rotate v two spaces to the left, leftmost two elements get rotated around
# to be new last elements
```

```
guyrot(v, -2)
#
# [1] 3 4 5 6 1 2
#
#
# Now issue a larger rotation, e.g. 19!
#
guyrot(v,19)
# [1] 6 1 2 3 4 5
#
# Its just the same as rotating by 1 since the input vector is of length 6
# and so rotating by 19 is the same as rotating by 6,6,6, and then 1!
#
```

HaarConcat *Generate a concatenated Haar MA process*

Description

This function generates a particular set of four concatenated Haar MA processes.

Usage

```
HaarConcat()
```

Arguments

None

Details

This function generates a realization of particular kind of non-stationary time series probability model. The returned time series is the result of concatenating 4 time series each of length 128 from the Haar MA process generator (HaarMA) of orders 1, 2, 3 and 4. The standard deviation of the innovations is 1. This function was used to generate the figure of the concatenated Haar MA process in Nason, von Sachs and Kroisandt. It produces a kind of time series that can be sparsely represented by the wavelet machinery but at the same time is non-stationary.

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

Value

A vector containing 512 observations from four concatenated Haar MA processes

Author(s)

G P Nason

See Also

HaarMA,ewspec

Examples

```
#
# Generate the concatenated Haar MA process.
#
MyHaarCC <- HaarConcat()
#
# Plot it
#
## Not run: ts.plot(MyHaarCC)
```

HaarMA *Generate Haar MA processes.*

Description

This function generates an arbitrary number of observations from a Haar MA process of any order with a particular variance.

Usage

```
HaarMA(n, sd=1, order=5)
```

Arguments

n	The number of observations in the realization that you want to create. Note that n does NOT have to be a power of two.
sd	The standard deviation of the innovations.
order	The order of the Haar MA process.

Details

A Haar MA process is a special kind of time series moving-average (MA) process. A Haar MA process of order k is a MA process of order 2^k. The coefficients of the Haar MA process are given by the filter coefficients of the discrete Haar wavelet at different scales.

For examples: the Haar MA process of order 1 is an MA process of order 2. The coefficients are $1/\sqrt{2}$ and $-1/\sqrt{2}$. The Haar MA process of order 2 is an MA process of order 4. The coefficients are 1/2, 1/2, -1/2, -1/2 and so on. It is possible to define other processes for other wavelets as well.

Any Haar MA process is a good examples of a (stationary) LSW process because it is sparsely representable by the locally-stationary wavelet machinery defined in Nason, von Sachs and Kroisandt.

Value

A vector containing a realization of a Haar MA process of the specified order, standard deviation and number of observations.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

See Also

HaarConcat, ewspec,

Examples

```
#
# Generate a Haar MA process of order 1 (high frequency series)
#
MyHaarMA <- HaarMA(n=151, sd=2, order=1)
#
# Plot it
#
## Not run: ts.plot(MyHaarMA)
#
# Generate another Haar MA process of order 3 (lower frequency), but of
# smaller variance
#
MyHaarMA2 <- HaarMA(n=151, sd=1, order=3)
#
# Plot it
#
## Not run: ts.plot(MyHaarMA2)
#
# Let's plot them next to each other so that you can really see the
# differences.
#
# Plot a vertical dotted line which indicates where the processes are
# joined
#
## Not run: ts.plot(c(MyHaarMA, MyHaarMA2))
## Not run: abline(v=152, lty=2)
```

image.wd *Produce image representation of nondecimated wavelet transform*

Description

Produces a representation of a nondecimated wavelet transform (time-ordered) as an image.

Usage

```
## S3 method for class 'wd'
image(x, strut = 10, type = "D", transform = I, ...)
```

Arguments

x	The wd.object that you wish to image
strut	The width of each coefficient in the image
type	Either "C" or "D" depending if you wish to image scaling function or wavelet coefficients respectively
transform	Apply a numerical transform to the coefficients before display
...	Other arguments

Details

Description says all

Value

None

Author(s)

G P Nason

See Also

logabs, nlevelsWT,wd

Examples

```
tmp <- wd(rnorm(256), type="station")
## Not run: image(tmp)
```

image.wst *Produce image representation of a wst class object*

Description

Produces an image representation of the coefficients contained within a wst.object class object.

Usage

```
## S3 method for class 'wst'
image(x, nv, strut = 10, type = "D", transform = I, ...)
```

Arguments

x	The wst object you wish to image
nv	An associated node vector, this argument is no longer used and should be omitted (in the S version it permitted coloration of particular bases)
strut	The number of pixels/width that each coefficient should be drawn with
type	Either "C" or "D" depending on whether you wish to image scaling function coefficients or wavelet ones
transform	A numerical transform you wish to apply to the coefficients before imaging
...	Other arguments

Details

Description says all

Value

None

Author(s)

G P Nason

See Also

logabs,wst

Examples

```
tmp <- wst(rnorm(1024))
## Not run: image(tmp)
## Not run: image(tmp, transform=logabs)
```

imwd *Two-dimensional wavelet transform (decomposition).*

Description

This function can perform two types of two-dimensional discrete wavelet transform (DWT). The standard transform (`type="wavelet"`) computes the 2D DWT according to Mallat's pyramidal algorithm (Mallat, 1989). The spatially ordered non-decimated 2D DWT (NDWT) (`type="station"`) contains all possible spatially shifted versions of the DWT. The order of computation of the DWT is O(n), and it is O(n log n) for the NDWT if n is the number of pixels.

Usage

```
imwd(image, filter.number=10, family="DaubLeAsymm", type="wavelet",
bc="periodic", RetFather=TRUE, verbose=FALSE)
```

Arguments

`image`	A square matrix containing the image data you wish to decompose. The side-length of this matrix must be a power of 2.
`filter.number`	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
`family`	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
`type`	specifies the type of wavelet transform. This can be "wavelet" (default) in which case the standard 2D DWT is performed (as in previous releases of WaveThresh). If type is "station" then the 2D spatially-ordered non-decimated DWT is performed. At present, only periodic boundary conditions can be used with the 2D spatially ordered non-decimated wavelet transform.
`bc`	specifies the boundary handling. If bc=="periodic" the default, then the function you decompose is assumed to be periodic on it's interval of definition, if bc=="symmetric" then the function beyond its boundaries is assumed to be a symmetric reflection of the function in the boundary. The symmetric option was the implicit default in releases prior to 2.2. Note that only periodic boundary conditions are valid for the 2D spatially-ordered non-decimated wavelet transform.
`RetFather`	If `TRUE` then this argument causes the scaling function coefficients at each resolution level to be returned as well as the wavelet coefficients. If `FALSE` then no scaling function coefficients are returned. The opportunity of returning father wavelet coefficients has been added since previous versions of WaveThresh.
`verbose`	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.

Details

The 2D algorithm is essentially the application of many 1D filters. First, the columns are attacked with the smoothing (H) and bandpass (G) filters, and the rows of each of these resultant images are attacked again with each of G and H, this results in 4 images. Three of them, GG, GH, and HG correspond to the highest resolution wavelet coefficients. The HH image is a smoothed version of the original and can be further attacked in exactly the same way as the original image to obtain GG(HH), GH(HH), and HG(HH), the wavelet coefficients at the second highest resolution level and HH(HH) the twice-smoothed image, which then goes on to be further attacked.

If `RetFather=TRUE` then the results of the HH smooth (the scaling function coefficients) are returned additionally.

There are now two methods of handling "boundary problems". If you know that your function is periodic (on it's interval) then use the bc="periodic" option, if you think that the function is symmetric reflection about each boundary then use bc="symmetric". If you don't know then it is wise to experiment with both methods, in any case, if you don't have very much data don't infer too much about your decomposition! If you have loads of data then don't worry too much about the boundaries. It can be easier to interpret the wavelet coefficients from a bc="periodic" decomposition, so that is now the default.

The spatially-ordered non-decimated DWT contains all spatial (toroidal circular) shifts of the standard DWT.

The standard DWT is orthogonal, the spatially-ordered non-decimated transform is most definitely not. This has the added disadvantage that non-decimated wavelet coefficients, even if you supply independent normal noise. This is unlike the standard DWT where the coefficients are independent (normal noise).

The two-dimensional packet-ordered non-decimated discrete wavelet transform is computed by the `wst2D` function.

Value

An object of class `imwd.object` containing the two-dimensional wavelet transform (possibly spatially-ordered non-decimated).

RELEASE

Version 3.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

`wd`, `imwd.object`, `filter.select`

Examples

```
data(lennon)
#
# Let's use the lennon test image
```

```
#
## Not run: image(lennon)
#
# Now let's do the 2D discrete wavelet transform
#
lwd <- imwd(lennon)
#
# Let's look at the coefficients
#
## Not run: plot(lwd)
```

`imwd.object` *Two-dimensional wavelet decomposition objects.*

Description

These are objects of classes

`imwd`

They represent a decomposition of an image with respect to a two-dimensional wavelet basis (or tight frame in the case of the two-dimensional (space-ordered) non-decimated wavelet decomposition).

Details

In previous releases the original image was stored as the "original" component of a imwd object. This is not done now as the resulting objects were excessively large.

Value

The following components must be included in a legitimate 'imwd' object.

`nlevelsWT`	number of levels in wavelet decomposition. If you raise 2 to the power of nlevels then you get the dimension of the image that you originally started with.
`type`	If `type="wavelet"` then the image was decomposed according to the 2D Mallat pyramidal algorithm. If `type="station"` then the image was decomposed using the 2D spatially ordered non-decimated wavelet transform.
`fl.dbase`	The first last database associated with the decomposition. For images, this list is not very useful as each level's components is stored as a list component, rather than being packaged up in a single vector as in the 1D case. Nevertheless the internals still need to know about fl.dbase to get the computations correct. See the help for `first.last` if you are a masochist.
`filter`	A filter object as returned by the `filter.select` function. This component records the filter used in the decomposition. The reconstruction routines use this component to find out what filter to use in reconstruction.

wNLx	The object will probably contain many components with names of this form. These are all the wavelet coefficients of the decomposition. In "wNLx" the "N" refers to the level number and the "x" refers to the direction of the coefficients with "1" being horizontal, "2" being vertical and "3" being diagonal and "4" corresonding to scaling function coefficients at the given resolution level. Note that the levels should be in numerically decreasing order, so if nlevelsWT is 5, then there will be w5L1, w5L2, w5L3 first, then down to w1L1, w1L2, and w1L3. Note that these coefficients store their data according to the `first.last` database `fl.dbase$first.last.d`, so refer to them using this. Note that if `type="wavelet"` then images at level N are subimages of side length `2^N` pixels. If the type component is `"station"` then each coefficient subimage is of the same dimension as the input image used to create this object.
w0Lconstant	This is the coefficient of the bottom level scaling function coefficient. So for examples, if you used Haar wavelets this would be the sample mean of the data (scaled by some factor depending on the number of levels, nlevelsWT).
bc	This component details how the boundaries were treated in the decomposition.

GENERATION

This class of objects is returned from the `imwd` function to represent a two-dimensional (possibly space-ordered non-decimated) wavelet decomposition of a function. Many other functions return an object of class imwd.

METHODS

The imwd class of objects has methods for the following generic functions: `compress`, `draw`, `imwr`, `nullevels.imwd`, `plot`, `print`, `summary`, `threshold.imwd`.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

`imwd`

`imwdc.object` *Two-dimensional compressed wavelet decomposition objects.*

Description

These are objects of classes

`imwdc`

They represent a decomposition of an image with respect to a two-dimensional wavelet basis

Details

In previous releases the original image was stored as the "original" component of a imwd object. This is not done now as the resulting objects were excessively large.

To uncompress this class of object back into an object of class `imwd.object` use the `uncompress.imwdc` function.

Value

The following components must be included in a legitimate 'imwdc' object.

`nlevelsWT` number of levels in wavelet decomposition. If you raise 2 to the power of nlevels then you get the dimension of the image that you originally started with.

`type` If `type="wavelet"` then the image was decomposed according to the 2D Mallat pyramidal algorithm. If `type="station"` then the image was decomposed using the 2D spatially ordered non-decimated wavelet transform.

`fl.dbase` The first last database associated with the decomposition. For images, this list is not very useful as each level's components is stored as a list component, rather than being packaged up in a single vector as in the 1D case. Nevertheless the internals still need to know about fl.dbase to get the computations correct. See the help for `first.last` if you are a masochist.

`filter` A filter object as returned by the `filter.select` function. This component records the filter used in the decomposition. The reconstruction routines use this component to find out what filter to use in reconstruction.

`wNLx` The object will probably contain many components with names of this form. These are all the wavelet coefficients of the decomposition. In "wNLx" the "N" refers to the level number and the "x" refers to the direction of the coefficients with "1" being horizontal, "2" being vertical and "3" being diagonal. Note that imwdc objects do not contain scaling function coefficients. This would negate the point of having a compressed object.

Each vector stores its coefficients using an object of class compressed, i.e. the vector is run-length encoded on zeroes.

Note that the levels should be in numerically decreasing order, so if nlevelsWT is 5, then there will be w5L1, w5L2, w5L3 first, then down to w1L1, w1L2, and w1L3. Note that these coefficients store their data according to the `first.last` database `fl.dbase$first.last.d`, so refer to them using this.

Note that if `type="wavelet"` then images at level N are subimages of side length `2^N` pixels. If the type component is `"station"` then each coefficient subimage is of the same dimension as the input image used to create this object.

`w0Lconstant` This is the coefficient of the bottom level scaling function coefficient. So for examples, if you used Haar wavelets this would be the sample mean of the data (scaled by some factor depending on the number of levels, nlevelsWT).

`bc` This component details how the boundaries were treated in the decomposition.

GENERATION

This class of objects is returned from the threshold.imwd function to represent a thresholded two-dimensional wavelet decomposition of a function. Some other functions return an object of class imwdc.

METHODS

The imwd class of objects has methods for the following generic functions: draw, imwr, nullevels, plot, print, summary, threshold.imwdc.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

imwd imwd.object, threshold.imwd, uncompress.imwdc.

Examples

```
#
# Perform the standard two-dimensional DWT
# on the lennon image.
#
data(lennon)

lwd <- imwd(lennon)
#
# Now let's see how many horizontal detail coefficients there are at
# scale 6
#
length(lwd$w6L1)
# [1] 4096
#
# So the horizontal detail ``image'' at scale contains 64x64=4096 coefficients.
# A lot!
#
# Now, suppose we threshold this
# two-dimensional wavelet decomposition object
#
lwdT <- threshold(lwd)
#
# First of all. What is the class of the detail coefficients now?
#
class(lwdT$w6L1)
# [1] "compressed"
#
```

```
# Aha. So this set of coefficients got compressed using the
# compress.default function.
#
# How many coefficients are being stored here?
#
lwdT$w6L1
# $position:
# [1]  173 2829 2832 2846
#
# $values:
# [1]  141.5455 -190.2810 -194.5714 -177.1791
#
# $original.length:
# [1] 4096
#
# attr(, "class"):
# [1] "compressed"
#
# Wow! Only 4 coefficients are not zero. Wicked compression!
```

---

imwr *Inverse two-dimensional wavelet transform.*

Description

Perform inverse two-dimensional wavelet transform using Mallat's, 1989 algorithm.

This function is generic.

Particular methods exist. For the `imwd` class object this generic function uses `imwr.imwd`. For the `imwdc` class object this generic function uses `imwr.imwdc`.

Usage

```
imwr(...)
```

Arguments

`...`	See individual help pages for details.

Details

See individual method help pages for operation and examples.

Value

A square matrix whose side length is a power of two that represents the inverse 2D wavelet transform of the input object x.

RELEASE

Version 2 Copyright Guy Nason 1993

Author(s)

G P Nason

See Also

imwd, imwr.imwd, imwr.imwdc.

---

imwr.imwd *Inverse two-dimensional discrete wavelet transform.*

Description

This functions performs the reconstruction stage of Mallat's pyramid algorithm (i.e. the inverse discrete wavelet transform) for images.

Usage

```
## S3 method for class 'imwd'
imwr(imwd, bc=imwd$bc, verbose=FALSE, ...)
```

Arguments

imwd	An object of class 'imwd'. This type of object is returned by 'imwd'.
bc	This argument specifies the boundary handling, it is best left to be the boundary handling specified by that in the supplied imwd (as is the default).
verbose	If this argument is true then informative messages are printed detailing the computations to be performed
...	any other arguments

Details

Details of the algorithm are to be found in Mallat (1989). Similarly to the decomposition function, imwd the inverse algorithm works by applying many 1D reconstruction algorithms to the coefficients. The filters in these 1D reconstructions are incorporated in the supplied imwd.object and originally created by the filter.select function in WaveThresh3.

This function is a method for the generic function imwr for class imwd.object. It can be invoked by calling imwr for an object of the appropriate class, or directly by calling imwr.imwd regardless of the class of the object.

Value

A matrix, of dimension determined by the original data set supplied to the initial decomposition (more precisely, determined by the `nlevelsWT` component of the `imwd.object`). This matrix is the highest resolution level of the reconstruction. If a `imwd` two-dimensional wavelet transform is followed immediately by a `imwr` inverse two-dimensional wavelet transform then the returned matrix will be exactly the same as the original image.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

`imwd`, `imwd.object`, `imwr`.

Examples

```
#
# Do a decomposition, then exact reconstruction
# Look at the error
#
test.image <- matrix(rnorm(32*32), nrow=32)
#
# Test image is just some sort of  square matrix whose side length
# is a power of two.
#
max( abs(imwr(imwd(test.image)) - test.image))
# [1] 1.014611e-11
```

`imwr.imwdc` *Inverse two-dimensional discrete wavelet transform.*

Description

Inverse two-dimensional discrete wavelet transform.

Usage

```
## S3 method for class 'imwdc'
imwr(imwd, verbose=FALSE, ...)
```

Arguments

imwd	An object of class imwdc. This type of object is returned by threshold.imwd and is a compress.imwd compressed version of an imwd object.
verbose	If this argument is true then informative messages are printed detailing the computations to be performed
...	other arguments to supply to the imwr function which is called after uncompressing the imwdc object.

Details

This function merely uncompresses the supplied imwdc.object and passes the resultant imwd object to the imwr.imwd function.

This function is a method for the generic function imwr for class imwdc.object. It can be invoked by calling imwr for an object of the appropriate class, or directly by calling imwr.imwdc regardless of the class of the object.

Value

A matrix, of dimension determined by the original data set supplied to the initial decomposition (more precisely, determined by the nlevelsWT component of the imwdc.object). This matrix is the highest resolution level of the reconstruction.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

compress.imwd, imwd, imwd.object, imwr.

Examples

```
#
# Do a decomposition, thresholding, then exact reconstruction
# Look at the error
#

test.image <- matrix(rnorm(32*32), nrow=32)

# Test image is just some sort of  square matrix whose side length
# is a power of two.
#
max( abs(imwr(threshold(imwd(test.image))) - test.image))
# [1] 62.34
#
```

```
# The answer is not zero (see contrasting examples in the help page for
# imwr.imwd because we have thresholded the
# 2D wavelet transform here).
```

`InvBasis` *Generic basis inversion for libraries*

Description

Will invert either a `wst` or `wp` object given that object and some kind of basis specification.

Usage

```
InvBasis(...)
```

Arguments

`...`	Usually a library representation and a basis specification

Details

Description says it all

Value

The reconstruction.

Author(s)

G P Nason

See Also

`InvBasis.wp`,`InvBasis.wst`,`MaNoVe`,`numtonv`

InvBasis.wp *Invert a wp library representation with a particular basis spec*

Description

Inverts a wp basis representation with a given basis specification, for example an output from the MaNoVe function.

Usage

```
## S3 method for class 'wp'
InvBasis(wp, nvwp, pktlist, verbose=FALSE, ...)
```

Arguments

wp	The wavelet packet object you wish to invert.
nvwp	A basis specification in the format of a node vector (wp) object, obtained, eg by the MaNoVe.wp function
pktlist	Another way of specifying the basis. If this argument is not specified then it is generated automatically from the nvwp argument. If it is specified then it overrides the one generated by nvwp
verbose	If TRUE then informative messages are printed.
...	Other arguments, not used

Details

Objects arising from a wp.object specification are a representation of a signal with respect to a library of wavelet packet basis functions. A particular basis specification can be obtained using the numtonv function which can pick an indexed basis function, or MaNoVe.wp which uses the Coifman-Wickerhauser minimum entropy method to select a basis. This function takes a wp.object and a particular basis description (in a nv.object node vector object) and inverts the representation with respect to that selected basis.

The function can alternatively take a packet list pktlist specification which overrides the node vector if supplied. If the pktlist is missing then one is generated internally from the nvwp object using the print.nvwp function.

Value

The inverted reconstruction

Author(s)

G P Nason

See Also

InvBasis,MaNoVe.wp,numtonv,print.nvwp,wp

Examples

```
#
# The example in InvBasis.wst can be used here, but replaced wst by wp
#
```

`InvBasis.wst` *Invert a wst library representation with a basis specification*

Description

Inverts a wst basis representation with a given basis specification, for example an output from the `MaNoVe` function.

Usage

```
## S3 method for class 'wst'
InvBasis(wst, nv, ...)
```

Arguments

`wst`	The wst object that you wish to invert
`nv`	The node vector, basis spec, that you want to pick out
`...`	Other arguments, that don't do anything here

Details

Objects arising from a `wst.object` specification are a representation of a signal with respect to a library of basis functions. A particular basis specification can be obtained using the `numtonv` function which can pick an indexed basis function, or `MaNoVe.wst` which uses the Coifman-Wickerhauser minimum entropy method to select a basis. This function takes a `wst.object` and a particular basis description (in a `nv.object` node vector object) and inverts the representation with respect to that selected basis.

Value

The inverted reconstruction

Author(s)

G P Nason

See Also

`numtonv`,`nv.object`,`MaNoVe.wst`,`threshold.wst`,`wst`

Examples

```
#
# Let's generate a noisy signal
#
x <- example.1()$y + rnorm(512, sd=0.2)
#
# You can plot this if you like
#
## Not run: ts.plot(x)
#
# Now take the nondecimated wavelet transform
#
xwst <- wst(x)
#
# Threshold it
#
xwstT <- threshold(xwst)
#
# You can plot this too if you like
#
## Not run: plot(xwstT)
#
# Now use Coifman-Wickerhauser to get a "good" basis
#
xwstTNV <- MaNoVe(xwstT)
#
# Now invert the thresholded wst using this basis specification
#
xTwr <- InvBasis(xwstT, xwstTNV)
#
# And plot the result, and superimpose the truth in dotted
#
## Not run: ts.plot(xTwr)
## Not run: lines(example.1()$y, lty=2)
```

`ipd` *Inductance plethysmography data.*

Description

Inductance plethysmography trace.

Usage

```
data(ipd)
```

Author(s)

G P Nason

Source

This data set contains 4096 observations of inductance plethsymography data sampled at 50Hz starting at 1229.98 seconds. This is a regular time series object.

I am grateful to David Moshal and Andrew Black of the Department of Anaesthesia, University of Bristol for permission to include this data set.

This data set was used in Nason, 1996 to illustrate noise reduction with wavelet shrinkage and using cross-validation for choosing the threshold.

A plethysmograph is an apparatus for measuring variations in the size of parts of the body. In this experiment the inductance plethysmograph consists of a coil of wire encapsulated in a belt. A radio-frequency carrier signal is passed through the wire and size variations change the inductance of the coil that can be detected as a change in voltage. When properly calibrated the output voltage of the inductance plethysmograph is proportional to the change in volume of the part of the body under examination.

It is of both clinical and scientific interest to discover how anaesthetics or analgesics may alter normal breathing patterns post-operatively. Sensors exist that measure blood oxygen saturation but by the time they indicate critically low levels the patient is often apnoeic (cease breathing) and in considerable danger. It is possible for a nurse to continually observe a patient but this is expensive, prone to error and requires training. In this examples the plethysmograph is arranged around the chest and abdomen of a set of patients and is used to measure the flow of air during breathing. The recordings below were made by the Department of Anaesthesia at the Bristol Royal Infirmary after the patients had undergone surgery under general anaesthetic. The data set (shown below) shows a section of plethysmograph recording lasting approximately 80 seconds. The two main sets of regular oscillations correspond to normal breathing. The disturbed behaviour in the centre of the plot where the normal breathing pattern disappears corresponds to the patient vomiting.

Examples

```
#
data(ipd)
## Not run: ts.plot(ipd)
```

`ipndacw` *Compute inner product matrix of discrete non-decimated autocorrelation wavelets.*

Description

This function computes the inner product matrix of discrete non-decimated autocorrelation wavelets.

Usage

```
ipndacw(J, filter.number = 10, family = "DaubLeAsymm", tol = 1e-100, verbose
         = FALSE, ...)
```

Arguments

`J`	Dimension of inner product matrix required. This number should be a negative integer.
`filter.number`	The index of the wavelet used to compute the inner product matrix.
`family`	The family of wavelet used to compute the inner product matrix.
`tol`	In the brute force computation for Daubechies compactly supported wavelets many inner product computations are performed. This tolerance discounts any results which are smaller than `tol` which effectively defines how long the inner product/autocorrelation products are.
`verbose`	If `TRUE` then informative messages are printed. Some of these can be quite fun as the function tells you whether precomputed matrices are being used, how much computation needs to be done and so forth.
`...`	any other arguments

Details

This function computes the inner product matrix of the discrete non-decimated autocorrelation wavelets. This matrix is used to correct the wavelet periodogram as a step to turning it into a evolutionary wavelet spectral estimate. The matrix returned by ipndacw is the one called A in the paper by Nason, von Sachs and Kroisandt.

For the Haar wavelet the matrix is computed by using the analytical formulae in the paper by Nason, von Sachs and Kroisandt and is hence very fast and efficient and can be used for large values of -J.

For other Daubechies compactly supported wavelets the matrix is computed directly by autocorrelating discrete non-decimated wavelets at different scales and then forming the inner products of these. A function that computes the autocorrelation wavelets themselves is `PsiJ`. This *brute force* computation is slow and memory inefficient hence `ipndacw` contains a mechanism that stores any inner product matrix that it creates according to a naming scheme defined by the convention defined in `rmname`. The stored matrices are assigned to the user-visible environment `WTEnv`.

These stored matrices can be used in future computations by the following automatic procedure:

1 The `rmget` looks to see whether previous computations have been performed that might be useful.

2 If a matrix of higher order is discovered then the appropriate top-left submatrix is returned, otherwise...

3 If the right order of matrix is found it is returned, otherwise ...

4 If a matrix of *smaller* order is found it is used as the top-left submatrix of the answer. The remaining elements to the right of and below the submatrix are computed and then the whole matrix is returned, otherwise...

5 If none are found then the whole matrix is computed in C and returned.

In this way a particular matrix for a given wavelet need only be computed once.

Value

A matrix of order (-J)x(-J) containing the inner product matrix of the discrete non-decimated autocorrelation matrices.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

See Also

ewspec, PsiJ, rmname, rmget, filter.select.

Examples

```
#
# Let us create the 4x4 inner product matrix for the Haar wavelet.
# We'll turn on the jolly verbose messages as well.
#
ipndacw(-4, filter.number=1, family="DaubExPhase", verbose=TRUE)
#Computing ipndacw
#Calling haarmat
#Took  0.0699999  seconds
#       -1     -2     -3     -4
#-1 1.5000 0.7500 0.3750 0.1875
#-2 0.7500 1.7500 1.1250 0.5625
#-3 0.3750 1.1250 2.8750 2.0625
#-4 0.1875 0.5625 2.0625 5.4375
#
# If we do this again it will use the precomputed version
#
ipndacw(-4, filter.number=1, family="DaubExPhase", verbose=TRUE)
#Computing ipndacw
#Returning precomputed version: using  4
#Took  0.08  seconds
#       -1     -2     -3     -4
#-1 1.5000 0.7500 0.3750 0.1875
#-2 0.7500 1.7500 1.1250 0.5625
#-3 0.3750 1.1250 2.8750 2.0625
#-4 0.1875 0.5625 2.0625 5.4375
#
# Let's use a smoother wavelet from the least-asymmetric family
# and generate the 6x6 version.
#
ipndacw(-6, filter.number=10, family="DaubLeAsymm", verbose=TRUE)
#Computing ipndacw
#Took  0.95  seconds
#             -1           -2           -3           -4           -5
```

```
#-1 1.839101e+00 3.215934e-01 4.058155e-04 8.460063e-06 4.522125e-08
#-2 3.215934e-01 3.035353e+00 6.425188e-01 7.947454e-04 1.683209e-05
#-3 4.058155e-04 6.425188e-01 6.070419e+00 1.285038e+00 1.589486e-03
#-4 8.460063e-06 7.947454e-04 1.285038e+00 1.214084e+01 2.570075e+00
#-5 4.522125e-08 1.683209e-05 1.589486e-03 2.570075e+00 2.428168e+01
#-6 5.161675e-10 8.941666e-08 3.366416e-05 3.178972e-03 5.140150e+00
#             -6
#-1 5.161675e-10
#-2 8.941666e-08
#-3 3.366416e-05
#-4 3.178972e-03
#-5 5.140150e+00
#-6 4.856335e+01
#
```

`irregwd` *Irregular wavelet transform (decomposition).*

Description

This function performs the irregular wavelet transform as described in the paper by Kovac and Silverman.

Usage

```
irregwd(gd, filter.number=2, family="DaubExPhase", bc="periodic", verbose=FALSE)
```

Arguments

`gd`	A grid structure which is the output of the `makegrid` function.
`filter.number`	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 2, the Daubechies extremal phase orthonormal compactly supported wavelet with 2 vanishing moments.
`family`	specifies the family of wavelets that you want to use. Two popular options are "DaubExPhase" and "DaubLeAsymm" but see the help for `filter.select` for more possibilities.
`bc`	specifies the boundary handling. If `bc="periodic"` the default, then the function you decompose is assumed to be periodic on it's interval of definition, if `bc="symmetric"` then the function beyond its boundaries is assumed to be a symmetric reflection of the function in the boundary. The symmetric option was the implicit default in releases prior to 2.2.
`verbose`	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.

Details

If one has irregularly spaced one-dimensional regression data (t,y), say. Then the function makegrid interpolates this to a regular grid and then the standard wavelet transform is used to transform the interpolated data. However, unlike the standard wavelet denoising set-up the interpolated data, y, values are correlated. Hence the wavelet coefficients of the interpolated will be correlated (even after using an orthogonal transform). Hence, in particular, the variance of each wavelet coefficient may well be different and so this routine also computes those variances using a fast algorithm (related to the two-dimensional wavelet transform).

When thresholding with threshold.irregwd the threshold function makes use of the information about the variance of each coefficient to modify the variance locally on a coefficient by coefficient basis.

Value

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

C	Vector of sets of successively smoothed versions of the interpolated data (see description of equivalent component of wd.object for further information.)
D	Vector of sets of wavelet coefficients of the interpolated data at different resolution levels. (see description of equivalent component of wd.object for further information.)
c	Vector that aids in calculation of variances of wavelet coefficients (used by threshold.irregwd).
nlevelsWT	The number of resolution levels. This depends on the length of the data vector. If length(data)=2^m, then there will be m resolution levels. This means there will be m levels of wavelet coefficients (indexed 0,1,2,...,(m-1)), and m+1 levels of smoothed data (indexed 0,1,2,...,m).
fl.dbase	There is more information stored in the C and D than is described above. In the decomposition "extra" coefficients are generated that help take care of the boundary effects, this database lists where these start and finish, so the "true" data can be extracted.
filter	A list containing information about the filter type: Contains the string "wavelet" or "station" depending on which type of transform was performed.
bc	How the boundaries were handled.
date	The date the transform was performed.

RELESASE

3.9.4 Code Copyright Arne Kovac 1997

Author(s)

Arne Kovac

See Also

makegrid, wd, wr.wd, accessC, accessc, accessD, putD, putC, filter.select, plot.irregwd, threshold.irregwd.

Examples

```
#
# See full examples at the end of the help for makegrid.
#
```

`irregwd.objects` *Irregular wavelet decomposition objects.*

Description

These are objects of classes

`wd`

They represent a decomposition of a function with respect to a wavelet basis. The function will have been interpolated to a grid and these objects represent the discrete wavelet transform `wd`.

Details

To retain your sanity the C and D coefficients should be extracted by the `accessC` and `accessD` functions and inserted using the `putC` and `putD` functions (or more likely, their methods), rather than by the `$` operator.

One can use the `accessc` function to obtain the `c` component.

Mind you, if you want to muck about with coefficients directly, then you'll have to do it yourself by working out what the fl.dbase list means (see `first.last` for a description.)

GENERATION

This class of objects is returned from the `irregwd` function. Some other functions that process these kinds of objects also return this class of object (such as `threshold.irregwd`.)

METHODS

The `irregwd` class of objects has methods for the following generic functions: `plot`, `threshold`,

STRUCTURE

All components in a legitimate 'irregwd' are identical to the components in an ordinary `wd.object` with the exception of `type` component and with the addition of the following component:

c vector that aids in the calculation of variances of wavelet coefficients (used by `threshold.irregwd`).

RELEASE

Version 3.9.4 Copyright Arne Kovac 1997, Help Copyright Guy Nason 2004

Author(s)

G P Nason

See Also

irregwd, threshold.irregwd, plot.irregwd,wd

IsEarly *Generic function to detect whether object is from an early version*

Description

Generic function to detect whether object is from an early version of WaveThresh

Usage

```
IsEarly(x)
```

Arguments

x	The object you want to see whether its from an early version

Details

Description says all

Value

Returns TRUE if object is from an earlier version of WaveThresh, FALSE if not.

Author(s)

G P Nason

See Also

ConvertMessage,IsEarly.default,IsEarly, IsEarly.wd

IsEarly.default — *Detects whether object is from an earlier version of WaveThresh*

Description

Detects whether object is from an earlier version of WaveThresh.

Usage

```
## Default S3 method:
IsEarly(x)
```

Arguments

x	Object to discern

Details

The default method always returns FALSE, i.e. unless the object is of a specific type handled by a particular method then it won't be from an earlier version.

Value

Always FALSE for the generic

Author(s)

G P Nason

See Also

IsEarly

IsEarly.wd — *Function to detect whether a wd object is from WaveThresh2 or not*

Description

Function to detect whether a wd object is from WaveThresh2 or not.

Usage

```
## S3 method for class 'wd'
IsEarly(x)
```

Arguments

x	The wd object that you are trying to check

Details

The function merely looks to see whether the wd object has a component called date. If it does not then it is from version 2. This routine is legacy and not very important anymore.

Value

Returns TRUE if from an earlier version of WaveThresh (v2), returns FALSE if not.

Author(s)

G P Nason

See Also

IsEarly

IsPowerOfTwo — *Decides whether vector elements are integral powers of two (returns NA if not).*

Description

This function checks to see whether its input is a power of two. If it is then it returns that power otherwise it returns NA.

Usage

```
IsPowerOfTwo(n)
```

Arguments

n	Vector of numbers that are to be checked whether it is a power of two.

Details

Function takes the log of the input, divides this by log(2) and if the result is integral then it knows the input is true power of two.

Value

If n is a power of two, then the power is returned otherwise NA is returned.

RELEASE

Version 3.6.0 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

nlevelsWT.default.

Examples

```
#
# Try and see whether 1,2,3 or 4 are powers of two!
#
IsPowerOfTwo(1:4)
# [1]  0  1 NA  2
#
# Yes, 1,2 and 4 are the 0, 1 and 2nd power of 2. However, 3 is not an
# integral power of two.
```

---

l2norm — *Compute L2 distance between two vectors of numbers.*

Description

Compute L2 distance between two vectors of numbers (square root of sum of squares of differences between two vectors).

Usage

```
l2norm(u,v)
```

Arguments

u	first vector of numbers
v	second vector of numbers

Details

Function simply computes the L2 distance between two vectors and is implemented as

`sqrt(sum((u-v)^2))`

Value

A real number which is the L2 distance between two vectors.

RELEASE

Version 3.6 Copyright Guy Nason 1995

Note

This function would probably be more accurate if it used the Splus function `vecnorm`.

Author(s)

G P Nason

See Also

`linfnorm`, `wstCV`, `wstCVl`.

Examples

```
#
# What is the L2 norm between the following sets of vectors
#
p <- c(1,2,3,4,5)
q <- c(1,2,3,4,5)
r <- c(2,3,4,5,6)
l2norm(p,q)
# [1] 0
l2norm(q,r)
# [1] 2.236068
l2norm(r,p)
# [1] 2.236068
```

`lennon` *John Lennon image.*

Description

A 256x256 matrix. Each entry of the matrix contains an image intensity value. The whole matrix represents an image of John Lennon

Usage

```
data(lennon)
```

Format

A 256x256 matrix. Each entry of the matrix contains an image intensity value. The whole matrix represents an image of John Lennon

Author(s)

G P Nason

Source

The John Lennon image was supplied uncredited on certain UNIX workstations as an examples image. I am not sure who the Copyright belongs to. Please let me know if you know

Examples

```
#
# This command produces the image seen above.
#
# image(lennon)
#
```

`levarr` *Subsidiary routine that generates a particular permutation*

Description

Not intended for casual user use. This function is used to provide the partition to reorder `wst.object` into `wd.object` (nondecimated time ordered) objects.

Usage

```
levarr(v, levstodo)
```

Arguments

`v`	the vector to permute
`levstodo`	the number of levels associated with the current level in the object you wish to permute

Details

Description says all

Value

A permutation of the `v` vector according to the number of levels that need handling

Author(s)

G P Nason

See Also

`getarrvec`, `convert.wd`, `convert.wst`

Examples

```
levarr(1:4, 3)
# [1] 1 3 2 4
```

`linfnorm` *Compute L infinity distance between two vectors of numbers.*

Description

Compute L infinity distance between two vectors of numbers (maximum absolute difference between two vectors).

Usage

```
linfnorm(u,v)
```

Arguments

`u`	first vector of numbers
`v`	second vector of numbers

Details

Function simply computes the L infinity distance between two vectors and is implemented as

`max(abs(u-v))`

Value

A real number which is the L infinity distance between two vectors.

RELEASE

Version 3.6 Copyright Guy Nason 1995

Note

This function would probably be more accurate if it used the Splus function `vecnorm`.

Author(s)

G P Nason

See Also

`l2norm`, `wstCV`, `wstCVl`.

Examples

```
#
# What is the L infinity norm between the following sets of vectors
#
p <- c(1,2,3,4,5)
q <- c(1,2,3,4,5)
r <- c(2,3,4,5,6)
linfnorm(p,q)
# [1] 0
linfnorm(q,r)
# [1] 1
linfnorm(r,p)
# [1] 1
```

`LocalSpec` *Compute Nason and Silverman smoothed wavelet periodogram.*

Description

This function is obsolete. Use the function `ewspec`. Performs the Nason and Silverman smoothed wavelet periodogram as described in Nason and Silverman (1995).

This function is generic.

Particular methods exist. For the wd class object this generic function uses `LocalSpec.wd`.

Usage

```
LocalSpec(...)
```

Arguments

...	See individual help pages for details.

Details

See individual method help pages for operation and examples.

Value

The LocalSpec of the wavelet object supplied. See method help files for examples.

RELEASE

Version 3.9 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

wd, wd.object, LocalSpec.wd

LocalSpec.wd — *Compute Nason and Silverman raw or smoothed wavelet periodogram.*

Description

This smoothing in this function is now obsolete. You should now use the function ewspec.

This function computes the Nason and Silverman raw or smoothed wavelet periodogram as described by Nason and Silverman (1995).

Usage

```
## S3 method for class 'wd'
LocalSpec(wdS, lsmooth="none", nlsmooth=FALSE, prefilter=TRUE,
  verbose=FALSE, lw.number=wdS$filter$filter.number,
  lw.family=wdS$filter$family, nlw.number=wdS$filter$filter.number,
  nlw.family=wdS$filter$family, nlw.policy="LSuniversal",
  nlw.levels=0:(nlevelsWT(wdS) - 1), nlw.type="hard", nlw.by.level=FALSE,
  nlw.value=0, nlw.dev=var, nlw.boundary=FALSE, nlw.verbose=FALSE,
  nlw.cvtol=0.01, nlw.Q=0.05, nlw.alpha=0.05, nlw.transform=I,
  nlw.inverse=I, debug.spectrum=FALSE, ...)
```

Arguments

Note that all options beginning "nlw" are only used if nlsmooth=T, i.e. iff NONLINEAR wavelet smoothing is used.

wdS	The stationary wavelet transform object that you want to smooth or square.
lsmooth	Controls the LINEAR smoothing. There are three options: "none", "Fourier" and "wavelet". They are described below. Note that Fourier begins with a capital "F".
nlsmooth	A switch to turn on (or off) the NONLINEAR wavelet shrinkage of (possibly LINEAR smoothed) local power coefficients. This option is either TRUE (to turn on the smoothing) or FALSE (to turn it off).
prefilter	If TRUE then apply a prefilter to the actual stationary wavelet coefficients at each level. This is a low-pass filter that cuts off all frequencies above the highest frequency allowed by the (Littlewood-Paley) wavelet that bandpassed the current level coefficients. If FALSE then no prefilter is applied.
verbose	If TRUE then the function chats about what it is doing. Otherwise it is silent.
lw.number	If wavelet LINEAR smoothing is used then this option controls the filter number of the wavelet within the family used to perform the LINEAR wavelet smoothing.

`lw.family` If wavelet LINEAR smoothing is used then this option controls the `family` of the wavelet used to perform the LINEAR wavelet smoothing.

`nlw.number` If NONLINEAR wavelet smoothing is also used then this option controls the `filter number` of the wavelet used to perform the wavelet shrinkage.

`nlw.family` If NONLINEAR wavelet smoothing is also used then this option controls the `family` of the wavelet used to perform the wavelet shrinkage.

`nlw.policy` If NONLINEAR wavelet smoothing is also used then this option controls the levels to use when performing wavelet shrinkage (see `threshold.wd` for different policy choices).

`nlw.levels` If NONLINEAR wavelet smoothing is also used then this option controls the levels to use when performing wavelet shrinkage (see `threshold.wd` for a detailed description of how levels can be chosen).

`nlw.type` If NONLINEAR wavelet smoothing is also used then this option controls the type of thresholding used in the wavelet shrinkage (either "hard" or "soft", but see `threshold.wd` for a list).

`nlw.by.level` If NONLINEAR wavelet smoothing is also used then this option controls whether level-by-level thresholding is used or if one threshold is chosen for all levels (see `threshold.wd`).

`nlw.value` If NONLINEAR wavelet smoothing is also used then this option controls if a manual (or similar) policy is supplied to `nlw.policy` then the nlw.value option carries the manual threshold value (see `threshold.wd`).

`nlw.dev` If NONLINEAR wavelet smoothing is also used then this option controls the type of variance estimator that is used in wavelet shrinkages (see `threshold.wd`). One possibility is the Splus var() function, another is the WaveThresh function `madmad`().

`nlw.boundary` If NONLINEAR wavelet smoothing is also used then this option controls whether boundary coefficients are also thresholded (see `threshold.wd`).

`nlw.verbose` If NONLINEAR wavelet smoothing is also used then this option controls whether the threshold function prints out messages as it thresholds levels (see `threshold.wd`).

`nlw.cvtol` If NONLINEAR wavelet smoothing is also used then this option controls the optimization tolerance is cross-validation wavelet shrinkage is used (see `threshold.wd`)

`nlw.Q` If NONLINEAR wavelet smoothing is also used then this option controls the Q value for wavelet shrinkage (see `threshold.wd`).

`nlw.alpha` If NONLINEAR wavelet smoothing is also used then this option controls the alpha value for wavelet shrinkage (see `threshold.wd`).

`nlw.transform` If NONLINEAR wavelet smoothing is also used then this option controls a transformation that is applied to the squared (and possibly linear smoothed) stationary wavelet coefficients before shrinkage. So, for examples, you might want to set `nlw.transform=log` to perform wavelet shrinkage on the logs of the squared (and possibly linear smoothed) stationary wavelet coefficients.

`nlw.inverse` If NONLINEAR wavelet smoothing is also used then this option controls the inverse transformation that is applied to the wavelet shrunk coefficients before they are put back into the stationary wavelet transform structure. So, for examples, if the `nlw.transform` is `log()` you should set the inverse to `nlw.inverse=exp`.

`debug.spectrum` If this option is `T` then spectrum plots are produced at each stage of the squaring/smoothing. Therefore if you put in the non-decimated wavelet transform of white noise you can get a fair idea of how the coefficients are filtered at each stage.

`...` any other arguments

Details

This smoothing in this function is now obsolete. Use the function `ewspec` instead. However, this function is still useful for computing the raw periodogram.

This function attempts to produce a picture of local time-scale power of a signal. There are two main components to this function: linear smoothing of squared coefficients and non-linear smoothing of these. Neither, either or both of these components may be used to process the data. The function expects a non-decimated wavelet transform object (of class wd, type="station") such as that produced by the `wd`() function with the type option set to "`station`". The following paragraphs describe the various methods of smoothing.

LINEAR SMOOTHING. There are three varieties of linear smoothing. None simply squares the coefficients. Fourier and wavelet apply linear smoothing methods in accordance to the prescription given in Nason and Silverman (1995). Each level in the SWT corresponds to a band-pass filtering to a frequency range [sl, sh]. After squaring we obtain power in the range [0, 2sl] and [2sl, 2sh]. The linear smoothing gets rid of the power in [2sl, 2sh]. The Fourier method simply applies a discrete Fourier transform (rfft) and cuts off frequencies above 2sl. The wavelet method is a bit more suble. The DISCRETE wavelet transform is taken of a level (i) and all levels within the DWT, j, where j>i are set to zero and then the inverse is taken. Approximately this performs the same operation as the Fourier method only faster. By default the same wavelets are used to perform the linear smoothing as were used to compute the stationary wavelet transform in the first place. This can be changed by altering `lw.number` and `lw.family`.

NONLINEAR SMOOTHING. After either of the linear smoothing options above it is possible to use wavelet shrinkage upon each level in the squared (and possibly Fourier or wavelet linear smoothed) to denoise the coefficients. This process is akin to smoothing the ordinary periodogram. All the usual wavelet shrinkage options are available as `nlw`.* where * is one of the usual `threshold.wd` options. By default the same wavelets are used to perform the wavelet shrinkage as were used to compute the non-decimated wavelet transform. These wavelets can be replaced by altering `nlw.number` and `nlw.family`. Also, it is possible to transform the squared (and possibly smoothed coefficients) before applying wavelet shrinkage. The transformation is effected by supplying an appropriate transformation function (AND ITS INVERSE) to `nlw.transform` and `nlw.inverse`. (For examples, `nlw.transform=log` and `nlw.inverse=exp` might be a good idea).

Value

An object of class `wd` a time-ordered non-decimated wavelet transform. Each level of the returned object contains a smoothed wavelet periodogram. Note that this is **not** the *corrected* smoothed wavelet periodogram, or the *evolutionary wavelet spectrum*. Use the function `ewspec` to compute the evolutionary wavelet spectrum.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason and Silverman, (1995).

See Also

ewspec,

Examples

```
#
# This function is obsolete. See ewspec()
#
# Compute the raw periodogram of the BabyECG
# data using the Daubechies least-asymmetric wavelet $N=10$.
#
data(BabyECG)
babywdS <- wd(BabyECG, filter.number=10, family="DaubLeAsymm", type="station")
babyWP <- LocalSpec(babywdS, lsmooth = "none", nlsmooth = FALSE)
## Not run: plot(babyWP, main="Raw Wavelet Periodogram of Baby ECG")
#
# Note that the lower levels of this plot are too large. This is partly because
# there are "too many" coefficients at the lower levels. For a better
# picture of the local spectral properties of this time series see
# the examples section of ewspec
#
# Other results of this function can be seen in the paper by
# Nason and Silverman (1995) above.
#
```

LocalSpec.wst *Obsolete function (use ewspec)*

Description

This function computes a local spectra as described in Nason and Silverman (1995). However, the function is obsolete and superceded by ewspec.

Usage

```
## S3 method for class 'wst'
LocalSpec(wst, ...)
```

Arguments

wst	The wst object to perform local spectral analysis on
...	Other arguments to LocalSpec.wd.

Details

Description says it all.

However, this function converts the `wst.object` object to a nondecimated `wd.object` and then calls `LocalSpec.wd`.

Value

Same value as `LocalSpec.wd`.

Author(s)

G P Nason

See Also

`ewspec`

---

logabs — *Take the logarithm of the squares of the argument*

Description

Take the log of the squares of the argument

Usage

```
logabs(x)
```

Arguments

x	A number

Details

Description says all

Value

Just the logarithm of the square of the argument

Author(s)

G P Nason

See Also

`image.wd`, `image.wst`

Examples

```
logabs(3)
# [1] 1.098612
```

LSWsim *Simulate arbitrary locally stationary wavelet process.*

Description

Simulates an arbitrary LSW process given a spectrum.

Usage

```
LSWsim(spec)
```

Arguments

spec	An object of class wd (the NDWT kind) which contains the spectral information for simulating your process. See examples below on how to create and manipulate this object.

Details

This function uses a spectral definition in spec to simulate a locally stationary wavelet process (defined by the Nason, von Sachs and Kroisandt, 2000, JRSSB paper).

The input object, spec, is a wd class object which contains a spectral description. In particular, all coefficients must be nonnegative and LSWsim() checks for this and returns an error if it is not so. Other than that the spectrum can contain pretty much anything. An object of this type can be easily created by the convenience routine cns. This creates an object of the correct structure but all elements are initially set to zero. The spectrum structure spec can then be filled by using the putD function.

The function works by first checking for non-negativity. Then it takes the square root of all coefficients. Then it multiplies all coefficients by a standard normal variate (from rnorm()) and multiples the finest level by 2, the next finest by 4, the next by 8 and so on. (This last scalar multiplication is intended to undo the effect of the average basis averaging which combines cofficients but divides by two at each combination). Finally, the modified spectral object is subjected to the convert function which converts the object from a wd time-ordered NWDT object to a wst packet-ordered object which can then be inverted using AvBasis.

Note that the NDWT transforms in WaveThresh are periodic so that the process that one simulates with this function is also periodic.

Value

A vector simulated from the spectral description given in the spec description. The returned vector will exhibit the spectral characteristics defined by spec.

RELEASE

Version 3.9 Copyright Guy Nason 2004

Author(s)

G P Nason

See Also

wd.object, putD, cns, AvBasis, convert, ewspec, plot.wst,

Examples

```
#
# Suppose we want to create a LSW process of length 1024 and with a spectral
# structure that has a squared sinusoidal character at level 4 and a burst of
# activity from time 800 for 100 observations at scale 9 (remember for a
# process of length 1024 there will be 9 resolution levels (since 2^10=1024)
# where level 9 is the finest and level 0 is the coarsest).
#
# First we will create an empty spectral structure for series of 1024 observations
#
#
myspec <- cns(1024)
#
# If you plot it you'll get a null spectrum (since every spectral entry is zero)
#
## Not run: plot(myspec, main="My Spectrum")
#
#
# Now let's add the desired spectral structure
#
# First the squared sine (remember spectra are positive)
#
myspec <- putD(myspec, level=4, sin(seq(from=0, to=4*pi, length=1024))^2)
#
# Let's create a burst of spectral info of size 1 from 800 to 900. Remember
# the whole vector has to be of length 1024.
#
burstat800 <- c(rep(0,800), rep(1,100), rep(0,124))
#
# Insert this (00000111000) type vector into the spectrum at fine level 9
#
myspec <- putD(myspec, level=9, v=burstat800)
#
# Now it's worth plotting this spectrum
#
## Not run: plot(myspec, main="My Spectrum")
#
# The squared sinusoid at level 4 and the burst at level 9 can clearly
# be seen
#
```

```
#
# Now simulate a random process with this spectral structure.
#
myLSWproc <- LSWsim(myspec)
#
# Let's see what it looks like
#
## Not run: ts.plot(myLSWproc)
#
#
# The burst is very clear but the sinusoidal structure is less apparent.
# That's basically it.
#
# You could now play with the spectrum (ie alter it) or simulate another process
# from it.
#
# [The following is somewhat of an aside but useful to those more interested
# in the LSW scene. We could now ask, so what? So you can simulate an
# LSW process. How can I be sure that it is doing so correctly? Well, here is
# a partial, computational, answer. If you simulate many realisations from the
# same spectral structure, estimate its spectrum, and then average those
# estimates then the average should tend to the spectrum you supplied. Here is a
# little function to do this (just for Haar but this function could easily be
# developed to be more general):
#
checkmyews <- function(spec, nsim=10){
        ans <- cns(2^nlevelsWT(spec))
        for(i in 1:nsim)        {
                cat(".")
                LSWproc <- LSWsim(spec)
                ews <- ewspec(LSWproc, filter.number=1, family="DaubExPhase",
                        WPsmooth=F)
                ans$D <- ans$D + ews$S$D
                ans$C <- ans$C + ews$S$C
        }
        ans$D <- ans$D/nsim
        ans$C <- ans$C/nsim
        ans
}
# If you supply it with a spectral structure (like myspec)
# from above and do enough simulations you'll get something looking like
# the original myspec structure. E.g. try
#
## Not run: plot(checkmyews(myspec, nsim=100))
##
# for fun. This type of check also gives you some idea of how much data
# you really need for LSW estimation for given spectral structures.]
#
```

`lt.to.name` *Convert desired level and orientation into code used by imwd*

Description

Function codes the name of a desired level and wavelet coefficient orientation into a string which is used by the 2D DWT functions to access and manipulate wavelet coefficients.

Usage

```
lt.to.name(level, type)
```

Arguments

`level`	Resolution level of coefficients that you want to extract or manipulate.
`type`	One of CC, CD, DC or DD indicating smoothed, horizontal, vertical or diagonal coefficients

Details

For the 1D wavelet transform (and others) the `accessC` and `accessD` function extracts wavelet coefficients from 1D wavelet decomposition objects. For `imwd.object` class objects, which are the 2D wavelet transforms of lattice objects (images) the wavelet coefficients are stored within components of the list object that underlies the imwd object.

This function provides an easy way to specify a resolution level and orientation in a human readable way and this function then produces the character string necessary to access the wavelet coefficients in an imwd object.

Note that this function *does not* actually extract any coefficients itself.

Value

A character string which codes the level and type of coefficients. It reads wXLY X is the resolution level and Y is an integer corresponding to the orientation (1=horizontal, 2=vertical, 3=diagonal, 4=smoothed).

Author(s)

G P Nason

See Also

`imwd`, `imwd.object`

Examples

```
#
# Generate the character string for the component of the imwd object
#
# The string associated with the diagonal detail at the third level...
#
lt.to.name(3, "DD")
# [1] "w3L3"
#
```

```
# Show how to access wavelet coefficients of imwd object.
#
# First, make up some data (using matrix/rnorm) and then subject it
# to an image wavelet transform.
#
tmpimwd <- imwd(matrix(rnorm(64),64,64))
#
# Get the horizontal coefficients at the 2nd level
#
tmpimwd[[ lt.to.name(2, "CD") ]]
# [1]  6.962251e-13  4.937486e-12  3.712157e-12 -3.064831e-12  6.962251e-13
# [6]  4.937486e-12  3.712157e-12 -3.064831e-12  6.962251e-13  4.937486e-12
# [11]  3.712157e-12 -3.064831e-12  6.962251e-13  4.937486e-12  3.712157e-12
# [16] -3.064831e-12
#
#
# If you want the coefficients returned as a matrix use the matrix function,
# i.e.
#
matrix(tmpimwd[[ lt.to.name(2, "CD") ]], 4,4)
#              [,1]          [,2]          [,3]          [,4]
#[1,]  6.962251e-13  6.962251e-13  6.962251e-13  6.962251e-13
#[2,]  4.937486e-12  4.937486e-12  4.937486e-12  4.937486e-12
#[3,]  3.712157e-12  3.712157e-12  3.712157e-12  3.712157e-12
#[4,] -3.064831e-12 -3.064831e-12 -3.064831e-12 -3.064831e-12
#
# Note that the dimensions of the matrix depend on the resolution level
# that you extract and dim = 2^level
```

---

madmad *Compute square of median absolute deviation (mad) function.*

Description

This function simply returns the square of the median absolute deviation (mad) function in S-Plus. This is required for supply to the `threshold` series of functions which require estimates of spread on the variance scale (not the standard deviation scale).

Usage

```
madmad(x)
```

Arguments

`x`	The vector for which you wish to compute the square of mad on.

Value

The square of the median absolute deviation of the coefficients supplied by `x`.

RELEASE

Version 3.4.1 Copyright Guy Nason 1994

Note

Its a MAD MAD world!

Author(s)

G P Nason

See Also

threshold

Examples

```
#
#
# Generate some normal data with mean 0 and sd of 8
# and we'll also contaminate it with an outlier of 1000000
# This is akin to signal wavelet coefficients mixing with the noise.
#
ContamNormalData <- c(1000000, rnorm(1000, mean=0, sd=8))
#
# What is the variance of the data?
#
var(ContamNormalData)
# [1] 999000792
#
# Wow, a seriously unrobust answer!
#
# How about the median absolute deviation?
#
mad(ContamNormalData)
# [1] 8.14832
#
# A much better answer!
#
# Now let's use madmad to get the answer on the variance scale
#
madmad(ContamNormalData)
# [1] 66.39512
#
# The true variance was 64, so the 66.39512 was a much better answer
# than that returned by the call to the variance function.
```

`make.dwwt` *Compute diagonal of the matrix WWT*

Description

Computes the values which specify the covariance structure of complex-valued wavelet coefficients.

Usage

```
make.dwwt(nlevels, filter.number = 3.1, family = "LinaMayrand")
```

Arguments

`nlevels` The number of levels of the wavelet decomposition.

`filter.number`, `family` Specifies the wavelet used; see filter.select for more details.

Details

If real-valued signals are decomposed by a discrete wavelet transform using a complex-valued Daubechies wavelet (as described by Lina & Mayrand (1995)), the resulting coefficients are complex-valued. The covariance structure of these coefficients are determined by the diagonal entries of the matrix WW^T. This function computes these values for use in shrinkage. For more details, see Barber & Nason (2004)

Value

A vector giving the diagonal elements of WW^T.

RELEASE

Part of the CThresh addon to WaveThresh. Copyright Stuart Barber and Guy Nason 2004.

Author(s)

Stuart Barber

See Also

`cthresh`

makegrid *Interpolate data to a grid.*

Description

This function takes a set of univariate (x,y) data with x arbitrary in (0,1) and linearly interpolates (x,y) to an equally spaced dyadic grid.

Usage

```
makegrid(t, y, gridn = 2^(floor(log(length(t)-1,2)) + 1))
```

Arguments

t	A vector of x data. Each of the entries of x must lie between 0 and 1.
y	A vector of y data. Each entry of y corresponds to the same-positioned entry in x and soy must be of the same length as x.
gridn	The number of grid points in the dyadic grid that the (x,y) gets interpolated to. By default this is the next power of two larger than the length of x.

Details

One method for performing wavelet regression on data that is not equally spaced nor of power of two length is that described in Kovac, (1997) and Kovac and Silverman, (2000).

The Kovac-Silverman algorithm linearly interpolates arbitrarily spaced (x,y) data to a dyadic grid and applies wavelet shrinkage to the interpolated data. However, if one assumes that the original data obeys a signal+noise model with iid data the interpolated data will be correlated due to the interpolation. This fact needs to be taken into account after taking the DWT and before thresholding one realizes that each coefficient has its own variance. The Kovac-Silverman algorithm computes this variance efficiently using knowledge of the interpolation scheme.

Value

An object of class `griddata`.

RELEASE

Version 3.9.6 Copyright Arne Kovac 1997 Copyright Guy Nason (help pages) 1999

Author(s)

Arne Kovac

See Also

accessc, irregwd, newsure, plot.irregwd, threshold.irregwd,

Examples

```
#
# Generate some values in (0,1), then sort them (for plotting)
#
tt <- sort(runif(100))
#
# Now evaluate the \code{\link{doppler}} function and add
# some noise.
#
yy <- doppler(tt) + rnorm(100, 0, 0.15)
#
# Now make the grid with this data
#
yygrid <- makegrid(t=tt, y=yy)
#
# Jolly good. Now let's take the wavelet transform of this gridded data.
# Note that we have to use the \code{\link{irregwd}} function
# of the gridded data as it computes the variances of the coefficients
# as well as the coefficients themselves.
#
yyirregwd <- irregwd(yygrid)
#
# You might want to plot the coefficients
#
# If you want to see the actual coefficients you have to first convert
# the class of the yyirregwd object to a wd object and then use
# \code{\link{plot.wd}} like this
#
yyirregwd2 <- yyirregwd
class(yyirregwd2) <- "wd"
## Not run: plot(yyirregwd2)
#
# If you want to see the variance factors (essentially the coefficient
# variances divided by the overall variance). Then just use
# \code{\link{plot.irregwd}}
#
## Not run: plot(yyirregwd)
#
# Ok. So you've seen the coefficients. Now let's do some thresholding.
#
yy.thresh.sure <- threshold(yyirregwd, policy="sure", type="soft", dev=madmad)
#
# And now do the reconstruct
#
yy.wr <- wr(yy.thresh.sure)
#
# And you can even plot the answer on the new grid!
#
## Not run: plot(yygrid$gridt, yy.wr, type="l")
#
# And superimpose the original data!
#
```

```
## Not run: points(tt, yy)
#
# This is sort of \code{Doppler} like!
```

---

makewpstDO *Help page for a function*

Description

Takes two time series: one a real-valued discrete-time time series, timeseries, the other, groups, a time series containing factor levels. This function performs a discriminant analysis of groups on a subset of the best-correlating nondecimated wavelet packets of timeseries

Usage

```
makewpstDO(timeseries, groups, filter.number=10, family="DaubExPhase",
 mincor=0.69999999999999996)
```

Arguments

`timeseries`	The time series which is the 'dependent variable', ie discrimination will be performed on the variables extracted from the non-decimated wavelet packet transform of this time series
`groups`	The factor levels as a time series
`filter.number`	The smoothness of the wavelet involved in the nondecimated wavelet packet transform. See `filter.select`
`family`	The wavelet family, see `filter.select`
`mincor`	Variables from the nondecimated wavelet packet transform with correlations less than this argument will be discarded in the first pass, and not considered as possible useful discriminants

Details

This function implements the 'discrimination' version of the "Wavelet packet transfer function modelling of nonstationary series" by Guy Nason and Theofanis Sapatinas, *Statistics and Computing*, **12**, 45-56.

The function first takes the non-decimated wavelet packet transform of `timeseries` using the `wpst` function. Then the set of nondecimated wavelet packets is put into matrix form using the `wpst2discr` function. The `Best1DCols` function selects those variables from the matrix whose correlation with the `groups` time series is greater than `mincor`. The selected variables are put into a reduced matrix.

The next step, `BMdiscr`, performs a linear discriminant analysis of the `groups` values onto the reduced matrix. In principle, one could have carried out a discriminant analysis using the full matrix of all the packets, but the problem is not well-conditioned and computationally efficient. The strategy adopted by Nason and Sapatinas is to do a "first pass" to select a large number of "likely" variables that might contribute something to discrimination, and then carry out a "second

pass" which performs a more detailed analysis to jointly determine which variables are the key ones for discrimination.

Note, using the discriminant model developed here, it is possible to use future values of `timeseries` and the model to predict future values of `groups`. See example below.

Value

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

`BPd`	Object returned from the `BMdiscr` function. Contains the reduced matrix and the discriminant object
`BP`	Object returned from the `Best1DCols` function, essentially the reduced matrix and the groups variable.
`filter`	The details of the wavelet filter used. This is used if the other components are used to perform discrimination on new data one needs to know what wavelet was used to perform the original nondecimated wavelet packet transform.

Author(s)

G P Nason

See Also

`basisplot.BP`, `Best1DCols`, `BMdiscr`, `wpst`, `wpst2discr`, `wpstCLASS`

Examples

```
#
# Use BabySS and BabyECG data for this example.
#
# Want to predict future values of BabySS from future values of BabyECG
#
# Build model on first 256 values of both
#
data(BabyECG)
data(BabySS)
BabyModel <- makewpstDO(timeseries=BabyECG[1:256], groups=BabySS[1:256],
 mincor=0.5)
#
# The results (ie print out answer)
#BabyModel
#Stationary wavelet packet discrimination object
#Composite object containing components:[1] "BPd"    "BP"     "filter"
#Fisher's discrimination: done
#BP component has the following information
#BP class object. Contains "best basis" information
#Components of object:[1] "nlevelsWT"    "BasisMatrix" "level"     "pkt"       "basiscoef"
#[6] "groups"
#Number of levels  8
#List of "best" packets
#Level id Packet id Basis coef
```

```
#[1,]        4         0  0.7340580
#[2,]        5         0  0.6811251
#[3,]        6         0  0.6443167
#[4,]        3         0  0.6193434
#[5,]        7         0  0.5967620
#[6,]        0         3  0.5473777
#[7,]        1        53  0.5082849
#
# You can plot the select basis graphically using
#
## Not run: basisplot(BabyModel$BP)
#
# An interesting thing are the final "best" packets, these form the
# "reduced" matrix, and the final discrimination is done on this
# In this case 7 wavelet packets were identified as being good for
# univariate high correlation.
#
# In the second pass lda analysis, using the reduced matrix, the following
# turns up as the best linear discriminant vectors
#
# The discriminant variables can be obtained by typing
#BabyModel$BPd$dm$scaling
#LD1        LD2
#[1,] 5.17130434  1.8961807
#[2,] 1.56487144 -3.5025251
#[3,] 1.69328553  1.1585477
#[4,] 3.63362324  8.4543247
#[5,] 0.15202947 -0.4530523
#[6,] 0.35659009 -0.3850318
#[7,] 0.09429836 -0.1281240
#
#
# Now, suppose we get some new data for the BabyECG time series.
# For the purposes of this example, this is just the continuing example
# ie BabyECG[257:512]. We can use our new discriminant model to predict
# new values of BabySS
#
BabySSpred <- wpstCLASS(newTS=BabyECG[257:512], BabyModel)
#
# Let's look at the first 10 (eg) values of this prediction
#
#BabySSpred$class[1:10]
#[1] 4 4 4 4 4 4 4 4 4 4
#Good. Now let's look at what the "truth" was:
#BabySS[257:267]
#[1] 4 4 4 4 4 4 4 4 4 4
#Good. However, the don't agree everywhere, let's do a cross classification
#between the prediction and the truth.
#
#> table(tmp2$class, BabySS[257:512])
#
#      1   2   3   4
#  1   4   1   1   0
```

```
#  2 116   0  23   3
#  4   2  12   0  94
#
#So class 3 and 4 agree pretty much, but class 1 has been mispredicted at class
#2 a lot.
```

---

makewpstRO — *Make a wavelet packet regression object from a dependent and independent time series variable.*

Description

The idea here is to try and build facilities to enable a transfer function model along the lines of that described by Nason and Sapatinas 2002 in *Statistics and Computing*. The idea is to turn the `timeseries` variable into a set of nondecimated wavelet packets which are already pre-selected to have some semblance of relationship to the `response` time series. The function does not actually perform any regression, in contrast to the related `makewpstDO` but returns a data frame which the user can use to build their own models.

Usage

```
makewpstRO(timeseries, response, filter.number = 10,
    family = "DaubExPhase", trans = logabs, percentage = 10)
```

Arguments

`timeseries`	The dependent variable time series. This series is decomposed using the `wpst` function into nondecimated wavelet packets, need to be a power of two length.
`response`	The independent or response time series.
`filter.number`	The type of wavelet used within `family`, see `filter.select`.
`family`	The family of wavelet, see `filter.select`
`trans`	A transform to apply to the nondecimated wavelet packet coefficients before any selection
`percentage`	The top `percentage` of nondecimated wavelet packets that correlated best with the `response` series will be preselected.

Details

The idea behind this methodology is that a `response` time series might not be directly related to the dependent `timeseries` time series, but it might be related to the nondecimated wavelet packets of the `timeseries`, these packets can pick out various features of the `timeseries` including certain delays, oscillations and others.

The best packets (the number if controlled by `percentage`), those that correlate best with `response` are selected and returned. The `response` and the best nondecimated wavelet packets are returned in a data frame object and then any convenient form of statistical modeling can be used to build a model of the `response` in terms of the packet variables.

Once a model has been built it can be interpreted in the usual way, but with respect to nondecimated wavelet packets.

Note that nondecimated wavelet packets are essential, as they are all of the same length as the original response series. If a decimated wavelet packet algorithm had been used then it is not clear what to do with the "gaps"!

If new `timeseries` data comes along the `wpstREGR` function can be used to extract the identical packets as the ones produced by this function (as the result of this function stores the identities of these packets). Then the statistical modelling that build the model from the output of this function, can be used to predict future values of the `response` time series from future values of the `timeseries` series.

Value

An object of class `wpstRO` containing the following items

`df`	A data frame containing the `response` time series and a number of columns/variables/packets that correlated with response series. These are all entitled "Xn" where n is some integer
`ixvec`	A packet index vector. After taking the nondecimated wavelet packet transform, all the packets are stored in a matrix. This vector indicates those that were preselected
`level`	The original level from which the preselected vectors came from
`pktix`	Another index vector, this time referring to the original wavelet packet object, not the matrix in which they subsequently got stored
`nlevelsWT`	The number of resolution levels in the original wavelet packet object
`cv`	The correlation vector. These are the values of the correlations of the packets with the response, then sorted in terms of decreasing absolute correlation
`filter`	The wavelet filter details
`trans`	The transformation function actually used

Author(s)

G P Nason

References

Nason, G.P. and Sapatinas, T. (2002) Wavelet packet transfer function modeling of nonstationary time series. *Statistics and Computing*, **12**, 45-56.

See Also

`makewpstDO`, `wpst`, `wpstREGR`

Examples

```
data(BabyECG)
baseseries <- BabyECG[1:256]
#
# Make up a FICTITIOUS response series!
#
response <- BabyECG[6:261]*3+52
#
# Do the modeling
#
BabeModel <- makewpstRO(timeseries=baseseries, response=response)
#Level: 0  ..........
#1  ..........
#2  ..........
#3  ..........
#4  ................
#5
#6
#7
#
#Contains SWP coefficients
#Original time series length:  256
#Number of bases:  25
#Some basis selection performed
#       Level Pkt Index Orig Index      Score
#[1,]     5         0        497  0.6729833
#[2,]     4         0        481  0.6120771
#[3,]     6         0        505  0.4550616
#[4,]     3         0        449  0.4309924
#[5,]     7         0        509  0.3779385
#[6,]     1        53        310  0.3275428
#[7,]     2        32        417 -0.3274858
#[8,]     2        59        444 -0.2912863
#[9,]     3        16        465 -0.2649679
#[10,]     1       110        367  0.2605178
#etc. etc.
#
#
# Let's look at the data frame component
#
names(BabeModel$df)
# [1] "response" "X1"       "X2"       "X3"       "X4"       "X5"
# [7] "X6"       "X7"       "X8"       "X9"       "X10"      "X11"
#[13] "X12"      "X13"      "X14"      "X15"      "X16"      "X17"
#[19] "X18"      "X19"      "X20"      "X21"      "X22"      "X23"
#[25] "X24"      "X25"
#
# Generate a formula including all of the X's (note we could use the .
# argument, but we later want to be more flexible
#
xnam <- paste("X", 1:25, sep="")
fmla1 <- as.formula(paste("response ~ ", paste(xnam, collapse= "+")))
```

```
#
# Now let's fit a linear model, the response on all the Xs
#
Babe.lm1 <- lm(fmla1, data=BabeModel$df)
#
# Do an ANOVA to see what's what
#
anova(Babe.lm1)
#Analysis of Variance Table
#
#Response: response
# Df Sum Sq Mean Sq  F value    Pr(>F)
#X1          1 214356  214356 265.7656 < 2.2e-16 ***
#X2          1  21188   21188  26.2701 6.289e-07 ***
#X3          1  30534   30534  37.8565 3.347e-09 ***
#X4          1    312     312   0.3871 0.5344439
#X5          1   9275    9275  11.4999 0.0008191 ***
#X6          1     35      35   0.0439 0.8343135
#X7          1    195     195   0.2417 0.6234435
#X8          1     94      94   0.1171 0.7324600
#X9          1    331     331   0.4103 0.5224746
#X10         1      0       0   0.0006 0.9810560
#X11         1    722     722   0.8952 0.3450597
#X12         1      0       0   0.0004 0.9850243
#X13         1     77      77   0.0959 0.7570769
#X14         1   2770    2770   3.4342 0.0651404 .
#X15         1      6       6   0.0072 0.9326155
#X16         1    389     389   0.4821 0.4881649
#X17         1     44      44   0.0544 0.8157015
#X18         1     44      44   0.0547 0.8152640
#X19         1   4639    4639   5.7518 0.0172702 *
#X20         1    490     490   0.6077 0.4364469
#X21         1    389     389   0.4823 0.4880660
#X22         1     85      85   0.1048 0.7463860
#X23         1   1710    1710   2.1198 0.1467664
#X24         1     12      12   0.0148 0.9033427
#X25         1     82      82   0.1019 0.7498804
#Residuals 230 185509     807
#---
#Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Looks like X1, X2, X3, X5, X14 and X19 are "significant". Also throw in
# X4 as it was a highly ranked preselected variable, and refit
#
fmla2 <- response ~ X1 + X2 + X3 + X4 + X5 + X14 + X19
Babe.lm2 <- lm(fmla2, data=BabeModel$df)
#
# Let's see the ANOVA table for this
#
anova(Babe.lm2)
#Analysis of Variance Table
#
#Response: response
```

```
# Df Sum Sq Mean Sq  F value    Pr(>F)
#X1          1 214356  214356 279.8073 < 2.2e-16 ***
#X2          1  21188   21188  27.6581 3.128e-07 ***
#X3          1  30534   30534  39.8567 1.252e-09 ***
#X4          1    312     312   0.4076 0.5238034
#X5          1   9275    9275  12.1075 0.0005931 ***
#X14         1   3095    3095   4.0405 0.0455030 *
#X19         1   4540    4540   5.9259 0.0156263 *
#Residuals 248 189989     766
#---
#Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# So, let's drop X4, refit, and then do ANOVA
#
Babe.lm3 <- update(Babe.lm2, . ~ . -X4)
anova(Babe.lm3)
#
# After viewing this, drop X14
#
Babe.lm4 <- update(Babe.lm3, . ~ . -X14)
anova(Babe.lm4)
#
# Let's plot the original series, and the "fitted" one
#
## Not run: ts.plot(BabeModel$df[["response"]])
## Not run: lines(fitted(Babe.lm4), col=2)
#
# Let's plot the wavelet packet basis functions associated with the model
#
## Not run: oldpar <- par(mfrow=c(2,2))
## Not run: z <- rep(0, 256)
## Not run: zwp <- wp(z, filter.number=BabeModel$filter$filter.number,
    family=BabeModel$filter$family)
## End(Not run)
## Not run: draw(zwp, level=BabeModel$level[1], index=BabeModel$pktix[1], main="", sub="")
## Not run: draw(zwp, level=BabeModel$level[2], index=BabeModel$pktix[2], main="", sub="")
## Not run: draw(zwp, level=BabeModel$level[3], index=BabeModel$pktix[3], main="", sub="")
## Not run: draw(zwp, level=BabeModel$level[5], index=BabeModel$pktix[5], main="", sub="")
## Not run: par(oldpar)
#
# Now let's do some prediction of future values of the response, given
# future values of the baseseries
#
newseries <- BabyECG[257:512]
#
# Get the new data frame
#
newdfinfo <- wpstREGR(newTS = newseries, wpstRO=BabeModel)
#
# Now use the best model (Babe.lm4) with the new data frame (newdfinfo)
# to predict new values of response
#
newresponse <- predict(object=Babe.lm4, newdata=newdfinfo)
```

```
#
# What is the "true" response, well we made up a response earlier, so let's
# construct the true response for this future data (in your case you'll
# have a separate genuine response variable)
#
trucfictresponse <- BabyECG[262:517]*3+52
#
# Let's see them plotted on the same plot
#
## Not run: ts.plot(trucfictresponse)
## Not run: lines(newresponse, col=2)
#
# On my plot they look tolerably close!
#
```

MaNoVe *Make Node Vector (using Coifman-Wickerhauser best-basis type algorithm)*

Description

This generic function chooses a "best-basis" using the Coifman-Wickerhauser (1992) algorithm. This function is generic. Particular methods exist: MaNoVe.wp and MaNoVe.wst.

Usage

```
MaNoVe(...)
```

Arguments

...	Methods may have other arguments

Details

Description says all.

Value

A node vector, which describes a particular basis specification relevant to the kind of object that the function was applied to.

Author(s)

G P Nason

See Also

MaNoVe.wp, MaNoVe.wst, wp.object, wst.object, wp, wst

MaNoVe.wp *Make Node Vector (using Coifman-Wickerhauser best-basis type algorithm) on wavelet packet object*

Description

This method chooses a "best-basis" using the Coifman-Wickerhauser (1992) algorithm applied to wavelet packet, wp.object, objects.

Usage

```
## S3 method for class 'wp'
MaNoVe(wp, verbose=FALSE, ...)
```

Arguments

wp	The wp object for which you wish to find the best basis for.
verbose	Whether or not to print out informative messages
...	Other arguments

Details

Description says all

Value

A wavelet packet node vector object of class nvwp, a basis description. This can be fed into a basis inversion using, say, the function InvBasis.

Author(s)

G P Nason

See Also

InvBasis, MaNoVe, MaNoVe.wst, wp.object, wp

Examples

```
#
# See example of use of this function in the examples section
# of the help of plot.wp
#
# A node vector vnv is created there that gets plotted.
#
```

MaNoVe.wst *Make Node Vector (using Coifman-Wickerhauser best-basis type algorithm) on nondecimated wavelet transform object*

Description

This method chooses a "best-basis" using the Coifman-Wickerhauser (1992) algorithm applied to nondecimated wavelet transform, wst.object, objects.

Usage

```
## S3 method for class 'wst'
MaNoVe(wst, entropy=Shannon.entropy, verbose=FALSE,
 stopper=FALSE, alg="C", ...)
```

Arguments

wst	The wst object for which you wish to find the best basis for.
entropy	The function used for computing the entropy of a vector
verbose	Whether or not to print out informative messages
stopper	Whether the computations are temporarily stopped after each packet. This can be useful in conjunction with the verbose argument so as to see computations proceed one packet at a time.
alg	If "C" then fast compiled C code is used (in which case the entropy function is ignored and the C code uses an internal Shannon entropy. Otherwise, slower R code is used but an arbitrary entropy argument can be used
...	Other arguments

Details

Description says all

Value

A wavelet node vector object, of class nv, a basis description. This can be fed into a basis inversion using, say, the function InvBasis.

Author(s)

G P Nason

See Also

InvBasis, MaNoVe, MaNoVe.wp, Shannon.entropy, wst.object, wst

Examples

```
#
# What follows is a simulated denoising example. We first create our
# "true" underlying signal, v. Then we add some noise to it with a signal
# to noise ratio of 6. Then we take the packet-ordered non-decimated wavelet
# transform and then threshold that.
#
# Then, to illustrate this function, we compute a "best-basis" node vector
# and use that to invert the packet-ordered NDWT using this basis. As a
# comparison we also use the Average Basis method
# (cf Coifman and Donoho, 1995).
#
# NOTE: It is IMPORTANT to note that this example DOES not necessarily
# use an appropriate or good threshold or necessarily the right underlying
# wavelet. I am trying to show the general idea and please do not "quote" this
# example in literature saying that this is the way that WaveThresh (or
# any of the associated authors whose methods it attempts to implement)
# does it. Proper denoising requires a lot of care and thought.
#
#
# Here we go....
#
# Create an example vector (the Donoho and Johnstone heavisine function)
#
v <- DJ.EX()$heavi
#
# Add some noise with a SNR of 6
#
vnoise <- v + rnorm(length(v), 0, sd=sqrt(var(v))/6)
#
# Take packet-ordered non-decimated wavelet transform (note default wavelet
# used which might not be the best option for denoising performance).
#
vnwst <- wst(vnoise)
#
# Let's take a look at the wavelet coefficients of vnoise
#
## Not run: plot(vnwst)

#
# Wow! A huge number of coefficients, but mostly all noise.
#
#
# Threshold the resultant NDWT object.
# (Once again default arguments are used which are certainly not optimal).
#
vnwstT <- threshold(vnwst)
#
# Let's have a look at the thresholded wavelet coefficients
#
## Not run: plot(vnwstT)
```

```
#
# Ok, a lot of the coefficients have been removed as one would expect with
# universal thresholding
#
#
# Now select packets for a basis using a Coifman-Wickerhauser algorithm
#
vnnv <- MaNoVe(vnwstT)
#
# Let's have a look at which packets got selected
#
vnnv
# Level :  9  Action is  R (getpacket Index:  1 )
# Level :  8  Action is  L (getpacket Index:  2 )
# Level :  7  Action is  L (getpacket Index:  4 )
# Level :  6  Action is  L (getpacket Index:  8 )
# Level :  5  Action is  R (getpacket Index:  17 )
# Level :  4  Action is  L (getpacket Index:  34 )
# Level :  3  Action is  L (getpacket Index:  68 )
# Level :  2  Action is  R (getpacket Index:  137 )
# Level :  1  Action is  R (getpacket Index:  275 )
# There are  10  reconstruction steps
#
# So, its not the regular decimated wavelet transform!
#
# Let's invert the representation with respect to this basis defined by
# vnnv
#
vnwrIB <- InvBasis(vnwstT, vnnv)
#
# And also, for completeness let's do an Average Basis reconstruction.
#
vnwrAB <- AvBasis(vnwstT)
#
# Let's look at the Integrated Squared Error in each case.
#
sum( (v - vnwrIB)^2)
# [1] 386.2501
#
sum( (v - vnwrAB)^2)
# [1] 328.4520
#
# So, for this limited example the average basis method does better. Of course,
# for *your* simulation it could be the other way round. "Occasionally", the
# inverse basis method does better. When does this happen? A good question.
#
# Let's plot the reconstructions and also the original
#
## Not run: plot(vnwrIB, type="l")
## Not run: lines(vnwrAB, lty=2)
## Not run: lines(v, lty=3)

#
```

```
# The dotted line is the original. Neither reconstruction picks up the
# spikes in heavisine very well. The average basis method does track the
# original signal more closely though.
#
```

`mfilter.select` *Provide filter coefficients for multiple wavelets.*

Description

This function returns the filter coefficients necessary for doing a discrete multiple wavelet transform (and its inverse).

Usage

```
mfilter.select(type = "Geronimo")
```

Arguments

`type`	The name for the multiple wavelet basis. The two possible types are "Geronimo" and "Donovan3"

.

Details

This function supplies the multiple wavelet filter coefficients required by the `mwd` function.

A multiple wavelet filter is somewhat different from a single wavelet filter. Firstly the filters are made up of matrices not single coefficients. Secondly there is no simple expression for the high pass coefficients G in terms of the low pass coefficients H, so both sets of coefficients must be specified. Note also that the transpose of the filter coefficients are used in the inverse transform, an unnecessary detail with scalar coefficients. There are two filters available at the moment. Geronimo is the default, and is recommended as it has been checked thoroughly. Donovan3 uses three orthogonal wavelets described in Donovan et al. but this coding has had little testing.

See Donovan, Geronimo and Hardin, 1996 and Geronimo, Hardin and Massopust, 1994.

This function fulfils the same purpose as the `filter.select` function does for the standard DWT `wd`.

Value

A list is returned with the following eight components which describe the filter:

`type`	The multiple wavelet basis type string.
`H`	A vector containing the low pass filter coefficients.
`G`	A vector containing the high pass pass filter coefficients.
`name`	A character string containing the full name of the filter.
`nphi`	The number of scaling functions in the multiple wavelet basis.

npsi	The number of wavelet functions in the multiple wavelet basis.
NH	The number of matrix coefficients in the filter. This is different from length(H).
ndecim	The decimation factor. I.e. the scale ratio between two successive resolution levels.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6)

Author(s)

Tim Downie

See Also

accessC.mwd, accessD.mwd, draw.mwd, mfirst.last, mwd.object, mwd, mwr, plot.mwd, print.mwd, putC.mwd, putD.mwd, summary.mwd, threshold.mwd, wd, wr.mwd.

Examples

```
#This function is currently used by `mwr' and `mwd' in decomposing and
#reconstructing, however you can view the coefficients.
#
# look at the filter coefficients for Geronimo multiwavelet
#
mfilter.select()
#$type:
#[1] "Geronimo"
#
#$name:
#[1] "Geronimo Multiwavelets"
#
#$nphi:
#[1] 2
#
#$npsi:
#[1] 2
#
#$NH:
#[1] 4
#
#$ndecim:
#[1] 2
#$H:
# [1]  0.4242641  0.8000000 -0.0500000 -0.2121320  0.4242641  0.0000000
# [7]  0.4500000  0.7071068  0.0000000  0.0000000  0.4500000 -0.2121320
#[13]  0.0000000  0.0000000 -0.0500000  0.0000000
#
#$G:
# [1] -0.05000000 -0.21213203  0.07071068  0.30000000  0.45000000 -0.70710678
#
```

```
# [7] -0.63639610  0.00000000  0.45000000 -0.21213203  0.63639610 -0.30000000
#[13] -0.05000000  0.00000000 -0.07071068  0.00000000
```

`mfirst.last` *Build a first/last database for multiple wavelet transforms.*

Description

This function is not intended for user use, but is used by various functions involved in computing and displaying multiple wavelet transforms.

Usage

```
mfirst.last(LengthH, nlevels, ndecim, type = "wavelet", bc = "periodic")
```

Arguments

`LengthH`	Number of filter matrix coefficients.
`nlevels`	Number of levels in the decomposition
`ndecim`	The decimation scale factor for the multiple wavelet basis.
`type`	Whether the transform is non-decimated or ordinary (wavelet). The non-decimated multiple wavelet transform is not yet supported.
`bc`	This argument determines how the boundaries of the the function are to be handled. The permitted values are periodic or `symmetric`

Details

Suppose you begin with `2^m`=2048 coefficient vectors. At the next level you would expect 1024 smoothed data vectors, and 1024 wavelet vectors, and if `bc="periodic"` this is indeed what happens. However, if `bc="symmetric"` you actually need more than 1024 (as the wavelets extend over the edges). The first last database keeps track of where all these "extras" appear and also where they are located in the packed vectors C and D of pyramidal coefficients within wavelet structures.

For examples, given a `first.last.c` row of

$$-2320$$

The ‘position’ of the coefficient vectors would be

$$c_{-2}, c_{-1}, c_0, c_1, c_2, c_3$$

In other words, there are 6 coefficients, starting at -2 and ending at 3, and the first of these (c_{-2}) appears at column 20 of the `$C` component matrix of the wavelet structure.

You can “do” first.last in your head for periodic boundary handling but for more general boundary treatments (e.g. symmetric) first.last is indispensable.

The numbers in first last databases were worked out from inequalities derived from: Daubechies, I. (1988).

Value

A first/last database structure, a list containing the following information:

`first.last.c` A `(m+1)x3` matrix. The first column specifies the real index of the first coefficient vector of the smoothed data at a level, the 2nd column is the real index of the last coefficient vector, the last column specifies the offset of the first smoothed datum at that level. The offset is used by the C code to work out where the beginning of the sequence is within a packed vector of the pyramid structure. The first and 2nd columns can be used to work out how many numbers there are at a level. If bc="periodic" then the pyramid is a true power of 2 pyramid, that is it starts with a power of 2, and the next level is half of the previous. If bc="symmetric" then the pyramid is nearly exactly a power of 2, but not quite, see the Details section for why this is so.

`nvecs.c` The number of C coefficient vectors.

`first.last.d` A `mx3` matrix. As for `first.last.c` but for the wavelet coefficients packed as the D component of a wavelet structure.

`nvecs.d` The number of `D` coefficient vectors.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6)

Author(s)

Tim Downie

See Also

`accessC.mwd`, `accessD.mwd`, `draw.mwd`, `mwd.object`, `mwd`, `mwr`, `plot.mwd`, `print.mwd`, `putC.mwd`, `putD.mwd`, `summary.mwd`, `threshold.mwd`, `wd`, `wr.mwd`.

Examples

```
#
#To see the housekeeping variables for a decomposition with
# 4 filter coefficient matices
# 5 resolution levels and a decimation scale of two
# use:
mfirst.last(4,5,2)
# $first.last.c:
#  First Last Offset
# [1,]     0    0     62
# [2,]     0    1     60
# [3,]     0    3     56
# [4,]     0    7     48
# [5,]     0   15     32
# [6,]     0   31      0
#
# $nvecs.c:
# [1] 63
```

```
#
# $first.last.d:
#  First Last Offset
# [1,]     0    0     30
# [2,]     0    1     28
# [3,]     0    3     24
# [4,]     0    7     16
# [5,]     0   15      0
#
# $nvecs.d:
# [1] 31
```

`modernise` *Generic function to upgrade a V2 WaveThresh object to V4*

Description

Not really used in practice. The function `IsEarly` can be used to tell if an object comes from an earlier version of wavethresh. Note that the earlier version only has a `wd.object` class object so there is only a method for that.

Usage

```
modernise(...)
```

Arguments

`...`	Other objects

Details

Description says all

Value

A modernised version of the object.

Author(s)

G P Nason

See Also

`IsEarly`,`modernise.wd`

modernise.wd — *Modernise a wd class object*

Description

Upgrade a version 2 `wd.object` to version 4. The function `IsEarly` can tell if the object comes from an earlier version of WaveThresh.

Usage

```
## S3 method for class 'wd'
modernise(wd, ...)
```

Arguments

`wd`	The wd object you wish to modernise
`...`	Other arguments

Details

Description says all.

Value

The modernised object.

Author(s)

G P Nason

mpostfilter — *Multiwavelet postfilter*

Description

A multiwavelet postfilter turns a multivariate sequence into a univariate sequence. As such, the postfilter is used on the inverse transform, it is the inverse of an earlier used prefilter.

Not intended for direct user use.

Usage

```
mpostfilter(C, prefilter.type, filter.type,
    nphi, npsi, ndecim, nlevels, verbose = FALSE)
```

Arguments

`C`	The multivariate sequence you wish to turn back into a univariate one using the inverse of an earlier prefilter operation.
`prefilter.type`	Controls the type of prefilter (see Tim Downie's PhD thesis, or references therein. Types include `Minimal`, `Identity`, `Repeat`, `Interp`, `default`, `Xia`, `Roach1`, `Roach3`, `Donovan3` or `Linear`
`filter.type`	The type of multiwavelet: can be `Geronimo` or `Donovan3`
`nphi`	The number of father wavelets in the system
`npsi`	The number of mother wavelets in the system
`ndecim`	The ndecim parameter (not apparently used here)
`nlevels`	The number of levels in the multiwavelet transform
`verbose`	If TRUE then informative messages are printed as the function progresses

Details

Description says all

Value

The appropriate postfiltered data.

Author(s)

Tim Downie

See Also

`mprefilter`,`mwd`

`mprefilter` *Multiwavelet prefilter*

Description

A multiwavelet prefilter turns a univariate sequence into a bivariate (in this case) sequence suitable for processing by a multiwavelet transform, such as `mwd`. As such, the prefilter is used on the forward transform.

Not intended for direct user use.

Usage

```
mprefilter(data, prefilter.type, filter.type, nlevels, nvecs.c,
    nphi, npsi, ndecim, verbose = FALSE)
```

Arguments

`data`	The univariate sequence that you wish to turn into a multivariate one
`prefilter.type`	Controls the type of prefilter (see Tim Downie's PhD thesis, or references therein. Types include `Minimal`, `Identity`, `Repeat`, `Interp`, `default`, `Xia`, `Roach1`, `Roach3`, `Donovan3` or `Linear`
`filter.type`	The type of multiwavelet: can be `Geronimo` or `Donovan3`
`nlevels`	The number of levels in the multiwavelet transform
`nvecs.c`	Parameter obtained from the mfirst.last function related to the particular filters
`nphi`	The number of father wavelets in the system
`npsi`	The number of mother wavelets in the system
`ndecim`	The ndecim parameter (not apparently used here)
`verbose`	If TRUE then informative messages are printed as the function progresses

Details

Description says all

Value

The appropriate prefiltered data.

Author(s)

Tim Downie

See Also

`mpostfilter`,`mwd`

---

mwd — *Discrete multiple wavelet transform (decomposition).*

Description

This function performs the discrete multiple wavelet transform (DMWT). Using an adaption of Mallat's pyramidal algorithm. The DMWT gives vector wavelet coefficients.

Usage

```
mwd(data, prefilter.type = "default", filter.type = "Geronimo",
        bc ="periodic", verbose = FALSE)
```

Arguments

`data`	A vector containing the data you wish to decompose. The length of this vector must be a power of 2 times the dimension of the DMWT (multiplicity of wavelets).
`prefilter.type`	This chooses the method of preprocessing required. The arguments will depend on filter.type, but "default" will always work.
`filter.type`	Specifies which multi wavelet filter to use, The options are "`Geronimo`" (dimension 2) or "`Donovan3`" (dimension 3). The latter has not been tested fully and may contain bugs. See the function `mfilter.select` for further details.
`bc`	specifies the boundary handling. If `bc=="periodic"` the default, then the function you decompose is assumed to be periodic on its interval of definition, if `bc=="symmetric"` then the function beyond its boundaries is assumed to be a symmetric reflection of the function in the boundary.
`verbose`	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.

Details

The code implements Mallat's pyramid algorithm adapted for multiple wavelets using Xia, Geronimo, Hardin and Suter, 1996. The method takes a data vector of length `2^J*M`, and preprocesses it. This has two effects, firstly it puts the data into matrix form and then filters it so that the DMWT can operate more efficiently Most of the technical details are similar to the single wavelet transform except for the matrix algebra considerations, and the prefiltering process. See Downie and Silverman (1998) for further details and how this transform can be used in a statistical context.

Value

An object of class `mwd`.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1996)

Author(s)

Tim Downie

See Also

`accessC.mwd`, `accessD.mwd`, `draw.mwd`, `mfirst.last`, `mfilter.select`, `mwd.object`, `mwr`, `plot.mwd`, `print.mwd`, `putC.mwd`, `putD.mwd`, `summary.mwd`, `threshold.mwd`, `wd`, `wr.mwd`.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
```

```
## Not run: ts.plot(test.data)
#
# Decompose test.data with multiple wavelet transform and
# plot the wavelet coefficients
#
tdmwd <- mwd(test.data)
## Not run: plot(tdmwd)
#[1] 1.851894 1.851894 1.851894 1.851894 1.851894 1.851894 1.851894
#
# You should see a plot with wavelet coefficients like in
#\code{\link{plot.wd}} but at each coefficient position
# there are two coefficients in two different colours one for each of
# the wavelets at that position.
#
# Note the scale for each level is returned by the function.
```

`mwd.object` *Multiple wavelet decomposition object (1D)*

Description

These are objects of class

`mwd`

They represent a decomposition of a function with respect to a multiple wavelet basis.

Details

To retain your sanity the C and D coefficients should be extracted by the `accessC` and `accessD` functions and put using the `putC` and `putD` functions, rather than by the `$` operator.

Value

The following components must be included in a legitimate 'mwd' object.

`C` a matrix containing each level's smoothed data, each column corresponding to one coefficient vector. The wavelet transform works by applying both a smoothing filter and a bandpass filter to the previous level's smoothed data. The top level contains data at the highest resolution level. Each of these levels are stored one after the other in this matrix. The matrix '`fl.dbase$first.last.c`' determines exactly which columns in the matrix, store each level.

`D` wavelet coefficient matrix. If you were to write down the discrete wavelet transform of a function then columns of D would be the vector coefficients of the wavelet basis function s. Like the C, they are also formed in a pyramidal manner, but stored in a linear matrix. The storage details are to be found in '`fl.dbase$first.last.d`'.

`nlevelsWT` The number of levels in the pyramidal decomposition that produces the coefficients. The precise number of levels depends on the number of different wavelet functions used and the preprocessing method used, as well as the number of data points used.

`fl.dbase` The first last database associated with this decomposition. This is a list consisting of 2 integers, and 2 matrices. The matrices detail how the coefficients are stored in the C and D components of the 'mwd.object'. See the help on `mfirst.last` for more information.

`filter` a list containing the details of the filter that did the decomposition. See `mfilter.select`.

`type` either `"wavelet"` indicating that the ordinary multiple wavelet transform was performed or `"station"` indicating that the non-decimated multiple wavelet transform was done.

`prefilter` Type of preprocessing or prefilter used. This will be specigic for the type of multiple wavelet used.

`date` The date that the transform was performed or the mwd object was last modified.

`bc` how the boundaries were handled

GENERATION

This class of objects is returned from the `mwd` function to represent a multiple wavelet decomposition of a function. Many other functions return an object of class mwd.

METHODS

The mwd class of objects has methods for the following generic functions: `accessC`, `accessD`, `draw`, `plot`, `print`, `putC`, `putD`, `summary`, `threshold`, `wr.mwd`.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie, 1995-6).

Author(s)

Tim Downie

See Also

`accessC.mwd`, `accessD.mwd`, `draw.mwd`, `mfirst.last`, `mfilter.select`, `mwd.object`, `mwr`, `plot.mwd`,`print.mwd`, `putC.mwd`, `putD.mwd`, `summary.mwd`, `threshold.mwd`, `wd`, `wr.mwd`.

mwr *Multiple discrete wavelet transform (reconstruction).*

Description

This function performs the reconstruction stage of Mallat's pyramid algorithm adapted for multiple wavelets (see Xia et al.(1996)), i.e. the discrete inverse *multiple* wavelet transform.

Usage

```
mwr(mwd, prefilter.type = mwd$prefilter, verbose = FALSE, start.level = 0,
 returnC = FALSE)
```

Arguments

`mwd`	A multiple wavelet decomposition object as returned by `mwd`.
`prefilter.type`	Usually best not to change this (i.e. not to use a different prefilter on the reconstruction to the one used on decomposition).
`verbose`	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.
`start.level`	The level you wish to start reconstruction at. The is usually the first (level 0).
`returnC`	If this is FALSE then a vector of the same length as the argument data supplied to the function `mwd` that constructed the supplied `mwd.object`. is returned, Ie. the reconstructed data. If true then the last level (highest resolution) C coefficients are returned in matrix form. This matrix has not been postprocessed.

Details

The code implements Mallat's pyramid algorithm adapted for multiple wavelet decompositions (Xia et al. 1996). In the reconstruction the quadrature mirror filters G and H are supplied with C0 and D0, D1, ... D(J-1) (the wavelet coefficients) and rebuild C1,..., CJ.

The matrix CJ is postprocessed which returns the full reconstruction

If `mwd.object` was obtained directly from `mwd` then the original function can be reconstructued exactly. Usually, the `mwd.object` has been modified in some way, for examples, some coefficients set to zero by `threshold`. Mwr then reconstructs the function with that set of wavelet coefficients.

See also Downie and Silverman, 1998

Value

Either a vector containing the final reconstruction or a matrix containing unpostprocessed coefficients.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1996)

Author(s)

Tim Downie

See Also

accessC.mwd, accessD.mwd, draw.mwd, mfirst.last, mfilter.select, mwd, mwd.object, plot.mwd, print.mwd, putC.mwd, putD.mwd, summary.mwd, threshold.mwd, wd, wr.mwd.

Examples

```
#
# Decompose and then exactly reconstruct test.data
#
test.data <- rnorm(128)
tdecomp <- mwd(test.data)
trecons <- mwr(tdecomp)
#
# Look at accuracy of reconstruction
max(abs(trecons - test.data))
#[1] 2.266631e-12
#
# See also the examples of using \code{\link{wr}} or mwr in
# the \code{examples} section of
# the help for \code{\link{threshold.mwd}}.
```

---

newsure — *Version of sure that acts as subsidiary for threshold.irregwd*

Description

Version of the sure function used as a subsidiary for threshold.irregwd.

Usage

```
newsure(s, x)
```

Arguments

s	Vector of standard deviations of coefficients
x	Vector of regular (ie non-normalized) coefficients

Details

Description says all

Value

The SURE threshold

Author(s)

Arne Kovac

See Also

sure, threshold.irregwd

---

nlevelsWT *Returns number of scale (resolution) levels.*

Description

Returns the number of scales (or resolutions) in various wavelet objects and for some objects returns the number of scales that would result if processed by a wavelet routine.

This function is generic.

One methods exists at present as most wavelet objects store the number of levels as the nlevelsWT component. The method that exists isnlevelsWT.default

Usage

```
nlevelsWT(...)
```

Arguments

...	See individual help pages for details.

Details

See individual method help pages for operation and examples.

Value

An integer representing the number of levels associated with the object.

RELEASE

Version 3.6.0 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

nlevelsWT.default

`nlevelsWT.default` *Returns number of levels associated with an object*

Description

This function returns the number of scale levels associated with either a wavelet type object or an atomic object.

Usage

```
## Default S3 method:
nlevelsWT(object, ...)
```

Arguments

`object`	An object for which you wish to determine how many levels it has or is associated with.
`...`	any other arguments

Details

This function first checks to see whether the input object has a component called nlevelsWT. If it does then it returns the value of this component. If it does not then it takes the length of the object and then uses the `IsPowerOfTwo` function to return the power of two which equals the length (if any) or NA if the length of the object is not a power of two.

Value

The number of resolution (scale) levels associated with the object.

Author(s)

Version 3.6.0 Copyright Guy Nason 1995

See Also

`nlevelsWT`

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
#
# Now, this vector is 512 elements long. What number of levels would any
# wavelet object be that was associated with this vector?
#
```

```
nlevelsWT(test.data)
# [1] 9
#
# I.e. 2^9=512. Let's check by taking the wavelet transform of the
# test data and seeing how many levels it actually has
#
nlevelsWT(wd(test.data))
# [1] 9
```

`nullevels` *Set whole resolution levels of coefficients equal to zero.*

Description

Generic function which sets whole resolution levels of coefficients equal to zero.

Particular methods exist. For objects of class:

imwd use the `nullevels.imwd` method.

wd use the `nullevels.wd` method.

wst use the `nullevels.wst` method.

See individual method help pages for operation and examples.

Usage

```
nullevels(...)
```

Arguments

...	See individual help pages for details.

Value

An object of the same class as x but with the specified levels set to zero.

RELEASE

Version 3.8.1 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

`nullevels.imwd` `nullevels.wd` `nullevels.wst` `wd.object`, `wd` `wst.object` `wst`

`nullevels.imwd` *Sets whole resolution levels of coefficients equal to zero in a imwd object.*

Description

Sets whole resolution levels of coefficients equal to zero in a `imwd.object`

Usage

```
## S3 method for class 'imwd'
nullevels(imwd, levelstonull, ...)
```

Arguments

`imwd`	An object of class `imwd`.
`levelstonull`	An integer vector specifying which resolution levels of coefficients of `imwd` that you wish to set to zero.
`...`	any other arguments

Details

Setting whole resolution levels of coefficients to zero can be very useful. For examples, one can construct a linear smoothing method by setting all coefficients above a particular resolution (the *primary resolution* equal to zero. Also setting particular levels equal to zero can also be useful for removing noise which is specific to a particular resolution level (as long as important signal is not also contained at that level).

Note that this function removes the horiztonal, diagonal and vertical detail coefficients at the resolution level specified. It does not remove the father wavelet coefficients at those resolution levels.

To remove individual coefficients on a systematic basis you probably want to look at the `threshold` function.

Value

An object of class `imwd` where the coefficients in resolution levels specified by levelstonull have been set to zero.

RELEASE

Version 3.9.5 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

nullevels, imwd, imwd.object, threshold.

Examples

```
#
# Do the wavelet transform of the Lennon image
#
data(lennon)
lenimwd <- imwd(lennon)
#
# Set scales (resolution levels) 2, 4 and 6 equal to zero.
#
lenwdNL <- nullevels(lenimwd, levelstonull=c(2,4,6))
#
# Now let's plot the coefficients using a nice blue-heat colour map
#
# You will see that coefficients at levels 2, 4 and 6 are black (i.e. zero)
# You can see that coefficients at other levels are unaffected and still
# show the Lennon coefficients.
#
## Not run: plot(lenwdNL)
```

---

nullevels.wd *Sets whole resolution levels of coefficients equal to zero in a wd object.*

Description

Sets whole resolution levels of coefficients equal to zero in a wd.object

Usage

```
## S3 method for class 'wd'
nullevels(wd, levelstonull, ...)
```

Arguments

wd	An object of class wd.
levelstonull	An integer vector specifying which resolution levels of coefficients of wd that you wish to set to zero.
...	any other arguments

Details

Setting whole resolution levels of coefficients to zero can be very useful. For examples, one can construct a linear smoothing method by setting all coefficients above a particular resolution (the *primary resolution* equal to zero. Also setting particular levels equal to zero can also be useful for

removing noise which is specific to a particular resolution level (as long as important signal is not also contained at that level).

Note that this function removes the horiztonal, diagonal and vertical detail coefficients at the resolution level specified. It does not remove the father wavelet coefficients at those resolution levels.

To remove individual coefficients on a systematic basis you probably want to look at the threshold function.

Value

An object of class wd where the coefficients in resolution levels specified by levelstonull have been set to zero.

RELEASE

Version 3.8.1 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

nullevels, wd, wd.object, threshold.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
#
# Do wavelet transform of test.data and plot the wavelet coefficients
#
wds <- wd(test.data)
## Not run: plot(wds)
#
# Now let us set all the coefficients in ODD resolution levels equal to zero!
#
# This is just to illustrate the capabilities of the function. I cannot
# imagine you wanting to do this in practice!
##
wdsnl <- nullevels(wds, levelstonull = c(1, 3, 5, 7))
#
# Now let's plot the result
#
## Not run: plot(wdsnl, scaling = "by.level")
#
# Lo and behold the odd levels have been set to zero!
```

`nullevels.wst` *Sets whole resolution levels of coefficients equal to zero in a wst object.*

Description

Sets whole resolution levels of coefficients equal to zero in a `wd` object.

Usage

```
## S3 method for class 'wst'
nullevels(wst, levelstonull, ...)
```

Arguments

`wst`	An object of class `wst`.
`levelstonull`	An integer vector specifying which resolution levels of coefficients of `wst` that you wish to set to zero.
`...`	any other arguments

Details

Setting whole resolution levels of coefficients to zero can be very useful. For examples, one can construct a linear smoothing method by setting all coefficients above a particular resolution (the *primary resolution* equal to zero. Also setting particular levels equal to zero can also be useful for removing noise which is specific to a particular resolution level (as long as important signal is not also contained at that level).

To remove individual coefficients on a systematic basis you probably want to look at the `threshold` function.

Value

An object of class `wst` where the coefficients in resolution levels specified by `levelstonull` have been set to zero.

RELEASE

Version 3.8.1 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

`nullevels`, `wst`, `wst.object`, `threshold`.

Examples

```
#
# Look at the examples for \code{\link{nullevels.wd}}.
# The operation is almost identical except that \code{\link{wst}}
# objects are replaced by \code{\link{wd}} ones.
```

---

numtonv *Convert an index number into a node vector object.*

Description

Convert an index number into a node `vector` object.

Usage

```
numtonv(number, nlevels)
```

Arguments

`number`	The index number of a particular basis within a wavelet object.
`nlevels`	The number of levels that the wavelet object has (can often be discovered using the `nlevels` function).

Details

A basis within a (e.g. non-decimated) wavelet object (such as a `wst.object`) is represented in WaveThresh by a `nv` or node vector.

A packet-ordered non-decimated wavelet transform object `wst` for short) which is the transform of a vector of length `n` contains `n` bases. Each basis can be indexed from 0 to `(n-1)` .

A `wst.object` is simply a fully populated binary tree. There are nlevels levels in the tree with a split at each level. The root of the tree is at level 0, there are two branches at level 1, four at level 2, eight at level 3 and so on. A path through the tree can be constructed by starting at the root and choosing "left" or "right" at each possible branch. For certain data situations this path is constructed using minimum entropy algorithms (for examples `MaNoVe`). This function (numtonv takes the numerical representation of a path and converts it into a `node.vector` form suitable for passing to `InvBasis` to invert the representation according to a basis specicified by number.

The least significant digit in number corresponds to deciding on the left/right decision at the fine leaves of the tree (high-frequency structure) and the most significant digit in number corresponds to deciding on the left/right decision at the root. Therefore gradually incrementing number from 0 to `2^{nlevels}-1` steps through all possible bases in the `wst` object ranging from all decisions being made "left" to all decisions being made "right". The "number" dividied by `2^{nlevels}` corresponds exactly to the binary number epsilon in Nason and Silverman (1995).

Value

An object of class `nv` (node vector). This contains information about a path through a wavelet object (a basis in a wavelet object).

RELEASE

Version 3.6.0 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

wst, wst.object, MaNoVe, nv.object, InvBasis, nlevels.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
#
# Make it noisy
#
ynoise <- test.data + rnorm(512, sd=0.1)
#
# Do packet ordered non-decimated wavelet transform
#
ynwst <- wst(ynoise)
#
# Now threshold the coefficients
#
ynwstT <- threshold(ynwst)
#
# Select basis number 9 (why not?)
#
NodeVector9 <- numtonv(9, nlevelsWT(ynwstT))
#
# Let's print it out to see what it looks like
# (nb, if you're repeating this examples, the basis might be different
# as you may have generated different pseudo random noise to me)
#
NodeVector9
# Level :  8  Action is  R (getpacket Index:  1 )
# Level :  7  Action is  L (getpacket Index:  2 )
# Level :  6  Action is  L (getpacket Index:  4 )
# Level :  5  Action is  R (getpacket Index:  9 )
# Level :  4  Action is  L (getpacket Index:  18 )
# Level :  3  Action is  L (getpacket Index:  36 )
# Level :  2  Action is  L (getpacket Index:  72 )
# Level :  1  Action is  L (getpacket Index:  144 )
# Level :  0  Action is  L (getpacket Index:  288 )
# There are  9  reconstruction steps
#
# The print-out describes the tree through ynwstT that corresponds to
# basis 9.
```

```
#
# The NodeVector9 and ynwstT objects could now be supplied to
# InvBasis.wst for inverting ynwstT according
# to the NodeVector9 or basis number 9.
```

---

nv.object *Node vector objects.*

Description

These are objects of classes

nv

They represent a basis in a packet-ordered non-decimated wavelet transform object.

Details

A nv object is a description of a basis which is a path through a packet ordered non-decimated wavelet transform. To view the basis just print it! See the examples in numtonv for a print out of its structure.

A similar object exists for describing a basis in a wavelet packet object see nvwp.

Value

The following components must be included in a legitimate 'nv' object.

node.list This is a complicated structure composed of one-dimensional array of nv$nlevelsWT lists. Each item in the array is itself a list having two components$upperctrl and upperl. Each component is described as follows:

upperctrl The 'upperctrl' item in each is the most important. It consists of a vector of characters. Each character refers to a node in the non-decimated wavelet tree at that level and can only be one of the characters L (for left), R (for right) and S (for stop). Each character in the vector informs reconstruction algorithms that, to do the best thing (whatever the best thing is in any particular case, e.g. select the minimum entropy node downwards), you should select the left/right node or stop at the current node.

upperl The 'upperl' vector is in 1-1 correspondance with the 'upperctrl' vector. Each entry is a number related in some way to the L/R/S entry. (For the minumum entropy this is the minmum entropy achieved by this selection).

nlevelsWT The number of levels in the wst object that was involved in the creation of the nv object. Nv objects describe a basis relative to a packet ordered non-decimated wavelet transform object and thus must know the number of levels in that object.

GENERATION

This class of objects is returned from the MaNoVe.wst and numtonv functions. The former returns the minimum entropy basis (most sparse basis) obtained using the Coifman-Wickerhauser, 1992 algorithm. The latter permits selection of a basis by an index number.

METHODS

The nv class of objects has methods for the following generic functions: print, nlevelsWT, InvBasis,

RELEASE

Version 3.6.0 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

wst, wst.object, numtonv, print, nlevelsWT, InvBasis, MaNoVe.wst.

---

plot.imwd *Draw a picture of the 2D wavelet coefficients using image*

Description

This function images 2D the absolute values discrete wavelet transform coefficients arising from a imwd.object object.

Usage

```
## S3 method for class 'imwd'
plot(x, scaling = "by.level", co.type = "abs",
 package = "R", plot.type = "mallat", arrangement = c(3, 3),
 transform = FALSE, tfunction = sqrt, ...)
## S3 method for class 'imwdc'
plot(x, verbose=FALSE, ...)
```

Arguments

x	The 2D imwd object you wish to depict
scaling	How coefficient scaling is performed. The options are by.level to scale the coefficients independently by level, anything else causes coefficients to be scaled globally
co.type	Can be "abs" for the absolute values of the coefficients to be plotted, can be "mabs" for the negative absolute values or "none" for none of this.
package	Can be "R" for the R package, or "S". The latter does less interesting things and results in a simpler plot
plot.type	If this argument is "mallat" the coefficients at different scales and orientations are packed into one image and plotted, a format originating from Mallat's early papers on this. The other possibility is "cols" which plots each combination of scale and direction on a separate plot. This latter format is useful for examining coefficients, especially at the coarser scales.

arrangement	If plot.type="cols" then this argument specifies how many rows and columns there are in the plot array.
transform	If FALSE then the coefficients are plotted as they are (subject to the co.type argument above), if TRUE then the transform function supplied by tfunction is applied to the coefficients.
tfunction	If transform=TRUE then this function gets applied to transform the coefficients before plotting
verbose	Print out informative messages
...	Supply other arguments to the call to the image function. This is very useful to, e.g., can the colours, or other aspects of the image

Details

Description says all

Value

If the package="S" argument is set then a matrix is returned containing the image that would have been plotted (and this only works if the plot.type="mallat" argument is set also.

Author(s)

G P Nason

See Also

imwd, imwd.object, threshold.imwd

Examples

```
data(lennon)
lwd <- imwd(lennon)
## Not run: plot(lwd)
## Not run: plot(lwd, col=grey(seq(from=0, to=1, length=100)), transform=TRUE)
```

---

plot.irregwd *Plot variance factors of wavelet transform coefficients for irregularly spaced wavelet transform object*

Description

This function plots the variance factors associated with the wavelet coefficients arising from a irregwd.objects irregularly spaced wavelet decomposition object.

Usage

```
## S3 method for class 'irregwd'
plot(x, xlabels, first.level = 1,
 main = "Wavelet Decomposition Coefficients", scaling = "by.level",
 rhlab = FALSE, sub, ...)
```

Arguments

x	The irregwd.objects object whose coefficients you wish to plot.
xlabels	A vector containing the "true" x-axis numbers that went with the vector that was transformed to produce the irregwd object supplied as the first argument to this function. If this argument is missing then the function tries to make up a sensible set of x-axis labels.
first.level	The first resolution level to begin plotting at. This argument can be quite useful when you want to supress some of the coarser levels in the diagram.
main	The main title of the plot.
scaling	How you want the coefficients to be scaled. The options are: global - one scale factor is chosen for the whole plot. The scale factor depends on the variance factor to be included on the plot that has the largest absolute value. The global option is useful when comparing factors that might appear anywhere in the plot; by.level - a scale factor is chosen for each resolution level in the plot. The scale factor for a level depends on the variance factor in that level that has the largest absolute value. The by.level option is useful when you wish to compare coefficients within a resolution level.
rhlab	If TRUE then a set of labels is produced on the right hand axis. The axis labels in this case refer to the scale factors used to scale each level and correspond to value of the largest variance factor (in absolute value) in each scale (when scaling=="by.level") or absolutely (when scaling="global"). If the rhlab=FALSE then no right hand axis labels are produced.
sub	A subtitle for the plot.
...	Other arguments supplied to the actual plot

Details

Produces a plot similar in style to the ones in Donoho and Johnstone, 1994. This function is basically the same as plot.wd except that variance factors and not coefficients are plotted. A variance factor is a number that quantifies the variability of a coefficient induced by the irregular design that was interpolated to a regular grid by the makegrid function which is used the by irregwd irregular wavelet transform function.

High values of the variance factor correspond to large variance in the wavelet coefficients but due to the irregular design, not the original noise structure on the coefficients.

Value

If rhlab==TRUE then the scaling factors applied to each scale level are returned. Otherwise NULL is returned.

Author(s)

Arne Kovac

Examples

```
#
# The help for makegrid contains an example
# of using this function.
#
```

`plot.mwd` *Use plot on an mwd object.*

Description

Plots the wavelet coefficients of a `mwd` class object.

Usage

```
## S3 method for class 'mwd'
plot(x, first.level = 1, main = "Wavelet Decomposition Coefficients",
        scaling = "compensated", rhlab = FALSE, sub = x$filter$name,
 NotPlotVal = 0.05, xlab = "Translate", ylab = "Resolution level",
 return.scale = TRUE, colour = (2:(npsi + 1)), ...)
```

Arguments

`x`	The `mwd` object whose coefficients you wish to plot.
`first.level`	The first resolution level to begin plotting at. This argument can be quite useful when you want to supress some of the coarser levels in the diagram.
`main`	The main title of the plot.
`scaling`	How you want the coefficients to be scaled. The options are: `"global"` - one scale factor is chosen for the whole plot. The scale factor depends on the coefficient to be included on the plot that has the largest absolute value. The global option is useful when comparing coefficients that might appear anywhere in the plot; `"by.level"` - a scale factor is chosen for each resolution level in the plot. The scale factor for a level depends on the coefficient in that level that has the largest absolute value. The `"by.level"` option is useful when you wish to compare coefficients within a resolution level. The other option is `"compensated"` which is the same as `"global"` except for that finer scales' coefficients are scaled up by a factor of SQRT(2) for `compensated`. This latter options is sometimes useful.
`rhlab`	If `T` then a set of labels is produced on the right hand axis. The axis labels in this case refer to the scale factors used to scale each level and correspond to value of the largest coefficient (in absolute value) in each scale (when `scaling=="by.level"`) or absolutely (when `scaling="global"`). If the `rhlab` argument is `FALSE` then no right hand axis labels are produced.

`sub`	A subtitle for the plot.
`NotPlotVal`	Doesn't seem to be implemented.
`xlab`	A title for the x-axis
`ylab`	A title for the y-axis
`return.scale`	If true (default) the scale for each resolution level is returned
`colour`	A vector of length `mwd$npsi`, the values of which are the colours used to plot the coefficients, one for each distinct type of wavelet (with apologies to our American cousins for spelling colour correctly!)
`...`	other arguments to be supplied to plot.

Details

Produces a plot similar to the ones in Donoho and Johnstone, 1994.

Wavelet coefficients for each resolution level are plotted one above the other, with the high resolution coefficients at the bottom, and the low resolution at the top. Each vector is represented by `mwd$npsi` lines one for each element in the coefficient vector. If colour is supported by the device each element will be represented by a different coulour. The coefficients are plotted using the `segment` function, with a large positive coefficient being plotted above an imaginary horizontal centre line, and a large negative coefficient plotted below it. The position of a coefficient along a line is indicative of the wavelet basis function's translate number.

The resolution levels are labelled on the left-hand side axis, and the maximum values of the absolute values of the coefficients for the particular level form the right-hand side axis.

The levels of coefficients can be scaled in three ways. If you are not interested in comparing the relative scales of coefficients from different levels, then the default scaling option, `"by.level"` is what you need. This computes the maximum of the absolute value of the coefficients at a particular level and scales the so that the fit nicely onto the plot. For this option, each level is scaled **DIFFERENTLY**. To obtain a uniform scale for all the levels specify the `"global"` option to the `scaling` argument. This will allow you to make inter-level comparisons.

Value

Axis labels for each resolution level unless `return.scale=F` when `NULL` is returned. The axis values are the maximum of the absolute value of the coefficients at that resolution level. They are returned because they are sometimes hard to read on the plot.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6).

Note

A plot of the coefficients contained within the `mwd` object at each resolution level is produced.

Author(s)

G P Nason

See Also

accessC.mwd, accessD.mwd, draw.mwd, mfirst.last, mfilter.select, mwd, mwd.object, mwr, print.mwd, putC.mwd, putD.mwd, summary.mwd, threshold.mwd, wd, wr.mwd.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
# Decompose test.data with multiple wavelet transform and
# plot the wavelet coefficients
#
tdmwd <- mwd(test.data)
## Not run: plot(tdmwd)
#[1] 1.851894 1.851894 1.851894 1.851894 1.851894 1.851894 1.851894
#
# You should see a plot with wavelet coefficients like in
# plot.wd but at each coefficient position
# there are two coefficients in two different colours one for each of
# the wavelets at that position.
#
# Note the scale for each level is returned by the function.
```

---

plot.nvwp — *Depict wavelet packet basis specfication*

Description

The nvwp class object (generated from MaNoVe.wp for example) contains a wavelet packet basis specification. This function produces a graphical depiction of such a basis.

Usage

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

Arguments

x	The wavelet packet node vector you wish to plot, nvwp class object
...	Other arguments to the central plot function

Details

The vertical axis indicates the resolution level, the horizontal axes indicates the packet index for the finest scales.

Value

Nothing

Author(s)

G P Nason

See Also

MaNoVe.wp,print.nvwp,wp

Examples

```
v <- rnorm(512)
vwp <- wp(v)
vnv <- MaNoVe(vwp)
## Not run: plot(vnv)
```

---

plot.wd — *Plot wavelet transform coefficients.*

Description

This function plots discrete wavelet transform coefficients arising from a wd object.

Usage

```
## S3 method for class 'wd'
plot(x,xlabvals, xlabchars, ylabchars, first.level = 0,
 main = "Wavelet Decomposition Coefficients", scaling = "global",
 rhlab = FALSE, sub, NotPlotVal = 0.005, xlab = "Translate",
 ylab = "Resolution Level",
 aspect = "Identity", ...)
```

Arguments

x	The wd class object you wish to plot
xlabvals	A vector containing the "true" x-axis numbers that went with the vector that was transformed to produce the wd object supplied as the first argument to this function. If this argument is missing then the function tries to make up a sensible set of x-axis labels.
xlabchars	Tickmark labels for the x axis
ylabchars	Tickmark labels for the y axis
first.level	The first resolution level to begin plotting at. This argument can be quite useful when you want to supress some of the coarser levels in the diagram.
main	The main title of the plot.

`scaling` How you want the coefficients to be scaled. The options are: `global` - one scale factor is chosen for the whole plot. The scale factor depends on the coefficient to be included on the plot that has the largest absolute value. The `global` option is useful when comparing coefficients that might appear anywhere in the plot; `by.level` - a scale factor is chosen for each resolution level in the plot. The scale factor for a level depends on the coefficient in that level that has the largest absolute value. The `by.level` option is useful when you wish to compare coefficients within a resolution level.

The two other options are compensated and super which are the same as `global` except for that finer scales' coefficients are scaled up by a factor of SQRT(2) for compensated and 2 for super. These latter two options are sometimes useful (more useful for non-decimated `wd` objects, where they act as a sort of `ipndacw` matrix operator).

`rhlab` If `TRUE` then a set of labels is produced on the right hand axis. The axis labels in this case refer to the scale factors used to scale each level and correspond to value of the largest coefficient (in absolute value) in each scale (when `scaling=="by.level"`) or absolutely (when `scaling="global"`). If the `rhlab` argument is `FALSE` then no right hand axis labels are produced.

`sub` A subtitle for the plot.

`NotPlotVal` This argument ensures that if all (scaled) coefficients in a resolution level are below `NotPlotVal` in absolute value then the whole resolution level is not plotted. This can be useful when plotting a `wd` object that is sparse (or has been thresholded and necessarily many coefficients might well be zero) as it speeds up the plot because whole levels do not have to be plotted (the function that does the plotting [`segments()`] is quite a slow function). Note that the value of `NotPlotVal` refers to `scaled` coefficients, those that have been scaled by this function (on any resolution level all coefficients are scaled to lie between -0.5 and 0.5).

`xlab` A title for the x-axis

`ylab` A title for the y-axis

`aspect` This argument describes the name (as a character string) of a function to be applied to the coefficients before plotting. By default the argument is `"Identity"`, i.e. the coefficients are plotted *as is*. This argument is most useful when a complex-valued wavelets are plotted you could use `"Mod"` to plot the modulus of the coefficients, or `"Re"` to plot the real parts of the coefficients or `"Arg"` to plot the argument of the coefficients. Also, the `aspect` argument can be useful for the ordinary wavelet transforms as well if you are interested in a particular transform of the coefficients.

`...` fine tuning

Details

Produces a plot similar to the ones in Donoho and Johnstone, 1994.

A wavelet decomposition of a signal consists of discrete wavelet coefficients at different scales (resolution levels) and locations. This function plots the coefficients as a pyramid (derived from Mallat's pyramid algorithm). See the examples below.

The resolution levels are stacked one above the other: coarse scale coefficients are always towards the top of the plot, fine scale coefficients are always located toward the bottom of the plot. The location of coefficients increases from left to right across the plot in synchrony with the input signal to the wd object. In other words the position of a coefficient along a line is indicative of the associated wavelet basis function's translate number. The actual coefficients are plotted using S-Plus's `segments()` function. This plots each coefficient as a vertical line with positive coefficients being plotted above an imaginary centre line and negative coefficients being plotted below.

The resolution levels are labelled on the left-hand side axis, and if `rhlab==T` the maximum values of the absolute values of the coefficients, for the particular level, are plotted on the right-hand axis.

The coefficients in the plot may be scaled in 4 ways. If you are interested in comparing coefficients in different levels then the default scaling option `scaling=="global"` is what you need. This works by finding the coefficient with the largest absolute value amongst all coeffients to be plotted and then scales all the other coefficients by the largest so that all coefficients lie in the range -1/2 to 1/2. The scaled coefficients are then plotted. If you are not interested in comparing relative resolution levels and want to see all that goes on within a particular scale then you should use the scaling option `scaling=="by.level"` which picks out the largest coefficient (in absolute value) from each level and scales each level separately. The "`compensated`" and super options are like the "`global`" option except that finer levels are scaled up (as discussed in the arguments list above): this can be useful when plotting non-decimated wavelet transform coefficients as it emphasizes the higher frequencies.

Value

If `rhlab==T` then the scaling factors applied to each scale level are returned. Otherwise NULL is returned.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Note

A plot of the coefficients contained within the wd object is produced.

Author(s)

G P Nason

See Also

wd

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
```

```
# Decompose test.data and plot the wavelet coefficients
#
wds <- wd(test.data)
## Not run: plot(wds)
#
# Now do the time-ordered non-decimated wavelet transform of the same thing
#
## Not run: wdS <- wd(test.data, type="station")
## Not run: plot(wdS)
#
# Next examples
# ------------
# The chirp signal is also another good examples to use.
#
# Generate some test data
#
test.chirp <- simchirp()$y
## Not run: ts.plot(test.chirp, main="Simulated chirp signal")
#
# Now let's do the time-ordered non-decimated wavelet transform.
# For a change let's use Daubechies least-asymmetric phase wavelet with 8
# vanishing moments (a totally arbitrary choice, please don't read
# anything into it).
#
chirpwdS <- wd(test.chirp, filter.number=8, family="DaubLeAsymm", type="station")
## Not run: plot(chirpwdS, main="TOND WT of Chirp signal")
```

`plot.wp` *Plot wavelet packet transform coefficients*

Description

This function plots wavelet packet transform coefficients arising from a `wp.object` object.

Usage

```
## S3 method for class 'wp'
plot(x, nvwp = NULL, main = "Wavelet Packet Decomposition",
    sub, first.level = 5, scaling = "compensated", dotted.turn.on = 5,
    color.force = FALSE, WaveletColor = 2, NodeVecColor = 3,
    fast = FALSE, SmoothedLines = TRUE, ...)
```

Arguments

`x`	The wp object whose coefficients you wish to plot.
`nvwp`	An optional associated wavelet packet node vector class object of class `nvwp`. This object is a list of packets in the wavelet packet table. If this argument is specified then it is possible to highlight the packets in the nvwp objects in a different color using the `NodeVecColor` argument

`main` The main title of the plot.

`sub` A subtitle for the plot.

`first.level` The first resolution level to begin plotting at. This argument can be quite useful when you want to supress some of the coarser levels in the diagram.

`scaling` How you want the coefficients to be scaled. The options are: `global` - one scale factor is chosen for the whole plot. The scale factor depends on the coefficient to be included on the plot that has the largest absolute value. The global option is useful when comparing coefficients that might appear anywhere in the plot; `by.level` - a scale factor is chosen for each resolution level in the plot. The scale factor for a level depends on the coefficient in that level that has the largest absolute value. The `by.level` option is useful when you wish to compare coefficients within a resolution level. The other option is `compensated` which is the same as global except for that finer scales' coefficients are scaled up by a factor of SQRT(2)

I don't know why compensated is the default option? That is probably silly!

`dotted.turn.on` The plot usually includes some dotted vertical bars that separate wavelet packets to make it clearer which packets are which. This option controls the coarsest resolution level at which dotted lines appear. All levels equal to and finer than this level will receive the vertical dotted lines.

`color.force` If FALSE then some "clever" code in CanUseMoreThanOneColor tries to figure out how many colours can be used (THIS HAS NOT BEEN MADE TO WORK IN R) and hence whether colour can be used to pick out wavelet packets or elements of a node vector. This option was designed to work with S. It doesn't work with R and so it is probably best to set `color.force=T`. In this way no interrogation is done and the lines/packets are plotted in the appropriate colours with no questions asked.

`WaveletColor` A colour specification for the colour for wavelet coefficients. Wavelet coefficients are a component of wavelet packet coefficients and this option allows them to be drawn in a different color. In R you can use names like "red", "blue" to select the colors. In R you'll also need to set the color.force option to TRUE.

`NodeVecColor` If a nvwp object is supplied this option can force coefficients that are part of that nvwp to be drawn in the specified color. See the explanation for the `WaveletColor` option above about specification in R.

`fast` This option no longer does anything.

`SmoothedLines` If TRUE then the scaling function coefficients are drawn using lines (and look like mini versions of the original). If FALSE then the scaling function coefficients are drawn using the `segments` function and look like a coarser shadowy version of the original.

`...` Other arguments to the plot command

Details

A wavelet packet object contains wavelet packet coefficients of a signal (usually obtained by the `wp` wavelet packet transform function). Given a wavelet packet object wp it possesses `nlevelsWT(wp)` resolution levels. In WaveThresh the coarsest level is level 0 and the finest is level nlevelsWT-1. For wavelet packets the number of packets at level j is 2^(nlevelsWT-j).

This function plots the wavelet packet coefficients. At the bottom of the plot the original input function (if present) is plotted. Then levels above the original plot successively coarser wavelet packet coefficients. From the Mallat transform point of view smoothing goes up off the the left of the picture and detail to the right. The packets are indexed from 0 to the number of packets going from left to right within each resolution level.

The function has the ability to draw wavelet coefficients in a different color using the `WaveletColor` argument.

Optionally, if a node vector wavelet packet object is also supplied, which contains the specification of a basis selected from the packet table, then packets in that node vector can be highlighted in a another colour determined by the `NodeVecColor`.

Packets are drawn on the plot and can be separated by vertical dotted lines. The resolution levels at which this happens can be controlled by the `dotted.turn.on` option. The coarsest resolution level to be drawn is controlled by the `first.level` option.

Value

Nothing

Author(s)

G P Nason

See Also

`MaNoVe`, `wp`, `wp.object`

Examples

```
#
# Generate some test data
#
v <- DJ.EX()$blocks
#
# Let's plot these to see what they look like
#
## Not run: plot(v, type="l")
#
# Do a wavelet packet transform
#
vwp <- wp(v)
#
# And create a node vector
#
vnv <- MaNoVe(vwp)
#
# Now plot the wavelet packets with the associated node vector
#
## Not run: plot(vwp, vnv, color.force=T, WaveletColor="red", dotted.turn.on=7)
#
# The wavelet coefficients are plotted in red. Packets from the node vector
```

```
# are depicted in green. The node vector gets plotted after the wavelet
# coefficients so the green packets overlay the red (retry the plot command
# but without the vnv object to see just the
# wavelet coefficients). The vertical dotted lines start at resolution
# level 7.
#
#
```

`plot.wst` *Plot packet-ordered non-decimated wavelet transform coefficients.*

Description

This function plots packet-ordered non-decimated wavelet transform coefficients arising from a `wst.object` object.

Usage

```
## S3 method for class 'wst'
plot(x, main = "Nondecimated Wavelet (Packet) Decomposition",
    sub, first.level = 5, scaling = "compensated", dotted.turn.on = 5,
    aspect = "Identity", ...)
```

Arguments

`x`	The wst object whose coefficients you wish to plot.
`main`	The main title of the plot.
`sub`	A subtitle for the plot.
`first.level`	The first resolution level to begin plotting at. This argument can be quite useful when you want to suppress some of the coarser levels in the diagram.
`scaling`	How you want the coefficients to be scaled. The options are: `global` - one scale factor is chosen for the whole plot. The scale factor depends on the coefficient to be included on the plot that has the largest absolute value. The global option is useful when comparing coefficients that might appear anywhere in the plot; `by.level` - a scale factor is chosen for each resolution level in the plot. The scale factor for a level depends on the coefficient in that level that has the largest absolute value. The `by.level` option is useful when you wish to compare coefficients within a resolution level. The other option is `compensated` which is the same as global except for that finer scales' coefficients are scaled up by a factor of SQRT(2) I don't know why compensated is the default option? It is a bit silly.
`dotted.turn.on`	The plot usually includes some dotted vertical bars that separate wavelet packets to make it clearer which packets are which. This option controls the coarsest resolution level at which dotted lines appear. All levels equal to and finer than this level will receive the vertical dotted lines.

aspect	A transform to apply to the coefficients before plotting. If the coefficients are complex-valued and aspect="Identity" then the modulus of the coefficients are plotted.
...	Other arguments to plot

Details

A packet-ordered non-decimated wavelet object contains coefficients of a signal (usually obtained by the `wst` packet-ordered non-decimated wavelet transform, but also functions that derive such objects, such as `threshold.wst`).

A packet-ordered nondecimated wavelet object, x, possesses `nlevelsWT(x)` resolution levels. In WaveThresh the coarsest level is level 0 and the finest is level `nlevelsWT-1`. For packet-ordered nondecimated wavelet the number of blocks (packets) at level `j` is `2^(nlevelsWT-j)`.

This function plots the coefficients. At the bottom of the plot the original input function (if present) is plotted. Then levels above the original plot successively coarser wavelet coefficients. Each packet of coefficients is plotted within dotted vertical lines. At the finest level there are two packets: one (the left one) correspond to the wavelet coefficients that would be obtained using the (standard) decimated wavelet transform function, `wd`, and the other packet are those coefficients that would have been obtained using the standard decimated wavelet transform after a unit cyclic shift.

For coarser levels there are more packets corresponding to different cyclic shifts (although the computation is not performed using shifting operations the effect is the same). For full details see Nason and Silverman, 1995.

Packets are drawn on the plot and can be separated by vertical dotted lines. The resolution levels at which this happens can be controlled by the `dotted.turn.on` option. The coarsest resolution level to be drawn is controlled by the `first.level` `option`.

It should be noted that the packets referred to here are just the blocks of nondecimated wavelet coefficients in a packet-ordering. These are different to wavelet packets (produced by `wp`*) and nondecimated wavelet packets (produced by* `wpst`*)*

Value

Nothing

Author(s)

G P Nason

See Also

`MaNoVe`,`threshold.wst`, `wst`, `wst.object`

Examples

```
#
# Generate some test data
#
v <- DJ.EX()$heavi
#
```

```
# Let's plot these to see what they look like
#
## Not run: plot(v, type="l")
#
# Do a packet-ordered non-decimated wavelet packet transform
#
vwst <- wst(v)
#
# Now plot the coefficients
#
## Not run: plot(vwst)
#
# Note that the "original" function is at the bottom of the plot.
# The finest scale coefficients (two packets) are immediately above.
# Increasingly coarser scale coefficients are above that!
#
```

`plot.wst2D` *Plot packet-ordered 2D non-decimated wavelet coefficients.*

Description

This function plots packet-ordered 2D non-decimated wavelet coefficients arising from a `wst2D` object.

Usage

```
## S3 method for class 'wst2D'
plot(x, plot.type="level", main="", ...)
```

Arguments

`x`	The `wst2D` object whose coefficients you wish to plot.
`plot.type`	So far the only valid argument is "level" which plots coefficients a level at a time.
`main`	The main title of the plot.
`...`	Any other arguments.

Details

The coefficients in a `wst2D` object are stored in a three-dimensional subarray called `wst2D`. The first index of the 3D array indexes the resolution level of coefficients: this function with `plot.type="level"` causes an image of coefficients to be plotted one for each resolution level.

The following corresponds to images produced on S+ graphics devices (e.g. image on `motif()`). Given a resolution level there are `4^(nlevelsWT-level)` packets within a level. Each packet can be addressed by a base-4 string of length `nlevels-level`. A zero corresponds to no shift, a 1 to a horizontal shift, a 2 to a vertical shift and a 3 to both a horizontal and vertical shift.

So, for examples, at resolution level `nlevelsWT-1` there are 4 sub-images each containing 4 sub-images. The main subimages correspond to (clockwise from bottom-left) no shift, horizontal shift, both shift and vertical shifts. The sub-images of the sub-images correspond to the usual smooth, horizontal detail, diagonal detail and vertical detail (clockwise, again from bottom left). Coarser resolution levels correspond to finer shifts! The following figure demonstrates the `nlevels-1` resolution level for the `ua` image (although the whole image has been rotated by 90 degrees clockwise for display here!):

Value

A plot of the coefficients contained within the `wst2D` object is produced.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

`getpacket.wst2D`, `putpacket.wst2D`, `wst2D`, `wst2D.object`.

Examples

```
#
# The above picture is one of a series produced by
#
#plot(uawst2D)
#
# Where the uawst2D object was produced in the EXAMPLES section
# of the help for \code{\link{wst2D}}
```

`plotdenwd` *Plot the wavelet coefficients of a p.d.f.*

Description

Plots the wavelet coefficients of a density function.

Usage

```
plotdenwd(wd, xlabvals, xlabchars, ylabchars, first.level=0,
 top.level=nlevelsWT(wd)-1,
 main="Wavelet Decomposition Coefficients", scaling="global",
 rhlab=FALSE, sub, NotPlotVal=0.005, xlab="Translate",
 ylab="Resolution Level", aspect="Identity", ...)
```

Arguments

`wd`	Wavelet decomposition object, usually output from `denwd`, possibly thresholded.
`xlabvals`	X-axis values at which the `xlabchars` will be printed
`xlabchars`	The x-label characters to be plotted at `xlabvals`
`ylabchars`	The y-label characters
`first.level`	This specifies how many of the coarse levels of coefficients are omitted from the plot. The default value of 0 means that all levels are plotted.
`top.level`	This tells the plotting rountine the true resolution level of the finest level of coefficients. The default results in the coarsest level being labelled 0. The "correct" value can be determined from the empirical scaling function coefficient object (output from denproj) as in the example below.
`main`	The title of the plot.
`scaling`	The type of scaling applied to levels within the plot. This can be "compensated", "by.level" or "global". See `plot.wd` for further details.
`rhlab`	Determines whether the scale factors applied to each level before plotting are printed as the right hand axis.
`sub`	The plot subtitle
`NotPlotVal`	If the maximum coefficient in a particular level is smaller than `NotPlotVal`, then the level is not plotted.
`xlab`	The x-axis label
`ylab`	The y-axis label
`aspect`	Function to apply to coefficients before plotting
`...`	Other arguments to the main plot routine

Details

Basically the same as `plot.wd` except that it copes with the zero boundary conditions used in density estimation. Note that for large filter number wavelets the high level coefficients will appear very squashed compared with the low level coefficients. This is a consequence of the zero boundary conditions and the use of the convention that each coefficient is plotted midway between two coefficients at the next highest level, as in `plot.wd`.

Value

Axis labels to the right of the picture (scale factors). These are returned as they are sometimes hard to read on the plot.

Author(s)

David Herrick

Examples

```
# Simulate data from the claw density, find the empirical
# scaling function coefficients, decompose them and plot
# the resulting wavelet coefficients

data <- rclaw(100)
datahr <- denproj(data, J=8, filter.number=2, family="DaubExPhase")
data.wd <- denwd(datahr)
## Not run: plotdenwd(data.wd, top.level=(datahr$res$J-1))
#
# Now use a smoother wavelet
#
datahr <- denproj(data, J=8, filter.number=10, family="DaubLeAsymm")
data.wd <- denwd(datahr)
## Not run: plotdenwd(data.wd, top.level=(datahr$res$J-1))
```

`plotpkt` *Sets up a high level plot ready to show the time-frequency plane and wavelet packet basis slots*

Description

Sets up a high level plot ready to add wavelet packet slots using, e.g. `addpkt`. This function is used by several routines to begin plotting graphical representations of the time-frequency plane and spaces for packets.

Usage

```
plotpkt(J)
```

Arguments

`J`	The number of resolution levels associated with the wavelet packet object you want to depict

Details

Description says all

Value

Nothing of interest

Author(s)

G P Nason

See Also

`addpkt`, `basisplot`, `basisplot.BP`, `basisplot.wp`, `plot.nvwp`

print.BP — *Print top best basis information for BP class object*

Description

The function Best1DCols works out what are the best packets in a selection of packets. This function prints out what the best packet are.

The Best1DCols is not intended for user use, and hence neither is this print method.

Usage

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

Arguments

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

Details

Description says all

Value

None.

Author(s)

G P Nason

See Also

Best1DCols

print.imwd — *Print out information about an imwd object in readable form.*

Description

This function prints out information about an imwd.object in a nice human-readable form.

Note that this function is automatically called by SPlus whenever the name of an imwd.object is typed or whenever such an object is returned to the top level of the S interpreter.

Usage

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

Arguments

x	An object of class imwd that you wish to print out.
...	This argument actually does nothing in this function!

Details

Prints out information about imwd objects in nice readable format.

Value

The last thing this function does is call summary.imwd so the return value is whatever is returned by this function.

RELEASE

Version 3.0 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

imwd.object, summary.imwd.

Examples

```
#
# Generate an imwd object.
#
tmp <- imwd(matrix(0, nrow=32, ncol=32))
#
# Now get R to use print.imwd
#
tmp
# Class 'imwd' : Discrete Image Wavelet Transform Object:
#        ~~~~  : List with 27 components with names
#              nlevelsWT fl.dbase filter type bc date w4L4 w4L1 w4L2 w4L3
# w3L4 w3L1 w3L2 w3L3 w2L4 w2L1 w2L2 w2L3 w1L4 w1L1 w1L2 w1L3 w0L4 w0L1
# w0L2 w0L3 w0Lconstant
#
# $ wNLx are LONG coefficient vectors !
#
# summary(.):
# ----------
# UNcompressed image wavelet decomposition structure
```

```
# Levels:  5
# Original image was 32 x 32  pixels.
# Filter was:  Daub cmpct on least asymm N=10
# Boundary handling:  periodic
```

`print.imwdc` *Print out information about an imwdc object in readable form.*

Description

This function prints out information about an `imwdc.object` in a nice human-readable form.

Note that this function is automatically called by SPlus whenever the name of an `imwdc.object` is typed or whenever such an object is returned to the top level of the S interpreter.

Usage

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

Arguments

`x`	An object of class imwdc that you wish to print out.
`...`	This argument actually does nothing in this function!

Details

Prints out information about `imwdc` objects in nice readable format.

Value

The last thing this function does is call `summary.imwdc` so the return value is whatever is returned by this function.

RELEASE

Version 2.2 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

`imwdc.object`, `summary.imwdc`.

Examples

```
#
# Generate an imwd object.
#
tmp <- imwd(matrix(0, nrow=32, ncol=32))
#
# Now get R to use print.imwd
#
tmp
# Class 'imwd' : Discrete Image Wavelet Transform Object:
#        ~~~~  : List with 27 components with names
#              nlevelsWT fl.dbase filter type bc date w4L4 w4L1 w4L2 w4L3
# w3L4 w3L1 w3L2 w3L3 w2L4 w2L1 w2L2 w2L3 w1L4 w1L1 w1L2 w1L3 w0L4 w0L1
# w0L2 w0L3 w0Lconstant
#
# $ wNLx are LONG coefficient vectors !
#
# summary(.):
# ----------
# UNcompressed image wavelet decomposition structure
# Levels:  5
# Original image was 32 x 32  pixels.
# Filter was:  Daub cmpct on least asymm N=10
# Boundary handling:  periodic
```

`print.mwd` *Use print() on a mwd object.*

Description

This function prints out information about an `mwd.object` in a nice human-readable form.

Note that this function is automatically called by SPlus whenever the name of an `mwd.object` is typed or whenever such an object is returned to the top level of the S interpreter.

Usage

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

Arguments

`x`	An object of class mwd that you wish to print out.
`...`	This argument actually does nothing in this function!

Details

Prints out information about `mwd` objects in nice readable format.

Value

The last thing this function does is call summary.mwd so the return value is whatever is returned by this function.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6)

Author(s)

G P Nason

See Also

accessC.mwd, accessD.mwd, draw.mwd, mfirst.last, mfilter.select,mwd, mwd.object, mwr, plot.mwd, putC.mwd, putD.mwd, summary.mwd, threshold.mwd, wd, wr.mwd.

Examples

```
#
# Generate an mwd object.
#
tmp <- mwd(rnorm(32))
#
# Now get Splus to use print.mwd
#
tmp
# Class 'mwd' : Discrete Multiple Wavelet Transform Object:
#  ~~~  : List with 10 components with names
#   C D nlevelsWT ndata filter fl.dbase type bc prefilter date
#
# $ C and $ D are LONG coefficient vectors !
#
# Created on : Tue Nov 16 13:16:07 GMT 1999
# Type of decomposition:  wavelet
#
# summary:
# ----------
# Length of original:  32
# Levels:  4
# Filter was:  Geronimo Multiwavelets
# Scaling fns:  2
# Wavelet fns:  2
# Prefilter:  default
# Scaling factor:  2
# Boundary handling:  periodic
# Transform type:  wavelet
# Date:  Tue Nov 16 13:16:07 GMT 1999
```

`print.nv` *Print a node vector object, also used by several other functions to obtain packet list information*

Description

Ostensibly prints out node vector information, but also produces packet indexing information for several functions.

Usage

```
## S3 method for class 'nv'
print(x, printing = TRUE, verbose = FALSE, ...)
```

Arguments

`x`	The `nv.object` that you wish to print
`printing`	If FALSE then nothing is printed. This argument is here because the results of the printing are also useful to many other routines where you want the results but are not bothered by actually seeing the results
`verbose`	Not actually used
`...`	Other arguments

Details

A node vector contains selected basis information, but this is stored as a tree object. Hence, it is not immediately obvious which basis elements have been stored. This function produces a list of the packets at each resolution level that have been selected in the basis. This information is so useful to other functions that the function is used even when printing is not the primary objective.

Value

A list containing two components: `indexlist` and `rvector`. The former is a list of packets that were selected at each resolution level. Rvector encodes a list of "rotate/non-rotate" instructions in binary. At each selected packet level a decision has to be made whether to select the LH or RH basis element, and this information is stored in `rvector`.

Author(s)

G P Nason

See Also

`InvBasis.wst`, `nv.object`, `plot.wp`

Examples

```
v <- rnorm(128)
vwst <- wst(v)
vnv <- MaNoVe(vwst)
print(vnv)
#Level :  6  Action is  R (getpacket Index:  1 )
#Level :  5  Action is  L (getpacket Index:  2 )
#Level :  4  Action is  L (getpacket Index:  4 )
#Level :  3  Action is  R (getpacket Index:  9 )
#Level :  2  Action is  L (getpacket Index:  18 )
#There are  6  reconstruction steps
#
# The L or R indicate whether to move to the left or the right basis function
# when descending the node tree
#
#
```

`print.nvwp` *Print a wavelet packet node vector object, also used by several other functions to obtain packet list information*

Description

Ostensibly prints out wavlet packet node vector information, but also produces packet indexing information for several functions.

Usage

```
## S3 method for class 'nvwp'
print(x, printing = TRUE, ...)
```

Arguments

`x`	The nvwp that you wish to print
`printing`	If FALSE then nothing is printed. This argument is here because the results of the printing are also useful to many other routines where you want the results but are not bothered by actually seeing the results
`...`	Other arguments

Details

A node vector contains selected basis information, but this is stored as a tree object. Hence, it is not immediately obvious which basis elements have been stored. This function produces a list of the packets at each resolution level that have been selected in the basis. This information is so useful to other functions that the function is used even when printing is not the primary objective.

Value

A list containing two components: `level` and `pkt`. These are the levels and packet indices of the select packets in the basis.

Author(s)

G P Nason

See Also

`InvBasis.wp`, `MaNoVe.wp`, `plot.nvwp`, `plot.wp`

Examples

```
v <- rnorm(128)
vwp <- wp(v)
vnv <- MaNoVe(vwp)
print(vnv)
#Level:  6  Packet:  1
#Level:  3  Packet:  0
#Level:  2  Packet:  4
#Level:  2  Packet:  13
#Level:  2  Packet:  15
#Level:  1  Packet:  5
#Level:  1  Packet:  10
#Level:  1  Packet:  13
#Level:  1  Packet:  14
#Level:  1  Packet:  15
#Level:  1  Packet:  16
#Level:  1  Packet:  20
#Level:  1  Packet:  21
#Level:  1  Packet:  24
#Level:  0  Packet:  8
#Level:  0  Packet:  9
#Level:  0  Packet:  12
#Level:  0  Packet:  13
#Level:  0  Packet:  14
#Level:  0  Packet:  15
#Level:  0  Packet:  22
#Level:  0  Packet:  23
#Level:  0  Packet:  24
#Level:  0  Packet:  25
#Level:  0  Packet:  34
#Level:  0  Packet:  35
#Level:  0  Packet:  36
#Level:  0  Packet:  37
#Level:  0  Packet:  38
#Level:  0  Packet:  39
#Level:  0  Packet:  44
#Level:  0  Packet:  45
#Level:  0  Packet:  46
#Level:  0  Packet:  47
```

```
#Level:  0  Packet:  50
#Level:  0  Packet:  51
#Level:  0  Packet:  56
#Level:  0  Packet:  57
#Level:  0  Packet:  58
#Level:  0  Packet:  59
```

`print.w2d` *Print method for printing w2d class objects*

Description

Prints information about a w2d class object. These objects are not typically directly used by a user.

Usage

```
## S3 method for class 'w2d'
print(x, ...)
```

Arguments

`x`	The w2d class object that you wish to print info about
`...`	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

`wpst2discr`

`print.w2m` *Print a w2m class object*

Description

These objects are the matrix representation of a nondecimated wavelet packet object

Usage

```
## S3 method for class 'w2m'
print(x, maxbasis = 10, ...)
```

Arguments

`x`	The w2m object to print
`maxbasis`	The maximum number of basis functions to report on
`...`	Other arguments

Details

Prints out information about a w2m object. This function gets called during `makewpstRO`, and so you can see its output in the example code in that help function

Value

None

Author(s)

G P Nason

See Also

`makewpstRO`,`wpst2m`

Examples

```
#
# See example in makewpstRO
#
```

`print.wd` *Print out information about an wd object in readable form.*

Description

This function prints out information about an `wd.object` in a nice human-readable form.

Note that this function is automatically called by SPlus whenever the name of an `wd.object` is typed or whenever such an object is returned to the top level of the S interpreter

Usage

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

Arguments

`x`	An object of class `wd` that you wish to print out.
`...`	This argument actually does nothing in this function!

Details

Prints out information about `wd` objects in nice readable format.

Value

The last thing this function does is call `summary.wd` so the return value is whatever is returned by this function.

RELEASE

Version 3.0 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

`wd.object`, `summary.wd`.

Examples

```
#
# Generate an wd object.
#
tmp <- wd(rnorm(32))
#
# Now get R to use print.wd
```

```
#
tmp
# Class 'wd' : Discrete Wavelet Transform Object:
#        ~~  : List with 8 components with names
#               C D nlevelsWT fl.dbase filter type bc date
#
# $ C and $ D are LONG coefficient vectors !
#
# Created on : Fri Oct 23 19:56:00 1998
# Type of decomposition:  wavelet
#
# summary(.):
# ----------
# Levels:  5
# Length of original:  32
# Filter was:  Daub cmpct on least asymm N=10
# Boundary handling:  periodic
# Transform type:  wavelet
# Date:  Fri Oct 23 19:56:00 1998
#
#
```

`print.wd3D` *Print out information about an wd3D object in a readable form.*

Description

This function prints out information about an `wd3D.object` in a readable form.

Note that this function is automatically called by SPlus whenever the name of an `wd3D.object` is typed or whenever such an object is returned to the top level of the S interpreter

Usage

```
## S3 method for class 'wd3D'
print(x, ...)
```

Arguments

`x`	An object of class `wd3D` that you wish to print out.
`...`	This argument actually does nothing in this function!

Details

Prints out information about `wd3D` objects in nice readable format.

Value

The last thing this function does is call `summary.wd3D` so the return value is whatever is returned by this function.

RELEASE

Version 3.9.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

accessD.wd3D, print.wd3D, putD.wd3D, putDwd3Dcheck, summary.wd3D, threshold.wd3D, wd3D, wd3D.object, wr3D.

Examples

```
#
# Generate an wd3D object.
#
tmp <- wd3D(array(rnorm(512), dim=c(8,8,8)))
#
# Now get R to use print.wd
#
tmp
#Class 'wd3d' : 3D DWT Object:
#       ~~~~  : List with 5 components with names
#              a filter.number family date nlevelsWT
#
#$ a is the wavelet coefficient array
#Dimension of a is [1] 8 8 8
#
#Created on : Wed Oct 20 17:24:15 BST 1999
#
#summary(.):
#----------
#Levels:  3
#Filter number was:  10
#Filter family was:  DaubLeAsymm
#Date:  Wed Oct 20 17:24:15 BST 1999
```

---

print.wp *Print out information about an wd object in readable form.*

Description

This function prints out information about an wp.object in a nice human-readable form.

Note that this function is automatically called by SPlus whenever the name of an wp.object is typed or whenever such an object is returned to the top level of the S interpreter

Usage

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

Arguments

x	An object of class wp that you wish to print out.
...	This argument actually does nothing in this function!

Details

Prints out information about wp objects in nice readable format.

Value

The last thing this function does is call summary.wp so the return value is whatever is returned by this function.

RELEASE

Version 3.0 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wp.object, summary.wp.

Examples

```
#
# Generate an wp object.
#
tmp <- wp(rnorm(32))
#
# Now get Splus to use print.wp
#
tmp
#
# Now get Splus to use print.wp
#
# tmp
# Class 'wp' : Wavelet Packet Object:
#        ~~  : List with 4 components with names
#               wp nlevelsWT filter date
#
# $wp is the wavelet packet matrix
#
# Created on : Fri Oct 23 19:59:01 1998
```

```
#
# summary(.):
# ----------
# Levels:  5
# Length of original:  32
# Filter was:  Daub cmpct on least asymm N=10
```

`print.wpst` *Prints out basic information about a wpst class object*

Description

Prints out basic information about a wpst class object generated by the, e.g., `wpst` function.

Note: stationary wavelet packet objects are now known as nondecimated wavelet packet objects.

Usage

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

Arguments

`x`	The wpst object that you wish to print info about
`...`	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

`wpst`

Examples

```
v <- rnorm(128)
vwpst <- wpst(v)
## Not run: print(vwpst)
#Class 'wpst' : Stationary Wavelet Packet Transform Object:
# ~~~  : List with 5 components with names
#  wpst nlevelsWT avixstart filter date
#
#$wpst is a coefficient vector
#
#Created on : Fri Mar  5 15:06:56 2010
#
#summary(.):
#----------
#Levels:  7
#Length of original:  128
#Filter was:  Daub cmpct on least asymm N=10
#Date:  Fri Mar  5 15:06:56 2010
```

`print.wpstCL` *Prints some information about a wpstCL object*

Description

Prints basic information about a wpstCL object

Usage

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

Arguments

`x`	wpstCL object to print info about
`...`	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

makewpstDO,wpstCLASS

Examples

```
#
# Use BabySS and BabyECG data for this example.
#
# Want to predict future values of BabySS from future values of BabyECG
#
# Build model on first 256 values of both
#
# See example in makewpstDO from which this one originates
#
data(BabyECG)
data(BabySS)
BabyModel <- makewpstDO(timeseries=BabyECG[1:256], groups=BabySS[1:256],
 mincor=0.5)
#
# Now, suppose we get some new data for the BabyECG time series.
# For the purposes of this example, this is just the continuing example
# ie BabyECG[257:512]. We can use our new discriminant model to predict
# new values of BabySS
#
BabySSpred <- wpstCLASS(newTS=BabyECG[257:512], BabyModel)
#
BabySSpred
#wpstCL class object
#Results of applying discriminator to time series
#Components:  BasisMatrix BasisMatrixDM wpstDO PredictedOP PredictedGroups
```

---

print.wpstDO *Print information about a wpstDO class object*

Description

Prints out the type of object, prints out the object's names, then uses print.BP to print out the best single packets.

Usage

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

Arguments

x	wpstDO object to print out
...	Other information to print

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

makewpstDO

Examples

```
#
# Use BabySS and BabyECG data for this example.
#
# Want to predict future values of BabySS from future values of BabyECG
#
# Build model on first 256 values of both
#
data(BabyECG)
data(BabySS)
BabyModel <- makewpstDO(timeseries=BabyECG[1:256], groups=BabySS[1:256],
 mincor=0.5)
#
# The results (ie print out answer)
BabyModel
#Stationary wavelet packet discrimination object
#Composite object containing components:[1] "BPd"    "BP"     "filter"
#Fisher's discrimination: done
#BP component has the following information
#BP class object. Contains "best basis" information
#Components of object:[1] "nlevelsWT"    "BasisMatrix" "level"     "pkt"       "basiscoef"
#[6] "groups"
#Number of levels  8
#List of "best" packets
#Level id Packet id Basis coef
#[1,]        4         0  0.7340580
#[2,]        5         0  0.6811251
#[3,]        6         0  0.6443167
#[4,]        3         0  0.6193434
#[5,]        7         0  0.5967620
#[6,]        0         3  0.5473777
#[7,]        1        53  0.5082849
#
```

`print.wpstRO` *Print a wpstRO class object*

Description

Prints out a representation of an wpstRO object

Usage

```
## S3 method for class 'wpstRO'
print(x, maxbasis = 10, ...)
```

Arguments

`x`	The wpstRO object to print
`maxbasis`	The maximum number of basis packets to report on
`...`	Other arguments

Details

Description says all

Value

None

Author(s)

G P Nason

See Also

`makewpstRO`

Examples

```
#
# See example in makewpstRO function
#
```

`print.wst` *Print out information about an wst object in readable form.*

Description

This function prints out information about an `wst.object` object in a nice human-readable form.

Usage

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

Arguments

`x`	The `wst.object` object to print info on
`...`	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

`wst`, `wst.object`

Examples

```
#
# Generate an wst object (a "nonsense" one for
# the example).
#
vwst <- wst(DJ.EX()$heavi)
#
# Now get Splus/R to use print.wst
#
vwst
#Class 'wst' : Stationary Wavelet Transform Object:
#       ~~~  : List with 5 components with names
#              wp Carray nlevelsWT filter date
#
#$wp and $Carray are the coefficient matrices
```

```
#
#Created on : Wed Sep 08 09:24:03 2004
#
#summary(.):
#----------
#Levels:  10
#Length of original:  1024
#Filter was:  Daub cmpct on least asymm N=10
#Date:  Wed Sep 08 09:24:03 2004
```

`print.wst2D` *Print out information about an wst2d object in a readable form.*

Description

This function prints out information about an `wst2D.object` in a nice human- readable form.

Note that this function is automatically called by SPlus whenever the name of an `wst2D.object` is typed or whenever such an object is returned to the top level of the S interpreter

Usage

```
## S3 method for class 'wst2D'
print(x, ...)
```

Arguments

`x`	An object of class `wst2D` that you wish to print out.
`...`	This argument actually does nothing in this function!

Details

Prints out information about `wst2D` objects in nice readable format.

Value

The last thing this function does is call `summary.wst2D` so the return value is whatever is returned by this function.

RELEASE

Version 3.9.6 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

`wst2D.object`, `summary.wst2D`.

Examples

```
#
# This examples uses the uawst2D object created in the EXAMPLES
# section of the help page for wst2D
#
#uawst2D
#Class 'wst2D' : 2D Stationary Wavelet Transform Object:
#       ~~~~~  : List with 4 components with names
#              wst2D nlevelsWT filter date
#
#$wst2D is the coefficient array
#
#Created on : Fri Nov  5 18:06:17 GMT 1999
#
#summary(.):
#----------
#Levels:  8
#Length of original:  256 x 256
#Filter was:  Daub cmpct on least asymm N=10
#Date:  Fri Nov  5 18:06:17 GMT 1999
```

PsiJ *Compute discrete autocorrelation wavelets.*

Description

This function computes discrete autocorrelation wavelets.

The inner products of the discrete autocorrelation wavelets are computed by the routine `ipndacw`.

Usage

```
PsiJ(J, filter.number = 10, family = "DaubLeAsymm", tol = 1e-100,
    OPLENGTH=10^7, verbose=FALSE)
```

Arguments

`J`	Discrete autocorrelation wavelets will be computed for scales -1 up to scale J. This number should be a negative integer.
`filter.number`	The index of the wavelet used to compute the discrete autocorrelation wavelets.
`family`	The family of wavelet used to compute the discrete autocorrelation wavelets.
`tol`	In the brute force computation for Daubechies compactly supported wavelets many inner product computations are performed. This tolerance discounts any results which are smaller than `tol` which effectively defines how long the inner product/autocorrelation products are.
`OPLENGTH`	This integer variable defines some workspace of length OPLENGTH. The code uses this workspace. If the workspace is not long enough then the routine will stop and probably tell you what OPLENGTH should be set to.
`verbose`	If `TRUE` then informative error messages are printed.

Details

This function computes the discrete autocorrelation wavelets. It does not have any direct use for time-scale analysis (e.g. ewspec). However, it is useful to be able to numerically compute the discrete autocorrelation wavelets for arbitrary wavelets and scales as there are still unanswered theoretical questions concerning the wavelets. The method is a brute force – a more elegant solution would probably be based on interpolatory schemes.

Horizontal scale. This routine returns only the values of the discrete autocorrelation wavelets and not their horiztonal positions. Each discrete autocorrelation wavelet is compactly supported with the support determined from the compactly supported wavelet that generates it. See the paper by Nason, von Sachs and Kroisandt which defines the horiztonal scale (but basically the finer scale discrete autocorrelation wavelets are interpolated versions of the coarser ones. When one goes from scale j to j-1 (negative j remember) an extra point is inserted between all of the old points and the discrete autocorrelation wavelet value is computed there. Thus as J tends to negative infinity the numerical approximation tends towards the continuous autocorrelation wavelet.

This function stores any discrete autocorrelation wavelet sets that it computes. The storage mechanism is not as advanced as that for ipndacw and its subsidiary routines rmget and firstdot but helps a little bit. The Psiname function defines the naming convention for objects returned by this function.

Sometimes it is useful to have the discrete autocorrelation wavelets stored in matrix form. The PsiJmat does this.

Note: intermediate calculations are stored in a user-visible environment called WTEnv. Previous versions of wavethresh stored this in the user's default data space (.GlobalEnv) but wavethresh did not ask permission nor notify the user. You can make these objects persist if you wish.

Value

A list containing -J components, numbered from 1 to -J. The [[j]]th component contains the discrete autocorrelation wavelet at scale j.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. echnical Report, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

See Also

ewspec, ipndacw, PsiJmat, Psiname.

Examples

```
#
# Let us create the discrete autocorrelation wavelets for the Haar wavelet.
# We shall create up to scale 4.
#
PsiJ(-4, filter.number=1, family="DaubExPhase")
#Computing PsiJ
#Returning precomputed version
#Took  0.00999999  seconds
#[[1]]:
#[1] -0.5  1.0 -0.5
#
#[[2]]:
#[1] -0.25 -0.50  0.25  1.00  0.25 -0.50 -0.25
#
#[[3]]:
# [1] -0.125 -0.250 -0.375 -0.500 -0.125  0.250  0.625  1.000  0.625  0.250
#[11] -0.125 -0.500 -0.375 -0.250 -0.125
#
#[[4]]:
# [1] -0.0625 -0.1250 -0.1875 -0.2500 -0.3125 -0.3750 -0.4375 -0.5000 -0.3125
#[10] -0.1250  0.0625  0.2500  0.4375  0.6250  0.8125  1.0000  0.8125  0.6250
#[19]  0.4375  0.2500  0.0625 -0.1250 -0.3125 -0.5000 -0.4375 -0.3750 -0.3125
#[28] -0.2500 -0.1875 -0.1250 -0.0625
#
# You can plot the fourth component to get an idea of what the
# autocorrelation wavelet looks like.
#
# Note that the previous call stores the autocorrelation wavelet
# in Psi.4.1.DaubExPhase. This is mainly so that it doesn't have to
# be recomputed.
#
# Note that the x-coordinates in the following are approximate.
#
## Not run: plot(seq(from=-1, to=1, length=length(Psi.4.1.DaubExPhase[[4]])),
 Psi.4.1.DaubExPhase[[4]], type="l",
 xlab = "t", ylab = "Haar Autocorrelation Wavelet")
## End(Not run)
#
#
# Now let us repeat the above for the Daubechies Least-Asymmetric wavelet
# with 10 vanishing moments.
# We shall create up to scale 6, a higher resolution version than last
# time.
#
p6 <- PsiJ(-6, filter.number=10, family="DaubLeAsymm", OPLENGTH=5000)
p6
##[[1]]:
# [1]  3.537571e-07  5.699601e-16 -7.512135e-06 -7.705013e-15  7.662378e-05
# [6]  5.637163e-14 -5.010016e-04 -2.419432e-13  2.368371e-03  9.976593e-13
#[11] -8.684028e-03 -1.945435e-12  2.605208e-02  6.245832e-12 -6.773542e-02
#[16]  4.704777e-12  1.693386e-01  2.011086e-10 -6.209080e-01  1.000000e+00
```

```
#[21] -6.209080e-01  2.011086e-10  1.693386e-01  4.704777e-12 -6.773542e-02
#[26]  6.245832e-12  2.605208e-02 -1.945435e-12 -8.684028e-03  9.976593e-13
#[31]  2.368371e-03 -2.419432e-13 -5.010016e-04  5.637163e-14  7.662378e-05
#[36] -7.705013e-15 -7.512135e-06  5.699601e-16  3.537571e-07
#
#[[2]]
# scale 2 etc. etc.
#
#[[3]]   scale 3 etc. etc.
#
#scales [[4]] and [[5]]...
#
#[[6]]
#...
#   remaining scale 6 elements...
#...
#[2371] -1.472225e-31 -1.176478e-31 -4.069848e-32 -2.932736e-41  6.855259e-33
#[2376]  5.540202e-33  2.286296e-33  1.164962e-42 -3.134088e-35  3.427783e-44
#[2381] -1.442993e-34 -2.480298e-44  5.325726e-35  9.346398e-45 -2.699644e-36
#[2386] -4.878634e-46 -4.489527e-36 -4.339365e-46  1.891864e-36  2.452556e-46
#[2391] -3.828924e-37 -4.268733e-47  4.161874e-38  3.157694e-48 -1.959885e-39
##
# Let's now plot the 6th component (6th scale, this is the finest
# resolution, all the other scales will be coarser representations)
#
#
# Note that the x-coordinates in the following are non-existant!
#
## Not run: ts.plot(p6[[6]], xlab = "t",
 ylab = "Daubechies N=10 least-asymmetric Autocorrelation Wavelet")
## End(Not run)
```

`PsiJmat` *Compute discrete autocorrelation wavelets but return result in matrix form.*

Description

This function computes discrete autocorrelation wavelets using the `PsiJ` function but it returns the results as a matrix rather than a list object.

Usage

```
PsiJmat(J, filter.number = 10, family = "DaubLeAsymm", OPLENGTH=10^7)
```

Arguments

`J`	Discrete autocorrelation wavelets will be computed for scales -1 up to scale J. This number should be a negative integer.

`filter.number`	The index of the wavelet used to compute the discrete autocorrelation wavelets.
`family`	The family of wavelet used to compute the discrete autocorrelation wavelets.
`OPLENGTH`	This integer variable defines some workspace of length OPLENGTH. The code uses this workspace. If the workspace is not long enough then the routine will stop and probably tell you what OPLENGTH should be set to.

Details

The discrete autocorrelation wavelet values are computed using the `PsiJ` function. This function merely organises them into a matrix form.

Value

A matrix containing -J rows and a number of columns less than OPLENGTH. Each row contains the values of the discrete autocorrelation wavelet for a different scale. Row one contains the scale -1 coefficients, row two contains the scale -2, and so on.

The number of columns is an odd number. The middle position of each row is the value of the discrete autocorrelation wavelet at zero — this is always 1. The discrete autocorrelation wavelet is symmetric about this point.

Important Apart from the central element none of the other columns line up in this way. This could be improved upon.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

See Also

`PsiJ`

Examples

```
#
# As a simple first examples we shall compute the matrix containing
# the discrete autocorrelation wavelets up to scale 3.
#
PsiJmat(-3, filter.number=1, family="DaubExPhase")
#Computing PsiJ
#Took  0.25  seconds
```

```
#       [,1]  [,2]   [,3] [,4]   [,5]  [,6]   [,7] [,8]   [,9] [,10]  [,11]
#[1,]  0.000  0.00  0.000  0.0  0.000  0.00 -0.500    1 -0.500  0.00  0.000
#[2,]  0.000  0.00  0.000  0.0 -0.250 -0.50  0.250    1  0.250 -0.50 -0.250
#[3,] -0.125 -0.25 -0.375 -0.5 -0.125  0.25  0.625    1  0.625  0.25 -0.125
#     [,12]  [,13] [,14]  [,15]
#[1,]   0.0  0.000  0.00  0.000
#[2,]   0.0  0.000  0.00  0.000
#[3,]  -0.5 -0.375 -0.25 -0.125
#
# Note that this contains 3 rows (since J=-3).
# Each row contains the same discrete autocorrelation wavelet at different
# scales and hence different resolutions.
# Compare to the output given by PsiJ for the
# equivalent wavelet and scales.
# Note also that apart from column 8 which contains 1 (the value of the
# ac wavelet at zero) none of the other columns line up. E.g. the value of
# this wavelet at 1/2 is -0.5: this appears in columns 9, 10 and 12
# we could have written it differently so that they should line up.
# I might do this in the future.
#
#
# Let's compute the matrix containing the discrete autocorrelation
# wavelets up to scale 6 using Daubechies N=10 least-asymmetric
# wavelets.
#
P6mat <- PsiJmat(-6, filter.number=10, family="DaubLeAsymm")
#
# What is the dimension of this matrix?
#
dim(P6mat)
#[1]    6 2395
#
# Hmmm. Pretty large, so we shan't print it out.
#
# However, these are the ac wavelets... Therefore if we compute their
# inner product we should get the same as if we used the ipndacw
# function directly.
#
P6mat
#             [,1]         [,2]         [,3]         [,4]         [,5]
#[1,] 1.839101e+00 3.215934e-01 4.058155e-04 8.460063e-06 4.522125e-08
#[2,] 3.215934e-01 3.035353e+00 6.425188e-01 7.947454e-04 1.683209e-05
#[3,] 4.058155e-04 6.425188e-01 6.070419e+00 1.285038e+00 1.589486e-03
#[4,] 8.460063e-06 7.947454e-04 1.285038e+00 1.214084e+01 2.570075e+00
#[5,] 4.522125e-08 1.683209e-05 1.589486e-03 2.570075e+00 2.428168e+01
#[6,] 5.161675e-10 8.941666e-08 3.366416e-05 3.178972e-03 5.140150e+00
#             [,6]
#[1,] 5.161675e-10
#[2,] 8.941666e-08
#[3,] 3.366416e-05
#[4,] 3.178972e-03
#[5,] 5.140150e+00
#[6,] 4.856335e+01
```

```
#
# Let's check it against the ipndacw call
#
ipndacw(-6, filter.number=10, family="DaubLeAsymm")
#             -1           -2           -3           -4           -5
#-1 1.839101e+00 3.215934e-01 4.058155e-04 8.460063e-06 4.522125e-08
#-2 3.215934e-01 3.035353e+00 6.425188e-01 7.947454e-04 1.683209e-05
#-3 4.058155e-04 6.425188e-01 6.070419e+00 1.285038e+00 1.589486e-03
#-4 8.460063e-06 7.947454e-04 1.285038e+00 1.214084e+01 2.570075e+00
#-5 4.522125e-08 1.683209e-05 1.589486e-03 2.570075e+00 2.428168e+01
#-6 5.161675e-10 8.941666e-08 3.366416e-05 3.178972e-03 5.140150e+00
#             -6
#-1 5.161675e-10
#-2 8.941666e-08
#-3 3.366416e-05
#-4 3.178972e-03
#-5 5.140150e+00
#-6 4.856335e+01
#
# Yep, they're the same.
#
```

`Psiname` *Return a PsiJ list object style name.*

Description

This function returns a character string according to a particular format for naming `PsiJ` objects.

Usage

```
Psiname(J, filter.number, family)
```

Arguments

`J`	A negative integer representing the order of the `PsiJ` object.
`filter.number`	The index number of the wavelet used to build the `PsiJ` object.
`family`	The wavelet family used to build the `PsiJ` object.

Details

Some of the objects computed by `PsiJ` take a long time to compute. Hence it is a good idea to store them and reuse them. This function generates a name according to a particular naming scheme that permits a search algorithm to easily find the matrices.

Each object has three defining characteristics: its *order*, *filter.number* and *family*. Each of these three characteristics are concatenated together to form a name.

This function performs exactly the same role as `rmname` except for objects produced by `PsiJ`.

Value

A character string containing the name of an object according to a particular naming scheme.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern

See Also

PsiJ

Examples

```
#
# What's the name of the order 4 Haar PsiJ object?
#
Psiname(-4, filter.number=1, family="DaubExPhase")
#[1] "Psi.4.1.DaubExPhase"
#
# What's the name of the order 12 Daubechies least-asymmetric wavelet PsiJ
# with 7 vanishing moments?
#
Psiname(-12, filter.number=7, family="DaubLeAsymm")
#[1] "Psi.12.7.DaubLeAsymm"
```

---

putC — *Put smoothed data (father wavelet) coefficients into wavelet structure*

Description

This generic function inserts smooths into various types of wavelet objects.

This function is generic.

Particular methods exist. For objects of class:

wd use the putC.wd method.

wp use the putC.wp method.

wst use the putC.wst method.

See individual method help pages for operation and examples.

See accessC if you wish to *extract* father wavelet coefficients. See putD if you wish to insert *mother* wavelet coefficients

Usage

```
putC(...)
```

Arguments

...	See individual help pages for details.

Value

A wavelet object of the same class as x with the new father wavelet coefficients inserted.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

putC.wd, putC.wp, putC.wst, accessC, putD.

---

putC.mwd *Put smoothed data into wavelet structure*

Description

The smoothed and original data from a multiple wavelet decomposition structure, mwd.object, (e.g. returned from mwd) are packed into a single matrix in that structure. This function copies the mwd.object, replaces some smoothed data in the copy, and then returns the copy.

Usage

```
## S3 method for class 'mwd'
putC(mwd, level, M, boundary = FALSE, index = FALSE, ...)
```

Arguments

mwd	Multiple wavelet decomposition structure whose coefficients you wish to replace.
level	The level that you wish to replace.
M	Matrix of replacement coefficients.
boundary	If boundary is FALSE then only the "real" data is replaced (and it is easy to predict the required length of M). If boundary is TRUE then you can replace the boundary values at a particular level as well (but it is hard to predict the required length of M, and the information has to be obtained from the mfirst.last database component of mwd).
index	If index is TRUE then the index numbers into the mwd$C array where the matrix M would be stored is returned. Otherwise, (default) the modified mwd.object is returned.
...	any other arguments

Details

The mwd function produces a wavelet decomposition structure.

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as a linear matrix of coefficients. PutC obtains information about where the smoothed data appears from the fl.dbase component of mwd, in particular the array fl.dbase$first.last.c which gives a complete specification of index numbers and offsets for mwd$C.

Note also that this function only *puts* information into mwd class objects. To *extract* coefficients from mwd structures you have to use the accessC.mwd function.

See Downie and Silverman, 1998.

Value

An object of class mwd.object if index is FALSE, otherwise the index numbers indicating where the M matrix would have been inserted into the mwd$C object are returned.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6).

Author(s)

G P Nason

See Also

accessC.mwd, accessD.mwd, draw.mwd, mfirst.last, mfilter.select, mwd, mwd.object, mwr, plot.mwd, print.mwd, putD.mwd, summary.mwd, threshold.mwd, wd, wr.mwd.

Examples

```
#
# Generate an mwd object
#
tmp <- mwd(rnorm(32))
#
# Now let's examine the finest resolution smooth...
#
accessC(tmp, level=3)
#           [,1]       [,2]       [,3]       [,4]        [,5]      [,6]
#[1,] -0.4669103 -1.3150580 -0.7094966 -0.1979214  0.32079986 0.5052254
#[2,] -0.7645379 -0.8680941  0.1004062  0.6633268 -0.05860848 0.5757286
#          [,7]       [,8]
#[1,] 0.5187380  0.6533843
#[2,] 0.2864293 -0.4433788
#
# A matrix. There are two rows one for each father wavelet in this
# two-ple multiple wavelet transform and at level 3 there are 2^3 columns.
#
# Let's set the coefficients of the first father wavelet all equal to zero
# for this examples
#
newcmat <- accessC(tmp, level=3)
newcmat[1,] <- 0
#
# Ok, let's insert it back at level 3
#
tmp2 <- putC(tmp, level=3, M=newcmat)
#
# And check it
#
accessC(tmp2, level=3)
#           [,1]       [,2]      [,3]      [,4]        [,5]      [,6]      [,7]
#[1,]  0.0000000  0.0000000 0.0000000 0.0000000  0.00000000 0.0000000 0.0000000
#[2,] -0.7645379 -0.8680941 0.1004062 0.6633268 -0.05860848 0.5757286 0.2864293
#           [,8]
#[1,]  0.0000000
#[2,] -0.4433788
#
# Yep, all the first father wavelet coefficients at level 3 are now zero.
```

---

putC.wd *Puts a whole resolution level of father wavelet coeffients into wd wavelet object.*

Description

Makes a copy of the wd object, replaces some father wavelet coefficients data in the copy, and then returns the copy.

Usage

```
## S3 method for class 'wd'
putC(wd, level, v, boundary=FALSE, index=FALSE, ...)
```

Arguments

wd	Wavelet decomposition object into which you wish to insert the father wavelet coefficients.
level	the resolution level at which you wish to replace the father wavelet coefficients.
v	the replacement data, this should be of the correct length.
boundary	If boundary is FALSE then only "real" data is replaced. If boundary is TRUE then the boundary (bookeeping) elements are replaced as well. Information about the lengths of the vectors can be found in the first.last database function and Nason and Silverman, 1994.
index	If index is TRUE then the index numbers into the 1D array where the coefficient insertion would take place are returned. If index is FALSE (default) then the modified wavelet decomposition object is returned.
...	any other arguments

Details

The function accessC obtains the father wavelet coefficients for a particular level. The function putC.wd replaces father wavelet coefficients at a particular resolution level and returns a modified wd object reflecting the change.

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as a linear vector. PutC.wd obtains information about where the smoothed data appears from the fl.dbase component of an wd.object, in particular the array

fl.dbase$first.last.c

which gives a complete specification of index numbers and offsets for

wd.object$C.

Note that this function is method for the generic function putC. When the wd.object is definitely a wd class object then you only need use the generic version of this function.

Note also that this function only puts information into wd class objects. To extract coefficients from a wd object you have to use the accessC function (or more precisely, the accessC.wd method).

Value

A wd class object containing the modified father wavelet coefficients.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

putC, wd.object, wd, accessC,putD, first.last,

Examples

```
#
# Generate an EMPTY wd object:
#
zero <- rep(0, 16)
zerowd <- wd(zero)
#
# Put some random father wavelet coefficients into the object at
# resolution level 2. For the decimated wavelet transform there
# are always 2^i coefficients at resolution level i. So we have to
# insert 4 coefficients
#
mod.zerowd <- putC( zerowd, level=2, v=rnorm(4))
#
# If you use accessC on mod.zerowd you would see that there were only
# coefficients at resolution level 2 where you just put the coefficients.
#
# Now, for a time-ordered non-decimated wavelet transform object the
# procedure is exactly the same EXCEPT that there are going to be
# 16 coefficients at each resolution level. I.e.
#
# Create empty TIME-ORDERED NON-DECIMATED wavelet transform object
#
zerowdS <- wd(zero, type="station")
#
# Now insert 16 random coefficients at resolution level 2
##
mod.zerowdS <- putC(zerowdS, level=2, v=rnorm(16))
#
# Once more if you use accessC on mod.zerowdS you will see that there are
# only coefficients at resolution level 2.
```

---

putC.wp — *Warning function when trying to insert father wavelet coefficients into wavelet packet object (wp).*

Description

There are no real smooths to insert in a wp wavelet packet object. This function returns an error message. To insert coefficients into a wavelet packet object you should use the putpacket collection of functions.

Usage

```
## S3 method for class 'wp'
putC(wp, ...)
```

Arguments

wp	Wavelet packet object.
...	any other arguments

Value

An error message!

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

putpacket, putpacket.wp.

---

putC.wst — *Puts a whole resolution level of father wavelet coeffients into wst wavelet object.*

Description

Makes a copy of the wst object, replaces a whole resolution level of father wavelet coefficients data in the copy, and then returns the copy.

Usage

```
## S3 method for class 'wst'
putC(wst, level, value, ...)
```

Arguments

wst	Packet-ordered non-decimated wavelet object into which you wish to insert the father wavelet coefficients.
level	the resolution level at which you wish to replace the father wavelet coefficients.
value	the replacement data, this should be of the correct length.
...	any other arguments

Details

The function accessC.wst obtains the father wavelet coefficients for a particular level. The function putC.wst replaces father wavelet coefficients at a particular resolution level and returns a modified wst object reflecting the change.

For the non-decimated wavelet transforms the number of coefficients at each resolution level is the same and equal to 2^nlevelsWT where nlevels is the number of levels in the wst.object. The number of coefficients at each resolution level is also, of course, the number of data points used to initially form the wst object in the first place.

Use the accessC.wst to extract whole resolution levels of father wavelet coefficients. Use accessD.wst and putD.wst to extract/insert whole resolution levels of mother wavelet coefficients. Use the getpacket.wst and putpacket.wst functions to extract/insert packets of coefficients into a packet-ordered non-decimated wavelet object.

Value

A wst class object containing the modified father wavelet coefficients

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wst.object, wst, putC, accessD.wst, putD.wst, getpacket.wst, putpacket.wst.

Examples

```
#
# Generate an EMPTY wst object:
#
zero <- rep(0, 16)
zerowst <- wst(zero)
#
# Put some random father wavelet coefficients into the object at
# resolution level 2. For the non-decimated wavelet transform there
# are always 16 coefficients at every resolution level.
#
mod.zerowst <- putC( zerowst, level=2, v=rnorm(16))
#
# If you use accessC  on mod.zerowd you would see that there were only
# coefficients at resolution level 2 where you just put the coefficients.
```

putD *Put mother wavelet coefficients into wavelet structure*

Description

This generic function inserts smooths into various types of wavelet objects.

This function is generic.

Particular methods exist. For objects of class:

wd use the putD.wd method.

wp use the putD.wp method.

wst use the putD.wst method.

See individual method help pages for operation and examples.

See accessD if you wish to *extract* mother wavelet coefficients. See putC if you wish to insert *father* wavelet coefficients.

Usage

```
putD(...)
```

Arguments

...	See individual help pages for details.

Value

A wavelet object of the same class as x with the new mother wavelet coefficients inserted.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

putD.wd, putD.wp, putD.wst, accessD, putC.

putD.mwd *Put wavelet coefficients into multiple wavelet structure*

Description

The wavelet coefficients from a multiple wavelet decomposition structure, mwd.object, (e.g. returned from mwd) are packed into a single matrix in that structure. This function copies the mwd.object, replaces some wavelet coefficients in the copy, and then returns the copy.

Usage

```
## S3 method for class 'mwd'
putD(mwd, level, M, boundary = FALSE, index = FALSE, ...)
```

Arguments

mwd	Multiple wavelet decomposition structure whose coefficients you wish to replace.
level	The level that you wish to replace.
M	Matrix of replacement coefficients.
boundary	If boundary is FALSE then only the "real" data is replaced (and it is easy to predict the required length of M). If boundary is TRUE then you can replace the boundary values at a particular level as well (but it is hard to predict the required length ofM, and the information has to be obtained from the mfirst.last database component of mwd).
index	If index is TRUE then the index numbers into the mwd$D array where the matrix M would be stored is returned. Otherwise, (default) the modified mwd.object is returned.
...	any other arguments

Details

The mwd function produces a wavelet decomposition structure.

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as a linear matrix of coefficients. PutD obtains information about where the wavelet coefficients appear from the fl.dbase component of mwd, in particular the array fl.dbase$first.last.d which gives a complete specification of index numbers and offsets for mwd$D.

Note also that this function only puts information into mwd class objects. To extract coefficients from mwd structures you have to use the accessD.mwd function.

See Downie and Silverman, 1998.

Value

An object of class mwd.object if index is FALSE, otherwise the index numbers indicating where the M matrix would have been inserted into the mwd$D object are returned.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6).

Author(s)

Tim Downie

See Also

accessC.mwd, accessD.mwd, draw.mwd, mfirst.last, mfilter.select, mwd, mwd.object, mwr, plot.mwd, print.mwd, putC.mwd, summary.mwd, threshold.mwd, wd, wr.mwd.

Examples

```
#
# Generate an mwd object
#
tmp <- mwd(rnorm(32))
#
# Now let's examine the finest resolution detail...
#
accessD(tmp, level=3)
#          [,1]       [,2]       [,3]      [,4]       [,5]        [,6]
#[1,] 0.8465672  0.4983564  0.3408087 0.1340325  0.5917774 -0.06804291
#[2,] 0.6699962 -0.2535760 -1.0344445 0.2068644 -0.4912086  1.16039885
#           [,7]       [,8]
#[1,] -0.6226445  0.2617596
#[2,] -0.4956576 -0.5555795
#
#
# A matrix. There are two rows one for each mother wavelet in this
# two-ple multiple wavelet transform and at level 3 there are 2^3 columns.
#
# Let's set the coefficients of the first mother wavelet all equal to zero
# for this examples
#
newdmat <- accessD(tmp, level=3)
newdmat[1,] <- 0
#
# Ok, let's insert it back at level 3
#
tmp2 <- putD(tmp, level=3, M=newdmat)
#
# And check it
#
accessD(tmp2, level=3)
#          [,1]      [,2]      [,3]      [,4]       [,5]     [,6]       [,7]
#[1,] 0.0000000  0.000000  0.000000 0.0000000  0.0000000 0.000000  0.0000000
#[2,] 0.6699962 -0.253576 -1.034445 0.2068644 -0.4912086 1.160399 -0.4956576
#           [,8]
#[1,]  0.0000000
#[2,] -0.5555795
```

```
#
#
# Yep, all the first mother wavelet coefficients at level 3 are now zero.
```

---

putD.wd *Puts a whole resolution level of mother wavelet coeffients into wd wavelet object.*

Description

Makes a copy of the wd object, replaces some mother wavelet coefficients data in the copy, and then returns the copy.

Usage

```
## S3 method for class 'wd'
putD(wd, level, v, boundary=FALSE, index=FALSE, ...)
```

Arguments

wd	Wavelet decomposition object into which you wish to insert the mother wavelet coefficients.
level	the resolution level at which you wish to replace the mother wavelet coefficients.
v	the replacement data, this should be of the correct length.
boundary	If boundary is FALSE then only "real" data is replaced. If boundary is TRUE then the boundary (bookeeping) elements are replaced as well. Information about the lengths of the vectors can be found in the first.last database function and Nason and Silverman, 1994.
index	If index is TRUE then the index numbers into the 1D array where the coefficient insertion would take place are returned. If index is FALSE (default) then the modified wavelet decomposition object is returned.
...	any other arguments

Details

The function accessD obtains the mother wavelet coefficients for a particular level. The function putD.wd replaces father wavelet coefficients at a particular resolution level and returns a modified wd object reflecting the change.

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as a linear vector. PutD.wd obtains information about where the smoothed data appears from the fl.dbase component of an wd.object, in particular the array

fl.dbase$first.last.d

which gives a complete specification of index numbers and offsets for

wd.object$D.

Note that this function is method for the generic function putD. When the wd.object is definitely a wd class object then you only need use the generic version of this function.

Note also that this function only puts information into wd class objects. To extract coefficients from a wd object you have to use the accessD function (or more precisely, the accessD.wd method).

Value

A wd class object containing the modified mother wavelet coefficients.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

putD, wd.object, wd, accessD,putD, first.last,

Examples

```
#
# Generate an EMPTY wd object:
#
zero <- rep(0, 16)
zerowd <- wd(zero)
#
# Put some random father wavelet coefficients into the object at
# resolution level 2. For the decimated wavelet transform there
# are always 2^i coefficients at resolution level i. So we have to
# insert 4 coefficients
#
mod.zerowd <- putD( zerowd, level=2, v=rnorm(4))
#
# If you plot mod.zerowd you will see that there are only
# coefficients at resolution level 2 where you just put the coefficients.
#
# Now, for a time-ordered non-decimated wavelet transform object the
# procedure is exactly the same EXCEPT that there are going to be
# 16 coefficients at each resolution level. I.e.
#
# Create empty TIME-ORDERED NON-DECIMATED wavelet transform object
#
zerowdS <- wd(zero, type="station")
#
# Now insert 16 random coefficients at resolution level 2
#
mod.zerowdS <- putD(zerowdS, level=2, v=rnorm(16))
#
# Once more if you plot mod.zerowdS then there will only be
```

```
# coefficients at resolution level 2.
```

---

putD.wd3D *Put wavelet coefficient array into a 3D wavelet object*

Description

This function put an array of wavelet coefficients, corresponding to a particular resolution level into a wd wavelet decomposition object.

The pyramid of coefficients in a wavelet decomposition (returned from the wd3D function, say) are packed into a single array in WaveThresh3.

Usage

```
## S3 method for class 'wd3D'
putD(x, v, ...)
```

Arguments

x	3D Wavelet decomposition object into which you wish to insert the wavelet coefficients.
v	This argument is a list with the following components: **a** A 3-dimensional array with each dimension of length equal to two to the power of lev which is the level at which you wish to insert the coefficients into x. **lev** The level at which you wish to insert the coefficients into x. **block** A character string indicating which coefficient block you wish to insert the coefficients into. This can be one of GGG, GGH, GHG, GHH, HGG, HGH, HHG. Additionally this can be HHH when the lev argument above is zero.
...	Other arguments

Details

The need for this function is a consequence of the pyramidal structure of Mallat's algorithm and the memory efficiency gain achieved by storing the pyramid as an array.

Note that this function is a method for the generic function putD.

Value

A new wd3D.object is returned with the coefficients at level lev in block given by block are replaced by the contents of a, if a is of the correct dimensions!

RELEASE

Version 3.9.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

accessD, accessD.wd3D, print.wd3D, putD, putDwd3Dcheck, summary.wd3D, threshold.wd3D, wd3D, wd3D.object, wr3D.

Examples

```
#
# Generate some test data
#
a <- array(rnorm(8*8*8), dim=c(8,8,8))
#
# Perform the 3D DWT
#
awd3D <- wd3D(a)
#
# Replace the second level coefficients by uniform random variables
# in block GGG (for some reason)
#
#
newsubarray <- list(a = array(runif(4*4*4), dim=c(4,4,4)), lev=2, block="GGG")
awd3D <- putD(awd3D,  v=newsubarray)
```

---

putD.wp — *Puts a whole resolution level of wavelet packet coeffients into wp wavelet object.*

Description

Makes a copy of the wp object, replaces a whole resolution level of wavelet packet coefficients data in the copy, and then returns the copy.

Usage

```
## S3 method for class 'wp'
putD(wp, level, value, ...)
```

Arguments

wp	Wavelet packet object into which you wish to insert the wavelet packet coefficients.
level	the resolution level at which you wish to replace the wavelet packet coefficients.
value	the replacement data, this should be of the correct length.
...	any other arguments

Details

The function accessD.wp obtains the wavelet packet coefficients for a particular level.

For wavelet packet transforms the number of coefficients at each resolution level is the same and equal to 2^nlevelsWT where nlevels is the number of levels in the wp.object. The number of coefficients at each resolution level is also, of course, the number of data points used to initially form the wp object in the first place.

Use the accessD.wp to extract whole resolution levels of wavelet packet coefficients.

We don't recommend that you use this function unless you really know what you are doing. Usually it is more convenient to manipulate individual *packets* of coefficients using getpacket/putpacket functions. If you must use this function to insert whole resolution levels of coefficients you must ensure that the data vector you supply is valid: i.e. contains packet coefficients in the right order.

Value

A wp class object containing the modified wavelet packet coefficients.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wp.object, wp, accessD, accessD.wp, getpacket.wp, putpacket.wp.

Examples

```
#
# Generate an EMPTY wp object:
#
zero <- rep(0, 16)
zerowp <- wp(zero)
#
# Put some random mother wavelet coefficients into the object at
# resolution level 2. For the wavelet packet transform there
# are always 16 coefficients at every resolution level.
#
mod.zerowp <- putD( zerowp, level=2, v=rnorm(16))
#
# If you plot mod.zerowp you will see that there are only
# coefficients at resolution level 2 where you just put the coefficients.
```

putD.wst *Puts a whole resolution level of mother wavelet coeffients into wst wavelet object.*

Description

Makes a copy of the wst object, replaces a whole resolution level of mother wavelet coefficients data in the copy, and then returns the copy.

Usage

```
## S3 method for class 'wst'
putD(wst, level, value, ...)
```

Arguments

wst	Packet-ordered non-decimated wavelet object into which you wish to insert the mother wavelet coefficients.
level	the resolution level at which you wish to replace the mother wavelet coefficients.
value	the replacement data, this should be of the correct length
...	any other arguments

Details

The function accessD.wst obtains the mother wavelet coefficients for a particular level. The function putD.wst replaces mother wavelet coefficients at a particular resolution level and returns a modified wst object reflecting the change.

For the non-decimated wavelet transforms the number of coefficients at each resolution level is the same and equal to 2^nlevelsWT where nlevels is the number of levels in the wst.object. The number of coefficients at each resolution level is also, of course, the number of data points used to initially form the wst object in the first place.

Use the accessD.wst to extract whole resolution levels of mother wavelet coefficients. Use accessC.wst and putC.wst to extract/insert whole resolution levels of father wavelet coefficients. Use the getpacket.wst and putpacket.wst functions to extract/insert packets of coefficients into a packet-ordered non-decimated wavelet object.

Value

A wst class object containing the modified mother wavelet coefficients.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wst.object, wst, putD, accessD.wst, putC.wst, getpacket.wst, putpacket.wst.

Examples

```
#
# Generate an EMPTY wst object:
#
zero <- rep(0, 16)
zerowst <- wst(zero)
#
# Put some random mother wavelet coefficients into the object at
# resolution level 2. For the non-decimated wavelet transform there
# are always 16 coefficients at every resolution level.
#
mod.zerowst <- putD( zerowst, level=2, v=rnorm(16))
#
# If you plot mod.zerowst you will see that there are only
# coefficients at resolution level 2 where you just put the coefficients.
```

---

putDwd3Dcheck *Check argument list for putD.wd3D*

Description

This function checks the argument list for putD.wd3D and is not meant to be directly called by any user.

Usage

```
putDwd3Dcheck(lti, dima, block, nlx)
```

Arguments

lti	At which level of the wd3D.object you wish to insert a block of coefficients.
dima	A vector, of length 3, which specifies the dimension of the block to insert.
block	A character string which specifies which block is being inserted (one of GGG, GGH, GHG, GHH, HGG, HGH, HHG, or HHH).
nlx	The number of levels in the wd3D.object that you wish to insert the coefficients into (can be obtained using the nlevelsWT function).

Details

This function merely checks that the dimensions and sizes of the array to be inserted into a wd3D.object using the putD.wd3D function are correct.

RELEASE

Version 3.9.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

accessD, putD, accessD.wd3D, print.wd3D, putD, summary.wd3D, threshold.wd3D, wd3D, wd3D.object, wr3D.

Examples

```
#
# Not intended to be used by the user!
#
```

---

putpacket *Insert a packet of coefficients into a wavelet object.*

Description

This generic function inserts packets of coefficients into various types of wavelet objects.

This function is generic.

Particular methods exist. For objects of class:

wp use the putpacket.wp method.

wst use the putpacket.wst method.

wst2D use the putpacket.wst2D method.

See individual method help pages for operation and examples.

Use the putC and putD function to insert whole resolution levels of coefficients simultaneously.

Usage

```
putpacket(...)
```

Arguments

...	See individual help pages for details.

Value

A wavelet object of the same class as x the input object. The returned wavelet object is the same as the input except that the appropriate packet of coefficients supplied is replaced.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

putpacket.wp, putpacket.wst, putpacket.wst2D, putD, putC, wp.object, wst.object, wst2D.object.

---

putpacket.wp *Inserts a packet of coefficients into a wavelet packet object (wp).*

Description

This function inserts a packet of coefficients into a wavelet packet (wp) object.

Usage

```
## S3 method for class 'wp'
putpacket(wp, level, index, packet , ...)
```

Arguments

wp	Wavelet packet object into which you wish to put the packet.
level	The resolution level of the coefficients that you wish to insert.
index	The index number within the resolution level of the packet of coefficients that you wish to insert.
packet	a vector of coefficients which is the packet you wish to insert.
...	any other arguments

Details

The coefficients in this structure can be organised into a binary tree with each node in the tree containing a packet of coefficients.

Each packet of coefficients is obtained by chaining together the effect of the two *packet operators* DG and DH: these are the high and low pass quadrature mirror filters of the Mallat pyramid algorithm scheme followed by decimation (see Mallat (1989b)).

Starting with data c^J at resolution level J containing 2^J data points the wavelet packet algorithm operates as follows. First DG and DH are applied to c^J producing d^{J-1} and c^{J-1} respectively. Each of these sets of coefficients is of length one half of the original data: i.e. 2^{J-1}. Each of these sets of coefficients is a set of *wavelet packet coefficients*. The algorithm then applies both DG and DH to both d^{J-1} and c^{J-1} to form a four sets of coefficients at level J-2. Both operators are used again on the four sets to produce 8 sets, then again on the 8 sets to form 16 sets and so on. At level j=J,...,0 there are 2^{J-j} packets of coefficients each containing 2^j coefficients.

This function enables whole packets of coefficients to be inserted at any resolution level. The `index` argument chooses a particular packet within each level and thus ranges from 0 (which always refer to the father wavelet coefficients), 1 (which always refer to the mother wavelet coefficients) up to 2^{J-j}.

Value

An object of class `wp.object` which is the same as the input `wp.object` except it now has a modified packet of coefficients.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

`wp`, `getpacket.wp`. `putpacket`.

Examples

```
#
# Take the wavelet packet transform of some random data
#
MyWP <- wp(rnorm(1:512))
#
# The above data set was 2^9 in length. Therefore there are
# coefficients at resolution levels 0, 1, 2, ..., and 8.
#
# The high resolution coefficients are at level 8.
# There should be 256 DG coefficients and 256 DH coefficients
#
length(getpacket(MyWP, level=8, index=0))
# [1] 256
length(getpacket(MyWP, level=8, index=1))
# [1] 256
#
# The next command shows that there are only two packets at level 8
#
#getpacket(MyWP, level=8, index=2)
# Index was too high, maximum for this level is  1
# Error in getpacket.wp(MyWP, level = 8, index = 2): Error occured
# Dumped
#
# There should be 4 coefficients at resolution level 2
#
# The father wavelet coefficients are (index=0)
getpacket(MyWP, level=2, index=0)
# [1] -0.9736576  0.5579501  0.3100629 -0.3834068
```

```
#
# The mother wavelet coefficients are (index=1)
#
getpacket(MyWP, level=2, index=1)
# [1]  0.72871405  0.04356728 -0.43175307  1.77291483
#
# Well, that exercised the getpacket.wp
# function. Now that we know that level 2 coefficients have 4 coefficients
# let's insert some into the MyWP object.
#
MyWP <- putpacket(MyWP, level=2, index=0, packet=c(21,32,67,89))
#
# O.k. that was painless. Now let's check that the correct coefficients
# were inserted.
#
getpacket(MyWP, level=2, index=0)
#[1] 21 32 67 89
#
# Yep. The correct coefficients were inserted.
```

`putpacket.wst` *Put a packet of coefficients into a packet ordered non-decimated wavelet object (wst).*

Description

This function inserts a packet of coefficients into a packet-ordered non-decimated wavelet object (`wst`) object. The `wst` objects are computed by the `wst` function amongst others.

Usage

```
## S3 method for class 'wst'
putpacket(wst, level, index, packet, ...)
```

Arguments

`wst`	Packet-ordered non-decimated wavelet object into which you wish to insert the packet.
`level`	The resolution level of the coefficients that you wish to insert.
`index`	The index number within the resolution level of the packet of coefficients that you wish to insert.
`packet`	A vector of coefficients that you wish to insert into the `wst` object. The length that the packet has to may be determined by extracting the same packet of coefficients using the `getpacket.wst` function and using the S-Plus length function to determine the length!
`...`	any other arguments

Details

This function actually calls the putpacket.wp to do the insertion.

In the future this function will be extended to insert father wavelet coefficients as well.

Value

An object of class wst.object containing the packet ordered non-decimated wavelet coefficients that have been modified: i.e. with packet inserted.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

getpacket.wst, putpacket, putpacket.wp, wst, wst.object.

Examples

```
#
# Take the packet-ordered non-decimated transform of some random data
#
MyWST <- wst(rnorm(1:512))
#
# The above data set was 2^9 in length. Therefore there are
# coefficients at resolution levels 0, 1, 2, ..., and 8.
#
# The high resolution coefficients are at level 8.
# There should be 256 coefficients at level 8 in index location 0 and 1.
#
length(getpacket(MyWST, level=8, index=0))
# [1] 256
length(getpacket(MyWST, level=8, index=1))
# [1] 256
#
# There should be 4 coefficients at resolution level 2
#
getpacket(MyWST, level=2, index=0)
# [1] -0.92103095  0.70125471  0.07361174 -0.43467375
#
# O.k. Let's insert the packet containing the numbers 19,42,21,32
#
NewMyWST <- putpacket(MyWST, level=2, index=0, packet=c(19,42,31,32))
#
# Let's check that it put the numbers in correctly by reaccessing that
# packet...
#
```

```
getpacket(NewMyWST, level=2, index=0)
# [1] 19 42 31 32
#
# Yep. It inserted the packet correctly.
```

---

putpacket.wst2D *Replace packet of coefficients in a two-dimensional non-decimated wavelet object (wst2D).*

Description

This function replaces a packet of coefficients from a two-dimensional non-decimated wavelet (wst2D) object and returns the modified object.

Usage

```
## S3 method for class 'wst2D'
putpacket(wst2D, level, index, type="S", packet, Ccode=TRUE, ...)
```

Arguments

wst2D	2D non-decimated wavelet object containing the coefficients you wish to replace.
level	The resolution level of the coefficients that you wish to replace. Can range from 0 to nlevelsWT(wpst)-1.
index	The index number within the resolution level of the packet of coefficients that you wish to replace. Index is a base-4 number which is r digits long. Each digit can be 0, 1, 2 or 3 corresponding to no shifts, horizontal shift, vertical shift or horizontal and vertical shifts. The number r indicates the depth of the resolution level from the data resolution i.e. where r = nlevelsWT - level. Where there is a string of more than one digit the left most digits correspond to finest scale shift selection, the right most digits to the coarser scales (I think).
packet	A square matrix of dimension 2^level which contains the new coefficients that you wish to insert.
type	This is a one letter character string: one of "S", "H", "V" or "D" for the smooth coefficients, horizontal, vertical or diagonal detail.
Ccode	If T then fast C code is used to obtain the packet, otherwise slow SPlus code is used. Unless you have some special reason always use the C code (and leave the argument at its default).
...	any other arguments

Details

The wst2D function creates a wst2D class object. Starting with a smooth the operators H, G, GS and HS (where G, H are the usual Mallat operators and S is the shift-by-one operator) are operated first on the rows and then the columns: i.e. so each of the operators HH, HG, GH, GG, HSH, HSG, GSH, GSG HHS, GHS, HGS, GGS HSHS, HSGS, GSHS and GSGS are applied. Then the same collection of operators is applied to all the derived smooths, i.e. HH, HSH, HHS and HSHS.

So the next level is obtained from the previous level with basically HH, HG, GH and GG but with extra shifts in the horizontal, vertical and horizontal and vertical directions. The index provides a way to enumerate the paths through this tree where each smooth has 4 children and indexed by a number between 0 and 3.

Each of the 4 children has 4 components: a smooth, horizontal, vertical and diagonal detail, much in the same way as for the Mallat 2D wavelet transform implemented in the WaveThresh function imwd.

Value

An object of class wst2D with coefficients at resolution level level, packet index and orientation given by type replaced by the matrix packet.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

getpacket.wst2D, wst2D, wst2D.object.

Examples

```
#
# Create a random image.
#
myrand <- matrix(rnorm(16), nrow=4, ncol=4)
#myrand
#            [,1]       [,2]        [,3]       [,4]
#[1,]  0.01692807  0.1400891 -0.38225727  0.3372708
#[2,] -0.79799841 -0.3306080  1.59789958 -1.0606204
#[3,]  0.29151629 -0.2028172 -0.02346776  0.5833292
#[4,] -2.21505532 -0.3591296 -0.39354119  0.6147043
#
# Do the 2D non-decimated wavelet transform
#
myrwst2D <- wst2D(myrand)
#
# Let's access the finest scale detail, not shifted in the vertical
# direction.
```

```
#
getpacket(myrwst2D, nlevelsWT(myrwst2D)-1, index=0, type="V")
#           [,1]       [,2]
#[1,] -0.1626819 -1.3244064
#[2,]  1.4113247 -0.7383336
#
# Let's put some zeros in instead...
#
zmat <- matrix(c(0,0,0,0), 2,2)
newwst2D <- putpacket(myrwst2D, nlevelsWT(myrwst2D)-1,
 index=0, packet=zmat, type="V")
#
# And now look at the same packet as before
#
getpacket(myrwst2D, nlevelsWT(myrwst2D)-1, index=0, type ="V")
#     [,1] [,2]
#[1,]    0    0
#[2,]    0    0
#
# Yup, packet insertion o.k.
```

`rcov` *Computes robust estimate of covariance matrix*

Description

Computes a robust correlation matrix from x.

Usage

```
rcov(x)
```

Arguments

x	Matrix that you wish to find robust covariance of. Number of variables is number of rows, number of observations is number of columns. This is the opposite way round to the convention expected by `var`, for example

Details

Method originates from Huber's "Robust Statistics" book. Note that the columns of `x` must be observations, this is the opposite way around to the usual way for functions like `var`.

Value

The robust covariance matrix

Author(s)

Tim Downie

See Also

threshold.mwd

Examples

```
#
# A standard normal data matrix with 3 variables, 100 observations
#
v <- matrix(rnorm(100*3), nrow=3, ncol=100)
#
# Robust covariance
#
rcov(v)
```

---

rfft *Real Fast Fourier transform*

Description

Compute a real Fast Fourier transform of x.

Usage

```
rfft(x)
```

Arguments

x	The vector whose Fourier transform you wish to take

Details

Given a vector x this function computes the real continuous Fourier transform of x, i.e. it regards x as points on a periodic function on [0,1] starting at 0, and finding the coefficients of the functions 1, $\sqrt{2}\cos(2\pi t)$, $\sqrt{2}\sin(2\pi t)$, etc. that gives the expansion of the interpolant of x. The number of terms in the expansion is the length of x. If x is of even length, the last coefficient will be that of a cosine term with no matching sine.

Value

Returns the Fourier coefficients

Author(s)

Bernard Silverman

See Also

LocalSpec.wd, rfftinv

Examples

```
x <- seq(from=0, to=2*pi, length=150)
s1 <- sin(10*x)
s2 <- sin(7*x)
s <- s1 + s2
w <- rfft(s)
## Not run: ts.plot(w)
#
# Should see two peaks, corresponding to the two sines at different frequencies
#
```

---

rfftinv *Inverse real FFT, inverse of rfft*

Description

Inverse function of rfft

Usage

```
rfftinv(rz, n = length(rz))
```

Arguments

rz	The Fourier coefficients to invert
n	The number of coefficients

Details

Just the inverse function of rfft.

Value

The inverse FT of the input

Author(s)

Bernard Silverman

See Also

rfft

rfftwt — *Weight a Fourier series sequence by a set of weights*

Description

Weight the real Fourier series `xrfft` of even length by a weight sequence `wt`. The first term of `xrfft` is left alone, and the weights are then applied to pairs of terms in `xrfft`. Note: `wt` is half the length of `xrfft`.

Usage

```
rfftwt(xrfft, wt)
```

Arguments

`xrfft`	The Fourier series sequence to weight
`wt`	The weights

Details

Description says all

Value

The weighted sequence

Author(s)

Bernard Silverman

See Also

`rfft`

rm.det — *Set coarse levels of a wavelets on the interval transform object to zero*

Description

Set the wavelet coefficients of certain coarse levels for a "wavelets on the interval" object equal to zero. The operation of this function is somewhat similar to the `nullevels` function, but for objects associated with the "wavelets on the interval code".

Usage

```
rm.det(wd.int.obj)
```

Arguments

`wd.int.obj`	the object whose coarse levels you wish to set to zero

Details

The "wavelets on the interval" code is contained within the `wd` function. All levels coarser than (but not including) the `wd.int.obj$current.scale` are set to zero.

Value

A `wd.object` of `type="interval"` containing the modified input object with certain coarse levels set to zero.

Author(s)

Piotr Fryzlewicz

See Also

`nullevels`, `wd`

---

rmget *Search for existing ipndacw matrices.*

Description

Returns the integer corresponding to the smallest order `ipndacw` matrix of greater than or equal to order than the order, J requested.

Not really intended for user use.

Usage

```
rmget(requestJ, filter.number, family)
```

Arguments

`requestJ`	A positive integer representing the order of the `ipndacw` matrix that is *required*.
`filter.number`	The index number of the wavelet used to build the `ipndacw` matrix that is required.
`family`	The wavelet family used to build the `ipndacw` matrix that is required.

Details

Some of the matrices computed by `ipndacw` take a long time to compute. Hence it is a good idea to store them and reuse them.

This function is asked to find an `ipndacw` matrix of a particular order, *filter.number* and *family*. The function steps through all of the directories in the `search()` list collecting names of all `ipndacw` matrices having the same *filter.number* and *family* characteristics. It then keeps any names where the *order* is larger than, or equal to, the requested order. This means that a suitable `ipndacw` matrix of the same or larger order is visible in one of the `search()` directories. The matrix name with the smallest `order` is selected and the *order* of the matrix is returned. The routine that called this function can then `get()` the matrix and either use it "as is" or extract the top-left hand corner of it if `requestJ` is less than the order returned by this function.

If no such matrix, as described by the previous paragraph, exists then this function returns `NULL`.

This function calls the subsidiary routine `firstdot`.

Value

If a matrix of order larger than or equal to the requested order exists somewhere on the search path *and* the `filter.number` and `family` is as specified then its order is returned. If more than one such matrix exists then the order of the smallest one larger than or equal to the requested one is returned.

If no such matrix exists the function returns NULL.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

See Also

`firstdot`, `ipndacw`, `rmname`.

Examples

```
#
# Suppose there are no matrices in the search path.
#
# Let's look for the matrix rm.4.1.DaubExPhase (Haar wavelet matrix of
# order 4)
#
rmget(requestJ=4, filter.number=1, family="DaubExPhase")
#NULL
```

```
#
# I.e. a NULL return code. So there were no suitable matrices.
#
#If we create two Haar ipndacw matrix of order 7 and 8
#
ipndacw(-7, filter.number=1, family="DaubExPhase")
ipndacw(-8, filter.number=1, family="DaubExPhase")
#
# Now let's repeat the earlier search
#
rmget(requestJ=4, filter.number=1, family="DaubExPhase")
#[1] 7
#
# So, as we the smallest Haar ipndacw matrix available larger than
# the requested order of 4 is "7".
#
```

`rmname` *Return a ipndacw matrix style name.*

Description

This function returns a character string according to a particular format for naming `ipndacw` matrices.

Usage

```
rmname(J, filter.number, family)
```

Arguments

`J`	A negative integer representing the order of the `ipndacw` matrix.
`filter.number`	The index number of the wavelet used to build the `ipndacw` matrix.
`family`	The wavelet family used to build the `ipndacw` matrix.

Details

Some of the matrices computed by `ipndacw` take a long time to compute. Hence it is a good idea to store them and reuse them. This function generates a name according to a particular naming scheme that permits a search algorithm to easily find the matrices.

Each matrix has three defining characteristics: its *order*, *filter.number* and *family*. Each of these three characteristics are concatenated together to form a name.

Value

A character string containing the name of a matrix according to a particular naming scheme.

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

References

Nason, G.P., von Sachs, R. and Kroisandt, G. (1998). Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum. *Technical Report*, Department of Mathematics University of Bristol/ Fachbereich Mathematik, Kaiserslautern.

See Also

ewspec, ipndacw,

Examples

```
#
# What's the name of the order 4 Haar matrix?
#
rmname(-4, filter.number=1, family="DaubExPhase")
#[1] "rm.4.1.DaubExPhase"
#
# What's the name of the order 12 Daubechies least-asymmetric wavelet
# with 7 vanishing moments?
#
rmname(-12, filter.number=7, family="DaubLeAsymm")
#[1] "rm.12.7.DaubLeAsymm"
```

---

rotateback *Cyclically shift a vector one place to the right*

Description

Cyclically shifts the elements of a vector one place to the right. The right-most element becomes the first element.

Usage

```
rotateback(v)
```

Arguments

v	The vector to shift

Details

Subsidiary function used by the av.basis function which is the R function component of the AvBasis.wst function.

Value

The rotated vector

Author(s)

G P Nason

Examples

```
#
# Here is a test vector
#
v <- 1:10
#
# Apply this function
#
rotateback(v)
#[1] 10  1  2  3  4  5  6  7  8  9
#
# A silly little function really!
```

---

rsswav — *Compute mean residual sum of squares for odd prediction of even ordinates and vice versa*

Description

Compute mean of residual sum of squares (RSS) for odd prediction of even ordinates and vice versa using wavelet shrinkage with a specified threshold. This is a subsidiary routine of the WaveletCV cross validation function. A version implemented in C exists called Crsswav.

Usage

```
rsswav(noisy, value = 1, filter.number = 10, family = "DaubLeAsymm",
    thresh.type = "hard", ll = 3)
```

Arguments

noisy	A vector of dyadic (power of two) length that contains the noisy data that you wish to compute the averaged RSS for.
value	The specified threshold.
filter.number	This selects the smoothness of wavelet that you want to perform wavelet shrinkage by cross-validation.

family	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
thresh.type	this option specifies the thresholding type which can be "hard" or "soft".
ll	The primary resolution that you wish to assume. No wavelet coefficients that are on coarser scales than ll will be thresholded.

Details

Note: a faster C based implementation of this function called Crsswav is available. It takes the same arguments and returns the same values.

Two-fold cross validation can be computed for a wd object using the "cv" policy option in threshold.wd. As part of this procedure for each threshold value that the CV optimisation algorithm selects a RSS value must be computed (the CV optimisation algorithm seeks to minimize this RSS value).

The RSS value computed is this. First, the even and odd indexed values are separated. The even values are used to construct an estimate of the odd true values using wavelet shrinkage with the given threshold. The sum of squares between the estimate and the noisy odds is computed. An equivalent calculation is performed by swapping the odds and evens. The two RSS values are then averaged and the average returned. This algorithm is described more fully in Nason, (1996).

Value

A list with the following components

ssq	The RSS value that was computed
df	The dof value computed on the thresholded wavelet transform of the data with the given threshold and thresholding options. (Although this is not really used for anything).
value	The value argument that was specified.
type	the thresh.type argument that was specified.
lev	The vector ll:(nlevelsWT(noisy)-1) (i.e. the levels that were thresholded).

Author(s)

G P Nason

See Also

Crsswav,threshold.wd, WaveletCV

ScalingFunction *Compute scaling functions on internally predefined grid*

Description

This is a subsidiary routine not intended to be called by a user: use draw instead. Generates scaling functions by inserting a Kronecker delta function into the bottom of the inverse DWT and repeating the inverting steps.

Usage

```
ScalingFunction(filter.number = 10, family = "DaubLeAsymm", resolution = 4096,
    itlevels = 50)
```

Arguments

filter.number	The filter number of the associated wavelet. See filter.select
family	The family of the associated wavelet. See filter.select
resolution	The nominal resolution, the actual grid size might be larger than this
itlevels	The number of complete filtering operations to generate the answer

Details

Description says all

Value

A list containing the x and y values of the required scaling function.

Author(s)

G P Nason

See Also

draw

`Shannon.entropy` *Compute Shannon entropy*

Description

Computes Shannon entropy of the squares of a set of coefficients.

Usage

```
Shannon.entropy(v, zilchtol=1e-300)
```

Arguments

`v`	A vector of coefficients (e.g. wavelet coefficients).
`zilchtol`	A small number. Any number smaller than this is considered to be zero for the purposes of this function.

Details

This function computes the Shannon entropy of the squares of a set of coefficients. The squares are used because we are only interested in the entropy of the energy of the coefficients, not their actual sign.

The entropy of the squares of `v` is given by `sum( v^2 * log(v^2) )`. In this implementation any zero coefficients (determined by being less than `zilchtol`) have a zero contribution to the entropy.

The Shannon entropy measures how "evenly spread" a set of numbers is. If the size of the entries in a vector is approximately evenly spread then the Shannon entropy is large. If the vector is sparsely populated or the entries are very different then the Shannon entropy is near zero. Note that the input vectors to this function usually have their norm normalized so that diversity of coefficients corresponds to sparsity.

Value

A number representing the Shannon entropy of the input vector.

RELEASE

Version 3.7.2 Copyright Guy Nason 1996

Author(s)

G P Nason

See Also

`MaNoVe.wst`, `wst`,

Examples

```
#
# Generate some test data
#
#
# A sparse set
#
Shannon.entropy(c(1,0,0,0))
#0
#
# A evenly spread set
#
Shannon.entropy( rep( 1/ sqrt(4), 4 ))
#1.386294
```

`simchirp` *Compute and return simulated chirp function.*

Description

This function computes and returns the coordinates of the reflected simulated chirp function described in Nason and Silverman, 1995. This function is a useful test function for evaluating wavelet shrinkage and time-scale analysis methodology as its frequency changes over time.

Usage

```
simchirp(n=1024)
```

Arguments

`n`	The number of ordinates from which to sample the chirp signal.

Details

This function computes and returns the x and y coordinates of the reflected chirp function described in Nason and Silverman, 1995.

The formula for the reflected simulated chirp is *formula*

The chirp returned is a discrete sample on `n` equally spaced points between -1 and 1.

Value

A list with two components:

`x`	a vector of length `n` containing the ordered x ordinates of the chirp from -1 to 1.
`y`	a vector of length `n` containing the corresponding y ordinates of the chirp.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

Examples

```
#
# Generate the chirp
#
test.data <- simchirp()$y
## Not run: ts.plot(test.data)
```

---

ssq — *Compute sum of squares difference between two vectors*

Description

Given two vectors, u and v, of length n, this function computes $\sum_{i=1}^{n}(u_i - v_i)^2$.

Usage

```
ssq(u,v)
```

Arguments

u	One of the vectors
v	The other of the vectors

Details

Description says all

Value

The sum of squares difference between the two vectors

Author(s)

G P Nason

Examples

```
ssq(c(1,2), c(3,4))
#[1] 8
```

summary.imwd *Print out some basic information associated with an imwd object*

Description

Prints out the number of levels, the dimensions of the original image from which the object came, the type of wavelet filter associated with the decomposition, the type of boundary handling.

Usage

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

Arguments

object	The object to print a summary about
...	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

imwd, threshold.imwd

Examples

```
m <- matrix(rnorm(32*32),nrow=32)
mimwd <- imwd(m)
summary(mimwd)
#UNcompressed image wavelet decomposition structure
#Levels:  5
#Original image was 32 x 32  pixels.
#Filter was:  Daub cmpct on least asymm N=10
#Boundary handling:  periodic
```

summary.imwdc *Print out some basic information associated with an imwdc object*

Description

Prints out the number of levels, the dimensions of the original image from which the object came, the type of wavelet filter associated with the decomposition, the type of boundary handling.

Usage

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

Arguments

object	The object to print a summary about
...	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

imwd, threshold.imwd

Examples

```
m <- matrix(rnorm(32*32),nrow=32)
mimwd <- imwd(m)
mimwdc <- threshold(mimwd)
summary(mimwdc)
#Compressed image wavelet decomposition structure
#Levels:  5
#Original image was 32 x 32  pixels.
#Filter was:  Daub cmpct on least asymm N=10
#Boundary handling:  periodic
```

summary.mwd *Use summary() on a mwd object.*

Description

This function prints out more information about an mwd.object in a nice human-readable form.

Usage

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

Arguments

object	An object of class mwd that you wish to print out more information.
...	Any other arguments.

Value

Nothing of any particular interest.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6)

Note

Prints out information about mwd objects in nice readable format.

Author(s)

Tim Downie

See Also

accessC.mwd, accessD.mwd, draw.mwd, mfirst.last, mfilter.select, mwd, mwd.object, mwr, plot.mwd, print.mwd, putC.mwd, putD.mwd, threshold.mwd, wd, wr.mwd.

Examples

```
#
# Generate an mwd object.
#
tmp <- mwd(rnorm(32))
#
# Now get Splus to use summary.mwd
#
summary(tmp)
# Length of original:  32
```

```
# Levels:  4
# Filter was:  Geronimo Multiwavelets
# Scaling fns:  2
# Wavelet fns:  2
# Prefilter:  default
# Scaling factor:  2
# Boundary handling:  periodic
# Transform type:  wavelet
# Date:  Tue Nov 16 13:55:26 GMT 1999
```

---

summary.wd *Print out some basic information associated with a wd object*

Description

Prints out the number of levels, the length of the original vector from which the object came, the type of wavelet filter associated with the decomposition, the type of boundary handling, the transform type and the date of production.

Usage

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

Arguments

object	The object to print a summary about
...	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

wd

Examples

```
vwd <- wd(1:8)
summary(vwd)
#Levels:  3
#Length of original:  8
#Filter was:  Daub cmpct on least asymm N=10
#Boundary handling:  periodic
#Transform type:  wavelet
#Date:  Mon Mar  8 21:30:32 2010
```

---

summary.wd3D *Print out some basic information associated with a wd3D object*

Description

Prints out the number of levels, the type of wavelet filter associated with the decomposition, and the date of production.

Usage

```
## S3 method for class 'wd3D'
summary(object, ...)
```

Arguments

object	The object to print a summary about
...	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

wd3D

Examples

```
test.data.3D <- array(rnorm(8*8*8), dim=c(8,8,8))
tdwd3D <- wd3D(test.data.3D)
summary(tdwd3D)
#Levels:  3
#Filter number was:  10
#Filter family was:  DaubLeAsymm
#Date:  Mon Mar  8 21:48:00 2010
```

---

summary.wp *Print out some basic information associated with a wp object*

Description

Prints out the number of levels, the length of the original vector from which the object came, the type of wavelet filter associated with the decomposition.

Usage

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

Arguments

object	The object to print a summary about
...	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

wp

Examples

```
vwp <- wp(rnorm(32))
summary(vwp)
#Levels:  5
#Length of original:  32
#Filter was:  Daub cmpct on least asymm N=10
```

summary.wpst *Print out some basic information associated with a wpst object*

Description

Prints out the number of levels, the length of the original vector from which the object came, the type of wavelet filter associated with the decomposition, and the date of production.

Usage

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

Arguments

object	The object to print a summary about
...	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

wpst

Examples

```
vwpst <- wpst(rnorm(32))
summary(vwpst)
#Levels:  5
#Length of original:  32
#Filter was:  Daub cmpct on least asymm N=10
#Date:  Mon Mar  8 21:54:47 2010
```

summary.wst *Print out some basic information associated with a wst object*

Description

Prints out the number of levels, the length of the original vector from which the object came, the type of wavelet filter associated with the decomposition, and the date of production.

Usage

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

Arguments

object	The object to print a summary about
...	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

wst

Examples

```
vwst <- wst(rnorm(32))
summary(vwst)
#Levels:  5
#Length of original:  32
#Filter was:  Daub cmpct on least asymm N=10
#Date:  Mon Mar  8 21:56:12 2010
```

summary.wst2D *Print out some basic information associated with a wst2D object*

Description

Prints out the number of levels, the dimensions of the original image from which the object came, the type of wavelet filter associated with the decomposition, and the date of production.

Usage

```
## S3 method for class 'wst2D'
summary(object, ...)
```

Arguments

object	The object to print a summary about
...	Other arguments

Details

Description says all

Value

Nothing

Author(s)

G P Nason

See Also

wst2D

Examples

```
m <- matrix(rnorm(32*32), nrow=32)
mwst2D <- wst2D(m)
summary(mwst2D)
#Levels:  5
#Length of original:  32 x 32
#Filter was:  Daub cmpct on least asymm N=10
#Date:  Mon Mar  8 21:57:55 2010
```

support *Returns support of compactly supported wavelets.*

Description

Returns the support for compactly supported wavelets. This information is useful for drawing wavelets for annotating axes.

Usage

```
support(filter.number=10, family="DaubLeAsymm", m=0, n=0)
```

Arguments

filter.number	The member index of a wavelet within the family. For Daubechies' compactly supported wavelet this is the number of vanishing moments which is related to the smoothness. See filter.select for more information on the wavelets.
family	The family of wavelets. See filter.select for more information on the wavelets.
m	Optional scale value (in usual wavelet terminology this is j)
n	Optional translation value (in usual wavelet terminology, this is k)

Details

It is useful to know the support of a wavelet when drawing it to annotate labels. Other functions, such as wavelet density estimation (CWavDE), also use this information.

Value

A list with the following components (each one is a single numeric value)

lh	Left hand support of the wavelet with scale m and translation n. These values change as m and n (although when m=0 the function confusingly returns the next coarser wavelet where you might expect it to return the mother. The mother is indexed by m=-1)
rh	As lh but returns the rh end.
psi.lh	left hand end of the support interval for the mother wavelet (remains unchanged no matter what m or n are)
psi.rh	right hand end of the support interval for the mother wavelet (remains unchanged no matter what m or n are)
phi.lh	left hand end of the support interval for the father wavelet (remains unchanged no matter what m or n are)
phi.rh	right hand end of the support interval for the father wavelet (remains unchanged no matter what m or n are)

Author(s)

G P Nason

See Also

CWavDE, draw.default, filter.select

Examples

```
#
# What is the support of a Haar wavelet?
#
support(filter.number=1, family="DaubExPhase", m=0, n=0)
#$lh
#[1] 0
#
#$rh
#[1] 2
#
#$psi.lh
#[1] 0
#
#$psi.rh
#[1] 1
#
#$phi.lh
#[1] 0
#
#$phi.rh
#[1] 1
#
# So the mother and father wavelet have support [0,1]
#
```

---

sure — *Computes the minimum of the SURE thresholding function*

Description

Computes the minimum of the SURE thresholding function for wavelet shrinkage as described in Donoho, D.L. and Johnstone, I.M. (1995) Adapting to unknown smoothness via wavelet shrinkage. *J. Am. Statist. Ass.*, **90**, 1200-1224.

Usage

```
sure(x)
```

Arguments

x	Vector of (normalized) wavelet coefficients. Coefficients should be supplied divided by their standard deviation, or some robust measure of scale

Details

SURE is a method for unbiasedly estimating the risk of an estimator. Stein (1981) showed that for a nearly arbitrary, nonlinear biased estimator, one can estimate its loss unbiasedly. See the Donoho and Johnstone, 1995 for further references and explanation. This function minimizes formula (11) from that paper.

Value

The absolute value of the wavelet coefficient that minimizes the SURE criteria

Author(s)

G P Nason

See Also

threshold

Examples

```
#
# Let's create "pretend" vector of wavelet coefficients contaminated with
# "noise".
#
v <- c(0.1, -0.2, 0.3, -0.4, 0.5, 99, 12, 6)
#
# Now, what's sure of this?
#
sure(v)
#
# [1] 0.5
#
#
# I.e. the large significant coefficients are 99, 12, 6 and the noise is
# anything less than this in abs value. So sure(v) is a good point to threshold
# at.
```

---

teddy *Picture of a teddy bear's picnic.*

Description

A 512x512 matrix. Each entry of the matrix contains an image intensity value.

Usage

```
data(teddy)
```

Format

A 512x512 matrix. Each entry of the matrix contains an image intensity value. The whole matrix represents an image of a teddy bear's picnic.

Author(s)

G P Nason

Source

Taken by Guy Nason.

Examples

```
#
# This command produces the image seen above.
#
# image(teddy)
#
```

---

test.dataCT *Test functions for wavelet regression and thresholding*

Description

This function evaluates the "blocks", "bumps", "heavisine" and "doppler" test functions of Donoho & Johnstone (1994b) and the piecewise polynomial test function of Nason & Silverman (1994). The function also generates data sets consisting of the specified function plus uncorrelated normally distributed errors.

Usage

```
test.dataCT(type = "ppoly", n = 512, signal = 1, rsnr = 7, plotfn = FALSE)
```

Arguments

`type`	Test function to be computed. Available types are "ppoly" (piecewise polynomial), "blocks", "bumps", "heavi" (heavisine), and "doppler".
`n`	Number of equally spaced data points on which the function is evaluated.
`signal`	Scaling parameter; the function will be scaled so that the standard deviation of the data points takes this value.
`rsnr`	Root signal-to-noise ratio. Specifies the ratio of the standard deviation of the function to the standard deviation of the simulated errors.
`plotfn`	If `plotfn=TRUE`, then the test function and the simulated data set are plotted

Value

A list with the following components:

`x`	The points at which the test function is evaluated.
`y`	The values taken by the test function.
`ynoise`	The simulated data set.
`type`	The type of function generated, identical to the input parameter type.
`rsnr`	The root signal-to-noise ratio of the simulated data set, identical to the input parameter rsnr.

Side effects

If `plotfn=T`, the test function and data set are plotted.

RELEASE

Part of the CThresh addon to WaveThresh. Copyright Stuart Barber and Guy Nason 2004.

Author(s)

Stuart Barber

`threshold` *Threshold coefficients*

Description

Modify coefficients by thresholding or shrinkage.

This function is generic.

Particular methods exist for the following objects:

wd object the `threshold.wd` function is used;

imwd object the `threshold.imwd` function is used;

imwdc object the `threshold.imwdc` function is used;

irregwd object the `threshold.irregwd` function is used;

wd3D object the `threshold.wd3D` function is used;

wp object the `threshold.wp` function is used;

wst object the `threshold.wst` function is used.

Usage

```
threshold(...)
```

Arguments

...	See individual help pages for details.

Details

See individual method help pages for operation and examples.

Value

Usually a copy of the input object but containing thresholded or shrunk coefficients.

RELEASE

Version 2 Copyright Guy Nason 1993

Author(s)

G P Nason

See Also

imwd.object, imwdc.object, irregwd object, threshold.imwd, threshold.imwdc, threshold.irregwd, threshold.wd, threshold.wd3D, threshold.wp, threshold.wst wd.object, wd3D.object, wp.object, wst.object.

---

threshold.imwd *Threshold two-dimensional wavelet decomposition object*

Description

This function provides various ways to threshold a imwd class object.

Usage

```
## S3 method for class 'imwd'
threshold(imwd, levels = 3:(nlevelsWT(imwd) - 1), type = "hard", policy =
        "universal", by.level = FALSE, value = 0, dev = var, verbose = FALSE,
        return.threshold = FALSE, compression = TRUE, Q = 0.05, ...)
```

Arguments

imwd	The two-dimensional wavelet decomposition object that you wish to threshold.
levels	a vector of integers which determines which scale levels are thresholded in the decomposition. Each integer in the vector must refer to a valid level in the imwd object supplied. This is usually any integer from 0 to nlevelsWT(wd)-1 inclusive. Only the levels in this vector contribute to the computation of the threshold and its application. (except for the fdr policy).

`type` determines the type of thresholding this can be "`hard`" or "`soft`".

`policy` selects the technique by which the threshold value is selected. Each policy corresponds to a method in the literature. At present the different policies are: "`universal`", "`manual`", "`fdr`", "`probability`". The policies are described in detail below.

`by.level` If FALSE then a global threshold is computed on and applied to all scale levels defined in levels. If TRUE a threshold is computed and applied separately to each scale level.

`value` This argument conveys the user supplied threshold. If the `policy="manual"` then value is the actual threshold value; if `policy="probability"` then `value` conveys the the user supplied quantile level.

`dev` this argument supplies the function to be used to compute the spread of the absolute values coefficients. The function supplied must return a value of spread on the variance scale (i.e. not standard deviation) such as the `var()` function. A popular, useful and robust alternative is the `madmad` function.

`verbose` if TRUE then the function prints out informative messages as it progresses.

`return.threshold` If this option is TRUE then the actual *value* of the threshold is returned. If this option is FALSE then a thresholded version of the input is returned.

`compression` If this option is TRUE then this function returns a comressed two-dimensional wavelet transform object of class `imwdc`. This can be useful as the resulting object will be smaller than if it was not compressed. The compression makes use of the fact that many coefficients in a thresholded object will be exactly zero. If this option is FALSE then a larger `imwd` object will be returned.

`Q` Parameter for the false discovery rate `"fdr"` policy.

`...` any other arguments

Details

This function thresholds or shrinks wavelet coefficients stored in a `imwd` object and by default returns the coefficients in a modified `imwdc` object. See the seminal papers by Donoho and Johnstone for explanations about thresholding. For a gentle introduction to wavelet thresholding (or shrinkage as it is sometimes called) see Nason and Silverman, 1994. For more details on each technique see the descriptions of each method below

The basic idea of thresholding is very simple. In a signal plus noise model the wavelet transform of an image is very sparse, the wavelet transform of noise is not (in particular, if the noise is iid Gaussian then so if the noise contained in the wavelet coefficients). Thus, since the image gets concentrated in few wavelet coefficients and the noise remains "spread" out it is "easy" to separate the signal from noise by keeping large coefficients (which correspond to true image) and delete the small ones (which correspond to noise). However, one has to have some idea of the noise level (computed using the dev option in threshold functions). If the noise level is very large then it is possible, as usual, that no image coefficients "stick up" above the noise.

There are many components to a successful thresholding procedure. Some components have a larger effect than others but the effect is not the same in all practical data situations. Here we give some rough practical guidance, although *you must refer to the papers below when using a*

particular technique. **You cannot expect to get excellent performance on all signals unless you fully understand the rationale and limitations of each method below**. I am not in favour of the "black-box" approach. The thresholding functions of WaveThresh3 are not a black box: experience and judgement are required!

Some issues to watch for:

levels The default of `levels = 3:(wd$nlevelsWT - 1)` for the `levels` option most certainly does not work globally for all data problems and situations. The level at which thresholding begins (i.e. the given threshold and finer scale wavelets) is called the *primary resolution* and is unique to a particular problem. In some ways choice of the primary resolution is very similar to choosing the bandwidth in kernel regression albeit on a logarithmic scale. See Hall and Patil, (1995) and Hall and Nason (1997) for more information. For each data problem you need to work out which is the best primary resolution. This can be done by gaining experience at what works best, or using prior knowledge. It is possible to "automatically" choose a "best" primary resolution using cross-validation (but not in WaveThresh).

Secondly the levels argument computes and applies the threshold at the levels specified in the `levels` argument. It does this for all the levels specified. Sometimes, in wavelet shrinkage, the threshold is computed using only the finest scale coefficients (or more precisely the estimate of the overall noise level). If you want your threshold variance estimate only to use the finest scale coefficients (e.g. with universal thresholding) then you will have to apply the `threshold.imwd` function twice. Once (with levels set equal to `nlevelsWT`(wd)-1) and with `return.threshold=TRUE` to return the threshold computed on the finest scale and then apply the threshold function with the `manual` option supplying the value of the previously computed threshold as the `value` options.

Note that the fdr policy does its own thing.

by.level for a `wd` object which has come from data with noise that is correlated then you should have a threshold computed for each resolution level. See the paper by Johnstone and Silverman, 1997.

Value

An object of class `imwdc` if the `compression` option above is TRUE, otherwise a `imwd` object is returned. In either case the returned object contains the thresholded coefficients. Note that if the `return.threshold` option is set to TRUE then the threshold values will be returned rather than the thresholded object.

RELEASE

Version 3.6 Copyright Guy Nason and others 1997

Note

This section gives a brief description of the different thresholding policies available. For further details see the *associated papers*. If there is no paper available then a small description is provided here. More than one policy may be good for problem, so experiment! They are arranged here in alphabetical order:

fdr See Abramovich and Benjamini, 1996. Contributed by Felix Abramovich.

manual specify a user supplied threshold using `value` to pass the value of the threshold. The `value` argument should be a vector. If it is of length 1 then it is replicated to be the same length as the `levels` vector, otherwise it is repeated as many times as is necessary to be the `levels` vector's length. In this way, different thresholds can be supplied for different levels. Note that the `by.level` option has no effect with this policy.

probability The `probability` policy works as follows. All coefficients that are smaller than the valueth quantile of the coefficients are set to zero. If `by.level` is false, then the quantile is computed for all coefficients in the levels specified by the "levels" vector; if `by.level` is true, then each level's quantile is estimated separately. The probability policy is pretty stupid - do not use it.

universal See Donoho and Johnstone, 1995.

Author(s)

G P Nason

References

The FDR code segments were kindly donated by Felix Abramovich.

See Also

`imwd`, `imwd.object`, `imwdc.object`. `threshold`.

Examples

```
#
# Let's use the lennon test image
#
data(lennon)
## Not run: image(lennon)
#
# Now let's do the 2D discrete wavelet transform
#
lwd <- imwd(lennon)
#
# Let's look at the coefficients
#
## Not run: plot(lwd)
#
# Now let's threshold the coefficients
#
lwdT <- threshold(lwd)
#
# And let's plot those the thresholded coefficients
#
## Not run: plot(lwdT)
#
# Note that the only remaining coefficients are down in the bottom
# left hand corner of the plot. All the others (black) have been set
# to zero (i.e. thresholded).
```

threshold.imwdc *Threshold two-dimensional compressed wavelet decomposition object*

Description

This function provides various ways to threshold a imwdc class object.

Usage

```
## S3 method for class 'imwdc'
threshold(imwdc, verbose=FALSE, ...)
```

Arguments

imwdc	The two-dimensional compressed wavelet decomposition object that you wish to threshold.
verbose	if TRUE then the function prints out informative messages as it progresses.
...	other arguments passed to the threshold.imwd function to control the thresholding characteristics such as policy, type of thresholding etc.

Details

This function performs exactly the same function as threshold.imwd except is accepts objects of class imwdc rather than imwd. Indeed, this function physically calls the threshold.imwd function after using the uncompress function to convert the input imwdc object into a imwd object.

Value

An object of class imwdc if the compression option is supplied and set to TRUE, otherwise a imwd object is returned. In either case the returned object contains the thresholded coefficients. Note that if the return.threshold option is set to TRUE then the threshold values will be returned rather than the thresholded object.

RELEASE

Version 3.6 Copyright Guy Nason and others 1997

Author(s)

G P Nason

References

The FDR code segments were kindly donated by Felix Abramovich.

See Also

imwd, imwd.object, imwdc.object, threshold, uncompress.

Examples

```
#
# See examples in \code{\link{threshold.imwd}}.
#
```

---

threshold.irregwd *hold irregularly spaced wavelet decomposition object*

Description

This function provides various ways to threshold a irregwd class object.

Usage

```
## S3 method for class 'irregwd'
threshold(irregwd,
    levels = 3:(nlevelsWT(wd) - 1), type = "hard", policy = "universal",
    by.level = FALSE, value = 0, dev = var, boundary = FALSE,
    verbose = FALSE, return.threshold = FALSE,
    force.sure=FALSE, cvtol = 0.01, Q = 0.05, alpha=0.05, ...)
```

Arguments

irregwd	The irregularly spaced wavelet decomposition object that you wish to threshold.
levels	a vector of integers which determines which scale levels are thresholded in the decomposition. Each integer in the vector must refer to a valid level in the irregwd object supplied. This is usually any integer from 0 to nlevelsWT(irregwd)-1 inclusive. Only the levels in this vector contribute to the computation of the threshold and its application.
type	determines the type of thresholding this can be "hard" or "soft".
policy	selects the technique by which the threshold value is selected. Each policy corresponds to a method in the literature. At present the different policies are: "universal", "LSuniversal", "sure", "cv", "fdr", "op1", "op2", "manual", "mannum", "probability". A description of the policies can be obtained by clicking on the above links.
by.level	If FALSE then a global threshold is computed on and applied to all scale levels defined in levels. If TRUE a threshold is computed and applied separately to each scale level.
value	This argument conveys the user supplied threshold. If the policy="manual" then value is the actual threshold value.
dev	this argument supplies the function to be used to compute the spread of the absolute values coefficients. The function supplied must return a value of spread on the variance scale (i.e. not standard deviation) such as the var() function. A popular, useful and robust alternative is the madmad function.

boundary	If this argument is TRUE then the boundary bookeeping values are included for thresholding, otherwise they are not.
verbose	if TRUE then the function prints out informative messages as it progresses.
return.threshold	If this option is TRUE then the actual *value* of the threshold is returned. If this option is FALSE then a thresholded version of the input is returned.
force.sure	If TRUE then the SURE threshold is computed on a vector even when that vector is very sparse. If FALSE then the normal SUREshrink procedure is followed whereby the universal threshold is used for sparse vectors of coefficients.
cvtol	Parameter for the cross-validation "cv" policy.
Q	Parameter for the false discovery rate "fdr" policy.
alpha	Parameter for Ogden and Parzen's first "op1" and "op2" policies.
...	other arguments

Details

This function thresholds or shrinks wavelet coefficients stored in a irregwd object and returns the coefficients in a modified irregwd object. The thresholding step is an essential component of denoising.

The basic idea of thresholding is very simple. In a signal plus noise model the wavelet transform of signal is very sparse, the wavelet transform of noise is not (in particular, if the noise is iid Gaussian then so if the noise contained in the wavelet coefficients). Thus since the signal gets concentrated in the wavelet coefficients and the noise remains "spread" out it is "easy" to separate the signal from noise by keeping large coefficients (which correspond to signal) and delete the small ones (which correspond to noise). However, one has to have some idea of the noise level (computed using the dev option in threshold functions). If the noise level is very large then it is possible, as usual, that no signal "sticks up" above the noise.

For thresholding of an *irregularly spaced wavelet decomposition* things are a little different. The original data are irregularly spaced (i.e. [x,y] where the x_i are irregularly spaced) and even if one assumes iid error on the original data once this has been interpolated to a grid by the makegrid function the interpolated data values are not independent. The irregwd function computes the wavelet transform of the interpolated data but also computes the variance of each coefficient using a fast transform. This variance information is stored in the c component of irregwd objects and this function, threshold.irregwd, makes use of this variance information when thresholding each coefficient. For more details see Kovac and Silverman, 2000

Some issues to watch for:

levels The default of levels = 3:(wd$nlevelsWT - 1) for the levels option most certainly does not work globally for all data problems and situations. The level at which thresholding begins (i.e. the given threshold and finer scale wavelets) is called the *primary resolution* and is unique to a particular problem. In some ways choice of the primary resolution is very similar to choosing the bandwidth in kernel regression albeit on a logarithmic scale. See Hall and Patil, (1995) and Hall and Nason (1997) for more information. For each data problem you need to work out which is the best primary resolution. This can be done by gaining experience at what works best, or using prior knowledge. It is possible to "automatically" choose a "best" primary resolution using cross-validation (but not yet in WaveThresh).

Secondly the levels argument computes and applies the threshold at the levels specified in the levels argument. It does this for all the levels specified. Sometimes, in wavelet shrinkage, the threshold is computed using only the finest scale coefficients (or more precisely the estimate of the overall noise level). If you want your threshold variance estimate only to use the finest scale coefficients (e.g. with universal thresholding) then you will have to apply the `threshold.wd` function twice. Once (with levels set equal to `nlevelsWT`(wd)-1 and with `return.threshold=TRUE` to return the threshold computed on the finest scale and then apply the threshold function with the manual option supplying the value of the previously computed threshold as the value options.

by.level for a `wd` object which has come from data with noise that is correlated then you should have a threshold computed for each resolution level. See the paper by Johnstone and Silverman, 1997.

Value

An object of class `irregwd`. This object contains the thresholded wavelet coefficients. Note that if the `return.threshold` option is set to `TRUE` then the threshold values will be returned rather than the thresholded object.

RELEASE

Version 3.6 Copyright Guy Nason 1997

Author(s)

Arne Kovac

See Also

`makegrid`, `irregwd`, `irregwd` object, `accessc`,

Examples

```
#
# See main examples of these functions in the help to makegrid
#
```

`threshold.mwd` *Use threshold on an mwd object.*

Description

Applies hard or soft thresholding to multiple wavelet decomposition object mwd.object.

Usage

```
## S3 method for class 'mwd'
threshold(mwd, levels = 3:(nlevelsWT(mwd) - 1), type = "hard",
    policy = "universal", boundary = FALSE, verbose = FALSE,
    return.threshold = FALSE, threshold = 0, covtol = 1e-09,
    robust = TRUE, return.chisq = FALSE,
    bivariate = TRUE, ...)
```

Arguments

`mwd`	The multiple wavelet decomposition object that you wish to threshold.
`levels`	a vector of integers which determines which scale levels are thresholded in the decomposition. Each integer in the vector must refer to a valid level in the `mwd` object supplied. This is usually any integer from 0 to `nlevelsWT`(wd)-1 inclusive. Only the levels in this vector contribute to the computation of the threshold and its application.
`type`	determines the type of thresholding this can be "`hard`" or "`soft`".
`policy`	selects the technique by which the threshold value is selected. Each policy corresponds to a method in the literature. At present the different policies are "`universal`", "`manual`", "`single`". The policies are described in detail below.
`boundary`	If this argument is `TRUE` then the boundary bookeeping values are included for thresholding, otherwise they are not.
`verbose`	if `TRUE` then the function prints out informative messages as it progresses.
`return.threshold`	If this option is `TRUE` then the actual *value* of the threshold is returned. If this option is FALSE then a thresholded version of the input is returned.
`threshold`	This argument conveys the user supplied threshold. If the `policy="manual"` then `value` is the actual threshold value. Any other `policy` means that the `threshold` value is ignored.
`covtol`	The tolerance for what constitutes a singular variance matrix. If smallest eigenvalue of the estimated variance matrix is less than `covtol` then it is assumed to be singular and no thresholding is done at that level. Note: do not confuse `covtol` with `cvtol` an argument in `threshold.wd`.
`robust`	If TRUE the variance matrix at each level is estimated using a robust method (mad) otherwise it is estimated using var().
`return.chisq`	If TRUE the vector of values to be thresholded is returned. These values are a quadratic form of each coefficient vector, and under normal assumptions the noise component will have a chi-squared distribution (see Downie and Silverman 1996).
`bivariate`	this line is in construction
`...`	any other arguments

Details

Thresholding modifies the coefficients within a mwd.object. The modification can be performed either with a "hard" or "soft" thresholding selected by the type argument.

Unless policy="single", the following method is applied. The columns of mwd$D are taken as coefficient vectors $D_{j,k}$. From these $\chi^2_{j,k} = D_{j,k} \cdot V_j^{-1}$. $D_{j,k}$ is computed, where V_j^{-1} is the inverse of the estimated variance of the coefficient vectors in that level (j). $\chi^2_{j,k}$ is a positive scalar which is to be thresholded in a similar manner to univariate hard or soft thresholding. To obtain the new values of $D_{j,k}$ shrink the vector by the same proportion as was the corresponding $\chi^2_{j,k}$ term. i

Value

An object of class mwd. This object contains the thresholded wavelet coefficients. Note that if the return.threshold option is set to TRUE then the threshold values will be returned, or if return.chisq the vector of values to be thresholded will be returned, rather than the thresholded object.

RELEASE

Version 3.9.6 (Although Copyright Tim Downie 1995-6).

Note

POLICIES

single If policy="single" then univariate thresholding is applied to each element of D as in (Strela et al 1999).

universal The universal threshold is computed using 2log(n) (See Downie & Silverman 1996) where n is the number of coefficient vectors to be thresholded.

manual The "manual" policy is simple. You supply a threshold value to the threshold argument and hard or soft thresholding is performed using that value

Author(s)

Tim Downie

See Also

accessC.mwd, accessD.mwd, draw.mwd, mfirst.last, mfilter.select, mwd, mwd.object, mwr, plot.mwd, print.mwd, putC.mwd, putD.mwd, summary.mwd, wd, wr.mwd.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
# Generate some noisy data
```

```
#
ynoise <- test.data + rnorm(512, sd=0.1)
##
# Plot it
#
## Not run: ts.plot(ynoise)
#
# Now take the discrete multiple wavelet transform
# N.b. I have no idea if the default wavelets here are appropriate for
# this particular examples.
#
ynmwd <- mwd(ynoise)
## Not run: plot(ynwd)
# [1] 2.020681 2.020681 2.020681 2.020681 2.020681 2.020681 2.020681
#
# Now do thresholding. We'll use the default arguments.
#
ynmwdT <- threshold(ynmwd)
#
# And let's plot it
#
## Not run: plot(ynmwdT)
#
# Let us now see what the actual estimate looks like
#
ymwr <- wr(ynmwdT)
#
# Here's the estimate...
#
## Not run: ts.plot(ywr1)
```

---

threshold.wd *Threshold (DWT) wavelet decomposition object*

Description

This function provides various ways to threshold a wd class object.

Usage

```
## S3 method for class 'wd'
threshold(wd, levels = 3:(nlevelsWT(wd) - 1), type = "soft", policy = "sure",
      by.level = FALSE, value = 0, dev = madmad, boundary = FALSE, verbose = FALSE,
        return.threshold = FALSE, force.sure = FALSE, cvtol = 0.01,
 cvmaxits=500, Q = 0.05, OP1alpha = 0.05,
        alpha = 0.5, beta = 1, C1 = NA, C2 = NA, C1.start = 100,
 al.check=TRUE, ...)
```

Arguments

wd	The DWT wavelet decomposition object that you wish to threshold.
levels	a vector of integers which determines which scale levels are thresholded in the decomposition. Each integer in the vector must refer to a valid level in the wd object supplied. This is usually any integer from 0 to nlevelsWT(wd)-1 inclusive. Only the levels in this vector contribute to the computation of the threshold and its application.
type	determines the type of thresholding this can be "hard" or "soft".
policy	selects the technique by which the threshold value is selected. Each policy corresponds to a method in the literature. At present the different policies are: "universal", "LSuniversal", "sure", "BayesThresh", "cv", "fdr", "op1", "op2", "manual", "mannum" and "probability". The policies are described in detail below.
by.level	If FALSE then a global threshold is computed on and applied to all scale levels defined in levels. If TRUE a threshold is computed and applied separately to each scale level.
value	This argument conveys the user supplied threshold. If the policy="manual" then value is the actual threshold value; if the policy="mannum" then value conveys the total number of ordered coefficients kept (from the largest); if policy="probability" then value conveys the the user supplied quantile level.
dev	this argument supplies the function to be used to compute the spread of the absolute values coefficients. The function supplied must return a value of spread on the variance scale (i.e. not standard deviation) such as the var() function. A popular, useful and robust alternative is the madmad function.
boundary	If this argument is TRUE then the boundary bookeeping values are included for thresholding, otherwise they are not.
verbose	if TRUE then the function prints out informative messages as it progresses.
return.threshold	If this option is TRUE then the actual *value* of the threshold is returned. If this option is FALSE then a thresholded version of the input is returned.
force.sure	If TRUE then the sure threshold is computed on a vector even when that vector is very sparse. If FALSE then the normal SUREshrink procedure is followed whereby the universal threshold is used for sparse vectors of coefficients.
cvtol	Parameter for the cross-validation "cv" policy.
cvmaxits	Maximum number of iterations allowed for the cross-validation "cv" policy.
Q	Parameter for the false discovery rate "fdr" policy.
OP1alpha	Parameter for Ogden and Parzen's first "op1" and "op2" policies.
alpha	Parameter for BayesThresh "BayesThresh" policy.
beta	Parameter for BayesThresh "BayesThresh" policy.
C1	Parameter for BayesThresh "BayesThresh" policy.
C2	Parameter for BayesThresh "BayesThresh" policy.
C1.start	Parameter for BayesThresh "BayesThresh" policy.

`al.check` If TRUE then the function checks that the levels are in ascending order. If they are not then this can be an indication that the default level arguments are not appropriate for this data set (`wd` object). However, a strange order might be appropriate for some reason if deliberately set, so setting this argument equal to FALSE turns off the check and warning.

`...` any other arguments

Details

This function thresholds or shrinks wavelet coefficients stored in a `wd` object and returns the coefficients in a modified `wd` object. See the seminal papers by Donoho and Johnstone for explanations about thresholding. For a gentle introduction to wavelet thresholding (or shrinkage as it is sometimes called) see Nason and Silverman, 1994. For more details on each technique see the descriptions of each method below

The basic idea of thresholding is very simple. In a signal plus noise model the wavelet transform of signal is very sparse, the wavelet transform of noise is not (in particular, if the noise is iid Gaussian then so if the noise contained in the wavelet coefficients). Thus since the signal gets concentrated in the wavelet coefficients and the noise remains "spread" out it is "easy" to separate the signal from noise by keeping large coefficients (which correspond to signal) and delete the small ones (which correspond to noise). However, one has to have some idea of the noise level (computed using the dev option in threshold functions). If the noise level is very large then it is possible, as usual, that no signal "sticks up" above the noise.

There are many components to a successful thresholding procedure. Some components have a larger effect than others but the effect is not the same in all practical data situations. Here we give some rough practical guidance, although *you must refer to the papers below when using a particular technique.* **You cannot expect to get excellent performance on all signals unless you fully understand the rationale and limitations of each method below.** I am not in favour of the "black-box" approach. The thresholding functions of WaveThresh3 are not a black box: experience and judgement are required!

Some issues to watch for:

levels The default of `levels = 3:(wd$nlevelsWT - 1)` for the `levels` option most certainly does not work globally for all data problems and situations. The level at which thresholding begins (i.e. the given threshold and finer scale wavelets) is called the *primary resolution* and is unique to a particular problem. In some ways choice of the primary resolution is very similar to choosing the bandwidth in kernel regression albeit on a logarithmic scale. See Hall and Patil, (1995) and Hall and Nason (1997) for more information. For each data problem you need to work out which is the best primary resolution. This can be done by gaining experience at what works best, or using prior knowledge. It is possible to "automatically" choose a "best" primary resolution using cross-validation (but not in WaveThresh).

Secondly the levels argument computes and applies the threshold at the levels specified in the `levels` argument. It does this for all the levels specified. Sometimes, in wavelet shrinkage, the threshold is computed using only the finest scale coefficients (or more precisely the estimate of the overall noise level). If you want your threshold variance estimate only to use the finest scale coefficients (e.g. with universal thresholding) then you will have to apply the `threshold.wd` function twice. Once (with levels set equal to `nlevelsWT`(wd)-1 and with `return.threshold=TRUE` to return the threshold computed on the finest scale and then apply

the threshold function with the manual option supplying the value of the previously computed threshold as the value options.

Thirdly, if you apply wavelet shrinkage to a small data set then you need to ensure you've chosen the `levels` argument appropriately. For example, if your original data was of length 8, then the associated `wd` wavelet decomposition object will only have levels 0, 1 and 2. So, the default argument for levels (starting at 3 and higher) will almost certainly be wrong. The code now warns for these situations.

by.level for a `wd` object which has come from data with noise that is correlated then you should have a threshold computed for each resolution level. See the paper by Johnstone and Silverman, 1997.

Value

An object of class `wd`. This object contains the thresholded wavelet coefficients. Note that if the `return.threshold` option is set to TRUE then the threshold values will be returned rather than the thresholded object.

RELEASE

Version 3.6 Copyright Guy Nason and others 1997

Note

POLICIES This section gives a brief description of the different thresholding policies available. For further details see *the associated papers*. If there is no paper available then a small description is provided here. More than one policy may be good for problem, so experiment! They are arranged here in alphabetical order:

BayesThresh See Abramovich, Silverman and Sapatinas, (1998). Contributed by Felix Abramovich and Fanis Sapatinas.

cv See Nason, 1996.

fdr See Abramovich and Benjamini, 1996. Contributed by Felix Abramovich.

LSuniversal See Nason, von Sachs and Kroisandt, 1998. This is used for smoothing of a wavelet periodogram and shouldn't be used generally.

manual specify a user supplied threshold using `value` to pass the value of the threshold. The `value` argument should be a vector. If it is of length 1 then it is replicated to be the same length as the `levels` vector, otherwise it is repeated as many times as is necessary to be the `levels` vector's length. In this way, different thresholds can be supplied for different levels. Note that the `by.level` option has no effect with this policy.

mannum You decided how many of the largest (in absolute value) coefficients that you want to keep and supply this number in value.

op1 See Ogden and Parzen, 1996. Contributed by Todd Ogden.

op2 See Ogden and Parzen, 1996. Contributed by Todd Ogden.

probability The `probability` policy works as follows. All coefficients that are smaller than the valueth quantile of the coefficients are set to zero. If `by.level` is false, then the quantile is computed for all coefficients in the levels specified by the "levels" vector; if `by.level` is true, then each level's quantile is estimated separately. The probability policy is pretty stupid - do not use it.

sure See Donoho and Johnstone, 1994.

universal See Donoho and Johnstone, 1995.

Author(s)

G P Nason

References

Various code segments detailed above were kindly donated by Felix Abramovich, Theofanis Sapatinas and Todd Ogden.

See Also

wd, wd.object, wr, wr.wd, threshold.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
# Generate some noisy data
#
ynoise <- test.data + rnorm(512, sd=0.1)
#
# Plot it
#
## Not run: ts.plot(ynoise)
#
# Now take the discrete wavelet transform
# N.b. I have no idea if the default wavelets here are appropriate for
# this particular examples.
#
ynwd <- wd(ynoise)
## Not run: plot(ynwd)
#
# Now do thresholding. We'll use a universal policy,
# and madmad deviance estimate on the finest
# coefficients and return the threshold. We'll also get it to be verbose
# so we can watch the process.
#
ynwdT1 <- threshold(ynwd, policy="universal", dev=madmad,
  levels= nlevelsWT(ynwd)-1, return.threshold=TRUE,
 verbose=TRUE)
# threshold.wd:
# Argument checking
# Universal policy...All levels at once
# Global threshold is:  0.328410967430135
#
```

```
# Why is this the threshold? Well in this case n=512 so sqrt(2*log(n)),
# the universal threshold,
# is equal to 3.53223. Since the noise is about 0.1 (because that's what
# we generated it to be) the threshold is about 0.353.
#
# Now let's apply this threshold to all levels in the noisy wavelet object
#
ynwdT1obj <- threshold(ynwd, policy="manual", value=ynwdT1,
 levels=0:(nlevelsWT(ynwd)-1))
#
# And let's plot it
#
## Not run: plot(ynwdT1obj)
#
# You'll see that a lot of coefficients have been set to zero, or shrunk.
#
# Let's try a Bayesian examples this time!
#
ynwdT2obj <- threshold(ynwd, policy="BayesThresh")
#
# And plot the coefficients
#
## Not run: plot(ynwdT2obj)
#
# Let us now see what the actual estimates look like
#
ywr1 <- wr(ynwdT1obj)
ywr2 <- wr(ynwdT2obj)
#
# Here's the estimate using universal thresholding
#
## Not run: ts.plot(ywr1)
#
# Here's the estimate using BayesThresh
#
## Not run: ts.plot(ywr2)
```

---

threshold.wd3D *Threshold 3D DWT object*

Description

This function provides various ways to threshold a wd3D class object.

Usage

```
## S3 method for class 'wd3D'
threshold(wd3D, levels = 3:(nlevelsWT(wd3D) - 1), type = "hard", policy =
        "universal", by.level = FALSE, value = 0, dev = var, verbose = FALSE,
        return.threshold = FALSE, ...)
```

Arguments

`wd3D`	The 3D DWT wavelet decomposition object that you wish to threshold.
`levels`	a vector of integers which determines which scale levels are thresholded in the decomposition. Each integer in the vector must refer to a valid level in the `wd3D` object supplied. This is usually any integer from 0 to `nlevelsWT`(wd3D)-1 inclusive. Only the levels in this vector contribute to the computation of the threshold and its application.
`type`	determines the type of thresholding this can be "`hard`" or "`soft`".
`policy`	selects the technique by which the threshold value is selected. Each policy corresponds to a method in the literature. At present the different policies are: "`universal`" and "`manual`". The policies are described in detail `below`.
`by.level`	If FALSE then a global threshold is computed on and applied to all scale levels defined in `levels`. If TRUE a threshold is computed and applied separately to each scale level.
`value`	This argument conveys the user supplied threshold. If the `policy="manual"` then value is the actual threshold value.
`dev`	this argument supplies the function to be used to compute the spread of the absolute values coefficients. The function supplied must return a value of spread on the variance scale (i.e. not standard deviation) such as the `var()` function. A popular, useful and robust alternative is the `madmad` function.
`verbose`	if TRUE then the function prints out informative messages as it progresses.
`return.threshold`	If this option is TRUE then the actual *value* of the threshold is returned. If this option is FALSE then a thresholded version of the input is returned.
`...`	any other arguments

Details

This function thresholds or shrinks wavelet coefficients stored in a `wd3D` object and returns the coefficients in a modified `wd3D` object. See the seminal papers by Donoho and Johnstone for explanations about thresholding. For a gentle introduction to wavelet thresholding (or shrinkage as it is sometimes called) see Nason and Silverman, 1994. For more details on each technique see the descriptions of each method below

The basic idea of thresholding is very simple. In a signal plus noise model the wavelet transform of signal is very sparse, the wavelet transform of noise is not (in particular, if the noise is iid Gaussian then so if the noise contained in the wavelet coefficients). Thus since the signal gets concentrated in the wavelet coefficients and the noise remains "spread" out it is "easy" to separate the signal from noise by keeping large coefficients (which correspond to signal) and delete the small ones (which correspond to noise). However, one has to have some idea of the noise level (computed using the dev option in threshold functions). If the noise level is very large then it is possible, as usual, that no signal "sticks up" above the noise.

There are many components to a successful thresholding procedure. Some components have a larger effect than others but the effect is not the same in all practical data situations. Here we give some rough practical guidance, although *you must refer to the papers below when using a particular technique.* **You cannot expect to get excellent performance on all signals unless you**

fully understand the rationale and limitations of each method below. I am not in favour of the "black-box" approach. The thresholding functions of WaveThresh3 are not a black box: experience and judgement are required!

Some issues to watch for:

levels The default of `levels = 3:(wd$nlevelsWT - 1)` for the `levels` option most certainly does not work globally for all data problems and situations. The level at which thresholding begins (i.e. the given threshold and finer scale wavelets) is called the *primary resolution* and is unique to a particular problem. In some ways choice of the primary resolution is very similar to choosing the bandwidth in kernel regression albeit on a logarithmic scale. See Hall and Patil, (1995) and Hall and Nason (1997) for more information. For each data problem you need to work out which is the best primary resolution. This can be done by gaining experience at what works best, or using prior knowledge. It is possible to "automatically" choose a "best" primary resolution using cross-validation (but not in WaveThresh).

Secondly the levels argument computes and applies the threshold at the levels specified in the `levels` argument. It does this for all the levels specified. Sometimes, in wavelet shrinkage, the threshold is computed using only the finest scale coefficients (or more precisely the estimate of the overall noise level). If you want your threshold variance estimate only to use the finest scale coefficients (e.g. with universal thresholding) then you will have to apply the `threshold.wd` function twice. Once (with levels set equal to `nlevelsWT`(wd)-1 and with `return.threshold=TRUE` to return the threshold computed on the finest scale and then apply the threshold function with the manual option supplying the value of the previously computed threshold as the value options.

by.level for a `wd` object which has come from data with noise that is correlated then you should have a threshold computed for each resolution level. See the paper by Johnstone and Silverman, 1997.

Value

An object of class `wd3D`. This object contains the thresholded wavelet coefficients. Note that if the return.threshold option is set to TRUE then the threshold values will be returned rather than the thresholded object.

RELEASE

Version 3.9.6 Copyright Guy Nason 1997.

Note

POLICIES

This section gives a brief description of the different thresholding policies available. For further details *see the associated papers*. If there is no paper available then a small description is provided here. More than one policy may be good for problem, so experiment! They are arranged here in alphabetical order:

manual specify a user supplied threshold using value to pass the value of the threshold. The value argument should be a vector. If it is of length 1 then it is replicated to be the same length as the `levels` vector, otherwise it is repeated as many times as is necessary to be the `levels`

vector's length. In this way, different thresholds can be supplied for different levels. Note that the `by.level` option has no effect with this policy.

universal See Donoho and Johnstone, 1995.

Author(s)

G P Nason

See Also

`threshold`, `accessD.wd3D`, `print.wd3D`, `putD.wd3D`, `putDwd3Dcheck`, `summary.wd3D`, `threshold.wd3D`, `wd3D.object`, `wr3D`.

Examples

```
#
# Generate some test data
#
test.data <- array(rnorm(8*8*8), dim=c(8,8,8))
testwd3D <- wd3D(test.data)
#
# Now let's threshold
#
testwd3DT <- threshold(testwd3D, levels=1:2)
#
# That's it, one can apply wr3D now to reconstruct
# if you like!
#
```

`threshold.wp` *Threshold wavelet packet decomposition object*

Description

This function provides various ways to threshold a `wp` class object.

Usage

```
## S3 method for class 'wp'
threshold(wp, levels = 3:(nlevelsWT(wp) - 1), dev = madmad,
 policy = "universal", value = 0, by.level = FALSE, type = "soft",
 verbose = FALSE, return.threshold = FALSE, cvtol = 0.01, cvnorm = l2norm,
 add.history = TRUE, ...)
```

Arguments

`wp`	The wavelet packet object that you wish to threshold.
`levels`	a vector of integers which determines which scale levels are thresholded in the decomposition. Each integer in the vector must refer to a valid level in the `wd` object supplied. This is usually any integer from 0 to `nlevelsWT`(wp)-1 inclusive. Only the levels in this vector contribute to the computation of the threshold and its application.
`policy`	selects the technique by which the threshold value is selected. Each policy corresponds to a method in the literature. At present the different policies are: "`universal`" and "`manual`". The policies are described in detail `below`.
`by.level`	If FALSE then a global threshold is computed on and applied to all scale levels defined in `levels`. If TRUE a threshold is computed and applied separately to each scale level.
`value`	This argument conveys the user supplied threshold. If the `policy="manual"` then value is the actual threshold value.
`dev`	this argument supplies the function to be used to compute the spread of the absolute values coefficients. The function supplied must return a value of spread on the variance scale (i.e. not standard deviation) such as the `var()` function. A popular, useful and robust alternative is the `madmad` function.
`type`	determines the type of thresholding this can be "`hard`" or "`soft`".
`verbose`	if TRUE then the function prints out informative messages as it progresses.
`return.threshold`	If this option is TRUE then the actual *value* of the threshold is returned. If this option is FALSE then a thresholded version of the input is returned.
`cvtol`	Not used, but reserved for future use
`cvnorm`	Not used, but reserved for future use
`add.history`	if `TRUE` then a history statement is added to the object for displaying.
`...`	any other arguments

Details

This function thresholds or shrinks wavelet coefficients stored in a `wp` object and returns the coefficients in a modified `wp` object. See the seminal papers by Donoho and Johnstone for explanations about thresholding. For a gentle introduction to wavelet thresholding (or shrinkage as it is sometimes called) see Nason and Silverman, 1994. For more details on each technique see the descriptions of each method below

The basic idea of thresholding is very simple. In a signal plus noise model the wavelet transform of signal is very sparse, the wavelet transform of noise is not (in particular, if the noise is iid Gaussian then so if the noise contained in the wavelet coefficients). Thus since the signal gets concentrated in the wavelet coefficients and the noise remains "spread" out it is "easy" to separate the signal from noise by keeping large coefficients (which correspond to signal) and delete the small ones (which correspond to noise). However, one has to have some idea of the noise level (computed using the dev option in threshold functions). If the noise level is very large then it is possible, as usual, that no signal "sticks up" above the noise.

There are many components to a successful thresholding procedure. Some components have a larger effect than others but the effect is not the same in all practical data situations. Here we give some rough practical guidance, although *you must refer to the papers below when using a particular technique.* **You cannot expect to get excellent performance on all signals unless you fully understand the rationale and limitations of each method below.** I am not in favour of the "black-box" approach. The thresholding functions of WaveThresh3 are not a black box: experience and judgement are required!

Some issues to watch for:

levels The default of `levels = 3:(wd$nlevelsWT - 1)` for the `levels` option most certainly does not work globally for all data problems and situations. The level at which thresholding begins (i.e. the given threshold and finer scale wavelets) is called the *primary resolution* and is unique to a particular problem. In some ways choice of the primary resolution is very similar to choosing the bandwidth in kernel regression albeit on a logarithmic scale. See Hall and Patil, (1995) and Hall and Nason (1997) for more information. For each data problem you need to work out which is the best primary resolution. This can be done by gaining experience at what works best, or using prior knowledge. It is possible to "automatically" choose a "best" primary resolution using cross-validation (but not in WaveThresh).

Secondly the levels argument computes and applies the threshold at the levels specified in the `levels` argument. It does this for all the levels specified. Sometimes, in wavelet shrinkage, the threshold is computed using only the finest scale coefficients (or more precisely the estimate of the overall noise level). If you want your threshold variance estimate only to use the finest scale coefficients (e.g. with universal thresholding) then you will have to apply the `threshold.wp` function twice. Once (with levels set equal to `nlevelsWT`(wd)-1 and with `return.threshold=TRUE` to return the threshold computed on the finest scale and then apply the threshold function with the manual option supplying the value of the previously computed threshold as the value options.

by.level for a `wd` object which has come from data with noise that is correlated then you should have a threshold computed for each resolution level. See the paper by Johnstone and Silverman, 1997.

Value

An object of class `wp`. This object contains the thresholded wavelet coefficients. Note that if the `return.threshold` option is set to TRUE then the threshold values will be returned rather than the thresholded object.

RELEASE

Version 3.6 Copyright Guy Nason and others1997.

Note

POLICIES

This section gives a brief description of the different thresholding policies available. For further details *see the associated papers*. If there is no paper available then a small description is provided here. More than one policy may be good for problem, so experiment! They are arranged here in alphabetical order:

universal See Donoho and Johnstone, 1995.

Author(s)

G P Nason

See Also

wp, wp.object, InvBasis, MaNoVe, threshold.

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
# Generate some noisy data
#
ynoise <- test.data + rnorm(512, sd=0.1)
#
# Plot it
#
## Not run: ts.plot(ynoise)
#
# Now take the discrete wavelet packet transform
# N.b. I have no idea if the default wavelets here are appropriate for
# this particular examples.
#
ynwp <- wp(ynoise)
#
# Now do thresholding. We'll use a universal policy,
# and madmad deviance estimate on the finest
# coefficients and return the threshold. We'll also get it to be verbose
# so we can watch the process.
#
ynwpT1 <- threshold(ynwp, policy="universal", dev=madmad)
#
# This is just another wp object. Is it sensible?
# Probably not as we have just thresholded the scaling function coefficients
# as well. So the threshold might be more sensibly computed on the wavelet
# coefficients at the finest scale and then this threshold applied to the
# whole wavelet tree??
```

---

threshold.wst *Threshold (NDWT) packet-ordered non-decimated wavelet decomposition object*

Description

This function provides various ways to threshold a wst class object

Usage

```
## S3 method for class 'wst'
threshold(wst, levels = 3:(nlevelsWT(wst) - 1), dev = madmad, policy =
 "universal", value = 0, by.level = FALSE, type = "soft", verbose
  = FALSE, return.threshold = FALSE, cvtol = 0.01, cvnorm = l2norm,
 add.history = TRUE, ...)
```

Arguments

wst	The packet ordered non-decimated wavelet decomposition object that you wish to threshold.
levels	a vector of integers which determines which scale levels are thresholded in the decomposition. Each integer in the vector must refer to a valid level in the wst object supplied. This is usually any integer from 0 to nlevelsWT(wst)-1 inclusive. Only the levels in this vector contribute to the computation of the threshold and its application.
dev	this argument supplies the function to be used to compute the spread of the absolute values coefficients. The function supplied must return a value of spread on the variance scale (i.e. not standard deviation) such as the var() function. A popular, useful and robust alternative is the madmad function
policy	selects the technique by which the threshold value is selected. Each policy corresponds to a method in the literature. At present the different policies are: "universal", "LSuniversal", "sure", "cv", "manual", The policies are described in detail below.
value	This argument conveys the user supplied threshold. If the policy="manual" then value is the actual threshold value.
by.level	If FALSE then a global threshold is computed on and applied to all scale levels defined in levels. If TRUE a threshold is computed and applied separately to each scale level.
type	determines the type of thresholding this can be "hard" or "soft".
verbose	if TRUE then the function prints out informative messages as it progresses.
return.threshold	If this option is TRUE then the actual *value* of the threshold is returned. If this option is FALSE then a thresholded version of the input is returned.
cvtol	Parameter for the cross-validation "cv" policy.
cvnorm	A function to compute the distance between two vectors. Two useful possibilities are l2norm and linfnorm. Selection of different metrics causes the cross-validation denoising method to optimize for different criteria.
add.history	If TRUE then the thresholding operation details are add to the returned wst object. This can be useful when later tracing how an object has been treated.
...	any other arguments

Details

This function thresholds or shrinks wavelet coefficients stored in a `wst` object and returns the coefficients in a modified `wst` object. The thresholding step is an essential component of denoising using the `packet-ordered non-decimated wavelet transform`. If the denoising is carried out using the `AvBasis` basis averaging technique then this software is an implementation of the Coifman and Donoho translation-invariant (TI) denoising. (Although it is the denoising technique which is translation invariant, not the packet ordered non-decimated transform, which is translation equivariant). However, the `threshold.wst` algorithm can be used in other denoising techniques in conjunction with the basis selection and inversion functions `MaNoVe` and `InvBasis`.

The basic idea of thresholding is very simple. In a signal plus noise model the wavelet transform of signal is very sparse, the wavelet transform of noise is not (in particular, if the noise is iid Gaussian then so if the noise contained in the wavelet coefficients). Thus since the signal gets concentrated in the wavelet coefficients and the noise remains "spread" out it is "easy" to separate the signal from noise by keeping large coefficients (which correspond to signal) and delete the small ones (which correspond to noise). However, one has to have some idea of the noise level (computed using the dev option in threshold functions). If the noise level is very large then it is possible, as usual, that no signal "sticks up" above the noise.

Many of the pragmatic comments for successful thresholding given in the help for `threshold.wd` hold true here: after all non-decimated wavelet transforms are merely organized collections of standard (decimated) discrete wavelet transforms. We reproduce some of the issues relevant to thresholding `wst` objects.

Some issues to watch for:

levels The default of `levels = 3:(nlevelsWT(wd) - 1)` for the `levels` option most certainly does not work globally for all data problems and situations. The level at which thresholding begins (i.e. the given threshold and finer scale wavelets) is called the `primary resolution` and is unique to a particular problem. In some ways choice of the primary resolution is very similar to choosing the bandwidth in kernel regression albeit on a logarithmic scale. See Hall and Patil, (1995) and Hall and Nason (1997) for more information. For each data problem you need to work out which is the best primary resolution. This can be done by gaining experience at what works best, or using prior knowledge. It is possible to "automatically" choose a "best" primary resolution using cross-validation (but not yet in WaveThresh).

Secondly the levels argument computes and applies the threshold at the levels specified in the `levels` argument. It does this for all the levels specified. Sometimes, in wavelet shrinkage, the threshold is computed using only the finest scale coefficients (or more precisely the estimate of the overall noise level). If you want your threshold variance estimate only to use the finest scale coefficients (e.g. with universal thresholding) then you will have to apply the `threshold.wd` function twice. Once (with levels set equal to `nlevelsWT`(wd)-1 and with `return.threshold=TRUE` to return the threshold computed on the finest scale and then apply the threshold function with the `manual` option supplying the value of the previously computed threshold as the `value` options.

by.level for a `wd` object which has come from data with noise that is correlated then you should have a threshold computed for each resolution level. See the paper by Johnstone and Silverman, 1997.

Value

An object of class `wst`. This object contains the thresholded wavelet coefficients. Note that if the `return.threshold` option is set to TRUE then the threshold values will be returned rather than the thresholded object.

RELEASE

Version 3.6 Copyright Guy Nason 1997

Note

This section gives a brief description of the different thresholding policies available. For further details *see the associated papers*. If there is no paper available then a small description is provided here. More than one policy may be good for problem, so experiment! Some of the policies here were specifically adapted to the This section gives a brief description of the different thresholding policies available. For further details see the associated papers. If there is no paper available then a small description is provided here. More than one policy may be good for problem, so experiment! Some of the policies here were specifically adapted to the `wst.object` but some weren't so beware. They are arranged here in alphabetical order:

cv See Nason, 1996.

LSuniversal See Nason, von Sachs and Kroisandt, 1998. This is used for smoothing of a wavelet periodogram and shouldn't be used generally.

manual specify a user supplied threshold using `value` to pass the value of the threshold. The `value` argument should be a vector. If it is of length 1 then it is replicated to be the same length as the `levels` vector, otherwise it is repeated as many times as is necessary to be the levels vector's length. In this way, different thresholds can be supplied for different levels. Note that the `by.level` option has no effect with this policy.

sure See Donoho and Johnstone, 1994 and Johnstone and Silverman, 1997.

universal See Donoho and Johnstone, 1995.

Author(s)

G P Nason

See Also

`AvBasis`, `AvBasis.wst`, `InvBasis`, `InvBasis.wst`, `MaNoVe`,`MaNoVe.wst`, `wst`, `wst.object`, `threshold`.

`TOgetthrda1` *Subsidiary routines for Ogden and Parzen's wavelet shrinkage methods*

Description

Corresponds to the wavelet thresholding routine developed by Ogden and Parzen (1994) Data dependent wavelet thresholding in nonparametric regression with change-point applications. *Tech Rep 176*, University of South Carolina, Department of Statistics.

Usage

```
TOgetthrda1(dat, alpha)
TOgetthrda2(dat, alpha)
TOkolsmi.chi2(dat)
TOonebyone1(dat, alpha)
TOonebyone2(dat, alpha)
TOshrinkit(coeffs, thresh)
```

Arguments

`dat`	data
`alpha`	a p-value, generally smoothing parameter
`coeffs`	Some coefficients to be shrunk
`thresh`	a threshold

Details

Not intended for direct use.

Value

Various depending on the function

Author(s)

Todd Ogden

See Also

`TOthreshda1`,`TOthreshda2`,`threshold`

TOthreshda1 *Data analytic wavelet thresholding routine*

Description

This function might be better called using the regular threshold function using the op1 policy.

Corresponds to the wavelet thresholding routine developed by Ogden and Parzen (1994) Data dependent wavelet thresholding in nonparametric regression with change-point applications. *Tech Rep 176*, University of South Carolina, Department of Statistics.

Usage

```
TOthreshda1(ywd, alpha = 0.05, verbose = FALSE, return.threshold = FALSE)
```

Arguments

ywd	The wd.object that you wish to threshold.
alpha	The smoothing parameter which is a p-value
verbose	Whether messages get printed
return.threshold	If TRUE then the threshold value gets returned rather than the actual thresholded object

Details

The TOthreshda1 method operates by testing the max of each set of squared wavelet coefficients to see if it behaves as the nth order statistic of a set of independent chi^2(1) r.v.'s. If not, it is removed, and the max of the remaining subset is tested, continuing in this fashion until the max of the subset is judged not to be significant.

In this situation, the level of the hypothesis tests, alpha, has default value 0.05. Note that the choice of alpha controls the smoothness of the resulting wavelet estimator – in general, a relatively large alpha makes it easier to include coefficients, resulting in a more wiggly estimate; a smaller alpha will make it more difficult to include coefficients, yielding smoother estimates.

Value

Returns the threshold value if return.threshold==TRUE otherwise returns the shrunk set of wavelet coefficients.

Author(s)

Todd Ogden

See Also

threshold,TOthreshda2, wd

TOthreshda2 *Data analytic wavelet thresholding routine*

Description

This function might be better called using the regular threshold function using the op2 policy.

Corresponds to the wavelet thresholding routine developed by Ogden and Parzen (1994) Data dependent wavelet thresholding in nonparametric regression with change-point applications. *Tech Rep 176*, University of South Carolina, Department of Statistics.

Usage

```
TOthreshda2(ywd, alpha = 0.05, verbose = FALSE, return.threshold = FALSE)
```

Arguments

ywd	The wd.object that you wish to threshold.
alpha	The smoothing parameter which is a p-value
verbose	Whether messages get printed
return.threshold	If TRUE then the threshold value gets returned rather than the actual thresholded object

Details

The TOthreshda2 method operates in a similar fashion to TOthreshda1 except that it takes the cumulative sum of squared coefficients, creating a sample "Brownian bridge" process, and then using the standard Kolmogorov-Smirnov statistic in testing.

In this situation, the level of the hypothesis tests, alpha, has default value 0.05. Note that the choice of alpha controls the smoothness of the resulting wavelet estimator – in general, a relatively large alpha makes it easier to include coefficients, resulting in a more wiggly estimate; a smaller alpha will make it more difficult to include coefficients, yielding smoother estimates.

Value

Returns the threshold value if return.threshold==TRUE otherwise returns the shrunk set of wavelet coefficients.

Author(s)

Todd Ogden

See Also

threshold,TOthreshda1, wd

tpwd *Tensor product 2D wavelet transform*

Description

Performs the tensor product 2D wavelet transform. This is a related, but different, 2D wavelet transform compared to imwd.

Usage

```
tpwd(image, filter.number = 10, family = "DaubLeAsymm", verbose = FALSE)
```

Arguments

image	The image you wish to subject to the tensor product WT
filter.number	The smoothness of wavelet, see filter.select
family	The wavelet family you wish to use
verbose	Whether or not you wish to print out informative messages

Details

The transform works by first taking the regular 1D wavelet transform across all columns in the image and storing these coefficients line by line back into the image. Then to this new image we apply the regular 1D wavelet transform across all rows in the image.

Hence, the top-left coefficient is the smoothed version both horizontally and vertically. The left-most row contains the image smoothed horiztonally, but then detail picked up amongst the horizontal smooths vertically.

Suggested by Rainer von Sachs.

Value

A list with the following components:

tpwd	A matrix with the same dimensions as the input image, but containing the tensor product wavelet transform coefficients.
filter.number	The filter number used
family	The wavelet family used
type	The type of transform used
bc	The boundary conditions used
date	When the transform occurred

Author(s)

G P Nason

See Also

imwd,tpwr

Examples

```
data(lennon)
ltpwd <- tpwd(lennon)
## Not run: image(log(abs(ltpwd$tpwd)), col=grey(seq(from=0, to=1, length=100)))
```

---

tpwr *Inverse tensor product 2D wavelet transform.*

Description

Performs the inverse transform to tpwd.

Usage

```
tpwr(tpwdobj, verbose = FALSE)
```

Arguments

tpwdobj	An object which is a list which contains the items indicated in the return value of tpwd
verbose	Whether informative messages are printed

Details

Performs the inverse transform to tpwd.

Value

A matrix, or image, containing the inverse tensor product wavelet transform of the image contained in the tpwd component of the tpwdobj object.

Author(s)

G P Nason

See Also

imwr,tpwd

Examples

```
data(lennon)
ltpwd <- tpwd(lennon)
#
# now perform the inverse and compare to the original
#
ltpwr <- tpwr(ltpwd)
sum((ltpwr - lennon)^2)
# [1] 9.22802e-10
```

---

uncompress *Uncompress objects*

Description

Uncompress objects.

This function is generic.

Particular methods exist. For the `imwdc.object` class object this generic function uses `uncompress.imwdc`. There is a default uncompression method: `uncompress.default` that works on vectors.

Usage

```
uncompress(...)
```

Arguments

...	See individual help pages for details.

Details

See individual method help pages for operation and examples

Value

A uncompressed version of the input.

RELEASE

Version 2.0 Copyright Guy Nason 1993

Author(s)

G P Nason

See Also

`uncompress.default`, `uncompress.imwdc`, `imwd`, `imwd.object`, `imwdc.object`, `threshold.imwd`

`uncompress.default` *Undo zero run-length encoding for a vector.*

Description

This function inverts the action carried out by the `compress.default` function.

Usage

```
## Default S3 method:
uncompress(v, verbose=FALSE, ...)
```

Arguments

`v`	The object to uncompress
`verbose`	Print an informative message whilst executing
`...`	Other arguments

Details

The inverse of `compress.default`

Value

The uncompressed, reinstated, vector.

Author(s)

G P Nason

See Also

`compress.default`, `uncompress`

Examples

```
uncompress(compress(c(1, rep(0,99), 1)))
#[1] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#[38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#[75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

uncompress.imwdc *Uncompress an imwdc class object*

Description

An imwdc.object is a run-length encoded object, essentially has all zeroes removed and only non-zero elements stored. This function undoes the compression.

Usage

```
## S3 method for class 'imwdc'
uncompress(x, verbose=FALSE, ...)
```

Arguments

x	The object to uncompress
verbose	If TRUE then print out messages
...	Other arguments

Details

Description says all, inverse of compress.imwd function.

Value

The uncompressed imwd.object.

Author(s)

G P Nason

See Also

imwd, compress.imwd

Examples

```
data(lennon)
#
# Do 2D wavelet transform on lennon image
#
lwd <- imwd(lennon)
#
# Do threshold the wavelet coefficients, a lot of zeroes are present
#
lmdT <- threshold(lwd)
#
# What is the class of the thresholded object?
```

```
#
class(lmdT)
#[1] "imwdc"
#
# note that the coefficients are stored efficiently in the imwdc class object
#
uncompress(lmdT)
#Class 'imwd' : Discrete Image Wavelet Transform Object:
#~~~~  : List with 30 components with names
#nlevelsWT fl.dbase filter w0Lconstant bc type w0L1 w0L2 w0L3 w1L1 w1L2
#w1L3 w2L1 w2L2 w2L3 w3L1 w3L2 w3L3 w4L1 w4L2 w4L3 w5L1 w5L2 w5L3 w6L1
#w6L2 w6L3 w7L1 w7L2 w7L3
#
#$ wNLx are LONG coefficient vectors !
#
#summary(.):
#----------
#UNcompressed image wavelet decomposition structure
#Levels:  8
#Original image was 256 x 256  pixels.
#Filter was:  Daub cmpct on least asymm N=10
#Boundary handling:  periodic
```

`wavegrow` *Interactive graphical tool to grow a wavelet synthesis*

Description

Use mouse to select which wavelets to enter a wavelet synthesis, continually plot the reconstruction and the wavelet tableaux.

Usage

```
wavegrow(n = 64, filter.number = 10, family = "DaubLeAsymm", type = "wavelet",
    random = TRUE, read.value = TRUE, restart = FALSE)
```

Arguments

`n`	Number of points in the decomposition
`filter.number`	The wavelet filter.number to use, see `filter.select`
`family`	The wavelet family to use in the reconstruction
`type`	If `"wavelet"` then carry out the regular wavelet transform, otherwise if `"station"` do the nondecimated transform.
`random`	If `TRUE` then iid Gaussian coefficients are inserted into the tableaux. If `FALSE` and `read.value=TRUE` then the user is promoted for a value, otherwise the value 1 is inserted into the tableaux at the selected point.
`read.value`	If `TRUE` then a value is read and used to insert that size of wavelet coefficient at the selected point. If `FALSE` then a coefficient of size 1 is inserted.

`restart` If `TRUE` then after a coefficient has been inserted, and plots done, the next selection causes all the coefficients to be reset to zero and a single coefficient inserted. This actually has the overall action of being able to select a coefficient location and view the size and shape of the wavelet produced.

Details

This function can perform many slightly different actions. However, the basic idea is for a tableaux of wavelet coefficients to be displayed in one graphics window, and the reconstruction of those coefficients to be displayed in another graphics window.

Hence, two graphics windows, capable of plotting and mouse interaction (e.g. X11, windows or quartz) with the locator function, are required to be active.

When the function starts up an initial random tableaux is displayed and its reconstruction.

The next step is for the user to select coefficients on the tableaux. What happens next specifically depends on the arguments above. By default selecting a coefficient causes that coefficient scale and location to be identified, then a random sample is taken from a N(0,1) random variable and assigned to that coefficient. Hence, the tableaux is updated, the reconstruction with the new coefficient computed and both are plotted.

If `type="wavelet"` is used then decimated wavelets are used, if `type="station"` then the time-ordered non-decimated wavelets are used.

If `random=FALSE` then new values for the coefficients are either selected (by asking the user for input) if `read.value=TRUE` or the value of 1 is input.

If `restart=TRUE` then the function merely displays the wavelet associated with the selected coefficient. Hence, this option is useful to demonstrate to people how wavelets from different points of the tableaux have different sizes, scales and locations.

If the mouse locator function is exited (this can be a right-click in some windowing systems, or pressing ESCAPE) then the function asks whether the user wishes to continue. If not then the function returns the current tableux. Hence, this function can be useful for users to build their own tabeleaux.

Value

The final tableaux.

Author(s)

G P Nason

See Also

wd

WaveletCV *Wavelet cross-validation*

Description

Two-fold wavelet shrinkage cross-validation (there is a faster C based version CWCV.)

Usage

```
WaveletCV(ynoise, x = 1:length(ynoise), filter.number = 10, family =
 "DaubLeAsymm", thresh.type = "soft", tol = 0.01, verbose = 0,
 plot.it = TRUE, ll=3)
```

Arguments

ynoise	A vector of dyadic (power of two) length that contains the noisy data that you wish to apply wavelet shrinkage by cross-validation to.
x	This function is capable of producing informative plots. It can be useful to supply the x values corresponding to the ynoise values. Further this argument is returned by this function which can be useful for later processors.
filter.number	This selects the smoothness of wavelet that you want to perform wavelet shrinkage by cross-validation.
family	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
thresh.type	this option specifies the thresholding type which can be "hard" or "soft".
tol	this specifies the convergence tolerance for the cross-validation optimization routine (a golden section search).
verbose	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.
plot.it	If this is TRUE then plots of the universal threshold (used to obtain an upper bound on the cross-validation threshold) reconstruction and the resulting cross-validation estimate are produced.
ll	The primary resolution that you wish to assume. No wavelet coefficients that are on coarser scales than ll will be thresholded.

Details

Note: a faster C based implementation of this function called CWCV is available. It takes the same arguments (although it has one extra minor argument) and returns the same values.

Compute the two-fold cross-validated wavelet shrunk estimate given the noisy data ynoise according to the description given in Nason, 1996.

You must specify a primary resolution given by ll. This must be specified individually on each data set and can itself be estimated using cross-validation (although I haven't written the code to do this).

Note. The two-fold cross-validation method performs very badly if the input data is correlated. In this case I would advise using other methods.

Value

A list with the following components

x	This is just the x that was input. It gets passed through more or less for convenience for the user.
ynoise	A copy of the input ynoise noisy data.
xvwr	The cross-validated wavelet shrunk estimate.
yuvtwr	The universal thresholded version (note this is merely a starting point for the cross-validation algorithm. It should not be taken seriously as an estimate. In particular its estimate of variance is likely to be inflated.)
xvthresh	The cross-validated threshold
uvthresh	The universal threshold (again, don't take this value too seriously. You might get better performance using the threshold function directly with specialist options.
xvdof	The number of non-zero coefficients in the cross-validated shrunk wavelet object (which is not returned).
uvdof	The number of non-zero coefficients in the universal threshold shrunk wavelet object (which also is not returned)
xkeep	always returns NULL!
fkeep	always returns NULL!

Author(s)

G P Nason

See Also

CWCV,Crsswav,rsswav,threshold.wd

Examples

```
#
# This function is best used via the policy="cv" option in
# the threshold.wd function.
# See examples there.
#
```

---

wd *Wavelet transform (decomposition).*

Description

This function can perform two types of discrete wavelet transform (DWT). The standard DWT computes the DWT according to Mallat's pyramidal algorithm (Mallat, 1989) (it also has the ability to compute the *wavelets on the interval* transform of Cohen, Daubechies and Vial, 1993).

The non-decimated DWT (NDWT) contains all possible shifted versions of the DWT. The order of computation of the DWT is O(n), and it is O(n log n) for the NDWT if n is the number of data points.

Usage

```
wd(data, filter.number=10, family="DaubLeAsymm", type="wavelet",
bc="periodic", verbose=FALSE, min.scale=0, precond=TRUE)
```

Arguments

`data` A vector containing the data you wish to decompose. The length of this vector must be a power of 2.

`filter.number` This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.

For the "wavelets on the interval" (`bc="interval"`) transform the filter number ranges from 1 to 8. See the table of filter coefficients indexed after the reference to Cohen, Daubechies and Vial, 1993.

`family` specifies the family of wavelets that you want to use. Two popular options are "DaubExPhase" and "DaubLeAsymm" but see the help for `filter.select` for more possibilities.

This argument is ignored for the "wavelets on the interval" transform (`bc="interval"`).

Note that, as of version 4.6.1 you can use the Lina-Mayrand complex-valued wavelets.

`type` specifies the type of wavelet transform. This can be "wavelet" (default) in which case the standard DWT is performed (as in previous releases of WaveThresh). If type is "station" then the non-decimated DWT is performed. At present, only periodic boundary conditions can be used with the non-decimated wavelet transform.

`bc` specifies the boundary handling. If `bc="periodic"` the default, then the function you decompose is assumed to be periodic on it's interval of definition, if `bc="symmetric"` then the function beyond its boundaries is assumed to be a symmetric reflection of the function in the boundary. The symmetric option was the implicit default in releases prior to 2.2. If `bc=="interval"` then the "wavelets on the interval algorithm" due to Cohen, Daubechies and Vial is used. (The `WaveThresh` implementation of the "wavelets on the interval transform" was coded by Piotr Fryzlewicz, Department of Mathematics, Wroclaw University of Technology, Poland; this code was largely based on code written by Markus Monnerjahn, RHRK, Universitat Kaiserslautern; integration into `WaveThresh` by `GPN`. See the nice project report by Piotr on this piece of code).

`verbose` Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.

`min.scale` Only used for the "wavelets on the interval transform". The wavelet algorithm starts with fine scale data and iteratively coarsens it. This argument controls how many times this iterative procedure is applied by specifying at which scale level to stop decomposiing.

`precond` Only used for the "wavelets on the interval transform". This argument specifies whether preconditioning is applied (called prefiltering in Cohen, Daubechies and Vial, 1993.) Preconditioning ensures that sequences like 1,1,1,1 or 1,2,3,4 map to zero high pass coefficients.

Details

If type=="wavelet" then the code implements Mallat's pyramid algorithm (Mallat 1989). For more details of this implementation see Nason and Silverman, 1994. Essentially it works like this: you start off with some data cm, which is a real vector of length 2^m, say.

Then from this you obtain two vectors of length $2^(m-1)$. One of these is a set of smoothed data, c(m-1), say. This looks like a smoothed version of cm. The other is a vector, d(m-1), say. This corresponds to the detail removed in smoothing cm to c(m-1). More precisely, they are the coefficients of the wavelet expansion corresponding to the highest resolution wavelets in the expansion. Similarly, c(m-2) and d(m-2) are obtained from c(m-1), etc. until you reach c0 and d0.

All levels of smoothed data are stacked into a single vector for memory efficiency and ease of transport across the SPlus-C interface.

The smoothing is performed directly by convolution with the wavelet filter (`filter.select(n)$H`, essentially low- pass filtering), and then dyadic decimation (selecting every other datum, see Vaidyanathan (1990)). The detail extraction is performed by the mirror filter of H, which we call G and is a band-pass filter. G and H are also known quadrature mirror filters.

There are now two methods of handling "boundary problems". If you know that your function is periodic (on it's interval) then use the bc="periodic" option, if you think that the function is symmetric reflection about each boundary then use bc="symmetric". You might also consider using the "wavelets on the interval" transform which is suitable for data arising from a function that is known to be defined on some compact interval, see Cohen, Daubechies, and Vial, 1993. If you don't know then it is wise to experiment with both methods, in any case, if you don't have very much data don't infer too much about your decomposition! If you have loads of data then don't infer too much about the boundaries. It can be easier to interpret the wavelet coefficients from a bc="periodic" decomposition, so that is now the default. Numerical Recipes implements some of the wavelets code, in particular we have compared our code to "wt1" and "daub4" on page 595. We are pleased to announce that our code gives the same answers! The only difference that you might notice is that one of the coefficients, at the beginning or end of the decomposition, always appears in the "wrong" place. This is not so, when you assume periodic boundaries you can imagine the function defined on a circle and you can basically place the coefficient at the beginning or the end (because there is no beginning or end, as it were).

The non-deciated DWT contains all circular shifts of the standard DWT. Naively imagine that you do the standard DWT on some data using the Haar wavelets. Coefficients 1 and 2 are added and difference, and also coefficients 3 and 4; 5 and 6 etc. If there is a discontinuity between 1 and 2 then you will pick it up within the transform. If it is between 2 and 3 you will loose it. So it would be nice to do the standard DWT using 2 and 3; 4 and 5 etc. In other words, pick up the data and rotate it by one position and you get another transform. You can do this in one transform that also does more shifts at lower resolution levels. There are a number of points to note about this transform.

Note that a time-ordered non-decimated wavelet transform object may be converted into a `packet-ordered non-decimated wavelet transform` object (and vice versa) by using the `convert` function.

The NDWT is translation equivariant. The DWT is neither translation invariant or equivariant. The standard DWT is orthogonal, the non-decimated transform is most definitely not. This has the added disadvantage that non-decimated wavelet coefficients, even if you supply independent normal noise. This is unlike the standard DWT where the coefficients are independent (normal noise).

You might like to consider growing wavelet syntheses using the `wavegrow` function.

Value

An object of class wd.

For boundary conditions apart from bc="interval" this object is a list with the following components.

C	Vector of sets of successively smoothed data. The pyramid structure of Mallat is stacked so that it fits into a vector. The function accessC should be used to extract a set for a particular level.
D	Vector of sets of wavelet coefficients at different resolution levels. Again, Mallat's pyramid structure is stacked into a vector. The function accessD should be used to extract the coefficients for a particular resolution level.
nlevelsWT	The number of resolution levels. This depends on the length of the data vector. If length(data)=2^m, then there will be m resolution levels. This means there will be m levels of wavelet coefficients (indexed 0,1,2,...,(m-1)), and m+1 levels of smoothed data (indexed 0,1,2,...,m).
fl.dbase	There is more information stored in the C and D than is described above. In the decomposition "extra" coefficients are generated that help take care of the boundary effects, this database lists where these start and finish, so the "true" data can be extracted.
filter	A list containing information about the filter type: Contains the string "wavelet" or "station" depending on which type of transform was performed.
date	The date the transform was performed.
bc	How the boundaries were handled.

If the "wavelets on the interval" transform is used (i.e. bc="interval") then the internal structure of the wd object is changed as follows.

- The coefficient vectors C and D have been replaced by a single vector transformed.vector. The new single vector contains just the transformed coefficients: i.e. the wavelet coefficients down to a particular scale (determined by min.scale above). The scaling function coefficients are stored first in the array (there will be 2^min.scale of them. Then the wavelet coefficients are stored as consecutive vectors coarsest to finest of length 2^min.scale, 2^(min.scale+1) up to a vector which is half of the length of the original data.)

  In any case the user is recommended to use the functions accessC, accessD, putC and putD to access coefficients from the wd object.
- The extra component current.scale records to which level the transform has been done (usually this is min.scale as specified in the arguments).
- The extra component filters.used is a vector of integers that record which filter index was used as each level of the decomposition. At coarser scales sometimes a wavelet with shorter support is needed.
- The extra logical component preconditioned specifies whether preconditioning was turned on or off.
- The component fl.dbase is still present but only contains data corresponding to the storage of the coefficients that are present in transformed.vector. In particular, since only one scale of the father wavelet coefficients is stored the component first.last.c of fl.dbase is now a

three-vector containing the indices of the first and last entries of the father wavelet coefficients and the offset of where they are stored in `transformed.vector`. Likewise, the component `first.last.d` of `fl.dbase` is still a matrix but there are now only rows for each scale level in the `transformed.vector` (something like `nlevelsWT(wd)-wd$current.scale`).

- The `filter` coefficient is also slightly different as the filter coefficients are no longer stored here (since they are hard coded into the wavelets on the interval transform.)

RELEASE

Version 3.5.3 Copyright Guy Nason 1994 Integration of "wavelets on the interval" code by Piotr Fryzlewicz and Markus Monnerjahn was at Version 3.9.6, 1999.

Author(s)

G P Nason

See Also

`wd.int`, `wr`, `wr.int`, `wr.wd`, `accessC`, `accessD`, `putD`, `putC`, `filter.select`, `plot.wd`, `threshold`, `wavegrow`

Examples

```
#
# Generate some test data
#
test.data <- example.1()$y
## Not run: ts.plot(test.data)
#
# Decompose test.data and plot the wavelet coefficients
#
wds <- wd(test.data)
## Not run: plot(wds)
#
# Now do the time-ordered non-decimated wavelet transform of the same thing
#
wdS <- wd(test.data, type="station")
## Not run: plot(wdS)
#
# Next examples
# ------------
# The chirp signal is also another good examples to use.
#
# Generate some test data
#
test.chirp <- simchirp()$y
## Not run: ts.plot(test.chirp, main="Simulated chirp signal")
#
# Now let's do the time-ordered non-decimated wavelet transform.
# For a change let's use Daubechies least-asymmetric phase wavelet with 8
# vanishing moments (a totally arbitrary choice, please don't read
```

```
# anything into it).
#
chirpwdS <- wd(test.chirp, filter.number=8, family="DaubLeAsymm", type="station")
## Not run: plot(chirpwdS, main="TOND WT of Chirp signal")
#
# Note that the coefficients in this plot are exactly the same as those
# generated by the packet-ordered non-decimated wavelet transform
# except that they are in a different order on each resolution level.
# See Nason, Sapatinas and Sawczenko, 1998
# for further information.
```

---

wd.dh *Compute specialized wavelet transform for density estimation*

Description

Computes the discrete wavelet transform, but with zero boundary conditions especially for density estimation.

Usage

```
wd.dh(data, filter.number = 10, family = "DaubLeAsymm", type = "wavelet",
    bc = "periodic", firstk = NULL, verbose = FALSE)
```

Arguments

data	The father wavelet coefficients
filter.number	The smoothness of the underlying wavelet to use, see filter.select
family	The wavelet family to use, see filter.select
type	The type of wavelet to use
bc	Type of boundarie conditions
firstk	A parameter that originates from denproj
verbose	If TRUE then informative messages are printed.

Details

This is a subsidiary routine, not intended for direct user use for density estimation. The main routines for wavelet density estimation are denwd, denproj, denwr.

The input to this function should be projected father wavelet coefficients as computed by denproj, but usually supplied to this function by denwd.

Thresholding should be carried out by the user independently of these functions.

Value

An object of class wd, but assumed on the basis of zero boundary conditions.

Author(s)

David Herrick

See Also

denproj, denwd

---

wd.int *Computes "wavelets on the interval" transform*

Description

This function actually computes the "wavelets on the interval" transform.

NOTE: It is not recommended that the casual user call this function. The "wavelets on the interval" transform is best called in WaveThresh via the wd function with the argument bc argument set to "interval".

Usage

```
wd.int(data, preferred.filter.number, min.scale, precond)
```

Arguments

data	The data that you wish to apply the "wavelets on the interval" transform to.
preferred.filter.number	Which wavelet to use to do the transform. This is an integer ranging from 1 to 8. See the Cohen, Daubeches and Vial (1993) paper. Wavelets that do not "overlap" a boundary are just like the ordinary Daubechies' wavelets.
min.scale	At which resolution level to transform to.
precond	If true performs preconditioning of the input vector to try and ensure that simple polynomial sequences (less than in order to the wavelet used) map to zero elements.

Details

(The WaveThresh implementation of the "wavelets on the interval transform" was coded by Piotr Fryzlewicz, Department of Mathematics, Wroclaw University of Technology, Poland; this code was largely based on code written by Markus Monnerjahn, RHRK, Universitat Kaiserslautern; integration into WaveThresh by GPN).

See the help on the "wavelets on the interval code" in the wd help page.

Value

A list containing the wavelet transform of the data. We again emphasize that this list is not intended for human consumption, use the wd function with the correct bc="interval" argument.

RELEASE

Version 3.9.6 (Although Copyright Piotr Fryzlewicz and Markus Monnerjahn 1995-9).

Author(s)

Piotr Fryzlewicz

See Also

wd, wr, wr.int.

Examples

```
#
# The user is expected to call the wr
# for inverting a "wavelets on the interval transform" and not to use
# this function explicitly
#
```

---

wd.object *Wavelet decomposition objects*

Description

These are objects of classes

wd

They represent a decomposition of a function with respect to a wavelet basis (or tight frame in the case of the (time-ordered) non-decimated wavelet decomposition).

Details

To retain your sanity the C and D coefficients should be extracted by the accessC and accessD functions and inserted using the putC and putD functions (or more likely, their methods), rather than by the $ operator.

Mind you, if you want to muck about with coefficients directly, then you'll have to do it yourself by working out what the fl.dbase list means (see first.last for a description.)

Note the *time-ordered non-decimated wavelet transform* used to be called the *stationary wavelet transform*. In fact, the non-decimated transform has several possible names and has been reinvented many times. There are two versions of the non-decimated transform: the coefficients are the same in each version just ordered differently within a resolution level. The two transforms are

- The function wd() with an argument type="station" computes the *time-ordered* non-decimated transform (see Nason and Silverman, 1995) which is useful in time-series applications (see e.g. Nason, von Sachs and Kroisandt, 1998).
- The function wst() computes the packets ordered non-decimated transform is useful for curve estimation type applications (see e.g. Coifman and Donoho, 1995).

Value

The following components must be included in a legitimate 'wd' object.

`C` a vector containing each level's smoothed data. The wavelet transform works by applying both a smoothing filter and a bandpass filter to the previous level's smoothed data. The top level contains data at the highest resolution level. Each of these levels are stored one after the other in this vector. The matrix

`fl.dbase$first.last.c`

determines exactly where each level is stored in the vector. Likewise, coefficients stored when the NDWT has been used should only be extracted using the "access" and "put" functions below.

`D` wavelet coefficients. If you were to write down the discrete wavelet transform of a function then these D would be the coefficients of the wavelet basis functions. Like the C, they are also formed in a pyramidal manner, but stored in a linear array. The storage details are to be found in

`fl.dbase$first.last.d`

Likewise, coefficients stored when the NDWT has been used should only be extracted using the "access" and "put" functions below.

`nlevelsWT` The number of levels in the pyramidal decomposition that produces the coefficients. If you raise 2 to the power of nlevels you get the number of data points used in the decomposition.

`fl.dbase` The first last database associated with this decomposition. This is a list consisting of 2 integers, and 2 matrices. The matrices detail how the coefficients are stored in the C and D components of the 'wd.object'. See the help on `first.last` for more information.

`filter` a list containing the details of the filter that did the decomposition

`type` either **wavelet** indicating that the ordinary wavelet transform was performed or **station** indicating that the time-ordered non-decimated wavelet transform was done.

`date` The date that the transform was performed or the wd was modified.

`bc` how the boundaries were handled

GENERATION

This class of objects is returned from the `wd` function to represent a (possibly time-ordered non-decimated) wavelet decomposition of a function. Many other functions return an object of class wd.

METHODS

The wd class of objects has methods for the following generic functions: `plot`, `threshold`, `summary`, `print`, `draw`.

RELAEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wd, wst

---

wd3D *Three-dimensional discrete wavelet transform*

Description

This function performs the 3D version of Mallat's discrete wavelet transform (see Mallat, 1989, although this paper does not describe in detail the 3D version the extension is trivial). The function assumes *periodic* boundary conditions.

Usage

```
wd3D(a, filter.number=10, family="DaubLeAsymm")
```

Arguments

a	A three-dimensional array constructed using the S-Plus `array()` function. Each dimension of the array should be equal to the same power of two.
filter.number	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
family	specifies the family of wavelets that you want to use. Two popular options are "DaubExPhase" and "DaubLeAsymm" but see the help for `filter.select` for more possibilities.

Details

This function implements a straightforward extension of Mallat's, (1989) one- and two-dimensional DWT. The algorithm recursively applies all possible combinations of the G and H detail and smoothing filters to each of the dimensions thus forming 8 different sub-blocks which we label HHH, GHH, HGH, GGH, HHG, GHG, HGG, and GGG. The algorithm recurses on the HHH component of each level (these are the father wavelet coefficients).

Making an analogy to the 2D transform where HH, HG, HG and GG is produced at each resolution level: the HG and GH correspond to "horizontal" and "vertical" detail and GG corresponds to "diagonal detail". The GGG corresponds to the 3D "diagonal" version, HGG corresponds to smoothing in dimension 1 and "diagonal" detail in dimensions 2 and 3, and so on. I don't think there are words in the English language which adequately describe "diagonal" in 3D — maybe cross detail?

Value

An object of class wd3D.

RELEASE

Version 3.9.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

wd, imwd, accessD.wd3D, print.wd3D, putD.wd3D, putDwd3Dcheck, summary.wd3D, threshold.wd3D, wd3D.object, wr3D.

Examples

```
#
# Generate some test data: 512 standard normal observations in an 8x8x8
# array.
#
test.data.3D <- array(rnorm(8*8*8), dim=c(8,8,8))
#
# Now do the 3D wavelet transform
#
tdwd3D <- wd3D(test.data.3D)
#
# See examples explaining  the 3D wavelet transform.
#
```

---

wd3D.object *Three-dimensional wavelet object*

Description

These are objects of classes

wd3D

They contain the 3D discrete wavelet transform of a 3D array (with each dimension being the same dyadic size).

Details

To retain your sanity the wavelet coefficients at any resolution level in directions, GGG, GGH, GHG, GHH, HGG, HGH, HHG should be extracted by the accessD() function and inserted using the putD function rather than by the $ operator.

Value

The following components must be included in a legitimate 'wd' object.

a	a three-dimensional array containing the 3D discrete wavelet coefficients. The coefficients are stored in a pyramid structure for efficiency.
nlevelsWT	The number of levels in the pyramidal decomposition that produces the coefficients. If you raise 2 to the power of nlevels you get the number of data points used in each dimension of the decomposition.
filter.number	the number of the wavelet family that did the DWT.
family	the family of wavelets that did the DWT.
date	the date that the transform was computed.

generation

This class of objects is returned from the wd3D function to represent a three-dimensional DWT of a 3D array. Other functions return an object of class wd3D.

methods

The wd3D class of objects has methods for the following generic functions: accessD, print, putD, summary, threshold.

release

Version 3.9.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

wd3D, accessD.wd3D, print.wd3D, putD.wd3D, putDwd3Dcheck, summary.wd3D, threshold.wd3D, wr3D.

Whistory *Obsolete function supposedly detailed history of object*

Description

The original idea behind this obsolete function was to interrogate an object and return the modifications that had been successively applied to the function. The reason for this was that after a long data analysis session one would end up with a whole set of, e.g., thresholded or otherwise modified objects and it would have been convenient for each object not only to store its current value but also the history of how it got to be that value.

Usage

```
Whistory(...)
```

Arguments

...	Arguments to pass to method

Details

Description says all

Value

No return value, although function was meant to print out a list times and dates when the object was modified.

Author(s)

G P Nason

See Also

Whistory.wst

Whistory.wst — *Obsolete function: as Whistory, but for wst objects*

Description

Obsolete function, see Whistory.

Usage

```
## S3 method for class 'wst'
Whistory(wst, all=FALSE, ...)
```

Arguments

wst	The object that you want to display the history for
all	Print the whole history list
...	Other arguments

Details

Description says all

Value

Nothing, but history information is printed.

Author(s)

G P Nason

See Also

Whistory

---

wp *Wavelet packet transform.*

Description

This function computes a wavelet packet transform (computed by the complete binary application of the DH and DG packet operators, as opposed to the Mallat discrete wavelet transform which only recurses on the DH operator [low pass]).

Usage

```
wp(data, filter.number=10, family="DaubLeAsymm", verbose=FALSE)
```

Arguments

data	A vector containing the data you wish to decompose. The length of this vector must be a power of 2.
filter.number	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
family	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
verbose	if TRUE then (un)helpful messages are printed during the execution.

Details

The paper by Nason, Sapatinas and Sawczenko, 1998 details this implementation of the wavelet packet transform. A more thorough reference is Wickerhauser, 1994.

Value

An object of class wp which contains the (decimated) wavelet packet coefficients.

RELEASE

Version 3.0 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

accessC.wp, accessD.wp, basisplot.wp, draw.wp,drawwp.default, filter.select, getpacket.wp, InvBasis.wp, MaNoVe.wp, plot.wp, print.wp, putC.wp, putD.wp, putpacket.wp, summary.wp, threshold.wp, wp.object.

Examples

```
v <- rnorm(128)
vwp <- wp(v)
```

---

wp.object *Wavelet Packet decomposition objects.*

Description

These are objects of classes wp They represent a decomposition of a function with respect to a set of wavelet packet functions.

Details

To retain your sanity we recommend that wavelet packets be extracted in one of two ways:

- use getpacket.wp to obtain individual packets.
- use accessD.wp to obtain all coefficients at a particular resolution level.

You can obtain the coefficients directly from the wp$wp component but you have to understand their organization described above.

Value

The following components must be included in a legitimate 'wp' object.

wp — a matrix containing the wavelet packet coefficients. Each row of the matrix contains coefficients with respect to a particular resolution level. There are nlevelsWT(wp)+1 rows in the matrix. Row nlevels(wp)+1 (the "bottom") row contains the "original" data used to produce the wavelet packet coefficients. Rows nlevels(wp) to row 1 contain coefficients at resolution levels nlevels(wp)-1 to 0 (so the first row contains coefficients at resolution level 0).

The columns contain the coefficients with respect to packets. A different packet length exists at each resolution level. The packet length at resolution level i is given by 2^i. However, the getpacket.wp function should be used to access individual packets from a wp object.

nlevelsWT	The number of levels in the wavelet packet decomposition. If you raise 2 to the power of nlevels you get the number of data points used in the decomposition.
filter	a list containing the details of the filter that did the decomposition (equivalent to the return value from the filter.select function).
date	The date that the transform was performed or the wp was modified.

GENERATION

This class of objects is returned from the wp function to represent a wavelet packet decomposition of a function. Many other functions return an object of class wp.

METHODS

The wp class of objects has methods for the following generic functions: InvBasis, MaNoVe, accessC, accessD, basisplot, draw. getpacket, nlevelsWT, plot, print, putC, putD, putpacket, summary, threshold.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wp

---

wpst *Non-decimated wavelet packet transform.*

Description

This function computes the non-decimated wavelet packet transform as described by Nason, Sapatinas and Sawczenko, 1998. The non-decimated wavelet packet transform (NWPT) contains all possible shifted versions of the wavelet packet transform.

Usage

```
wpst(data, filter.number=10, family="DaubLeAsymm", FinishLevel)
```

Arguments

`data`	A vector containing the data you wish to decompose. The length of this vector must be a power of 2.
`filter.number`	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
`family`	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
`FinishLevel`	At which level to stop decomposing. The full decomposition decomposes to level 0, but you could stop earlier.

Details

This function computes the packet-ordered non-decimated wavelet packet transform of data as described by Nason, Sapatinas and Sawczenko, 1998. It assumes periodic boundary conditions. The order of computation of the NWPT is $O(n^2)$ if n is the number of input data points.

Packets can be extracted from the `wpst.object` produced by this function using the `getpacket.wpst` function. Whole resolution levels of non-decimated wavelet packet coefficients in time order can be obtained by using the `accessD.wpst` function.

Value

An object of class `wpst` containing the discrete packet-ordered non-decimated wavelet packet coefficients.

RELEASE

Version 3.8.8 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

`accessD`, `accessD.wpst`, `filter.select`, `getpacket`, `getpacket.wpst`, `makewpstDO`

Examples

```
v <- rnorm(128)
vwpst <- wpst(v)
```

`wpst2discr` *Reshape/reformat packet coefficients into a multivariate data set*

Description

The packet coefficients of a nondecimated wavelet packet object are stored internally in an efficient form. This function takes the nondecimated wavelet packets and stores them as a matrix (multivariate data set). Each column in the returned matrix corresponds to an individual packet, each row corresponds to a time index in the original packet or time series.

Usage

```
wpst2discr(wpstobj, groups)
```

Arguments

`wpstobj`	A wpst class object, output from `wpst` say
`groups`	A time series containing the group membership at each time point

Details

Description says it all

Value

An object of class w2d which is a list containing the following items:

`m`	The matrix containing columns of packet information.
`groups`	Passes through the `group` argument from input.
`level`	Each column corresponds to a packet, this vector contains the information on which resolution level each packet comes from
`pktix`	Like for `level` but for packet indices
`nlevelsWT`	The number of resolution levels in total, from the wpst object

Author(s)

G P Nason

See Also

`makewpstDO`, `wpst`

`wpst2m` *Converts a nondecimated wavelet packet object to a (large) matrix with packets stored as columns*

Description

Takes a nondecimated wavelet packet transform, takes the packets one packet at a time and stores them in a matrix. The packets are rotated on extraction and storage in the matrix in an attempt to align them, they are also optionally transformed by `trans`. The rotation is performed by `compgrot`.

Note that the coefficients are of some series, not the basis functions themselves.

Usage

```
wpst2m(wpstobj, trans = identity)
```

Arguments

`wpstobj`	The nondecimated wavelet packet object to store
`trans`	The optional transform to apply to the coefficients

Details

Description says all

Value

A list, of class w2m, with the following components:

`m`	The matrix containing the packets
`level`	A vector containing the levels from where the packets in m come from
`pktix`	A vector containing the packet indices from where the packets in m come from
`nlevelsWT`	The number of resolution levels from the original wpst object

Author(s)

G P Nason

See Also

`makewpstRO`, `print.w2m`

Examples

```
#
# Not intended to be directly used by users
#
```

wpstCLASS *Predict values using new time series values via a non-decimated wavelet packet discrimination object.*

Description

Given a timeseries (`timeseries`) and another time series of categorical values (`groups`) the `makewpstDO` produces a model that permits discrimination of the `groups` series using a discriminant analysis based on a restricted set of non-decimated wavelet packet coefficients of `timeseries`. The current function enables new `timeseries` data, to be used in conjunction with the model to generate new, predicted, values of the `groups` time series.

Usage

```
wpstCLASS(newTS, wpstDO)
```

Arguments

`newTS`	A new segment of time series values, of the same time series that was used as the dependent variable used to construct the wpstDO object
`wpstDO`	An object that uses values of a dependent time series to build a discriminatory model of a groups time series. Output from the `makewpstDO` function

Details

This function performs the same nondecimated wavelet packet (NDWPT) transform of the `newTS` data that was used to analyse the original `timeseries` and the details of this transform are stored within the `wpstDO` object. Then, using information that was recorded in `wpstDO` the packets with the same level/index are extracted from the new NDWPT and formed into a matrix. Then the linear discriminant variables, again stored in `wpstDO` are used to form predictors of the original `groups` time series, ie new values of `groups` that correspond to the new values of `timeseries`.

Value

The prediction using the usual R `predict.lda` function. The predicted values are stored in the `class` component of that list.

Author(s)

G P Nason

See Also

`makewpstDO`

Examples

```
#
# See example at the end of help page for makewpstDO
#
```

---

wpstREGR *Construct data frame using new time series using information from a previously constructed wpstRO object*

Description

The makewpstRO function takes two time series, performs a nondecimated wavelet packet transform with the "dependent" variable one, stores the "best" packets (those that individually correlate with the response series) and returns the data frame that contains the response and the best packets. The idea is that the user then performs some kind of modelling between response and packets. This function takes a new "dependent" series and returns the best packets in a new data frame in the same format as the old one. The idea is that the model and the new data frame can be used together to predict new values for the response

Usage

```
wpstREGR(newTS, wpstRO)
```

Arguments

newTS	The new "dependent" time series
wpstRO	The previously constructed wpstRO object made by makewpstRO

Details

Description says it all

Value

New values of the response time series

Author(s)

G P Nason

References

See reference to Nason and Sapatinas paper in the help for makewpstRO.

See Also

makewpstRO, wpst

Examples

```
#
# See extended example in makewpstRO help, includes example of using this fn
#
```

---

wr *Wavelet reconstruction (inverse DWT).*

Description

Performs inverse discrete wavelet transform.

This function is generic.

Particular methods exist. For the wd class object this generic function uses wr.wd.

Usage

```
wr(...)
```

Arguments

...	See individual help pages for details.

Details

See individual method help pages for operation and examples.

Value

Usually the wavelet reconstruction of x. Although the return value varies with the precise method used.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wd, wd.object, wr.wd

`wr.int` *Computes inverse "wavelets on the interval" transform.*

Description

This function actually computes the inverse of the "wavelets on the interval" transform.

Usage

```
## S3 method for class 'int'
wr(wav.int.object, ...)
```

Arguments

`wav.int.object`	A list with components defined by the return from the `wd.int` function.
`...`	any other arguments

Details

(The WaveThresh implementation of the “wavelets on the interval transform” was coded by Piotr Fryzlewicz, Department of Mathematics, Wroclaw University of Technology, Poland; this code was largely based on code written by Markus Monnerjahn, RHRK, Universitat Kaiserslautern; integration into WaveThresh by `GPN`).

See the help on the "wavelets on the interval code" in the wd help page.

Value

The inverse wavelet transform of the wav.int.object supplied.

RELEASE

Version 3.9.6 (Although Copyright Piotr Fryzlewicz and Markus Monnerjahn 1995-9).

Note

It is not recommended that the casual user call this function. The "wavelets on the interval" transform is best called in `WaveThresh` via the `wd` function with the argument bc argument set to `"interval"`.

Author(s)

Piotr Fryzlewicz and Markus Monnerjahn

See Also

`wd.int`, `wd`, `wr`.

Examples

```
#
# The user is expected to call the wr
# for inverting a "wavelets on the interval transform".
#
```

---

wr.mwd *Multiple wavelet reconstruction for mwd objects*

Description

This function is method for the function to apply the inverse multiple wavelet transform for mwd.object objects.

Usage

```
## S3 method for class 'mwd'
wr(...)
```

Arguments

...	Arguments to the mwr function.

Details

The function is merely a wrapper for mwr

Value

The same return value as for mwr.

Author(s)

Tim Downie

See Also

mwd, mwr

wr.wd — *Wavelet reconstruction for wd class objects (inverse discrete wavelet transform).*

Description

This function performs the reconstruction stage of Mallat's pyramid algorithm (Mallat 1989), i.e. the discrete inverse wavelet transform. The actual transform is performed by some C code, this is dynamically linked into S (if your machine can do this).

Usage

```
## S3 method for class 'wd'
wr(wd, start.level = 0, verbose = FALSE, bc = wd$bc,
 return.object = FALSE, filter.number = wd$filter$filter.number,
 family = wd$filter$family, ...)
```

Arguments

`wd`	A wavelet decomposition object as returned by `wd`, and described in the help for that function and the help for `wd.object`.
`start.level`	The level you wish to start reconstruction at. The is usually the first (level 0). This argument is ignored for a wd object computed using the "wavelets on the interval" transform (i.e. using the `bc="interval"` option of `wd`.
`verbose`	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.
`bc`	The boundary conditions used. Usually these are determined by those used to create the supplied wd object, but you sometimes change them with possibly silly results.
`filter.number`	The filter number of the wavelet used to do the reconstruction. Again, as for bc, you should probably leave this argument alone. Ignored if the bvc component of the `wd` object is `"interval"`.
`family`	The type of wavelet used to do the reconstruction. You can change this argument from the default but it is probably NOT wise. Ignored if the bvc component of the `wd` object is `"interval"`.
`return.object`	If this is F then the top level of the reconstruction is returned (this is the reconstructed function at the highest resolution). Otherwise if it is T the whole wd reconstructed object is returned. Ignored if the `bvc` component of the `wd` object is `"interval"`.
`...`	any other arguments

Details

The code implements Mallat's inverse pyramid algorithm. In the reconstruction the quadrature mirror filters G and H are supplied with c0 and d0, d1, ... d(m- 1) (the wavelet coefficients) and rebuild c1,..., cm.

If the bc component of the wd object is "interval" then the wr.int function which implements the inverse "wavelet on the interval" transform due to Cohen, Daubechies and Vial, 1993 is used instead.

Value

Either a vector containing the top level reconstruction or an object of class wd containing the results of the reconstruction, details to be found in help for wd.object.

RELEASE

Version 3 Copyright Guy Nason 1994 Integration of "wavelets on the interval" code by Piotr Fryzlewicz and Markus Monnerjahn was at Version 3.9.6, 1999.

Author(s)

G P Nason

See Also

wd, wr.int, accessC, accessD, filter.select, plot.wd, threshold

Examples

```
#
# Take the wd object generated in the examples to wd (called wds)
#
# Invert this wd object
#
#yans <- wr(wds)
#
# Compare it to the original, called y
#
#sum((yans-y)^2)
#[1] 9.805676e-017
#
# A very small number
#
```

wr3D *Inverse DWT for 3D DWT object.*

Description

Performs the inverse DWT for wd3D.object, i.e. 3D DWT objects.

Usage

```
wr3D(obj)
```

Arguments

obj	A wd3D.object 3D DWT object as returned by wd3D.

Details

The code implements a 3D version of Mallat's inverse pyramid algorithm.

Value

A 3D array containing the inverse 3D DWT of obj.

RELEASE

Version 3.9.6 Copyright Guy Nason 1997

Author(s)

G P Nason

See Also

wr, accessD.wd3D, print.wd3D, putD.wd3D, putDwd3Dcheck, summary.wd3D, threshold.wd3D, wd3D, wd3D.object.

Examples

```
#
# Now let's take the object generated by the last stage in the EXAMPLES
# section of threshold.wd3D and invert it!
#
#testwr <- wr3D(testwd3DT)
#
# You'll find that testwr is an array of dimension 8x8x8!
#
```

wst *Packet-ordered non-decimated wavelet transform.*

Description

Computes the packet-ordered non-decimated wavelet transform (TI-transform). This algorithm is functionally equivalent to the time-ordered non-decimated wavelet transform (computed by wd with the type="station" argument).

Usage

```
wst(data, filter.number=10, family="DaubLeAsymm", verbose=FALSE)
```

Arguments

data	A vector containing the data you wish to decompose. The length of this vector must be a power of 2.
filter.number	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments. Note: as of version 4.6 you can use the Lina-Mayrand complex-valued compactly supported wavelets.
family	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
verbose	Controls the printing of "informative" messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.

Details

The packet-ordered non-decimated wavelet transform is more properly known as the TI-transform described by Coifman and Donoho, 1995. A description of this implementation can be found in Nason and Silverman, 1995.

The coefficients produced by this transform are exactly the same as those produced by the wd function with the type="station" option *except* in that function the coefficients are *time-ordered*. In the wst function the coefficients are produced by a wavelet packet like algorithm with a *cyclic rotation* step instead of processing with the father wavelet mirror filter at each level.

The coefficients produced by this function are useful in curve estimation problems in conjunction with the thresholding function threshold.wst and either of the inversion functions AvBasis.wst and InvBasis.wst The coefficients produced by the time-ordered non-decimated wavelet transform are more useful for time series applications: e.g. the evolutionary wavelet spectrum computation performed by ewspec. Note that a time-ordered non-decimated wavelet transform object may be converted into a packet-ordered non-decimated wavelet transform object (and vice versa) by using the convert function.

Value

An object of class: wst. The help for the wst describes the intricate structure of this class of object.

RELEASE

Version 3.5.3 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

wst.object, threshold.wst, AvBasis.wst, InvBasis.wst, filter.select, convert, ewspec, plot.wst,

Examples

```
#
# Let's look at the packet-ordered non-decimated wavelet transform
# of the data we used to do the time-ordered non-decimated wavelet
# transform exhibited in the help page for wd.
#
test.data <- example.1()$y
#
# Plot it to see what it looks like (piecewise polynomial)
#
## Not run: ts.plot(test.data)
#
# Now let's do the packet-ordered non-decimated wavelet transform.
#
TDwst <- wst(test.data)
#
# And let's plot it....
#
## Not run: plot(TDwst)
#
# The coefficients in this plot at each resolution level are the same
# as the ones in the non-decimated transform plot in the wd
# help page except they are in a different order. For more information
# about how the ordering works in each case see
# Nason, Sapatinas and Sawczenko, 1998.
#
# Next examples
# ------------
# The chirp signal is also another good examples to use.
#
#
# Generate some test data
#
test.chirp <- simchirp()$y
## Not run: ts.plot(test.chirp, main="Simulated chirp signal")
```

```
#
# Now let's do the packet-ordered non-decimated wavelet transform.
# For a change let's use Daubechies extremal phase wavelet with 6
# vanishing moments (a totally arbitrary choice, please don't read
# anything into it).
#
chirpwst <- wst(test.chirp, filter.number=6, family="DaubExPhase")
## Not run: plot(chirpwst, main="POND WT of Chirp signal")
```

`wst.object` *(Packet ordered) Nondecimated wavelet transform decomposition objects.*

Description

These are objects of class `wst` They represent a decomposition of a function with respect to a set of (all possible) shifted wavelets.

Details

To retain your sanity we recommend that the coefficients from a `wst` object be extracted in one of two ways:

- use `getpacket.wst` to obtain individual packets of either father or mother wavelet coefficients.
- use `accessD.wst` to obtain all mother coefficients at a particular resolution level.
- use `accessC.wst` to obtain all father coefficients at a particular resolution level.

You can obtain the coefficients directly from the `wst$wp` component (mother) or `wst$Carray` component (father) but you have to understand their organization described above.

Value

The following components must be included in a legitimate 'wst' object.

`wp` a matrix containing the packet ordered non-decimated wavelet coefficients. Each row of the matrix contains coefficients with respect to a particular resolution level. There are `nlevelsWT(wst)+1` rows in the matrix. Row `nlevels(wst)+1` (the "bottom") row contains the "original" data used to produce the wavelet packet coefficients. Rows `nlevels(wst)` to row 1 contain coefficients at resolution levels `nlevels(wst)-1` to 0 (so the first row contains coefficients at resolution level 0).

The columns contain the coefficients with respect to packets. A different packet length exists at each resolution level. The packet length at resolution level `i` is given by `2^i`. However, the `getpacket.wst` function should be used to access individual packets from a `wst` object.

`Carray` A matrix of the same dimensions and format as `wp` but containing the father wavelet coefficients.

nlevelsWT	The number of levels in the decomposition. If you raise 2 to the power of nlevels you get the number of data points used in the decomposition.
filter	a list containing the details of the filter that did the decomposition (equivalent to the return value from the filter.select function).
date	The date that the transform was performed or the wst was modified.

GENERATION

This class of objects is returned from the wst function which computes the *packets-ordered* non-decimated wavelet transform (effectively all possible shifts of the standard discrete wavelet transform).

Many other functions return an object of class wst.

METHODS

The wst class of objects has methods for the following generic functions: AvBasis, InvBasis, LocalSpec, MaNoVe, accessC, accessD, convert, draw. getpacket. image. nlevelsWT, nullevels, plot, print, putC, putD, putpacket, summary, threshold.

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wst

---

wst2D *(Packet-ordered) 2D non-decimated wavelet transform.*

Description

This function computes the (packet-ordered) 2D non-decimated wavelet transform

Usage

```
wst2D(m, filter.number=10, family="DaubLeAsymm")
```

Arguments

`m`	A matrix containing the image data that you wish to decompose. Each dimension of the matrix must be the same power of 2.
`filter.number`	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
`family`	specifies the family of wavelets that you want to use. Two popular options are "DaubExPhase" and "DaubLeAsymm" but see the help for `filter.select` for more possibilities.

Details

The `wst2D` computes the (packet-ordered) 2D non-decimated discrete wavelet transform. Such a transform may be used in wavelet shrinkage of images using the `AvBasis.wst2D` function to perform an "average-basis" inverse. Such a transform was used to denoise images in the paper by Lang, Guo, Odegard, Burrus and Wells, 1995.

The algorithm works by mixing the HH, GH, HG and GG image operators of the 2D (decimated) discrete wavelet transform (see Mallat, 1989 and the implementation in WaveThresh called `imwd`) with the shift operator S (as documented in Nason and Silverman, 1995) to form new operators (as given in the help to `getpacket.wst2D`).

Subimages can be obtained and replaced using the `getpacket.wst2D` and `putpacket.wst2D` functions.

This function is a 2D analogue of the (packet-ordered) non-decimated discrete wavelet transform implemented in WaveThresh as `wst`.

Value

An object of class `wst2D`.

RELEASE

Version 3.9.5 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

`AvBasis.wst2D`, `getpacket.wst2D`, `imwd`, `plot.wst2D`, `print.wst2D`, `putpacket.wst2D`, `summary.wst2D`, `wst2D.object`.

Examples

```
#
# We shall use the lennon image.
#
data(lennon)
```

```
#
#
# Now let's apply the (packet-ordered) 2D non-decimated DWT to it...
# (using the default wavelets)
#
uawst2D <- wst2D(lennon)
#
# One can use the function plot.wst2D to get
# a picture of all the resolution levels. However, let's just look at them
# one at a time.
#
# How many levels does our uawst2D object have?
#
nlevelsWT(uawst2D)
#[1] 8
#
# O.k. Let's look at resolution level 7
#
## Not run: image(uawst2D$wst2D[8,,])
#
#
# There are four main blocks here (each of 256x256 pixels) which themselves
# contain four sub-blocks. The primary blocks correspond to the no shift,
# horizontal shift, vertical shift and "horizontal and vertical" shifts
# generated by the shift S operator. Within each of the 256x256 blocks
# we have the "usual" Mallat smooth, horizontal, vertical and diagonal
# detail, with the smooth in the top left of each block.
#
# Let's extract the smooth, with no shifts at level 7 and display it
#
## Not run: image(getpacket(uawst2D, level=7, index=0, type="S"))
#
#
# Now if we go two more resolution levels deeper we have now 64x64 blocks
# which contain 32x32 subblocks corresponding to the smooth, horizontal,
# vertical and diagonal detail.
#
#
# Groovy eh?
```

`wst2D.object` *(Packet ordered) Two-dimensional nondecimated wavelet transform decomposition objects.*

Description

These are objects of class `wst2D` They represent a decomposition of a function with respect to a set of (all possible) shifted two-dimensional wavelets. They are a 2D extension of the `wst.object`.

Details

To retain your sanity we recommend that the coefficients from a `wst2D` object be extracted or replaced using

- `getpacket.wst2D` to obtain individual packets of either father or mother wavelet coefficients.
- `putpacket.wst2D` to insert coefficients.

You can obtain the coefficients directly from the `wst2D$wst2D` component but you have to understand their organization described above.

Value

The following components must be included in a legitimate `wst2D` object.

`wst2D` This a three-dimensional array. Suppose that the original image that created the `wst2D` object is n x n. Then the dimension of the `wst2D` array is [nlevelsWT, 2n, 2n]. The first index of the array refers to the resolution level of the array with "resolution level = index - 1" (so, e.g. the coarsest scale detailed is stored at index 1 and the finest at index nlevels). For a given resolution level (selected first index) the associated 2n x 2n matrix contains the two-dimensional non-decimated wavelet coefficients for that level packed as follows.

At the finest resolution level the 2n x 2n coefficient image may be broken up into four n x n subimages. Each of the four images corresponds to data shifts in the horizontal and vertical directions. The top left image corresponds to "no shift" and indeed the top left image corresponds to the coefficients obtained using the decimated 2D wavelet transform as obtained using the `imwd` function. The top right image corresponds to a horizontal data shift; the bottom left to a vertical data shift and the bottom right corresponds to both horizontal and vertical data shift.

Within each of the four n x n images named in the previous paragraph are again 4 subimages each of dimension n/2 x n/2. These correspond to (starting at the top left and moving clockwise) the smooth (CC), horizontal detail (DC), diagonal detail (DD) and vertical detail (CD).

At coarser resolution levels the coefficients are smaller submatrices corresponding to various levels of data shifts and types of detail (smooth, horizontal, vertical, diagonal).

We strongly recommend the use of the `getpacket.wst2D` and `putpacket.wst2D` functions to remove and replace coefficients from `wst2D`

objects.

`nlevelsWT` The number of levels in the decomposition. If you raise 2 to the power of 2 `nlevels` you get the number of data points used in the decomposition.

`filter` a list containing the details of the filter that did the decomposition (equivalent to the return value from the `filter.select` function).

`date` The date that the transform was performed or the `wst2D` was modified.

GENERATION

This class of objects is returned from the wst2D function which computes the *packets-ordered* two-dimensional non-decimated wavelet transform (effectively all possible shifts of the standard two-dimensional discrete wavelet transform).

Many other functions return an object of class wst2D.

METHODS

The wst2D class of objects has methods for the following generic functions: AvBasis, getpacket. plot, print, putpacket, summary,

RELEASE

Version 3.5.3 Copyright Guy Nason 1994

Author(s)

G P Nason

See Also

wst2D

---

wstCV *Performs two-fold cross-validation estimation using packet-ordered non-decimated wavelet transforms and one, global, threshold.*

Description

Performs Nason's 1996 two-fold cross-validation estimation using packet-ordered non-decimated wavelet transforms and one, global, threshold.

Usage

```
wstCV(ndata, ll = 3, type = "soft", filter.number = 10, family =
 "DaubLeAsymm", tol = 0.01, verbose = 0, plot.it = FALSE, norm =
 l2norm, InverseType = "average", uvdev = madmad)
```

Arguments

ndata	the noisy data. This is a vector containing the signal plus noise. The length of this vector should be a power of two.
ll	the primary resolution for this estimation. Note that the primary resolution is *problem-specific*: you have to find out which is the best value.
type	whether to use hard or soft thresholding. See the explanation for this argument in the threshold.wst function.

`filter.number` This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.

`family` specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".

`tol` the cross-validation tolerance which decides when an estimate is sufficiently close to the truth (or estimated to be so).

`verbose` If `TRUE` then informative messages are printed during the progression of the function, otherwise they are not.

`plot.it` If `TRUE` then a plot of the progress of optimising the error estimate for different values of the threshold is generated as the algorithm proceeds. The algorithm tries to minimize the error estimate so you should see a "bowl" developing. After each iteration the error estimate is plotted with the iteration number so you should see the numbers tend to the bottom of the bowl.

`norm` which measure of distance to judge the dissimilarity between the estimates. The functions `l2norm` and `linfnorm` are suitable examples.

`InverseType` The possible options are "average" or "minent". The former uses basis averaging to form estimates of the unknown function. The "minent" function selects a basis using the Coifman and Wickerhauser, 1992 algorithm to select a basis to invert.

`uvdev` Universal thresholding is used to generate an upper bound for the ideal threshold. This argument provides the function that computes an estimate of the variance of the noise for use with the universal threshold calculation (see `threshold.wst`).

Details

This function implements the cross-validation method detailed by Nason, 1996 for computing an estimate of the error between an estimate and the "truth". The difference here is that it uses the `packet ordered non-decimated wavelet transform` rather than the standard Mallat `wd` discrete wavelet transform. As such it is an examples of the translation-invariant denoising of Coifman and Donoho, 1995 but uses cross-validation to choose the threshold rather than SUREshrink.

Note that the procedure outlined above can use `AvBasis` basis averaging or basis selection and inversion using the Coifman and Wickerhauser, 1992 best-basis algorithm

Value

A list returning the results of the cross-validation algorithm. The list includes the following components:

`ndata` a copy of the input noisy data

`xvwr` a reconstruction of the best estimate computed using this algorithm. It is the inverse (computed depending on what the InverseType argument was) of the `xvwrWSTt` component.

`xvwrWSTt` a thresholded version of the packet-ordered non-decimated wavelet transform of the noisy data using the best threshold discovered by this cross-validation algorithm.

uvt	the universal threshold used as the upper bound for the algorithm that tries to discover the optimal cross-validation threshold. The lower bound is always zero.
xvthresh	the best threshold as discovered by cross-validation. Note that this is one number, the global threshold. The wstCVl function should be used to compute a level-dependent threshold.
xkeep	a vector containing the various thresholds used by the optimisation algorithm in trying to determine the best one. The length of this vector cannot be predetermined but depends on the noisy data, thresholding method, and optimisation tolerance.
fkeep	a vector containing the value of the estimated error used by the optimisation algorithm in trying to minimize the estimated error. The length, like that of xkeep cannot be predetermined for the same reasons.

RELEASE

Version 3.6 Copyright Guy Nason 1995

Note

If plot.it is TRUE then a plot indicating the progression of the optimisation algorithm is plotted.

Author(s)

G P Nason

See Also

GetRSSWST, linfnorm, linfnorm, threshold.wst, wst, wst.object, wstCVl.

Examples

```
#
# Example PENDING
#
```

---

wstCVl — *Performs two-fold cross-validation estimation using packet-ordered non-decimated wavelet transforms and a (vector) level-dependent threshold.*

Description

Performs Nason's 1996 two-fold cross-validation estimation using packet-ordered non-decimated wavelet transforms and a (vector) level-dependent threshold.

Usage

```
wstCVl(ndata, ll = 3, type = "soft", filter.number = 10, family = "DaubLeAsymm",
 tol = 0.01, verbose = 0, plot.it = FALSE, norm = l2norm, InverseType = "average",
 uvdev = madmad)
```

Arguments

`ndata`	the noisy data. This is a vector containing the signal plus noise. The length of this vector should be a power of two.
`ll`	the primary resolution for this estimation. Note that the primary resolution is *problem-specific*: you have to find out which is the best value.
`type`	whether to use hard or soft thresholding. See the explanation for this argument in the `threshold.wst` function.
`filter.number`	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
`family`	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
`tol`	the cross-validation tolerance which decides when an estimate is sufficiently close to the truth (or estimated to be so).
`verbose`	If `TRUE` then informative messages are printed during the progression of the function, otherwise they are not.
`plot.it`	Whether or not to produce a plot indicating progress.
`norm`	which measure of distance to judge the dissimilarity between the estimates. The functions `l2norm` and `linfnorm` are suitable examples.
`InverseType`	The possible options are "average" or "minent". The former uses basis averaging to form estimates of the unknown function. The "minent" function selects a basis using the Coifman and Wickerhauser, 1992 algorithm to select a basis to invert.
`uvdev`	Universal thresholding is used to generate an upper bound for the ideal threshold. This argument provides the function that computes an estimate of the variance of the noise for use with the universal threshold calculation (see `threshold.wst`).

Details

This function implements a modified version of the cross-validation method detailed by Nason, 1996 for computing an estimate of the error between an estimate and the "truth". The difference here is that it uses the packet ordered non-decimated wavelet transform rather than the standard Mallat wd discrete wavelet transform. As such it is an examples of the translation-invariant denoising of Coifman and Donoho, 1995 but uses cross-validation to choose the threshold rather than SUREshrink.

Further, this function computes level-dependent thresholds. That is, it can compute a different threshold for each resolution level.

Note that the procedure outlined above can use `AvBasis` basis averaging or basis selection and inversion using the Coifman and Wickerhauser, 1992 best-basis algorithm

Value

A list returning the results of the cross-validation algorithm. The list includes the following components:

- `ndata` — a copy of the input noisy data
- `xvwr` — a reconstruction of the best estimate computed using this algorithm. It is the inverse (computed depending on what the InverseType argument was) of the `xvwrWSTt` component.
- `xvwrWSTt` — a thresholded version of the packet-ordered non-decimated wavelet transform of the noisy data using the best threshold discovered by this cross-validation algorithm.
- `uvt` — the universal threshold used as the upper bound for the algorithm that tries to discover the optimal cross-validation threshold. The lower bound is always zero.
- `xvthresh` — the best threshold as discovered by cross-validation. Note that this is vector, a level-dependent threshold with one threshold value for each resolution level. The first entry corresponds to level `ll`, the last entry corresponds to level `nlevelsWT(ndata)-1` and the entries in between linearly to the levels in between. The `wstCV` function should be used to compute a global threshold.
- `optres` — The results from performing the optimisation using the `nlminb` function from Splus. This object contains many interesting components with information about how the optimisation went. See the `nlminb` help page for information.

RELEASE

Version 3.6 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

`GetRSSWST`, `linfnorm`, `linfnorm`, `threshold.wst`, `wst`, `wst.object`, `wstCV`

Examples

```
#
# Example PENDING
#
```

WTEnv *Environment that exists to store intermediate calculations for re-use within the same R session.*

Description

Environment that stores results of long calculations so that they can be made available for immediate reuse.

Details

This environment is created on package load by wavethresh. The results of some intermediate calculations get stored in here (notably by PsiJ, PsiJmat and ipndacw). The reason for this is that the calculations are typically lengthy and it saves wavethresh time to search the WTEnv for pre-computed results. For example, ipndacw computes matrices of various orders. Matrices of low order form the upper-left corner of matrices of higher order so higher order matrix calculations can make use of the lower order instances.

A similar functionality was present in wavethresh in versions 4.6.1 and prior to this. In previous versions computations were saved in the users current data directory. However, the user was never notified about this nor permission sought.

The environment WTEnv disappears when the package disappears and the R session stops - and results of all intermediate calculations disappear too. This might not matter if you never use the larger objects (as it will not take much time to recompute).

RELEASE

Version 3.9 Copyright Guy Nason 1998

Author(s)

G P Nason

See Also

ipndacw, PsiJ, PsiJmat

Examples

```
#
# See what it is
#
WTEnv
#<environment: 0x102fc3830>
#
# Compute something that uses the environment
#
fred <- PsiJ(-5)
#
```

```
# Now let's see what got put in
#
ls(envir=WTEnv)
#[1] "Psi.5.10.DaubLeAsymm"
```

---

wvcvlrss *Computes estimate of error for function estimate.*

Description

This function is merely a call to the GetRSSWST function.

Usage

```
wvcvlrss(threshold, ndata, levels, type, filter.number, family, norm,
 verbose, InverseType)
```

Arguments

threshold	the value of the threshold that you wish to compute the error of the estimate at
ndata	the noisy data. This is a vector containing the signal plus noise. The length of this vector should be a power of two.
levels	the levels over which you wish the threshold value to be computed (the threshold that is used in computing the estimate and error in the estimate). See the explanation for this argument in the threshold.wst function.
type	whether to use hard or soft thresholding. See the explanation for this argument in the threshold.wst function.
filter.number	This selects the smoothness of wavelet that you want to use in the decomposition. By default this is 10, the Daubechies least-asymmetric orthonormal compactly supported wavelet with 10 vanishing moments.
family	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm".
norm	which measure of distance to judge the dissimilarity between the estimates. The functions l2norm and linfnorm are suitable examples.
verbose	If TRUE then informative messages are printed during the progression of the function, otherwise they are not.
InverseType	The possible options are "average" or "minent". The former uses basis averaging to form estimates of the unknown function. The "minent" function selects a basis using the Coifman and Wickerhauser, 1992 algorithm to select a basis to invert.

Details

This function is merely a call to the GetRSSWST function with a few arguments interchanged. In particular, the first two arguments are interchanged. This is to make life easier for use with the nlminb function which expects the first argument of the function it is trying to optimise to be the variable that the function is optimised over.

Value

A real number which is estimate of the error between estimate and truth at the given threshold.

RELEASE

Version 3.6 Copyright Guy Nason 1995

Author(s)

G P Nason

See Also

GetRSSWST.

Examples

```
#
# This function performs the error estimation step for the
# wstCVl function and so is not intended for
# user use.
#
```

---

wvmoments *Compute moments of wavelets or scaling function*

Description

Numerically compute moments of wavelets or scaling function

Usage

```
wvmoments(filter.number = 10, family = "DaubLeAsymm", moment = 0,
    scaling.function = FALSE)
```

Arguments

filter.number	The smoothness of wavelet or scaling function to compute moments for, see filter.select
family	The wavelet family to use, see filter.select
moment	The moment to compute
scaling.function	If FALSE then a wavelet is used in the moment calculation, alternatively if TRUE the associated scaling function is used.

Details

Given a wavelet $\psi(x)$ this function computes the mth moment $\int x^m \psi(x)\, dx$.

Note that for low order moments the integration function often fails for the usual numerical reasons (this never happened in S!). It might be that fiddling with the tolerances will improve this situation.

Value

An object of class `integrate` containing the integral and other pieces of interesting information about the moments calculation.

Author(s)

G P Nason

See Also

`draw.default`

Examples

```
wvmoments(filter.number=5, family="DaubExPhase", moment=5)
#-1.317600 with absolute error < 7.5e-05
```

---

wvrelease — *Prints out the release number of the WaveThresh package*

Description

PRints out the release number of the WaveThresh package, and some copyright info.

Usage

```
wvrelease()
```

Arguments

None.

Details

Description says all

Value

Nothing

Author(s)

G P Nason

Examples

```
wvrelease()
```

Index

∗ **algebra**
- ipndacw, 156
- l2norm, 165
- linfnorm, 168
- numtonv, 217
- WTEnv, 394

∗ **arith**
- IsPowerOfTwo, 164
- madmad, 179
- nlevelsWT, 210
- nlevelsWT.default, 211

∗ **array**
- GenW, 119
- getarrvec, 121

∗ **character**
- Psiname, 265

∗ **classes**
- imwd.object, 144
- imwdc.object, 145
- mwd.object, 206
- nv.object, 219
- wd.object, 362
- wd3D.object, 365
- wp.object, 369
- wst.object, 384
- wst2D.object, 387

∗ **datagen**
- mfirst.last, 199

∗ **datasets**
- BabyECG, 42
- BabySS, 44
- ipd, 155
- lennon, 166
- teddy, 316

∗ **dplot**
- addpkt, 35
- drawbox, 103
- drawwp.default, 104
- makegrid, 182
- plotpkt, 237
- ScalingFunction, 301

∗ **error**
- ConvertMessage, 72
- IsEarly, 162
- IsEarly.default, 163
- putC.wp, 271

∗ **hplot**
- basisplot, 46
- basisplot.BP, 47
- basisplot.wp, 48
- draw, 91
- draw.default, 92
- draw.imwd, 95
- draw.imwdc, 96
- draw.mwd, 98
- draw.wd, 99
- draw.wp, 100
- draw.wst, 102
- image.wd, 140
- image.wst, 141
- plot.imwd, 220
- plot.irregwd, 221
- plot.mwd, 223
- plot.nvwp, 225
- plot.wd, 226
- plot.wp, 229
- plot.wst, 232
- plot.wst2D, 234
- plotdenwd, 235
- wavegrow, 352

∗ **iplot**
- wavegrow, 352

∗ **manip**
- accessC, 19
- accessc, 18
- accessC.mwd, 20
- accessC.wd, 21
- accessC.wp, 23

accessC.wst, 24
accessD, 25
accessD.mwd, 26
accessD.wd, 27
accessD.wd3D, 29
accessD.wp, 31
accessD.wpst, 32
accessD.wst, 34
av.basis, 36
AvBasis, 37
AvBasis.wst, 38
AvBasis.wst2D, 40
bestm, 52
cns, 58
compare.filters, 59
compgrot, 60
compress, 62
compress.default, 63
compress.imwd, 64
convert, 67
convert.wd, 68
convert.wst, 70
cthresh, 74
ewspec, 105
find.parameters, 112
first.last, 113
first.last.dh, 116
getpacket, 123
getpacket.wp, 124
getpacket.wpst, 126
getpacket.wst, 128
getpacket.wst2D, 130
GetRSSWST, 133
griddata objects, 134
HaarMA, 138
levarr, 167
LSWsim, 175
lt.to.name, 177
make.dwwt, 181
mfilter.select, 197
modernise, 201
modernise.wd, 202
mwr, 208
nullevels, 212
nullevels.imwd, 213
nullevels.wd, 214
nullevels.wst, 216
print.wd3D, 249
print.wp, 250
PsiJ, 259
PsiJmat, 262
putC, 266
putC.mwd, 267
putC.wd, 269
putC.wst, 272
putD, 274
putD.mwd, 275
putD.wd, 277
putD.wd3D, 279
putD.wp, 280
putD.wst, 282
putDwd3Dcheck, 283
putpacket, 284
putpacket.wp, 285
putpacket.wst, 287
putpacket.wst2D, 289
rm.det, 294
rmget, 295
rmname, 297
Shannon.entropy, 302
simchirp, 303
test.dataCT, 317
threshold, 318
threshold.irregwd, 324
uncompress, 349
uncompress.default, 350
uncompress.imwdc, 351
wpst2m, 373
wr, 376
wvcvlrss, 395

∗ **math**

AutoBasis, 36
conbar, 65
guyrot, 136
logabs, 174
mpostfilter, 202
mprefilter, 203
mwd, 204
newsure, 209
rfft, 292
rfftinv, 293
rfftwt, 294
rotateback, 298
ssq, 304
support, 314
sure, 315

tpwd, 347
tpwr, 348
wavethresh-package, 7
wd, 355
wd.dh, 360
wp, 368
wpst, 370
wr.mwd, 378
wst, 382
wst2D, 385
wstCV, 389
wstCVl, 391
wvmoments, 396

∗ **methods**
LocalSpec, 169

∗ **misc**
c2to4, 54
CanUseMoreThanOneColor, 55
IsEarly.wd, 163
wvrelease, 397

∗ **models**
dof, 89

∗ **multivariate**
Best1DCols, 51
BMdiscr, 53
makewpstDO, 184
makewpstRO, 187
rcov, 291
wpst2discr, 372
wpstCLASS, 374
wpstREGR, 375

∗ **nonlinear**
imwr, 148
imwr.imwd, 149
imwr.imwdc, 150
summary.mwd, 307
threshold.imwd, 319
threshold.imwdc, 323
threshold.mwd, 326
threshold.wd, 329
threshold.wd3D, 334
threshold.wp, 337
threshold.wst, 340
wd, 355
wd.int, 361
wd3D, 364
wp, 368
wpst, 370
wr.int, 377
wr.wd, 379
wr3D, 381
wst, 382
wst.object, 384
wst2D, 385
wstCV, 389
wstCVl, 391

∗ **nonparametric**
DJ.EX, 88
doppler, 90
example.1, 108

∗ **print**
print.BP, 238
print.nv, 243
print.nvwp, 244
print.w2d, 246
print.w2m, 247
print.wd, 248
print.wpst, 252
print.wpstCL, 253
print.wpstDO, 254
print.wpstRO, 256
print.wst, 257
summary.imwd, 305
summary.imwdc, 306
summary.wd, 308
summary.wd3D, 309
summary.wp, 310
summary.wpst, 311
summary.wst, 312
summary.wst2D, 313

∗ **robust**
rcov, 291

∗ **smooth**
BAYES.THR, 49
Chires5, 56
Chires6, 57
Crsswav, 73
Cthreshold, 76
CWavDE, 77
CWCV, 79
dclaw, 81
dencvwd, 82
denplot, 83
denproj, 84
denwd, 86
denwr, 87

FullWaveletCV, 118
HaarConcat, 137
imwd, 142
imwd.object, 144
imwdc.object, 145
imwr, 148
imwr.imwd, 149
imwr.imwdc, 150
InvBasis, 152
InvBasis.wp, 153
InvBasis.wst, 154
irregwd, 159
irregwd.objects, 161
LocalSpec.wd, 170
MaNoVe, 192
MaNoVe.wp, 193
MaNoVe.wst, 194
plot.wd, 226
rsswav, 299
summary.mwd, 307
threshold.imwd, 319
threshold.imwdc, 323
threshold.mwd, 326
threshold.wd, 329
threshold.wd3D, 334
threshold.wp, 337
threshold.wst, 340
TOgetthrda1, 344
TOthreshda1, 345
TOthreshda2, 346
WaveletCV, 354
wd, 355
wd.dh, 360
wd.int, 361
wd.object, 362
wd3D, 364
wp, 368
wp.object, 369
wpst, 370
wr.int, 377
wr.wd, 379
wr3D, 381
wst, 382
wst.object, 384
wst2D, 385
wst2D.object, 387
wstCV, 389
wstCVl, 391

∗ **ts**
Best1DCols, 51
BMdiscr, 53
checkmyews, 55
LocalSpec.wst, 173
makewpstDO, 184
makewpstRO, 187
wpst2discr, 372
wpstCLASS, 374
wpstREGR, 375

∗ **utilities**
compress, 62
filter.select, 109
firstdot, 117
print.imwd, 238
print.imwdc, 240
print.mwd, 241
print.wst2D, 258
uncompress, 349
Whistory, 366
Whistory.wst, 367

accessC, *18*, 19, *22*, *25–27*, *111*, *115*, *124*, *160*, *161*, *178*, *206*, *207*, *267*, *270*, *271*, *358*, *359*, *362*, *370*, *380*, *385*
accessc, 18, *160*, *161*, *182*, *326*
accessC.mwd, 20, *99*, *198*, *200*, *205*, *207*, *209*, *225*, *242*, *268*, *276*, *307*, *328*
accessC.wd, *19*, *20*, 21, *270*
accessC.wp, *19*, *20*, 23, *369*
accessC.wst, *19*, *20*, 24, *273*, *282*, *384*
accessD, *18*, *20*, *23*, 25, *28–30*, *32*, *33*, *35*, *111*, *115*, *124*, *160*, *161*, *178*, *206*, *207*, *274*, *277*, *278*, *280*, *281*, *284*, *358*, *359*, *362*, *365*, *366*, *370*, *371*, *380*, *385*
accessD.mwd, *21*, 26, *27*, *99*, *198*, *200*, *205*, *207*, *209*, *225*, *242*, *268*, *276*, *307*, *328*
accessD.wd, *23*, *25*, *26*, 27, *278*
accessD.wd3D, *25*, 29, *250*, *280*, *284*, *337*, *365*, *366*, *381*
accessD.wp, *25*, *26*, 31, *281*, *369*
accessD.wpst, *25*, 32, *54*, *127*, *371*
accessD.wst, *25*, *26*, 34, *273*, *282*, *283*, *384*
addpkt, 35, *48*, *104*, *237*
AutoBasis, 36
av.basis, 36, *39*, *66*, *299*

AvBasis, 37, *37*, *134*, *175, 176*, *342, 343*, *385*, *389, 390*, *392*
AvBasis.wst, *36, 37*, 38, *38*, *299*, *343*, *382, 383*
AvBasis.wst2D, 40, *386*

BabyECG, 42
BabySS, 44
basisplot, *35*, 46, *237*, *370*
basisplot.BP, *35*, *46*, 47, *185*, *237*
basisplot.wp, *35*, *46*, 48, *125*, *237*, *369*
BAYES.THR, 49
Best1DCols, *47*, 51, *53*, *184, 185*, *238*
bestm, 52
BMdiscr, 53, *184, 185*

c2to4, 54
CanUseMoreThanOneColor, 55
checkmyews, 55
Chires5, 56, *57*, *86*
Chires6, 57, *57, 58*, *86*
cns, *56*, 58, *175, 176*
compare.filters, 59, *111*
compgrot, 60, *373*
compress, 62, *63–65*, *145*
compress.default, *62*, 63, *65*, *350*
compress.imwd, *62*, 64, *65*, *151*, *351*
conbar, *37*, 65
convert, 67, *69*, *71*, *122*, *175, 176*, *357*, *382, 383*, *385*
convert.wd, *67*, 68, *68*, *121, 122*, *167*
convert.wst, *67, 68*, 70, *121, 122*, *167*
ConvertMessage, 72, *162*
Crsswav, 73, *299, 300*, *355*
cthresh, 74, *113*, *181*
Cthreshold, 76
CWavDE, 77, *314, 315*
CWCV, *77*, 79, *118, 119*, *354, 355*

dclaw, 81
dencvwd, 82, *116*
denplot, 83
denproj, *56–58*, *82, 83*, 84, *84*, *86*, *360, 361*
denwd, 86, *86*, *236*, *360, 361*
denwr, *86*, 87, *360*
DJ.EX, 88, *91*, *109*
dof, 89
doppler, *88, 89*, 90
draw, 91, *111*, *145*, *147*, *207*, *301*, *363*, *370*, *385*
draw.default, *78*, *91*, 92, *92*, *95–97*, *99, 100*, *102, 103*, *315*, *397*
draw.imwd, *91, 92*, 95
draw.imwdc, *91, 92*, 96
draw.mwd, *21*, *27*, 98, *198*, *200*, *205*, *207*, *209*, *225*, *242*, *268*, *276*, *307*, *328*
draw.wd, *91, 92*, 99
draw.wp, *91, 92*, 100, *105*, *125*, *369*
draw.wst, *91, 92*, 102
drawbox, 103
drawwp.default, *101*, 104, *369*

ewspec, *17*, *42*, *44*, *56*, *59*, 105, *138, 139*, *158*, *169, 170*, *172–174*, *176*, *260*, *298*, *382, 383*
example.1, *88, 89*, 108

family, *56, 57*, *171*, *296*
filter.select, *23*, *29*, *37*, *49*, *56, 57*, *59, 60*, *66*, *74*, *76*, *78*, *94–97*, *99–104*, *106*, *108*, 109, *115*, *143, 144*, *146*, *149*, *158–160*, *184*, *187*, *197*, *301*, *314, 315*, *347*, *352*, *356*, *359, 360*, *364*, *369–371*, *380*, *383*, *385, 386*, *388*, *396*
find.parameters, *76*, 112
first.last, 113, *116*, *144–146*, *161*, *270, 271*, *277, 278*, *362, 363*
first.last.dh, 116
firstdot, 117, *260*, *296*
FullWaveletCV, 118
function, *378*

GenW, 119
getarrvec, *68–71*, 121, *167*
getpacket, *19*, *23–25*, *31, 32*, *37*, 123, *281*, *370, 371*, *385*, *389*
getpacket.wp, 124, *124*, *281*, *286*, *369*
getpacket.wpst, *33*, *124*, 126, *371*
getpacket.wst, *24, 25*, *34, 35*, *124*, 128, *273*, *282, 283*, *287, 288*, *384*
getpacket.wst2D, 130, *235*, *290*, *386*, *388*
GetRSSWST, 133, *391*, *393*, *395, 396*
griddata objects, 134
guyrot, 136

HaarConcat, 137, *139*

HaarMA, *137*, 138, *138*

image, *385*
image.wd, 140, *174*
image.wst, 141, *174*
imwd, *17*, *62–65*, *82, 83*, *95–97*, *111*, *115*, *131*, 142, *145*, *147–151*, *178*, *213, 214*, *221*, *239*, *290*, *305, 306*, *319–323*, *347–349*, *351*, *365*, *386*, *388*
imwd.object, *62*, *65*, *92*, *95, 96*, *143*, 144, *146, 147*, *149–151*, *178*, *213, 214*, *220, 221*, *238, 239*, *319*, *322, 323*, *349*, *351*
imwdc.object, *62*, *65*, *92*, *96, 97*, 145, *151*, *240*, *319*, *322, 323*, *349*, *351*
imwr, *111*, *145*, *147*, 148, *149–151*, *348*
imwr.imwd, *148*, 149, *149*, *151*
imwr.imwdc, *148, 149*, 150
InvBasis, *39*, *48*, *105*, 152, *153*, *193, 194*, *217, 218*, *220*, *340*, *342, 343*, *370*, *385*
InvBasis.wp, *66*, *125*, *152*, 153, *245*, *369*
InvBasis.wst, *152*, 154, *243*, *343*, *382, 383*
ipd, 155
ipndacw, *108*, 156, *227*, *259, 260*, *295–298*, *394*
irregwd, *18, 19*, *135*, 159, *160–162*, *182*, *222*, *324–326*
irregwd.objects, *18, 19*, 161, *221, 222*
IsEarly, *72*, 162, *162–164*, *201, 202*
IsEarly.default, *72*, *162*, 163
IsEarly.wd, *162*, 163
IsPowerOfTwo, 164, *211*

l2norm, *133*, 165, *168*, *341*, *390*, *392*, *395*
lennon, 166
levarr, *69*, *71*, *121, 122*, 167
linfnorm, *133, 134*, *166*, 168, *341*, *390–393*, *395*
LocalSpec, *106–108*, 169, *385*
LocalSpec.wd, *169*, 170, *170*, *173, 174*, *292*
LocalSpec.wst, 173
logabs, *140, 141*, 174
LSWsim, *56*, *59*, 175
lt.to.name, 177

madmad, *171*, 179, *320*, *324*, *330*, *335*, *338*, *341*
make.dwwt, *76*, *112*, 181
makegrid, *19*, *135*, *159, 160*, 182, *222*, *325, 326*
makewpstDO, *47*, *52, 53*, 184, *187, 188*, *254, 255*, *371, 372*, *374*
makewpstRO, *53*, 187, *247*, *256*, *373*, *375*
MaNoVe, *36*, *39*, *152–154*, 192, *193, 194*, *217, 218*, *231*, *233*, *340*, *342, 343*, *370*, *385*
MaNoVe.wp, *48*, *125*, *153*, *192*, 193, *194*, *225, 226*, *245*, *369*
MaNoVe.wst, *154*, *192, 193*, 194, *219, 220*, *302*, *343*
mfilter.select, *21*, *27*, *99*, 197, *205*, *207*, *209*, *225*, *242*, *268*, *276*, *307*, *328*
mfirst.last, *21*, *27*, *99*, *198*, 199, *205*, *207*, *209*, *225*, *242*, *268*, *276*, *307*, *328*
modernise, 201
modernise.wd, *201*, 202
mpostfilter, 202, *204*
mprefilter, 203, *203*
mwd, *20, 21*, *26, 27*, *98, 99*, *197, 198*, *200*, *203*, 204, *204, 205*, *207–209*, *223–225*, *241, 242*, *267, 268*, *275, 276*, *307*, *327, 328*, *378*
mwd.object, *20, 21*, *26, 27*, *98, 99*, *198*, *200*, *205*, 206, *207–209*, *225*, *241, 242*, *267, 268*, *275, 276*, *307*, *328*, *378*
mwr, *21*, *99*, *198*, *200*, *205*, *207*, 208, *225*, *242*, *268*, *276*, *307*, *328*, *378*

newsure, *182*, 209
nlevels, *127*, *217, 218*
nlevelsWT, *18*, *105*, *127*, *131*, *140*, *150, 151*, 210, *211*, *220*, *283*, *319*, *321*, *324*, *327*, *330, 331*, *335, 336*, *338, 339*, *341, 342*, *370*, *385*
nlevelsWT.default, *165*, *210*, 211
nullevels, *147*, 212, *214–216*, *294, 295*, *385*
nullevels.imwd, *145*, *212*, 213
nullevels.wd, *212*, 214
nullevels.wst, *212*, 216
numtonv, *39*, *152–154*, 217, *219, 220*
nv.object, *153, 154*, *218*, 219, *243*

pclaw *(*dclaw*)*, 81
plot, *145*, *147*, *161*, *207*, *363*, *370*, *385*, *389*
plot.imwd, 220
plot.imwdc *(*plot.imwd*)*, 220
plot.irregwd, *19*, *160*, *162*, *182*, 221

plot.mwd, *21*, *27*, *99*, *198*, *200*, *205*, *207*, *209*, 223, *242*, *268*, *276*, *307*, *328*
plot.nvwp, *35*, 225, *237*, *245*
plot.wd, *222*, 226, *236*, *359*, *380*
plot.wp, *55*, *125*, 229, *243*, *245*, *369*
plot.wst, *176*, 232, *383*
plot.wst2D, 234, *386*
plotdenwd, *86*, 235
plotpkt, *35*, *48*, 237
print, *145*, *147*, *207*, *220*, *363*, *366*, *370*, *385*, *389*
print.BP, 238, *254*
print.imwd, 238
print.imwdc, 240
print.mwd, *21*, *27*, *99*, *198*, *200*, *205*, *207*, *209*, *225*, 241, *268*, *276*, *307*, *328*
print.nv, 243
print.nvwp, *153*, *226*, 244
print.w2d, 246
print.w2m, 247, *373*
print.wd, 248
print.wd3D, 249, *250*, *280*, *284*, *337*, *365*, *366*, *381*
print.wp, 250, *369*
print.wpst, 252
print.wpstCL, 253
print.wpstDO, 254
print.wpstRO, 256
print.wst, 257
print.wst2D, 258, *386*
PsiJ, *157*, *158*, 259, *262*, *263*, *265*, *266*, *394*
PsiJmat, *260*, 262, *394*
Psiname, *260*, 265
putC, *22*, *160*, *161*, *206*, *207*, 266, *270*, *271*, *273*, *274*, *284*, *285*, *358*, *359*, *362*, *370*, *385*
putC.mwd, *21*, *27*, *99*, *198*, *200*, *205*, *207*, *209*, *225*, *242*, 267, *276*, *307*, *328*
putC.wd, *22*, *23*, *266*, *267*, 269
putC.wp, *266*, *267*, 271, *369*
putC.wst, *266*, *267*, 272, *282*, *283*
putD, *28*, *160*, *161*, *175*, *176*, *206*, *207*, *267*, *271*, 274, *278–280*, *283–285*, *358*, *359*, *362*, *365*, *366*, *370*, *385*
putD.mwd, *21*, *27*, *99*, *198*, *200*, *205*, *207*, *209*, *225*, *242*, *268*, 275, *307*, *328*
putD.wd, *23*, *28*, *274*, 277
putD.wd3D, *250*, 279, *283*, *337*, *365*, *366*, *381*
putD.wp, *274*, 280, *369*
putD.wst, *273*, *274*, 282
putDwd3Dcheck, *250*, *280*, 283, *337*, *365*, *366*, *381*
putpacket, *105*, *271*, *272*, *281*, 284, *286*, *288*, *370*, *385*, *389*
putpacket.wp, *125*, *272*, *281*, *284*, 285, *285*, *288*, *369*
putpacket.wst, *273*, *282–285*, 287
putpacket.wst2D, *132*, *235*, *284*, *285*, 289, *386*, *388*

rclaw, *84*
rclaw *(*dclaw*)*, 81
rcov, 291
rfft, 292, *293*, *294*
rfftinv, *292*, 293
rfftwt, 294
rm.det, 294
rmget, *117*, *157*, *158*, *260*, 295
rmname, *157*, *158*, *265*, *296*, 297
rotateback, *37*, 298
rsswav, *73*, 299, *355*

ScalingFunction, *94*, 301
Shannon.entropy, *194*, 302
simchirp, 303
ssq, 304
summary, *145*, *147*, *207*, *363*, *366*, *370*, *385*, *389*
summary.imwd, *239*, 305
summary.imwdc, *240*, 306
summary.mwd, *21*, *27*, *99*, *198*, *200*, *205*, *207*, *209*, *225*, *242*, *268*, *276*, 307, *328*
summary.wd, *248*, 308
summary.wd3D, *249*, *250*, *280*, *284*, 309, *337*, *365*, *366*, *381*
summary.wp, *251*, 310, *369*
summary.wpst, 311
summary.wst, 312
summary.wst2D, *258*, 313, *386*
support, 314
sure, *209*, *210*, 315, *330*, *341*

teddy, 316
test.dataCT, *76*, 317
threshold, *17*, *23*, *29*, *39*, *76*, *77*, *80*, *89*, *90*, *106*, *108*, *111*, *161*, *179*, *180*, *207*, *208*, *213–216*, *316*, 318, *322*, *323*,

333, 337, 340, 343–346, 359, 363, 366, 370, 380, 385
threshold.imwd, 62, 64, 65, 145, 147, 151, 221, 305, 306, 318, 319, 319, 323, 349
threshold.imwdc, 147, 318, 319, 323
threshold.irregwd, 19, 160–162, 182, 209, 210, 318, 319, 324
threshold.mwd, 21, 27, 99, 198, 200, 205, 207, 209, 225, 242, 268, 276, 292, 307, 326
threshold.wd, 50, 73, 77, 80, 90, 171, 172, 300, 318, 319, 327, 329, 342, 355
threshold.wd3D, 250, 280, 284, 318, 319, 334, 337, 365, 366, 381
threshold.wp, 125, 318, 319, 337, 369
threshold.wst, 133, 154, 233, 318, 319, 340, 382, 383, 389–393, 395
TOgetthrda1, 344
TOgetthrda2 (TOgetthrda1), 344
TOkolsmi.chi2 (TOgetthrda1), 344
TOonebyone1 (TOgetthrda1), 344
TOonebyone2 (TOgetthrda1), 344
TOshrinkit (TOgetthrda1), 344
TOthreshda1, 344, 345, 346
TOthreshda2, 344, 345, 346
tpwd, 347, 348
tpwr, 348, 348

uncompress, 64, 323, 349, 350
uncompress.default, 349, 350
uncompress.imwdc, 146, 147, 349, 351

wavegrow, 39, 352, 357, 359
WaveletCV, 73, 299, 300, 354
wavethresh (wavethresh-package), 7
wavethresh-package, 7
wd, 17, 18, 21–23, 27–29, 34, 50, 58, 67–71, 86, 89, 90, 94, 99, 107, 108, 111, 115, 119–122, 134, 140, 143, 160–162, 170, 172, 175, 197, 198, 200, 205, 207, 209, 212, 214–216, 225–228, 233, 242, 248, 268–271, 276–279, 295, 307, 308, 321, 326, 328–333, 336, 338, 339, 342, 345, 346, 353, 355, 358, 360–365, 376, 377, 379, 380, 382, 390
wd.dh, 116, 360
wd.int, 359, 361, 377
wd.object, 22, 66, 69–71, 77, 82, 86–88, 90, 92, 94, 99, 100, 108, 140, 160, 161, 167, 170, 174, 176, 201, 202, 212, 214, 215, 248, 270, 271, 277, 278, 295, 319, 333, 345, 346, 362, 376, 379, 380
wd3D, 29, 249, 250, 279, 280, 284, 309, 334–336, 364, 364, 366, 381
wd3D.object, 249, 250, 279, 280, 283, 284, 319, 337, 365, 365, 381
Whistory, 366, 367, 368
Whistory.wst, 367, 367
wp, 23, 31, 32, 37, 48, 101, 105, 124, 125, 153, 192, 193, 226, 230, 231, 233, 251, 271, 280, 281, 285, 286, 310, 337–340, 368, 368–370
wp.object, 48, 92, 100, 101, 153, 192, 193, 229, 231, 250, 251, 281, 285, 286, 319, 340, 369, 369
wpst, 32, 33, 37, 61, 122, 126, 127, 184, 185, 187, 188, 233, 252, 311, 370, 371, 372, 375
wpst2discr, 51, 52, 136, 184, 185, 246, 372
wpst2m, 247, 373
wpstCLASS, 136, 185, 254, 374
wpstREGR, 188, 375
wr, 23, 29, 65, 66, 94, 111, 115, 120, 333, 359, 362, 376, 377, 381
wr.int, 359, 362, 377, 380
wr.mwd, 27, 99, 198, 200, 205, 207, 209, 225, 242, 268, 276, 307, 328, 378
wr.wd, 22, 87, 94, 111, 115, 160, 333, 359, 376, 379
wr3D, 250, 280, 284, 337, 365, 366, 381
wst, 17, 24, 25, 34, 35, 37–39, 61, 67–71, 102, 121, 122, 128, 129, 141, 154, 175, 192, 194, 212, 216–220, 233, 257, 272, 273, 282, 283, 287, 288, 302, 312, 341–343, 362, 364, 382, 383–386, 391, 393
wst.object, 25, 35, 37–39, 61, 68, 69, 71, 92, 102, 103, 122, 129, 141, 154, 167, 174, 192, 194, 212, 216–218, 220, 232, 233, 257, 273, 282, 283, 285, 288, 319, 343, 383, 384, 387, 391, 393
wst2D, 40, 41, 130–132, 143, 234, 235, 258, 289, 290, 313, 385, 386, 389

wst2D.object, *41*, *132*, *235*, *258*, *285*, *290*, *386*, 387
wstCV, *134*, *166*, *168*, 389, *393*
wstCVl, *134*, *166*, *168*, 391, *391*
WTEnv, *157*, *260*, 394
wvcvlrss, 395
wvmoments, 396
wvrelease, 397