

# The wavethresh Package

August 7, 2006

**Version** 2.2-9

**Note** — Version also in `wvrelease()` in `./R/release.R`

**Date** 2006-08-07

**Author** Guy Nason <G.P.Nason@Bristol.ac.uk> of R-port: Arne Kovac (1997) and Martin Maechler (1999)

**Maintainer** Martin Maechler <maechler@stat.math.ethz.ch>

**Title** Software to perform wavelet statistics and transforms.

**Description** Software to perform 1-d and 2-d wavelet statistics and transforms

**Depends** R (>= 2.1)

**License** GPL version 2 or later

## R topics documented:

<code>accessC</code>	2
<code>accessD</code>	4
<code>compress</code>	5
<code>compress.default</code>	6
<code>compress.imwd</code>	7
<code>compressed.object</code>	8
<code>dof</code>	9
<code>draw</code>	10
<code>draw.default</code>	10
<code>draw.wd</code>	12
<code>filter.select</code>	13
<code>first.last</code>	14
<code>imwd</code>	16
<code>imwd.object</code>	17
<code>imwdc.object</code>	18
<code>imwr</code>	19
<code>imwr.imwd</code>	19
<code>lennon</code>	21
<code>lt.to.name</code>	21
<code>pack8bit</code>	22
<code>plot.imwd</code>	23

plot.imwdc . . . . .	25
plot.wd . . . . .	26
print.imwd . . . . .	27
print.wd . . . . .	28
putC . . . . .	28
putD . . . . .	29
summary.imwd . . . . .	30
summary.wd . . . . .	31
support . . . . .	32
threshold . . . . .	33
threshold.imwd . . . . .	34
threshold.imwdc . . . . .	35
threshold.wd . . . . .	36
uncompress . . . . .	39
uncompress.default . . . . .	39
uncompressed.object . . . . .	40
wd . . . . .	41
wd.object . . . . .	43
wr . . . . .	44
wvrelease . . . . .	46

<b>Index</b>	<b>47</b>
--------------	-----------

---

accessC

*Get Smoothed Data from Wavelet Structure*

---

## 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

```
accessC(wd.obj, level = wd.obj$nlevels, boundary=FALSE)
```

## Arguments

<code>wd.obj</code>	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.
<code>level</code>	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).
<code>boundary</code>	logical; if <code>TRUE</code> 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 <code>boundary</code> 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.

## Details

The `wd` (`wr`) 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  $0, 1, \dots, m$ . So

```
> accessC(wd.obj, level=m)
```

pulls out the original data, as does

```
> accessC(wd.obj)
```

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

```
> accessC(wd.obj, 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 `wd.obj`, in particular the array `fl.dbase$first.last.c` which gives a complete specification of index numbers and offsets for `wd.obj$C`.

Note that this and the `accessD` function only work with objects of class `wd`, see `wd.object`.

Further note that this function only gets information from 'wd' class objects. To put coefficients etc. into `wd` structures you have to use the "putC" function.

## Value

A vector of the extracted data.

## See Also

For background information, `wr` and `wd`. Further, `accessD`, `filter.select`, `plot.wd`, `threshold`, `putC`, `putD`.

## Examples

```
## Get the 3rd level of smoothed data from a decomposition
accessC(wd(rnorm(2^7)), level=3)

example(wd)
str(accessC(wds))
## Plot the time series from a reconstruction
plot.ts(accessC(wr(wds, return.obj = TRUE)))
```

---

 accessD

*Get wavelet expansion coefficients from wavelet structure.*


---

### Description

The coefficients from a wavelet expansion in a wavelet decomposition structure (returned from [wd](#) or [wr](#)) are packed into a single vector in that structure. This function extracts the coefficients corresponding to a particular resolution level.

### Usage

```
accessD(wd.obj, level, boundary=FALSE)
```

### Arguments

<code>wd.obj</code>	Wavelet decomposition structure from which you wish to extract the expansion coefficients.
<code>level</code>	The level that you wish to extract. If the "original" data has $2^m$ data points then there are $m$ possible levels that you could want to access, indexed by $0, 1, \dots, (m-1)$ .
<code>boundary</code>	If this argument is 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 using periodic boundary handling then this option has no effect.

### Details

The `wd` (`wr`) function produces a **wavelet decomposition (reconstruction) 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 vector. `AccessD` obtains information about where the coefficients appear from the `fl.dbase` component of `wd.obj`, in particular the array `fl.dbase$first.last.d` which gives a complete specification of index numbers and offsets for `wd.obj$D`.

Note that this function and `accessC` only work on objects of class `wd`. Also, you have to use `putD` to put wavelet coefficients into a `wd` object.

### Value

A vector of the extracted coefficients.

### See Also

[wr](#) and [wd](#) for background information; [accessC](#), [filter.select](#), [threshold](#), [putC](#), [putD](#).

**Examples**

```
example(wd)

## Get the 3rd level coefficients of a decomposition
accessD(wds, level=3)

## Do a qqnorm plot to assess the normality of some coefficients
qqnorm(accessD(wds, level=8))
```

---

`compress`*Compression - Generic function (wavelet)*

---

**Description**

Compresses  $x$  (typically by removing zeroes). This is the generic function.

**Usage**

```
compress(x, ...)
```

**Arguments**

<code>x</code>	an R object.
<code>...</code>	methods may have additional arguments.

**Details**

Functions with names beginning in `compress.` will be methods for this function,  
See their individual help pages, e.g., [compress.imwd](#), [compress.default](#), for operation.

**Value**

a compressed version of the supplied argument.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**See Also**

[threshold](#), [uncompress](#).

---

compress.default    *Compress a (Wavelet) Vector*

---

### Description

Compression of a vector by removal of zero elements.

### Usage

```
## Default S3 method:
compress(x, verbose = getOption("verbose"), ...)
```

### Arguments

x	numeric vector to compress
verbose	logical; if true then report on compression activity.
...	further arguments to be passed to or from methods.

### Details

Images are large objects. Thresholded 2d wavelet objects (imwd) are also large, but many of their elements are zero. Compress 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 a method for the generic function `compress()` for class `default`. It can be invoked by calling `compress(x)` for an object  $x$  of the appropriate class, or directly by calling `compress.default(x)` 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

Release 2.2 Copyright Guy Nason 1993

### BUGS

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

### See Also

[compressed.object](#), [uncompressed.object](#); [rle](#) for another kind of compression.

**Examples**

```
#
# Compress a vector with lots of zeroes
#
str(compress(c(rep(0,100),99)))
## List of 3
## $ position      : int 101
## $ values        : num 99
## $ original.length: int 101
## - attr(*, "class")= chr "compressed"

## Try to compress a vector with not many zeroes
compress(1:10)
## (uncompressed)
```

---

compress.imwd

*Compression and Decompression for 'imwd' Objects*


---

**Description**

Compresses or decompresses a (thresholded) imwd object (`imwd.object`) by removing or padding with zero elements.

**Usage**

```
## S3 method for class 'imwd':
compress(x, verbose = getOption("verbose"), ...)
## S3 method for class 'imwdc':
uncompress(x, verbose = getOption("verbose"), ...)
```

**Arguments**

<code>x</code>	class imwd object to compress or class imwdc to decompress.
<code>verbose</code>	logical; if true then report on compression activity.
<code>...</code>	further arguments to be passed to or from methods.

**Details**

Thresholded imwd objects are usually very large and contain many zero elements. This function compresses these objects into smaller "imwdc" objects by using the `compress.default` function and removing the zeroes.

`compress.imwd` is a method for the generic function `compress` for class imwd. The user shouldn't need to use this function directly.

`uncompress.imwd` uncompresses a imwdc object back into a imwd one. It is a method for the generic function `uncompress()` for class imwdc. It can be invoked by calling `uncompress(x)` for an object `x` of the appropriate class, or directly by calling `uncompress.imwdc(x)` regardless of the class of the object.

**Value**

An object of class "imwdc" representing the compressed imwd object, see `imwdc.object`.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**See Also**

[compress.default](#), [uncompress.default](#), [threshold](#), [imwd.object](#), [imwdc.object](#).

**Examples**


---

compressed.object    *Compressed (Wavelet) Object*

---

**Description**

These are objects of class "compressed". They are lists with components representing a vector, see below.

**Details**

The [compress.default](#) function tries to compress a vector. If it can it produces an object of class `compressed`, otherwise an object of class `uncompressed`.

**Value**

The following components must be included in a legitimate `compressed` object.

<code>position</code>	a vector containing the position of the non-zero elements in the original vector.
<code>values</code>	a vector, corresponding position-by-position to the position vector, containing the values of the original vector.
<code>original.length</code>	the original length of the uncompressed vector.

**GENERATION**

By [compress.default](#)

**METHODS**

The "compressed" class of objects has methods for the following generic functions: `uncompress.default`

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**See Also**

[compress.](#)

---

dof *Degrees of Freedom of Wavelet*

---

**Description**

Compute degrees of freedom for wavelet thresholding model.

**Usage**

```
dof(wd)
```

**Arguments**

wd                    The wavelet object

**Details**

The wavelet thresholding is a non-linear transform and `dof` returns the approximate number of degrees of freedom.

**Value**

The number of degrees of freedom.

**References**

see [wd](#) for a list.

**See Also**

[threshold](#)

**Examples**

```
example(wd)
threshy <- threshold(wds)

# Compare degrees of freedom
dof(wds)            # 1024
dof(threshy)        # about 23
```

---

draw	<i>Draw Wavelets - Generic function</i>
------	---

---

### Description

Draw a picture of a wavelet. This is a generic function. There are methods for 1D and 2D wavelets, see below.

### Usage

```
draw(x, ...)
```

### Arguments

x	an R object.
...	methods may have additional arguments. In addition, the high-level graphics control arguments described under <a href="#">par</a> and the arguments to <a href="#">title</a> may be supplied to this function.

### Side Effects

a plot is created on the current graphics device.

### See Also

[draw.default](#), [draw.wd](#), [draw.imwd](#), [draw.imwdc](#) for examples and [wd.object](#) for background.

---

draw.default	<i>Draw a Picture of a Wavelet</i>
--------------	------------------------------------

---

### Description

Draws pictures of basic (Daubechies') wavelets in the wavethresh package.

### Usage

```
## Default S3 method:
draw(x = filter.number, filter.number = 2,
     family = c("DaubExPhase", "DaubLeAsymm"),
     resolution = 1024, verbose = FALSE, plot = TRUE,
     main = paste("Wavelet Picture", if(enhance) " (Enhanced)", sep=""),
     sub = zwd$filter$name, xlab = "x", ylab = expression(psi(x)), dimension = 1,
     twodplot = persp, enhance = TRUE, efactor = 0.05, ...)
```

**Arguments**

<code>x, filter.number</code>	integer; number of the filter in the wavelet family specified. The range of possible numbers depends on the family.
<code>family</code>	character, the family of wavelets to use, defaults to the first of the list shown above.
<code>resolution</code>	integer specifying the resolution to use in drawing pictures. A higher resolution gives a better quality picture but will take longer to draw (especially in two dimensions).
<code>verbose</code>	logical, if TRUE, report the progress of drawing.
<code>plot</code>	logical, if true then plot the appropriate wavelet on the current active graphics device, otherwise the information about the wavelet is returned.
<code>main, sub</code>	main and sub-title for the plot, see <code>title</code> .
<code>xlab, ylab</code>	labels for the x and y axis.
<code>dimension</code>	If this is 1, then a one-dimensional version of the wavelet is drawn, if this is 2, then a two-d picture is drawn with the <code>twodplot</code> function.
<code>twodplot</code>	function such as <code>persp</code> that can do something interesting with a matrix.
<code>enhance</code>	logical; with Daubechies' wavelets their effective support is much less than their actual support. If <code>enhance</code> is true, the function tries to draw the wavelet on its effective support.
<code>efactor</code>	numeric, defining the effective support. Define $z0$ to be <code>efactor</code> times the maximum absolute value of the wavelet $w(x)$ . Define the set $A = \{x :  w(x)  > z0\}$ , and the effective support is the smallest interval (in 1D, square in 2D) containing $A$ .
<code>...</code>	further arguments to <code>plot</code> or <code>twodplot</code> .

**Details**

For Daubechies' compactly supported wavelets there is no known closed form expression. However, it is possible to make use of the inverse wavelet transform to draw pictures of the wavelets. The idea is to set all but one wavelet coefficient equal to zero and that one coefficient equal to one in a wavelet expansion. Then the inverse transform (reconstruction) is applied to the expansion and a picture of one wavelet is produced.

A method similar to the one we present here is presented in Daubechies (1992) in Section 6.5 on the cascade algorithm.

The principle is simple, but the implementation to get good pictures is surprisingly tricky. Ideally you want to put in the non-zero coefficient at the lowest resolution level as possible, this will give you as much detail as possible. However, the support of some of the large-scale wavelets is greater than the union of the supports of all the high-resolution small-scale wavelets and so it may not be possible to draw a complete wavelet.

This function analyses the supports of the wavelets at different levels and finds a coefficient to set to one by choosing the wavelet at the lowest possible resolution and whose support is within the union of supports of the highest resolution wavelets. If there is a choice of such wavelets, then the middle-most one is selected.

**Value**

If `plot` is false then no actual drawing is done, however, a list is returned with `dimension + 1` components: If `dimension` is one, the list has components `x` and `y`, representing the domain of definition (or effective support) of the wavelet and the value of the wavelet at `x`.

If `dimension == 2`, the list has three components `x`, `y` and `z`.

**Side Effects**

a picture of a wavelet is drawn on the currently active graphics device if `plot=TRUE`.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**See Also**

[draw](#), [draw.wd](#) etc.

**Examples**

```
op <- par(mfrow=c(5,2), oma = c(0,0, 4, 0),
          mgp = c(1.2, .8, 0), mar = .1 + c(4,4, .5,1))
for(fn in 1:10) {
  draw.default(filter.number= fn, col = "blue", main = NULL, xlab= "")
  abline(h=0,v=0, lty=3, lwd=.5, col = "gray")
}
mtext(paste("draw.default(*, family = '", formals(draw.default)$family[[2]], "'"),
        side = 3, line = 1, outer = TRUE,
        cex = par("cex.main"), font = par("font.main"))
par(op)

# Draw a 2-dimensional Daubechies least asymmetric wavelet
draw.default(filter.number=6, family="DaubLeAsymm", dim=2, resolution=128)
```

---

draw.wd

*Draw Wavelet Corresponding to Wavelet Object*

---

**Description**

Draws picture of the wavelet associated with (1D or 2D) wavelet decomposition object `wd`, as opposed to drawing `wd` itself (which is achieved by the corresponding `plot` method, such as [plot.wd](#)).

**Usage**

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

**Arguments**

x	Object of class <code>wd</code> , <code>imwd</code> , or <code>imwdc</code> , typically from a wavelet decomposition using the <code>wd</code> , <code>imwd</code> , or <code>threshold</code> function.
resolution	integer specifying the resolution to use in drawing. A higher resolution gives a better quality picture but will take longer to draw (especially in two dimensions).
...	further arguments to <code>draw.default</code> .

**Details**

Sometimes it is more convenient to use this function rather than `draw.default`, since you want a picture of the wavelet that did your decomposition.

**See Also**

`draw.default`, `wd`.

**Examples**

```
example(wd)
# Draw a picture of the wavelets that were used
draw(wds); abline(h=0,v=0, lty=3, lwd=.5)

example(imwd)
# Draw a picture of the 2D wavelet that used
draw(imwdL)
```

---

filter.select

*Wavelet Filter Coefficients*

---

**Description**

This function stores the filter coefficients necessary for doing a discrete wavelet transform (and its inverse).

**Usage**

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

**Arguments**

filter.number	integer, selecting the desired filter; takes a value dependent upon the family that you select.
family	character string, selecting the type of wavelet.
constant	numeric; 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.)

## Details

This function contains three types of filter. Two types can be selected with family set to "DaubExPhase" (as per default), 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 as proposed in Daubechies (1988). Setting family to "DaubLeAsymm" gives you Daubechies least asymmetric wavelets also from Daubechies (1988), 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.

This function is currently used by `wr` and `wd` in decomposing and reconstructing, however you may wish to look at the coefficients.

## Value

A list is returned with four components describing the filter:

<code>H</code>	A vector containing the filter coefficients.
<code>name</code>	A character string containing the name of the filter.
<code>family</code>	A character string containing the family of the filter.
<code>filter.number</code>	The filter number used to select the filter.

## NOTE

The filter coefficients should always sum to  $\sqrt{2}$ . This is a useful check on their validity.

## See Also

`wd`, `wr`, for background information; `accessC`, `accessD`, `imwd`, `imwr`, `threshold`.

## Examples

```
## look at the filter coefficients for N=2 :
str(f2 <- filter.select(2))
##- List of 4
##- $ H           : num [1:4]  0.483  0.837  0.224 -0.129
##- $ name        : chr "Daub cmpct on ext. phase N=2"
##- $ family      : chr "DaubExPhase"
##- $ filter.number: num 2

all.equal(sum(f2 $H), sqrt(2))# TRUE
```

## Description

This function is not intended for user use, but is used by various functions involved in computing and displaying wavelet transforms.

**Usage**

```
first.last (LengthH, DataLength, bc = c("periodic", "symmetric"))
```

**Arguments**

LengthH            length of the filter used to produce a wavelet decomposition.  
 DataLength        length of the data before transforming; must be a power of 2, say  $2^m$ .  
 bc                 character string, determining how the boundaries of the the function are to be handled; one of "periodic" (default) or "symmetric".

**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 example, given a `first.last.c` row of

```
-2 3 20
```

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 with the following components:

`first.last.c` A  $(m+1) \times 3$  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  $m \times 3$  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**

Release 2.2 Copyright Guy Nason 1993

**BUGS**

None, I hope. However, with hindsight, I should have implemented the periodic version first. The symmetric boundary stuff confused a lot of people (including me)!

**References**

The numbers in first/last databases were worked out from inequalities derived from:

Daubechies, I. (1988). *Orthonormal bases of compactly supported wavelets*; Communications on Pure and Applied Mathematics, **41**, 909-996.

**See Also**

`wr`, `accessC`, `accessD`, `filter.select`, `threshold`, `wd`, `imwd`, `imwr`.

**Examples**

```
## If you're twisted then you may just want to look at one of these.

first.last(length(filter.select(2)), 64)
```

---

imwd

---

*2D Discrete Wavelet Transform (Image W. Decomposition)*


---

**Description**

This function performs the decomposition stage of Mallat's pyramid algorithm i.e. the discrete wavelet transform for images.

**Usage**

```
imwd(image, filter.number=2,
      bc = c("periodic", "symmetric"), verbose = getOption("verbose"))
```

**Arguments**

<code>image</code>	numeric square matrix containing the image. The number of rows in the image must be a power of 2. Since the matrix is square, this is also the number of columns.
<code>filter.number</code>	integer; the filter that you wish to use to decompose the function. The filters are obtained from the <code>filter.select</code> function and are the compactly supported orthonormal wavelets as described in Daubechies, I.
<code>bc</code>	boundary treatment. The periodic (default) treatment causes the decomposition to act as if the function you are trying to decompose is periodic (on it's own interval). The other option is symmetric, which used to be the default. This causes the decomposition to act as if the function extended by symmetric reflection at each end.
<code>verbose</code>	logical; if true then informative messages are printed whilst the computations are performed.

**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.

After each attack the dimension of the images is halved. After many attacks you will obtain four real numbers, one of which correspond to the image smoothed many times.

Exact details of the algorithm are to be found in Mallat 1989.

**Value**

An object of class `imwd`, a list containing the wavelet coefficients (see [imwd.object](#)).

**See Also**

`wd`, for background information; `imwr` for reconstruction, `plot.imwd`, `draw.imwd`.

**Examples**

```
# Do a decomposition of an image
#
data(lennon)
imwdL <- imwd(lennon)
# Look at the coefficients --> example(plot.imwd)
```

---

`imwd.object`

*2d Wavelet Decomposition Object*

---

**Description**

These are objects of class "`imwd`". They represent a decomposition of a 2D function with respect to a 2D 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.

**Value**

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

<code>nlevels</code>	number of levels in wavelet decomposition. If you raise 2 to the power of <code>nlevels</code> then you get the dimension of the image that you originally started with.
<code>fl.dbase</code>	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 <code>fl.dbase</code> to get the computations correct.

filter	A filter object as returned by "filter.select". 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 the levels should be in numerically decreasing order, so if nlevels 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, see <a href="#">first.last</a> .
w0Lconstant	This is the coefficient of the bottom level scaling function coefficient. So for example, if you used Haar wavelets this would be the sample mean of the data (scaled by some factor depending on the number of levels, nlevels).
bc	This component details how the boundaries were treated in the decomposition.

## GENERATION

This class of objects is typically returned from [imwd\(.\)](#) to represent a wavelet decomposition of an image (or other 2D function).

## METHODS

The "imwd" class of objects has methods for the following generic functions: [plot](#), [threshold](#), [summary](#), [print](#), [draw](#), [imwr](#), [compress](#).

## RELEASE

Release 2.2 Copyright Guy Nason 1993

## See Also

[imwd](#) for examples; [compress](#), [draw](#).

---

imwdc.object

*Compressed 2D wavelet decomposition object*

---

## Description

These are objects of class "imwdc". They represent a compressed decomposition of a 2D function with respect to a 2D wavelet basis.

## Value

The imwdc object is the same as a imwd object (see [imwd.object](#)), except that the wNLx and w0Lconstant components have been compressed using [compress.default](#).

## GENERATION

This class of objects is returned from the [threshold](#) and [compress](#) functions to represent a wavelet decomposition of a image (or other 2D function).

**METHODS**

The "imwdc" class of objects has methods for the following generic functions: `plot`, `threshold`, `summary`, `print`, `draw`, `imwr`, `uncompress`.

**See Also**

[imwd.object](#), [uncompress](#).

---

imwr

---

*2D Inverse Discrete Wavelet Transform (Image W. Reconstruction)*


---

**Description**

Generic function for 2-dimensional wavelet reconstruction.

**Usage**

```
imwr(imwd, ...)
```

**Arguments**

`imwd` a wavelet decomposition object.  
`...` methods may have additional arguments.

**See Also**

the principal method, [imwr.imwd](#), and [imwr.imwdc](#).

---

imwr.imwd

---

*2D Inverse Discrete Wavelet Transform (Image W. Reconstruction)*


---

**Description**

These functions perform 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$bc, verbose = getOption("verbose"), ...)
## S3 method for class 'imwdc':
imwr(imwd, bc=imwd$bc, verbose = getOption("verbose"), ...)
```

**Arguments**

imwd	object of class <code>imwd</code> or <code>imwdc</code> respectively; typically returned by <code>imwd</code> and <code>threshold.imwd</code> .
bc	character, specifying the boundary handling. It is best left to be the boundary handling specified by default.
verbose	logical; if true then informative messages are printed detailing the computations to be performed.
...	further arguments to be passed to or from methods.

**Details**

Details of the algorithm are to be found in Mallat (1989). As for "imwd" the algorithm works by applying many 1D reconstruction algorithms to the coefficients. The filters used are those described in Daubechies (1988).

This function is a method for the generic function `imwr()` for class `imwd`. It can be invoked by calling `imwr(x)` for an object `x` of the appropriate class, or directly by calling `imwr.imwd(x)` 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 `nlevels` component of the `imwd.object`). This matrix is the highest resolution level of the reconstruction. If a `imwd` (decomposition) is followed immediately by a `imwr` (reconstruction) then the returned matrix will be exactly the same as the original image.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**References**

see `wd` for a list.

**See Also**

`imwd`, `plot`, `threshold`

**Examples**

```
example(imwd)
# Look at the error
summary(abs(c(imwr(imwdL) - lennon)))#around 1e-9

## Threshold after decomposing an image -- automagically compresses:
(tdi <- threshold(imwdL))

## Now reconstruct; imwr calling imwr.imwdc directly
filled.contour(answer <- imwr(tdi))
```

lennon

*Image of John Lennon***Description**

The `lennon` matrix has 256 rows and 256 columns and integer values in `0:192` which are *inverted* image gray scales, i.e., 0 means white and 192 is dark. There are a few dark speckles on the image.

John Lennon (19xx-19xx) was *the* composer/leader of the legendary Beatles.

**Author(s)**

R format and 8bit packing by Martin Maechler

**Source**

From Guy Nason, `<G.P.Nason@bristol.ac.uk>`

**Examples**

```
data(lennon)
str(lennon)

tlennon <- table(lennon)
plot(names(tlennon), sqrt(tlennon), type = "h",
      ylab = "tlennon __ sqrt scaled", yaxt = "n",
      main = "Gray value distribution of `Lennon'")
atl <- pretty(tlennon, 8)
axis(2, at=sqrt(atl), labels = formatC(atl,wid=1), las=2)

image(lennon, zlim = c(0, 192), col = gray(127:0/128))# white to dark
image(lennon, zlim = c(-10, 240), col = gray(127:0/128))
```

lt.to.name

*Convert notations (wavelet)***Description**

Convert level/notation into `imwd.object` component names.

Not for user use!

**Usage**

```
lt.to.name(level, type)
```

**Arguments**

`level` the level of the decomposition.

`type` the type, one of CD, DC or DD, reflecting the application of filters g,h in combination.

**Value**

A string in the `imwd.object` wavelet coefficient name format.

---

pack8bit

*Packing of 8-bit Integers*

---

**Description**

In image processing, often pictures are stored as long array of values in `0:255`. For (file) storage, these can be packed into 32 bit integers. These functions are for packing and unpacking.

**Usage**

```
pack8bit(v8)
unpack8bit(v)
```

**Arguments**

<code>v8</code>	integer vector with 8-bit elements in <code>0..255</code> .
<code>v</code>	integer vector (at least 32 bits).

**Details**

If the length of `v8` is *not* a multiple of 4, `v8` is effectively padded with one to three 0s. The implementation is straightforward and short; look at the function definitions.

**Value**

`pack8bit(v8)` returns an integer vector of length `ceiling(length(v8) / 4)`.  
`unpack8bit(v)` returns a vector of values from `0:255` of length `4 * length(v)`.

**Author(s)**

Martin Maechler

**See Also**

[compress](#) etc for wavelet compression.

**Examples**

```
pack8bit(0:11)
(pack8bit(0:11) == pack8bit(0:9)) # both need 3 32-bit ints; only 1st is =
## BUG:
all(250:255 == unpack8bit(pack8bit(250:255)))
stopifnot(0:255 == unpack8bit(pack8bit(0:255)))
```

---

plot.imwd

*Plot Method for an 'imwd' object*


---

## Description

Images Wavelet Coefficients of a imwd class object

## Usage

```
## S3 method for class 'imwd':
plot(x, scaling="by.level", co.type= c("abs", "mabs", "none"),
     plot.type = c("mallat", "cols"), arrangement = c(3,3),
     plot = exists("image",mode="function"), transform = FALSE,
     tfunction = sqrt, col = heat.colors(32), ...)
```

## Arguments

x	object of class imwd containing a wavelet decomposition of an image.
scaling	character string, determining the scaling applied to each sub-image. The values can be "by.level" which means each image is scaled independently to the whole dynamic range of the image, otherwise the whole image is scaled as a whole (as in previous versions). This argument only takes effect when the plot.type is "mallat".
co.type	character string, specifying a transform to be applied to the coefficients before display. If co.type=="abs", then the absolute values of the coefficients will be displayed, if co.type=="mabs", then the negative of the absolute values will be displayed. These two arguments ensure that large coefficients, whether positive or negative, will look different to small coefficients. If co.type=="none", then no transforming will take place (as in previous releases).
plot	logical; if TRUE, <code>image</code> is used to display the result.
plot.type	If plot.type=="mallat", then the image within image type of plot as exemplified within Mallat (1989) will be produced. If plot.type=="rows", then the individual level/orientation plots are produced in an array according to the arrangement argument.
arrangement	Determines the parameter to pass to <code>mfrow</code> to set up the display for multiple plots per page. The default, <code>c(3,3)</code> specifies 3 rows and 3 columns.
transform	logical; if TRUE, then the function given by argument <code>tfunction</code> is applied.
tfunction	a (vectorizing) function; a transform to apply to the coefficients if <code>transform</code> is true.
col	vector of colors to use for <code>image</code> .
...	potential further graphics parameters.

## Details

If `plot.type=="mallat"` then the picture produced is the same as the one described by Mallat 1989. After a decomposition there will be exactly the same number of coefficients as there were pixels in the original image. This function is a good way of plotting all the coefficients, but it means that you can't really see some of the lower resolution images very clearly.

If `plot.type=="rows"`, then each sub-image of the decomposition is plotted at the same size so you can see the lower resolution images. By default the arrangement of each of the sub-images is on a 3x3 array, so three levels can fit onto one plot. If you are using a screen device then it may be desirable to turn on the "ask before you plot" option with `dev.ask()`, since the coefficients plot may run to a few pages with the "rows" `plot.type`.

It is not always easy to see exactly what's going on, so there are a few arguments that try to manipulate the (sub)image(s). The scaling argument only works when the `plot.type=="mallat"` is in force. If scaling is `by.level` then each subimage is scaled independently, if not then the whole image is scaled at once. The `co.type` works for both plot types and causes absolute values, negative absolute values or just the coefficients to be plotted - this is useful for enhancing various effects. The most flexible transformation is provided when `transform==TRUE`, then the function `tfunction` is applied to all the coefficients and can produce some useful contrast enhancing effects.

At each stage of the decomposition we decompose an image into four images, each of which are half the size of the original (see "imwd" for what these are). Three of the images correspond to wavelet coefficients in the horizontal, vertical and diagonal directions and these form the bottom-right, top-left and top-right sections of the displayed image. The fourth image is further decomposed into another four subimages. Three of which get put into the bottom-right, top-left, and top-right sections OF THE BOTTOM-LEFT part of the previous image and so on and so on. It is impossible to explain this properly in a simple fashion, so the only way to really understand what is going on is to do it yourself and see what happens.

This function is a method for the generic function `plot()` for class `imwd`. It can be invoked by calling `plot(x)` for an object `x` of the appropriate class, or directly by calling `plot.imwd(x)` regardless of the class of the object.

## Value

The matrix of wavelet coefficients packed in Mallat form, returned `invisible` if `plot` is `true`.

## Side Effects

An image of the wavelet coefficients is produced if `plot` is `true`.

## See Also

`imwd`, for references and background.

## Examples

```
example(imwd)
# See the wavelet coefficient in Mallat's form
c.gray <- gray(127:0 / 128)
plot(imwdL, col = c.gray)
plot(imwdL, col = c.gray, scaling = "none")
plot(imwdL, col = c.gray, scaling = "none", co.type = "none")
plot(imwdL, col = c.gray, plot.type = "cols")
```

---

plot.imwdc *Plot Method for an 'imwdc' object*

---

### Description

Images (visualizes) wavelet coefficients of an `imwdc` class object.

### Usage

```
## S3 method for class 'imwdc':  
plot(x, verbose = getOption("verbose"), ...)
```

### Arguments

<code>x</code>	object of class "imwdc", maybe from a thresholding using <code>threshold()</code> .
<code>verbose</code>	logical, passed to <code>uncompress</code> .
<code>...</code>	other arguments to <code>plot.imwd</code> .

### Side Effects

A plot of image coefficients is performed This function is a method for the generic function `plot()` for class `imwdc`. It can be invoked by calling `plot(x)` for an object `x` of the appropriate class, or directly by calling `plot.imwdc(x)` regardless of the class of the object.

### RELEASE

Release 2.2 Copyright Guy Nason 1993

### See Also

`plot.imwd`, `plot`

### Examples

```
example(imwd)  
# Look at the error  
## Threshold after decomposing an image -- automagically compresses:  
summary(tdi <- threshold(imwdL))  
plot(tdi) # bug?
```

plot.wd

*Plot Method for a 'wd' object***Description**

Plot wavelet coefficients of an object of class "wd".

**Usage**

```
## S3 method for class 'wd':
plot(x, xlabels, first.level = 1,
     main = "Wavelet Decomposition Coefficients", sub = x$filter$name,
     xlab = "Translate", ylab = "Resolution Level",
     scaling="by.level", rhlab = FALSE,
     col = par("fg"), lty = par("lty"), lwd = par("lwd"), ...)
```

**Arguments**

x	object of class wd, containing a wavelet decomposition of a function.
xlabels	if supplied, this argument should be a vector containing the x-axis for the plot. For example, if you are trying to regress y on x, then you might want to put "x" in as the x-axis. Otherwise, the translates will be plotted.
first.level	integer, determining how many of the low resolution levels are plotted. The default, first.level=1 means that 1 coefficient is plotted.
main, sub, xlab, ylab	main and sub-title, x- and y- axis label of plot
scaling	type of scaling applied to levels within the plot. Either "by.level" or "global".
rhlab	logical; determines whether the scale factors applied to each level before plotting are printed as the right hand axis.
col, lty, lwd	graphical parameters, passed to <a href="#">segments</a> .
...	other arguments to be supplied to <code>plot.wd</code> , see <a href="#">plot</a> .

**Details**

The picture produced is similar to those in Donoho and Johnstone 1992. 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. 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 two 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 to the right of the picture. These 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.

**Side Effects**

A plot of the wavelet coefficients at each resolution level is produced.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**See Also**

[wd](#) and [wd.object](#)

**Examples**

```
example(wd)

plot(wds, rhlab = TRUE) # plotting the wavelet coefficients
```

---

print.imwd	<i>Print an 'imwd' or 'imwdc' object</i>
------------	--

---

**Description**

Prints out information about the object and then calls `summary()` on it.

**Usage**

```
## S3 method for class 'imwd':
print(x, ...)
print.imwdc(x, ...)
```

**Arguments**

x	an object of class <code>imwd</code> or <code>imwdc</code> , respectively.
...	further arguments to be passed to or from methods.

**Details**

Contributed in release 2.2 by Martin Maechler

**See Also**

[imwd](#), [imwd.object](#), [imwdc.object](#).

---

```
print.wd
```

*Print Method for a 'wd' object.*

---

**Description**

Prints out information about an wd object and then calls `summary()` on it.

**Usage**

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

**Arguments**

`x`                    an object of class wd  
`...`                further arguments to be passed to or from methods.

**Side Effects**

Information about the wd object is printed out.

**See Also**

[wd](#), [wd.object](#).

---

```
putC
```

*Put Smoothed Data Into Wavelet Structure*

---

**Description**

Makes a copy of the wd object, replaces some smoothed data in the copy, and then returns it.

**Usage**

```
putC(wd, level, v, boundary=FALSE)
```

**Arguments**

`wd`                    object of class wd that is to be copied and have smoothed data replaced.  
`level`                integer; the level at which the replacement is to take place.  
`v`                     the replacement data which should be of the correct length.  
`boundary`            logical; if FALSE then only the "real" data is replaced (and it is easy to predict the required length of `v`).  
If `boundary` is TRUE, you can replace the boundary values at a particular level as well (but it is hard to predict the required length of `v`, and the information has to be obtained from the `first.last` database component of `wd`).  
`boundary` has no meaning if `wd` was obtained with the periodic boundary handling method (`bc`).

**Details**

The function `accessC` obtains the smoothed data for a particular level. This function, `putC` replaces data at a particular level and returns a modified `wd` object reflecting the change.

This function is probably not particularly useful, but it is present for completeness. It is required because of the pyramidal nature of the smoothed data points being packed into a vector.

**Value**

A `wd` class object containing the replaced data, see `wd.object`.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**See Also**

`wd.object`, `accessC`

**Examples**

```
example(wd)
## Put the numbers 1:64 into level 6
summary(newws <- putC(wds, level=6, v=1:64, boundary=FALSE))
#
# If you look at the C component of new, you will see that
# some numbers have changed at the appropriate position.
all.equal(wds,newws)
##>[1] "Component C: ... difference: 2.1912"
```

---

putD

*Put Wavelet Coefficients Into Wavelet Structure*

---

**Description**

Makes a copy of the `wd` object, replaces some wavelet coefficients into the copy, and then returns the copy.

**Usage**

```
putD(wd, level, v, boundary=FALSE)
```

**Arguments**

<code>wd</code>	object of class <code>wd</code> that is to be copied and have wavelet coefficients replaced.
<code>level</code>	integer; the level at which the replacement is to take place.
<code>v</code>	the replacement coefficients which should be of the correct length.
<code>boundary</code>	logical; if <code>FALSE</code> then only the "real" coefficients are replaced (and it is easy to predict the required length of <code>v</code> , just the correct power of 2). If <code>boundary</code> is <code>TRUE</code> , then you can replace the boundary coefficients as well (but it is hard to predict the required length of <code>v</code> , and the information has to be extracted from the <code>first.last</code> database component of <code>wd</code> ). <code>boundary</code> has no meaning if <code>wd</code> was obtained with the periodic boundary handling method ( <code>bc</code> ).

**Details**

The function `accessD` obtains the wavelet coefficients for a particular level. This function, `putD` replaces coefficients at a particular level and returns a modified `wd` object reflecting the change.

As opposed to the utility of `putC`, the `putD` function is actually quite useful. It is fun to replace coefficients, since then you can dream up your own functions, get pictures of the wavelets etc. etc.

**Value**

A `wd` class object containing the replaced coefficients, see `wd.object`.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**See Also**

`wd.object`, `accessD`, `draw`

**Examples**

```
example(wd)

## Set all the wavelet coefficients to zero
for(i in 0:(wds$nlevels - 1))
  wds <- putC(wds, level=i, v=rep(0,2^i))

## and now see what you've done
plot(wds)
matplot(x, cbind(wr(wd(y)), wr(wds)), type = 'l')
```

---

summary.imwd

*Summary Methods for 'imwd' and 'imwdc' Object*

---

**Description**

The number of levels, the dimension of the original image, name of the filter and the type of boundary handling are printed.

**Usage**

```
## S3 method for class 'imwd':
summary(object, ...)
summary.imwdc(object, ...)
```

**Arguments**

`object` The image wavelet decomposition of class "imwd" or "imwdc", respectively. This is assumed to be the result of some image wavelet decomposition.

`...` further arguments to be passed to or from methods.

**RELEASE**

Release 2.2, Copyright Guy Nason 1993

**See Also**

[summary](#), [imwd.object](#), [imwdc.object](#).

**Examples**

```
example(imwd)
summary(imwdL)
```

---

summary.wd

*Summary Method for 'wd' (1D Wavelet) Object*

---

**Description**

The number of levels, the length of the original data sequence, name of the filter and type of boundary handling are printed.

**Usage**

```
summary.wd(object, ...)
```

**Arguments**

object	The wavelet decomposition of class "wd". This is assumed to be the result of some wavelet decomposition.
...	further arguments to be passed to or from methods.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**See Also**

[wd.object](#), [summary](#).

**Examples**

```
summary(wd(1:512)) # silly
```

---

support *Compute support of wavelet*

---

### Description

Computes the support of a wavelet

### Usage

```
support(filter.number = 2, family = c("DaubExPhase", "DaubLeAsymm"),
        m = 0, n = 0)
```

### Arguments

filter.number	integer; the number within the wavelet family whose support you wish to compute.
family	character string giving the family of wavelets that should be used.
m	the dilation number.
n	the translation number.

### Details

To draw a wavelet it is important to know it's support. `support` provides this information. If a new family of wavelets is added then their support needs to be determined and this function modified. This function needn't be called by the user in normal use. If the wavelet's aren't compactly supported then the support will not be a simple closed interval!

### Value

A list containing the support of the wavelets. The list contains the following components:

lh	The left-hand end of the interval of the support of the wavelet.
rh	The right-hand end
psi.lh	The left-hand end of the support of the mother wavelet
psi.rh	The right-hand end
phi.lh	The left-hand end of the support of the scale function (father wavelet)
phi.rh	The right-hand end

### RELEASE

Release 2.2 Copyright Guy Nason 1993

### BUGS

As the example shows below, when  $m=0$  and  $n=0$  the lh and rh don't show the mother wavelet's support, but the wavelet above the mother wavelet. The calling functions allow for this.

### See Also

[wr](#), [draw](#).

**Examples**

```
str(support())
## List of 6
## $ lh      : num -2
## $ rh      : num 4
## $ psi.lh  : num -1
## $ psi.rh  : num 2
## $ phi.lh  : num 0
## $ phi.rh  : num 3
```

---

threshold

*Thresholding Wavelets - Generic function*

---

**Description**

Generic function for soft or hard thresholding of wavelets.

**Usage**

```
threshold(x, ...)
```

**Arguments**

`x` an R object.  
`...` methods may have additional arguments.

**Value**

thresholding is applied to the object and a thresholded version is returned.

**References**

see [wd](#) for a list.

**See Also**

[threshold.wd](#), [threshold.imwd](#) and [threshold.imwdc](#) for the wavelet methods and **examples**.

[wd](#), [wr](#), [imwd](#), etc.

---

threshold.imwd      *Threshold an 'imwd' Object (2D Wavelet)*

---

## Description

Applies hard or soft thresholding to wavelet decomposition object of class `imwd`.

## Usage

```
## S3 method for class 'imwd':
threshold(x, levels=3:(x$levels-1), type = c("hard", "soft"),
  policy = c("universal", "manual", "probability"),
  by.level = FALSE, value = 0, dev = var,
  verbose = getOption("verbose"), return.threshold = FALSE,
  compression = TRUE, ...)
```

## Arguments

<code>x</code>	object of class "imwd", typically from a wavelet decomposition using the <a href="#">imwd</a> function.
<code>levels</code>	integer vector determining which levels are thresholded in the decomposition.
<code>type</code>	character, determining the type of thresholding; either "hard" or "soft".
<code>policy</code>	character indicating the threshold to use, can be "universal", "manual", or "probability".
<code>by.level</code>	logical; if <code>FALSE</code> then a global threshold is applied to <i>all</i> the levels specified by <code>levels</code> , otherwise a threshold is computed and applied separately to each level.
<code>value</code>	numeric user-supplied threshold for the "manual" policy, or the user-supplied quantile level for the "probability" policy.
<code>dev</code>	deviance function. The default <code>var</code> is to use the variance, but you can insert your own measure of deviance.
<code>verbose</code>	logical, if true then <code>threshold()</code> spurts informative messages at you.
<code>return.threshold</code>	logical, if true then the actual threshold is returned, otherwise the thresholded object is returned.
<code>compression</code>	logical; if true the thresholded object is compressed and then returned, otherwise it is returned unaltered.
<code>...</code>	further arguments to be passed to or from methods.

## Details

Thresholding modifies the coefficients within a `imwd` wavelet decomposition object. The modification can be performed either with a "hard" or "soft" thresholding selected by the `type` argument. See the "Details" section of [threshold.wd](#) for an explanation of thresholding types and policies. The thresholding process forces many coefficients to zero. From release 2.2 onwards the thresholded `imwd` object is compressed by [compress](#) and returned as a much smaller `imwdc` object. An `imwdc` object is easily converted into an `imwd` object by [uncompress](#), but all relevant functions will handle the `imwdc` object.

Note that the coefficients for the horizontal, diagonal and vertical detail at a particular level are treated as one. In future releases we may add the capability to treat the details differently, although this would increase the complexity of the argument specification.

This function is a method for the generic function `threshold()` for class `imwd`. It can be invoked by calling `threshold(x)` for an object `x` of the appropriate class, or directly by calling `threshold.imwd(x)` regardless of the class of the object.

### Value

If `compression` is `true` then an object of class `"imwdc"` that has been thresholded and compressed, otherwise an uncompressed, but thresholded object of class `"imwd"`.

### RELEASE

Release 2.2 Copyright Guy Nason 1993

### BUGS

There should be an optimal policy as well, although universal comes close.

### References

see [wd](#) for a list.

### See Also

For examples, see [imwr.imwd](#), [imwr](#), [imwd](#), [compress](#), [uncompress](#), [imwd.object](#), [imwdc.object](#)

---

threshold.imwdc      *Threshold an 'imwdc' Object (2D Wavelet)*

---

### Description

Applies hard or soft thresholding to wavelet decomposition object of class `imwdc`.

### Usage

```
## S3 method for class 'imwdc':
threshold(x, verbose = getOption("verbose"), ...)
```

### Arguments

<code>x</code>	object of class <code>imwdc</code> , maybe after a previous thresholding.
<code>verbose</code>	print information messages about what's going on
<code>...</code>	other arguments to <code>threshold</code>

**Details**

This function thresholds just like `threshold.imwd`, except that the input is an "imwdc" object, not a "imwd" one, which suggests that the object has already been thresholded (because `threshold.imwd` returns an "imwdc" object). Because the object is likely to have been thresholded a warning message stating this is printed.

However, it is entirely possible that you would wish to impose a higher threshold on an already thresholded object, and so this function does just this.

This function is a method for the generic function `threshold()` for class `imwdc`. It can be invoked by calling `threshold(x)` for an object `x` of the appropriate class, or directly by calling `threshold.imwdc(x)` regardless of the class of the object.

**Value**

An object of class "imwdc", containing the thresholded object.

**RELEASE**

Release 2.2 Copyright Guy Nason 1993

**BUGS**

There should be an optimal policy as well, although universal comes close.

**References**

see `wd` for a list.

**See Also**

For examples, see `imwr.imwd`, `imwr`, `imwd`, `compress`, `uncompress`, `imwd.object`, `imwdc.object`

---

threshold.wd

*Threshold a 'wd' Object (1D Wavelet)*

---

**Description**

Applies hard or soft thresholding with different policies to a wavelet decomposition object.

**Usage**

```
## S3 method for class 'wd':
threshold(x, levels=3:(x$nlevels-1), type = c("hard", "soft"),
  policy = c("universal", "manual", "probability"), by.level = FALSE,
  value = 0, dev = var, boundary = FALSE,
  verbose = getOption("verbose"), return.threshold = FALSE, ...)
```

**Arguments**

<code>x</code>	Object of class "wd", typically from a wavelet decomposition using the <code>wd</code> function.
<code>levels</code>	integer vector determining which levels are thresholded in the decomposition.
<code>type</code>	character, determining the type of thresholding; either "hard" or "soft".
<code>policy</code>	character indicating the threshold to use, can be "universal", "manual", or "probability".
<code>by.level</code>	logical; if FALSE then a global threshold is applied to <i>all</i> the levels specified by <code>levels</code> , otherwise a threshold is computed and applied separately to each level.
<code>value</code>	numeric user-supplied threshold for the "manual" policy, or the user-supplied quantile level for the "probability" policy.
<code>dev</code>	deviance function. The default <code>var</code> is to use the variance, but you can insert your own measure of deviance.
<code>boundary</code>	logical, if true then the boundary correction values are included for thresholding, otherwise not.
<code>verbose</code>	logical, if true then <code>threshold()</code> spurts informative messages at you.
<code>return.threshold</code>	logical, if true then the actual threshold is returned, otherwise the thresholded object is returned.
<code>...</code>	further arguments to be passed to or from methods.

**Details**

Thresholding modifies the coefficients within a `wd` wavelet decomposition object. The modification can be performed either with a "hard" or "soft" thresholding selected by the `type` argument.

Hard thresholding simply compares a coefficient with a threshold. If it is larger in absolute magnitude it is left alone, if it is smaller it is set to zero. The "soft threshold" formula is

$$\text{soft}(w) = \text{sgn}(w) \max(|w| - t, 0)$$

where  $w$  is the wavelet coefficient,  $t$  is the threshold and  $\text{sgn}(w)$  is the sign of  $w$ . Soft thresholding causes  $w$  to be replaced by  $\text{soft}(w)$ .

There are many ways that the threshold can be computed; we term this the "policy". A universal policy computes a threshold based on Donoho and Johnstone's "universal thresholds". The threshold is  $\sqrt{2 \cdot \log(n)} \cdot \text{noise}$ , where `noise` is computed as  $\sqrt{\text{dev}(w)}$ , i.e. some measure of the variability of the coefficients, and  $n$  is the number of data points (or number of wavelet coefficients). By default "dev" is "var", so the noise level is estimated using the sample standard deviation. You can use any other such estimate by writing your own function and supplying it as the "dev" argument. For example you could create the function `myvar` by

```
myvar <- function(d) mad(d)^2
```

This computes the square of the mean absolute deviation of the data. It is squared because "dev" should be a measure on the variance scale, not the standard deviation (you know what I mean). If you make the "by.levels" argument `T`, then a separate threshold is computed for each level in the "levels" vector. This means that the variance is estimated separately for each level.

The "manual" policy is simple. You supply a threshold value ("value") and hard or soft thresholding is performed using that value. The "value" argument is 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

necessary to be the length vector's length. In this way, different thresholds can be given for different levels. Note that the "by.level" argument has no effect with this policy.

The "probability" policy works as follows. All coefficients that are smaller than the "value"th 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.

This function is a method for the generic function `threshold()` for class `wd`. It can be invoked by calling `threshold(x)` for an object `x` of the appropriate class, or directly by calling `threshold.wd(x)` regardless of the class of the object.

### Value

An object of class "wd" that has been thresholded.

### RELEASE

Release 2.2 Copyright Guy Nason 1993

### BUGS

There should be an optimal policy as well, although universal comes close.

### References

see [wd](#) for a list.

### See Also

[wr](#), [wd](#)

### Examples

```
# "Standard" example:
example(wd)
#
# Threshold it
#
(thH.wds <- threshold(wds))
thS.wds <- threshold(wds, type="soft")
#
# Reconstruct from the thresholded coefficients
#
str(   trecH <- wr(thH.wds))
summary(trecS <- wr(thS.wds))
#
# Plot the data, the true & reconstructed functions
#
plot(x,y, col = "gray", cex = .6) # the data
lines(x, fx, col="blue", lwd=1) # true function
lines(x, trecH, col= "orchid", lty=2, lwd=2)
lines(x, trecS, col= "tomato", lty=4, lwd=2)
legend(0,5.6, c("true", paste(c("hard","soft"), "thresh.")),
      col = c("blue","orchid","tomato"), lwd= c(1,2,2), lty = c(1,2,4))
```

---

uncompress                      *Decompression - Generic Function (Wavelet)*

---

### Description

Decompresses `x` by padding with zeroes This is the generic function.

### Usage

```
uncompress(x, ...)
```

### Arguments

`x`                      an R object.  
`...`                    methods may have additional arguments.

### Details

Functions with names beginning in `uncompress.` will be methods for this function, See their individual help pages, e.g., [uncompress.imwdc](#), [uncompress.default](#), for operation.

### Value

an uncompressed version of the supplied argument.

### See Also

[compress.](#)

---

`uncompress.default` *Decompress a Compressed or Uncompressed Object*

---

### Description

The [compress](#) function compresses a sparse vector by removing zeroes. This function performs the inverse transformation.

### Usage

```
## Default S3 method:
uncompress(x, verbose = getOption("verbose"), ...)
```

### Arguments

`x`                      The compressed or uncompressed object to uncompress.  
`verbose`                logical; if true then informative messages are printed.  
`...`                    further arguments to be passed to or from methods.

**Details**

See `compress.default` to see what this function does to vectors. The `uncompress.default` function takes objects that have been output from `compress.default(.)` and restores them to their original values. If the input to `compress.default` was a sparse vector, then an object of class `compressed` would be returned. This class of object (see `compressed.object`) is a list containing the position and values of all the non-zero elements of the original vector. The `uncompress.default` function reconstitutes the original vector.

Sometimes compression is not worthwhile (i.e. the original vector is not sparse enough). In this case `compress.default()` returns the original vector in a list with class `uncompressed`. The `uncompress.default()` function's task in this case is simple: only return the vector part of `uncompressed`.

**Value**

The uncompressed vector.

**See Also**

`compress`, `compressed.object`.

**Examples**

```
## Compress a sparse vector and look at it
str(cv <- compress( vv <- c(rep(0,100),564) ))

## Uncompress the vector, (uncompress.default is used)
str(ucv <- uncompress(cv))
all(ucv == vv)

## a bit less sparse:
vv[sample(seq(vv), 5)] <- 1:5
str(cv <- compress(vv))
all(vv == uncompress(cv))
```

---

uncompressed.object

*Uncompressed (Wavelet) Object*

---

**Description**

These are objects of class "uncompressed". They are lists with components representing a vector, see below.

Typically, these are generated by `compress.default`.

**Value**

The following components must be included in a legitimate `uncompressed` object.

`vector`            the original uncompressed vector

**See Also**

`compressed.object` for details; `compress`.

wd

*Discrete Wavelet Transform (Decomposition).***Description**

This function performs the decomposition stage of Mallat's pyramid algorithm (Mallat 1989), i.e., the discrete wavelet transform. The actual transform is performed by some C code, this is dynamically linked into R.

**Usage**

```
wd(data, filter.number = 2, family = c("DaubExPhase", "DaubLeAsymm"),
    bc = c("periodic", "symmetric"), verbose = getOption("verbose"))
```

**Arguments**

<code>data</code>	a vector containing the data you wish to decompose. The length of this vector must be a power of 2.
<code>filter.number</code>	This selects the wavelet that you want to use in the decomposition. By default this is 2, the Daubechies orthonormal compactly supported wavelet N=2 (Daubechies (1988)), extremal phase family.
<code>family</code>	specifies the family of wavelets that you want to use. The options are "DaubExPhase" and "DaubLeAsymm". The "DaubExPhase" wavelets were provided with previous versions of wavethresh.
<code>bc</code>	character, specifying the boundary handling. If <code>bc=="periodic"</code> the default, then the function you decompose is assumed to be periodic on its interval of definition, if <code>bc == "symmetric"</code> , the function beyond its boundaries is assumed to be a symmetric reflection of the function in the boundary.
<code>verbose</code>	logical, controlling the printing of 'informative' messages whilst the computations progress. turned off by default.

**Details**

The code implements Mallat's pyramid algorithm (Mallat 1989). Essentially it works like this: you start off with some data  $c_m$ , 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  $c_m$ . The other is a vector,  $d_{(m-1)}$ , say. This corresponds to the detail removed in smoothing  $c_m$  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  $c_0$  and  $d_0$ .

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"`. 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).

### Value

An object of class `wd`, see `wd.object` for details. Basically, this object is a list with the following components

<code>C</code>	Vector of sets of successively smoothed data. The pyramid structure of Mallat is stacked so that it fits into a vector. <code>accessC</code> should be used to extract these.
<code>D</code>	Vector of sets of wavelet coefficients at different resolution levels, stacking Mallat's pyramid structure into a vector. The function <code>accessD</code> should be used to extract the coefficients for a particular resolution level.
<code>nlevels</code>	The number of resolution levels, depending on the length of the data vector. If <code>length(data) = 2^m</code> , then there will be <code>m</code> resolution levels.
<code>fl.dbase</code>	The first/last database associated with this decomposition, see <code>wd.object</code> and <code>first.last</code> for details.
<code>filter</code>	A list containing information about the filter
<code>bc</code>	How the boundaries were handled.

### RELEASE

Release 2.2 Copyright Guy Nason 1993

### References

- Any book on wavelets, especially
- Chui, C. K. (1992) *An Introduction to Wavelets*; Academic Press, London.
- Daubechies, I. (1988) *Orthonormal bases of compactly supported wavelets*; Communications on Pure and Applied Mathematics, **41**, 909-996.
- Daubechies, I. (1992) *Ten Lectures on Wavelets*; CBMS-NSF Regional Conference Series in Applied Mathematics.
- Donoho, D and Johnstone, I. (1994) *Ideal Spatial Adaptation by Wavelet Shrinkage*; Biometrika; **81**, 425-455.
- Mallat, S. G. (1989) *A theory for multiresolution signal decomposition: the wavelet representation*; IEEE Transactions on Pattern Analysis and Machine Intelligence, **11**, No. 7, 674-693.
- Vaidyanathan, P. P. (1990) *Multirate digital filters, filter banks, polyphase networks and applications: A tutorial*; Proceedings of the IEEE, **78**, No. 1, 56-93.

**See Also**

[wr](#), [threshold](#), [plot.wd](#), [accessC](#), [accessD](#), [filter.select](#).

**Examples**

```
## Example from Nason's 1993 report
f.ssl <- function(x) sin(x) + sin(2*x) + log(1+x)
m <- 10 ; n <- 2^m
x <- seq(0, 10*pi, length = n)
fx <- f.ssl(x)
y <- fx + rnorm(n, s= .3)

## Decompose the test data
summary(wds <- wd(y))
```

---

wd.object

*Wavelet decomposition object (1D)*

---

**Description**

These are objects of class "wd" They represent a decomposition of a function with respect to a 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.

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.

**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 containing data at the highest resolution level. This the "pyramid structure of Mallat". Each of these levels are stored one after the other in this vector. The matrix <code>fl.dbase\$first.last.c</code> determines exactly where each level is stored in the vector. Usually, <a href="#">accessC</a> should be used to extract C components. |
| 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 <code>fl.dbase\$first.last.d</code> . Usually, <a href="#">accessD</a> should be used to extract D components.   |
| nlevels | The number of resolution levels in the pyramidal decomposition that produces the coefficients. Therefore, $2^{nlevels} \equiv 2^m$ is the number of data points used in the decomposition. This means there will be $m$ levels of wavelet coefficients (indexed $0, 1, 2, \dots, m-1$ ), and $m+1$ levels of smoothed data (indexed $0, 1, 2, \dots, m$ ).   |

<code>fl.dbase</code>	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 <code>wd.object</code> . 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. See <code>first.last</code> for more information.
<code>filter</code>	a list containing the details of the filter that did the decomposition
<code>bc</code>	how the boundaries were handled

## GENERATION

This class of objects is returned from the `wd` function to represent a wavelet decomposition of a function. Other functions also return a `wd.object`

## METHODS

The "wd" class of objects has methods for the following generic functions: `plot`, `threshold`, `summary`, `print`, `draw`.

## RELEASE

Release 2.2 Copyright Guy Nason 1993

## See Also

`wd` for examples and background.

---

`wr`

*Discrete wavelet transform (reconstruction).*

---

## Description

This function performs the reconstruction stage of Mallat’s pyramid algorithm, i.e., the discrete inverse wavelet transform.

## Usage

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

## Arguments

<code>wd</code>	A wavelet decomposition object as returned by <code>wd</code> , see <code>wd.object</code> .
<code>start.level</code>	integer; the level at which to start reconstruction. This is usually the first (level 0).
<code>bc</code> , <code>filter.number</code> , <code>family</code>	by default part of the <code>wd</code> object, but can specified differently by the “knowing”.
<code>verbose</code>	logical, controlling the printing of “informative” messages whilst the computations progress. Such messages are generally annoying so it is turned off by default.

```
return.object
```

logical; 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 T the whole wd reconstructed object is returned.

## Details

The code implements Mallat's pyramid algorithm (Mallat 1989). In the reconstruction the quadrature mirror filters G and H are supplied with  $c_0$  and  $d_0, d_1, \dots, d_{m-1}$  (the wavelet coefficients) and rebuild  $c_1, \dots, c_m$ .

If wd was obtained directly from `wd()` then the original function can be reconstructed **exactly** as  $c_m$  and can be sought with `accessC(wd.object, level=wd.object$levels)`.

Usually, the wd object has been modified in some way, for example, some coefficients set to zero by threshold. `WR` then reconstructs the function with that set of wavelet coefficients.

## 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 the documentation for `wd.object`.

## RELEASE

Release 2.2 Copyright Guy Nason 1993

## References

see `wd` for a list.

## See Also

`wd`, `accessC`, `accessD`, `filter.select`, `threshold`.

## Examples

```
# Decompose and then exactly reconstruct test.data
example(wd)#-> wds has wd() result
rec.wds <- wr(wds)
rec.wds.obj <- wr(wds, return.object = TRUE)
rec.wds2 <- accessC(rec.wds.obj, level=rec.wds.obj$nlevels)
all(rec.wds == rec.wds2)# since wr() internally uses accessC()

# Look at accuracy of reconstruction
summary(abs(rec.wds - y)) #~ 10^-11

# Reconstruct a hard.thresholded object, look at the wavelet coefficients
summary(thr.wds <- wr(threshold(wds, type="hard") ) )
```

wvrelease

*Identify version of wavelet software.*

---

**Description**

Prints a message identifying the version of the wavelet software.

**Usage**

```
wvrelease(do.cat = interactive())
```

**Arguments**

`do.cat` logical; if true, `cat(.)` a few lines to the console, otherwise just invisibly return.

**Details**

When attaching the wavelet software directory to your own data directory it is useful to put a "wvrelease()" function call just after the attach. This ensures that you have attached to the correct directory and that you are using the most up to date software. (This function is mainly useful to those of you that have received earlier versions of this software).

**Value**

a list list with components

`major` the major release number

`R.minor` the R port version (of the major release)

**Side Effects**

Prints a message identifying the version of the wavelet software.

**Examples**

```
str(r <- wvrelease())
```

# Index

## \*Topic **classes**

- compressed.object, 7
- imwd.object, 16
- imwdc.object, 17
- uncompressed.object, 39
- wd.object, 42

## \*Topic **datasets**

- lennon, 20

## \*Topic **dplot**

- support, 31

## \*Topic **hplot**

- draw, 9
- draw.default, 9
- draw.wd, 11
- plot.imwd, 22
- plot.imwdc, 24
- plot.wd, 25

## \*Topic **manip**

- accessC, 1
- accessD, 3
- compress, 4
- compress.default, 5
- compress.imwd, 6
- putC, 27
- putD, 28
- uncompress, 38
- uncompress.default, 38

## \*Topic **math**

- filter.select, 12
- wd, 40
- wr, 43

## \*Topic **methods**

- imwd.object, 16
- imwdc.object, 17
- wd.object, 42

## \*Topic **misc**

- first.last, 13

## \*Topic **models**

- dof, 8

## \*Topic **nonlinear**

- imwd, 15
- imwr, 18
- imwr.imwd, 18

- summary.imwd, 29
- summary.wd, 30
- threshold, 32
- threshold.imwd, 33
- threshold.imwdc, 34
- threshold.wd, 35
- wd, 40
- wr, 43
- wvrelease, 45

## \*Topic **smooth**

- filter.select, 12
- imwd, 15
- imwd.object, 16
- imwdc.object, 17
- imwr, 18
- imwr.imwd, 18
- plot.imwd, 22
- plot.wd, 25
- summary.imwd, 29
- summary.wd, 30
- threshold, 32
- threshold.imwd, 33
- threshold.imwdc, 34
- threshold.wd, 35
- wd, 40
- wd.object, 42
- wr, 43
- wvrelease, 45

## \*Topic **utilities**

- compress, 4
- first.last, 13
- lt.to.name, 20
- pack8bit, 21
- print.imwd, 26
- print.wd, 27
- uncompress, 38

accessC, 1, 4, 13, 15, 28, 41, 42, 44

accessD, 2, 3, 13, 15, 29, 41, 42, 44

compress, 4, 6, 7, 17, 21, 33–35, 38, 39

compress.default, 4, 5, 6, 7, 17, 39

compress.imwd, 4, 6

compressed.object, 5, 7, 39

- dof, 8
- draw, 9, 11, 17, 29, 31
- draw.default, 9, 9, 12
- draw.imwd, 9, 16
- draw.imwd (*draw.wd*), 11
- draw.imwdc, 9
- draw.imwdc (*draw.wd*), 11
- draw.wd, 9, 11, 11
  
- filter.select, 2, 4, 12, 15, 42, 44
- first.last, 13, 17, 41, 43
  
- image, 22
- imwd, 12, 13, 15, 15, 17, 19, 23, 26, 32–35
- imwd.object, 6, 7, 16, 16–18, 26, 30, 34, 35
- imwdc.object, 6, 7, 17, 26, 30, 34, 35
- imwr, 13, 15, 16, 18, 34, 35
- imwr.imwd, 18, 18, 34, 35
- imwr.imwdc, 18
- imwr.imwdc (*imwr.imwd*), 18
- invisible, 23
  
- lennon, 20
- lt.to.name, 20
  
- pack8bit, 21
- par, 9
- persp, 10
- plot, 11, 19, 24, 25
- plot.imwd, 16, 22, 24
- plot.imwdc, 24
- plot.wd, 2, 11, 25, 42
- print.imwd, 26
- print.imwdc (*print.imwd*), 26
- print.wd, 27
- putC, 2, 4, 27, 29, 42
- putD, 2–4, 28, 42
  
- rle, 5
  
- segments, 25
- summary, 30
- summary.imwd, 29
- summary.imwdc (*summary.imwd*), 29
- summary.wd, 30
- support, 31
  
- threshold, 2, 4, 7, 8, 12, 13, 15, 17, 19, 24, 32, 35, 42, 44
- threshold.imwd, 19, 32, 33, 35
- threshold.imwdc, 32, 34
- threshold.wd, 32, 33, 35
- title, 9, 10
  
- uncompress, 4, 18, 24, 33–35, 38
- uncompress.default, 7, 38, 38
- uncompress.imwdc, 38
- uncompress.imwdc (*compress.imwd*), 6
- uncompressed.object, 5, 39
- unpack8bit (*pack8bit*), 21
  
- wd, 1–4, 8, 12, 13, 15, 16, 19, 26, 27, 32, 34–37, 40, 43, 44
- wd.object, 2, 9, 26–30, 41, 42, 43, 44
- wr, 2–4, 13, 15, 31, 32, 37, 42, 43
- wvrelease, 45