

# Package ‘adlift’

April 17, 2009

**Version** 0.9-6

**Date** 26/1/2009

**Title** An adaptive lifting scheme algorithm

**Author** Matt Nunes <matt.nunes@bristol.ac.uk> Marina Knight <Marina.Knight@bristol.ac.uk>

**Maintainer** Matt Nunes <matt.nunes@bristol.ac.uk>

**Description** Adaptive Wavelet transforms for signal denoising

**Depends** EbayesThresh

**License** GPL

**URL** <http://www2.maths.bris.ac.uk/~maman/Adlift.html>

**Repository** CRAN

**Date/Publication** 2009-01-26 13:46:31

## R topics documented:

AdaptNeigh . . . . .	2
AdaptNeighmp . . . . .	5
AdaptPred . . . . .	7
AdaptPredmp . . . . .	9
adjustx . . . . .	12
Amatdual . . . . .	13
artlev1 . . . . .	14
as.column . . . . .	16
as.row . . . . .	17
basifns . . . . .	18
condno . . . . .	20
CubicPred . . . . .	21
CubicPredmp . . . . .	23
denoise . . . . .	25
denoisehetero . . . . .	26

denoiseheteromp . . . . .	28
denoiseheteroprop . . . . .	30
dojitter . . . . .	31
fwtnp . . . . .	32
fwtnpmp . . . . .	35
getnbrs . . . . .	38
heterovar . . . . .	39
intervals . . . . .	41
invtnp . . . . .	42
invtnpmp . . . . .	44
lengthintervals . . . . .	47
LinearPred . . . . .	48
LinearPredmp . . . . .	50
make.signal2 . . . . .	52
matcond . . . . .	53
modjitter . . . . .	54
motorcycledata . . . . .	55
PointsUpdate . . . . .	56
PointsUpdatemp . . . . .	58
postmean.cauchy . . . . .	60
pts . . . . .	61
QuadPred . . . . .	62
QuadPredmp . . . . .	63
Rmatsolve . . . . .	65
transmatdual . . . . .	66
UndoPointsUpdate . . . . .	68
UndoPointsUpdatemp . . . . .	69
<b>Index</b>	<b>72</b>

---

AdaptNeigh

*AdaptNeigh*


---

## Description

This function performs the prediction lifting step over neighbourhoods and interpolation schemes.

## Usage

```
AdaptNeigh(pointsin, X, coeff, nbrs, remove, intercept, neighbours)
```

## Arguments

pointsin	The indices of gridpoints still to be removed.
X	the vector of grid values.
coeff	the vector of detail and scaling coefficients at that step of the transform.

<code>nbrs</code>	the indices (into $X$ ) of the neighbours to be used in the prediction step. Note that the value to this input is not important, since the procedure checks the neighbourhoods structure in the minimisation step anyway, but is for standardisation of arguments to the non-adaptive prediction schemes.
<code>remove</code>	the index (into $X$ ) of the point to be removed.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform. (Note that this is actually a dummy argument, since it is not necessary for the computation of the detail coefficient in <code>AdaptNeigh</code> , though is used for standardising its arguments with other prediction schemes for use in the <code>fwtnp</code> function).
<code>neighbours</code>	the number of neighbours to be considered in the computation of predicted values and detail coefficients.

### Details

The procedure performs adaptive regression (through `AdaptPred`) over the three types of regression and also over the  $3 * neighbours$  configurations of neighbours. The combination (type of regression, configuration of neighbours) is chosen which gives the smallest detail coefficient (in absolute value).

### Value

<code>results.</code>	<i>This is a ten item list giving the regression information chosen from the detail coefficient minimisation (i.e. the information supplied to <code>AdaptNeigh</code> by <code>AdaptPred</code>):</i>
<code>Xneigh</code>	matrix of $X$ values corresponding to the neighbours of the removed point. The matrix consists of columns $1, X[nbrs], X[nbrs]^2, \dots$ depending on the order of the prediction used and whether or not an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
<code>bhat</code>	The regression coefficients used in prediction.
<code>weights</code>	the prediction weights for the neighbours.
<code>pred</code>	the predicted function value obtained from the regression.
<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.
<code>int</code>	if TRUE, an intercept was used in the regression.
<code>scheme</code>	a character vector denoting the type of regression used in the prediction ("Linear", "Quad" or "Cubic").
<code>details</code>	a vector of the detail coefficients from which <code>AdaptPred</code> selects the minimum value. There are six entries. The first three entries represent the detail coefficients from regression with no intercept in increasing order of prediction. The second three details are values for regression with intercept.
<code>minindex</code>	the index into <code>details</code> ( <code>results[[9]]</code> ) which produces the minimum value.

<code>newinfo.</code>	<i>A six item list containing extra information to be used in the main transform procedure (<code>fwtnp</code>) obtained from the minimisation in <code>AdaptNeigh</code>:</i>
<code>clo</code>	boolean value telling the configuration of the neighbours which produce the overall minimum detail coefficient.
<code>totalminindex</code>	the index into <code>mindetails</code> (below) indicating the overall minimum detail coefficient produced by the procedure.
<code>nbrs</code>	the indices into <code>X</code> of the neighbours used in the best prediction scheme.
<code>index</code>	the indices into <code>pointsin</code> of the neighbours used in the best prediction.
<code>mindetails</code>	a vector of $3 \times \text{neighbours}$ entries giving the minimum details produced by each call of <code>AdaptPred</code> in <code>AdaptNeigh</code> (for the different number and configuration of neighbours).
<code>minindices</code>	vector of $3 \times \text{neighbours}$ entries giving the index (out of 6) of the schemes which produce the best predictions by each call of <code>AdaptPred</code> in <code>AdaptNeigh</code> .

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[AdaptPred](#), [fwtnp](#)

**Examples**

```
#
# Generate some doppler data: 500 observations.
#
tx <- runif(500)
ty <- make.signal2("doppler", x=tx)
#
# Compute the neighbours of point 173 (2 neighbours on each side)
#
out <- getnbrs(tx, 173, order(tx), 2, FALSE)

#
# Perform the adaptive lifting step
#
an <- AdaptNeigh(order(tx), tx, ty, out$nbrs, 173, FALSE, 2)
#
an[[1]][[8]]

an[[2]][[3]]

#shows best prediction when removing point 173, with the neighbours used
```

---

 AdaptNeighmp

*AdaptNeighmp*


---

### Description

This function performs the prediction lifting step over neighbourhoods and interpolation schemes, for multiple point data.

### Usage

```
AdaptNeighmp(pointsin, X, coefflist, coeff, nbrs, newnbrs, remove, intercept,
  neighbours, mpdet, g)
```

### Arguments

<code>pointsin</code>	The indices of gridpoints still to be removed.
<code>X</code>	the vector of grid values.
<code>coeff</code>	the vector of detail and scaling coefficients at that step of the transform.
<code>coefflist</code>	the list of detail and multiple scaling coefficients at that step of the transform.
<code>nbrs</code>	the indices (into <code>X</code> ) of the neighbours to be used in the prediction step.
<code>newnbrs</code>	as <code>nbrs</code> , but repeated according to the multiple point structure of the grid.
<code>remove</code>	the index (into <code>X</code> ) of the point to be removed.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform.
<code>neighbours</code>	the number of neighbours in the computation of the predicted value. This is not actually used specifically in <code>AdaptNeighmp</code> , since this is known already from <code>nbrs</code> .
<code>mpdet</code>	how the mutple point detail coefficients are computed. Possible values are "ave", in which the multiple detail coefficients produced when performing the multiple predictions are averaged, or "min", where the overall minimum detail coefficient is taken.
<code>g</code>	the group structure of the multiple point data.

### Details

The procedure performs adaptive regression (through `AdaptPred`) over the three types of regression and also over the  $3 \times \text{neighbours}$  configurations of neighbours. The combination (type of regression, configuration of neighbours) is chosen which gives the smallest detail coefficient (in absolute value).

**Value**

<code>results.</code>	<i>This is a ten item list giving the regression information chosen from the detail coefficient minimisation (i.e, the information supplied to <code>AdaptNeigh</code> by <code>AdaptPred</code>):</i>
<code>Xneigh</code>	matrix of X values corresponding to the neighbours of the removed point. The matrix consists of columns 1, $X[nbrs]$ , $X[nbrs]^2$ , ... depending on the order of the prediction used and whether or not an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
<code>bhat</code>	The regression coefficients used in prediction.
<code>weights</code>	the prediction weights for the neighbours.
<code>pred</code>	the predicted function value obtained from the regression.
<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.
<code>int</code>	if TRUE, an intercept was used in the regression.
<code>scheme</code>	a character vector denoting the type of regression used in the prediction ("Linear", "Quad" or "Cubic").
<code>details</code>	a vector of the detail coefficients from which <code>AdaptPred</code> selects the minimum value. There are six entries. The first three entries represent the detail coefficients from regression with no intercept in increasing order of prediction. The second three details are values for regression with intercept.
<code>minindex</code>	the index into <code>details</code> ( <code>results[[9]]</code> ) which produces the minimum value.
<code>clo</code>	boolean value telling the configuration of the neighbours which produce the overall minimum detail coefficient.
<code>totalminindex</code>	the index into <code>mindetails</code> (below) indicating the overall minimum detail coefficient produced by the procedure.
<code>nbrs</code>	the indices into X of the neighbours used in the best prediction scheme.
<code>index</code>	the indices into <code>pointsin</code> of the neighbours used in the best prediction.
<code>mindetails</code>	a vector of $3 * neighbours$ entries giving the minimum details produced by each call of <code>AdaptPred</code> in <code>AdaptNeigh</code> (for the different number and configuration of neighbours).
<code>minindices</code>	vector of $3 * neighbours$ entries giving the index (out of 6) of the schemes which produce the best predictions by each call of <code>AdaptPred</code> in <code>AdaptNeigh</code> .

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[AdaptPredmp](#), [fwtncmp](#)

**Examples**

```

#read in data with multiple values...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel
short<-adjustx(times,accel,"mean")
X<-short$sepx
coeff<-short$sepx
g<-short$g

coefflist<-list()
for (i in 1:length(g)){
coefflist[[i]]<-accel[g[[i]]]
}

#work out neighbours of point to be removed (31)

out<-getnbrs(X,31,order(X),2,TRUE)
nbrs<-out$n

nbrs

newnbrs<-NULL
for (i in 1:length(nbrs)){
newnbrs<-c(newnbrs,rep(nbrs[i],times=length(g[[nbrs[i]]])))
}

#work out repeated neighbours using g...
newnbrs

AdaptNeighmp(order(X),X,coefflist,coeff,nbrs,newnbrs,31,TRUE,2,"ave",g)

```

---

AdaptPred

*AdaptPred*


---

**Description**

This function performs the prediction lifting step over intercept and regression order.

**Usage**

```
AdaptPred(pointsin, X, coeff, nbrs, remove, intercept, neighbours)
```

**Arguments**

pointsin	The indices of gridpoints still to be removed.
X	the vector of grid values

<code>coeff</code>	the vector of detail and scaling coefficients at that step of the transform.
<code>nbrs</code>	the indices (into $X$ ) of the neighbours to be used in the prediction step. Note that the value to this input is not important, since the procedure checks the neighbourhoods structure in the minimisation step anyway, but is for standardisation of arguments to the non-adaptive prediction schemes.
<code>remove</code>	the index (into $X$ ) of the point to be removed.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform. (Note that this is actually a dummy argument, since it is not necessary for the computation of the detail coefficient in <code>AdaptPred</code> (the intercept is part of the adaptiveness), though is used for standardising its arguments with other prediction schemes for use in the <code>fwtnp</code> function).
<code>neighbours</code>	the number of neighbours to be considered in the computation of predicted values and detail coefficients.

### Details

The procedure performs adaptive regression (through `AdaptPred`) over the three types of regression and also over intercept. The combination (type of regression, intercept) is chosen which gives the smallest detail coefficient (in absolute value).

### Value

<code>results</code>	<i>This is a ten item list giving the regression information chosen from the detail coefficient minimisation:</i>
<code>Xneigh</code>	matrix of $X$ values corresponding to the neighbours of the removed point. The matrix consists of columns 1, $X[nbrs]$ , $X[nbrs]^2$ , ... depending on the order of the prediction used and whether or not an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
<code>bhat</code>	The regression coefficients used in prediction.
<code>weights</code>	the prediction weights for the neighbours.
<code>pred</code>	the predicted function value obtained from the regression.
<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.
<code>int</code>	if TRUE, an intercept was used in the regression.
<code>scheme</code>	a character vector denoting the type of regression used in the prediction ("Linear", "Quad" or "Cubic").
<code>details</code>	a vector of the detail coefficients from which <code>AdaptPred</code> selects the minimum value. There are six entries. The first three entries represent the detail coefficients from regression with no intercept in increasing order of prediction. The second three details are values for regression with intercept.
<code>minindex</code>	the index into <code>details</code> ( <code>results[[9]]</code> ) which produces the minimum value.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[AdaptNeigh](#), [CubicPred](#), [fwtnp](#), [LinearPred](#), [QuadPred](#)

**Examples**

```
#
# Generate some doppler data: 500 observations.
#
tx <- runif(500)
ty<-make.signal2("doppler",x=tx)
#
# Compute the neighbours of point 173 (2 neighbours on each side)
#
out<-getnbrs(tx,173,order(tx),2,FALSE)

#
# Perform the adaptive lifting step
#
ap<-AdaptPred(order(tx),tx,ty,out$nbrs,173,FALSE,2)
#
ap[[10]]

#this corresponds to no intercept and highest order regression (Cubic)...
#
#and let's check it...
ap[[7]]

ap[[8]]
```

---

AdaptPredmp

*AdaptPredmp*

---

**Description**

This function performs the prediction lifting step over intercept and regression order, for multiple point data.

**Usage**

```
AdaptPredmp(pointsin, X, coefflist, coeff, nbrs, newnbrs, remove, intercept,
  neighbours, mpdet, g)
```

**Arguments**

<code>pointsin</code>	The indices of gridpoints still to be removed.
<code>X</code>	the vector of grid values.
<code>coeff</code>	the vector of detail and scaling coefficients at that step of the transform.
<code>coefflist</code>	the list of detail and multiple scaling coefficients at that step of the transform.
<code>nbrs</code>	the indices (into <code>X</code> ) of the neighbours to be used in the prediction step.
<code>newnbrs</code>	as <code>nbrs</code> , but repeated according to the multiple point structure of the grid.
<code>remove</code>	the index (into <code>X</code> ) of the point to be removed.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform.
<code>neighbours</code>	the number of neighbours in the computation of the predicted value. This is not actually used specifically in <code>AdaptPredmp</code> , since this is known already from <code>nbrs</code> .
<code>mpdet</code>	how the mutple point detail coefficients are computed. Possible values are "ave", in which the multiple detail coefficients produced when performing the multiple predictions are averaged, or "min", where the overall minimum detail coefficient is taken.
<code>g</code>	the group structure of the multiple point data.

**Details**

The procedure performs adaptive regression (through `AdaptPred`) over the three types of regression and also over intercept. The combination (type of regression, intercept) is chosen which gives the smallest detail coefficient (in absolute value).

**Value**

<code>results.</code>	<i>This is a ten item list giving the regression information chosen from the detail coefficient minimisation:</i>
<code>Xneigh</code>	matrix of <code>X</code> values corresponding to the neighbours of the removed point. The matrix consists of columns <code>1, X[newnbrs], X[newnbrs]^2, ...</code> depending on the order of the prediction used and whether or not an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
<code>bhat</code>	The regression coefficients used in prediction.
<code>weights</code>	the prediction weights for the neighbours.
<code>pred</code>	the predicted function value obtained from the regression.
<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.
<code>int</code>	if TRUE, an intercept was used in the regression.
<code>scheme</code>	a character vector denoting the type of regression used in the prediction ("Linear", "Quad" or "Cubic").

`details` a vector of the detail coefficients from which `AdaptPredmp` selects the minimum value. There are six entries. The first three entries represent the detail coefficients from regression with no intercept in increasing order of prediction. The second three details are values for regression with intercept.

`minindex` the index into `details(results[[9]])` which produces the minimum value.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[AdaptNeighmp](#), [CubicPredmp](#), [fwt npmp](#), [LinearPredmp](#), [QuadPredmp](#)

**Examples**

```
#read in data with multiple values...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel
short<-adjustx(times,accel,"mean")
X<-short$sepx
coeff<-short$sepx
g<-short$g

coefflist<-list()
for (i in 1:length(g)){
  coefflist[[i]]<-accel[g[[i]]]
}

#work out neighbours of point to be removed (31)

out<-getnbrs(X,31,order(X),2,TRUE)
nbrs<-out$n

nbrs

newnbrs<-NULL
for (i in 1:length(nbrs)){
  newnbrs<-c(newnbrs,rep(nbrs[i],times=length(g[[nbrs[i]]])))
}

#work out repeated neighbours using g...
newnbrs

AdaptPredmp(order(X),X,coefflist,coeff,nbrs,newnbrs,31,TRUE,2,"ave",g)
```

---

`adjustx`*adjustx*

---

### Description

This function produces new grid values to cope with data with repeated grid values according to the method chosen to deal with it.

### Usage

```
adjustx(x, f, type = "mean")
```

### Arguments

<code>x</code>	a vector of the original (repeated) gridpoints.
<code>f</code>	the vector of function values associated to the grid vector <code>X</code> .
<code>type</code>	The method used to cope with the multiple points. "mean" averages all function values with the same grid value. The "jitter" option adds a small amount to all but one of each repeated grid value, and associates the function values to these new gridpoints. In this way, the each gridpoint value corresponds uniquely to the function values.

### Details

The function compares `x` to `unique(x)` to find the occurrences of repeated grid values, and stores the information in `groups`. In the "jitter" case, this is then used to modify the original gridpoints by adding an epsilon to the repeated values. In the case of `type="mean"`, the new gridpoints are, in fact `unique(x)`, and the information is used to average the groups of original function values to construct `sepf`.

### Value

<code>sepx</code>	the vector of new gridpoints.
<code>sepf</code>	the function values associated to <code>sepx</code> .
<code>groups</code>	a list of indices into <code>x</code> showing where the original repeated grid values occurred.

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

### See Also

[fwtnp](#),

**Examples**

```

#read in the motorcycle crash data
#
data(motorcycledata)

#
dim(motorcycledata)

#check data.
#
times<-motorcycledata$time
accel<-motorcycledata$accel

a<-adjustx(times,accel,"mean")
#
#note the repeated values in the original grid data
#
#display new data vectors
a$sepx
#
a$sepf
#
#and now the new adjusted data has length 94.
#

```

Amatdual

*Amatdual***Description**

Combines filter matrices to produce a refinement matrix A for a wavelet transform.

**Usage**

```
Amatdual(steps, pointsin, removelist, nbrs, weights, alpha)
```

**Arguments**

steps	a value indicating which refinement matrix to construct. It refers to the number of points already removed during the transform.
pointsin	The indices of gridpoints still to be removed.
removelist	a vector of indices into X of the lifted coefficients during the transform (in the order of removal).
nbrs	indices of the neighbours used in the last step of the decomposition.
weights	the prediction weights obtained from the regression in the prediction step of the transform.
alpha	the update weights used to update lengths and coeff.

**Details**

The function uses the prediction and update weights to construct the filter matrices `Hdual` and `Gdual`. Combining these two matrices results in the refinement matrix `Adual`.

**Value**

<code>Adual</code>	the refinement matrix for the particular step of the transform.
<code>Hdual</code>	the high-pass filter matrix for the current step of the transform.
<code>Gdual</code>	the low-pass filter matrix for the current step of the transform.
<code>o</code>	the indices of <code>nbrs</code> into the vector of <code>pointsin</code> and the steps removed points of the transform.
<code>alpha</code>	the update weights used to update <code>lengths</code> and <code>coeff</code> .
<code>weights</code>	the prediction weights obtained from the regression in the prediction step of the transform.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[transmatdual](#)

**Examples**

```
#
x<-runif(256)
y<-make.signal2("doppler",x=x)
a<-fwtnp(x,y,LocalPred=AdaptNeigh,neighbours=2)
#
A<-Amatdual(90,a$pointsin,a$removelist,a$neighbrs[[90]],a$gamlist[[90]],a$alphalist[[90]])
#
A$Adual
#
#the 90th refinement matrix for the transform above.
#
```

---

artlev1

*artlev1*


---

**Description**

This function splits the coefficients into levels according to increasing quantiles of the removed interval lengths.

**Usage**

```
artlev1(y, rem)
```

**Arguments**

`y` a vector of the removed interval lengths (in the order of `removelist`).

`rem` vector of indices of the removed points (from the output of the forward transform).

**Details**

The function finds the median of the removed interval lengths, and takes all points in indices with removed interval lengths at most this value as the first artificial level. These indices are now not considered in later groups. The cut-off value,  $q$ , is now increased to the 75th percentile, and the indices at most this value are grouped into the second level. The procedure is continued with successive percentiles  $(1+q)/2$  until all indices are grouped. At each stage, the level size is checked to ensure it has at least 10 elements, and if not, the level is taken together with the next level (i.e. the present percentile is ignored, and increased to the  $q$  value).

**Value**

`p` a list of the grouped indices of `removelist` (in decreasing group size) indicating thresholding groups.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[denoise](#),

**Examples**

```
#create test signal data
#
x<-runif(100)
y<-make.signal2("blocks",x=x)
#
#perform forward transform...
#
out<-fwtnp(x,y,LocalPred=AdaptNeigh,neighbours=2)
#
al<-artlev1(out$lengthsremove,out$removelist)
#
#
# the indices of removelist split into levels:
al
#
```

---

`as.column`*as.column*

---

**Description**

This function returns a given vector as a column (with dimension).

**Usage**

```
as.column(x)
```

**Arguments**

`x` any vector or array.

**Details**

`x` can either be a vector with no dimension attributes (a list of values), a vector with dimensions, or a matrix/array. If `x` is a matrix/array, the function gives `x` if `ncol(x)` is less than or equal to `nrow(x)`, or its transpose if `ncol(x)` is greater than or equal to `nrow(x)`. For any input, the input is given non-null dimensions.

**Value**

`y` a vector identical to `x`, but given as a column.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[as.row](#)

**Examples**

```
vector<-1:8
#
vector
#
#...vector has no dimension attributes
#
as.column(vector)
#
#...gives output dimension of (8,1)
#
F<-matrix(c(6,2,2,10,6,17),3,2)
#
#
```

```

as.column(F)

#
#the function has no effect on F
#
F<-t(F)
F
#now has dimension (2,3)...
#
as.column(F)
#
#the output is made to have more rows than columns

```

---

as.row

*as.row*


---

## Description

This function returns a given vector as a row (with dimension).

## Usage

```
as.row(x)
```

## Arguments

`x` any vector or array

## Details

`x` can either be a vector with no dimension attributes (a list of values), a vector with dimensions, or a matrix/array. If `x` is a matrix/array, the function gives `x` if `ncol(x)` is greater than or equal to `nrow(x)`, or its transpose if `ncol(x)` is less than or equal to `nrow(x)`. For any input, the input is given non-null dimensions.

## Value

`y` a vector identical to `x`, but given as a row.

## Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

## See Also

[as.column](#)

**Examples**

```

X<-0:5
#
X
#
as.row(X)
#
#puts input into row (matrix)
#
Y<-matrix(0:5,6,1)
#
Y
#
as.row(Y)
#
#input forced into a row.
#

```

---

basisfns

*basisfns*


---

**Description**

This function plots all mother and father wavelets associated with a given wavelet transform.

**Usage**

```

basisfns(x, f, pred, neigh, int, clo, keep, plot.f = FALSE, plot.bas = FALSE,
         separate = FALSE)

```

**Arguments**

<code>x</code>	a gridpoint vector.
<code>f</code>	the vector of associated function values.
<code>pred</code>	The type of regression to be performed. Possible options are <a href="#">LinearPred</a> , <a href="#">QuadPred</a> , <a href="#">CubicPred</a> , <a href="#">AdaptPred</a> and <a href="#">AdaptNeigh</a> .
<code>neigh</code>	The number of neighbours over which the regression is performed at each step. If <code>closest</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
<code>int</code>	Indicates whether or not the regression curve includes an intercept.
<code>clo</code>	Refers to the configuration of the chosen neighbours. If <code>closest</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.
<code>keep</code>	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.

<code>plot.f</code>	a boolean value indicating whether to plot the original function or not. If so, the signal is plotted with vertical coloured lines, showing which prediction method was used on the different parts of the signal. The plot also shows which grid-points correspond to scaling functions.
<code>plot.bas</code>	subset of <code>1:length(f)</code> , denoting which basis functions to plot. Each basis function is colour-coded according to which prediction scheme was used in the lifting of the corresponding gridpoint.
<code>separate</code>	a boolean argument indicating if the basis functions should be plotted on a single graphsheet.

### Details

The procedure constructs  $W$ , the matrix representation of the forward transform specified in the arguments to the function, and then uses the inverse matrix to calculate the vectors of basis function values: to work out the basis function values, one inverts the transform with a delta vector, with a one in the position corresponding to the basis function required. Since this is equivalent to pre-multiplying the delta vector by the matrix representation for the inverse transform ( $W^{-1}$ ), the basis function values are precisely the columns of  $W^{-1}$ . The procedure then plots the basis functions (each on a separate graphsheet, if chosen), colour coded according to the prediction scheme used or whether it is a scaling function.

### Value

<code>out</code>	the output from the forward transform which is specified in the arguments to this function
<code>pointsin</code>	the vector of indices of points still to be removed.
<code>schhist</code>	a character string vector of the prediction scheme used for the prediction of each gridpoint (in the order of $x$ ).
<code>inthist</code>	vector of boolean values indicating whether an intercept was used in the prediction steps during the transform (in the order of $x$ ).
<code>basmat</code>	a matrix of wavelet basis function values. The row $i$ represents the function values corresponding to the grid for the basis function associated to the gridpoint $i$ .

### Note

If `plot.bas=T`, since the function produces one graph for each gridpoint, R or Splus is likely to exceed the total number of open devices for large datasets.

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

### See Also

[fwtnp](#), [transmatdual](#)

**Examples**

```
#create test signal data
#
x<-runif(100)
y<-make.signal2("blocks",x=x)
#
#perform procedure...
#
a<-basisfns(x,y,AdaptNeigh,2,TRUE,TRUE,2,FALSE,c(1,14,15),FALSE)
#
#this produces plots of three basis functions all on one graph.
```

---

condno

*condno*


---

**Description**

This function uses a specified norm to compute the condition number of a matrix representation of a wavelet transform.

**Usage**

```
condno(W, type)
```

**Arguments**

<code>W</code>	a matrix which represents a wavelet transform.
<code>type</code>	a character string denoting which norm to use when computing the condition number. Possible values are "l1", or one of the standard norm types, "F" (Frobenius norm), "i" (infinity norm), "m" (max modulus of a matrix) or "1" (1-norm).

**Details**

The function computes the condition number as  $\text{condno} = \|W\| * \|W^{-1}\|$ .

**Value**

`condno` the condition number of the matrix `W`.

**Note**

The matrix `W` must be invertible.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**Examples**

```

#create test signal data
#
x<-runif(100)
y<-make.signal2("blocks",x=x)
#
a<-transmatdual(x,y,Pred=AdaptNeigh,neigh=2)
#
#computes the transition matrix for the specified options
#
W<-a$Wnew
#
condno(W,"F")
#
condno(W,"11")
#
condno(W,"1")
#

```

---

CubicPred

*CubicPred*


---

**Description**

This function performs the prediction lifting step using a cubic regression curve given a configuration of neighbours.

**Usage**

```
CubicPred(pointsin, X, coeff, nbrs, remove, intercept, neighbours)
```

**Arguments**

pointsin	The indices of gridpoints still to be removed.
X	the vector of grid values.
coeff	the vector of detail and scaling coefficients at that step of the transform.
nbrs	the indices (into X) of the neighbours to be used in the prediction step.
remove	the index (into X) of the point to be removed.
intercept	Boolean value for whether or not an intercept is used in the prediction step of the transform.
neighbours	the number of neighbours in the computation of the predicted value. This is not actually used specifically in <i>CubicPred</i> , since this is known already from <i>nbrs</i> .

**Details**

The procedure performs cubic regression using the given neighbours using an intercept if chosen. The regression coefficients (`weights`) are used to predict the new function value at the removed point. If there are not enough neighbours to generate a cubic regression curve, the order of prediction is decreased until it is possible (i.e. to `QuadPred`, then `LinearPred`).

**Value**

<code>Xneigh</code>	matrix of X values corresponding to the neighbours of the removed point. The matrix consists of columns $X[nbrs]$ , $X[nbrs]^2$ , $X[nbrs]^3$ augmented with a column of ones if an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
<code>bhat</code>	The regression coefficients used in prediction.
<code>weights</code>	the prediction weights for the neighbours.
<code>pred</code>	the predicted function value obtained from the regression.
<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[fwtnp](#), [LinearPred](#), [QuadPred](#)

**Examples**

```
#
# Generate some doppler data: 500 observations.
#
tx <- runif(500)
ty<-make.signal2("doppler",x=tx)
#
# Compute the neighbours of point 173 (2 neighbours on each side)
#
out<-getnbrs(tx,173,order(tx),2,FALSE)

#
# Perform cubic prediction based on the neighbours (without intercept)
#
cp<-CubicPred(order(tx),tx,ty,out$nbrs,173,FALSE,2)
#
cp$bhat

#
#the coefficients which define the cubic regression curve
```

```
#
cp$pred

#
#the predicted value from the regression curve
#
```

---

CubicPredmp

*CubicPredmp*


---

### Description

This function performs the prediction lifting step using a cubic regression curve given a configuration of neighbours, for multiple point data.

### Usage

```
CubicPredmp(pointsin, X, coefflist, coeff, nbrs, newnbrs, remove, intercept,
neighbours, mpdet, g)
```

### Arguments

pointsin	The indices of gridpoints still to be removed.
X	the vector of grid values.
coeff	the vector of detail and scaling coefficients at that step of the transform.
coefflist	the list of detail and multiple scaling coefficients at that step of the transform.
nbrs	the indices (into X) of the neighbours to be used in the prediction step.
newnbrs	as nbrs, but repeated according to the multiple point structure of the grid.
remove	the index (into X) of the point to be removed.
intercept	Boolean value for whether or not an intercept is used in the prediction step of the transform.
neighbours	the number of neighbours in the computation of the predicted value. This is not actually used specifically in <code>CubicPredmp</code> , since this is known already from <code>nbrs</code> .
mpdet	how the mutple point detail coefficients are computed. Possible values are "ave", in which the multiple detail coefficients produced when performing the multiple predictions are averaged, or "min", where the overall minimum detail coefficient is taken. Note that this is taken to standardise the input when <code>LocalPredmp</code> is called.
g	the group structure of the multiple point data. Note that this is taken to standardise the input when <code>LocalPredmp</code> is called.

**Details**

The procedure performs cubic regression using the given neighbours using an intercept if chosen. The regression coefficients (`weights`) are used to predict the new function value at the removed point.

**Value**

<code>Xneigh</code>	matrix of X values corresponding to the neighbours of the removed point. The matrix consists of the column <code>X[newnbrs]</code> augmented with a column of ones if an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
<code>bhat</code>	The regression coefficients used in prediction.
<code>weights</code>	the prediction weights for the neighbours.
<code>pred</code>	the predicted function value obtained from the regression.
<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[fwtntpmp](#), [LinearPredmp](#), [QuadPredmp](#)

**Examples**

```
#read in data with multiple values...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel

short<-adjustx(times,accel,"mean")
X<-short$sepx
coeff<-short$sepx
g<-short$g

coefflist<-list()
for (i in 1:length(g)){
coefflist[[i]]<-accel[g[[i]]]
}

#work out neighbours of point to be removed (31)

out<-getnbrs(X,31,order(X),2,TRUE)
nbrs<-out$n

nbrs
```

```

newnbrs<-NULL
for (i in 1:length(nbrs)){
newnbrs<-c(newnbrs,rep(nbrs[i],times=length(g[[nbrs[i]]])))
}

#work out repeated neighbours using g...
newnbrs

CubicPredmp(order(X),X,coefflist,coeff,nbrs,newnbrs,31,TRUE,2,"ave",g)

```

---

denoise

*denoise*


---

## Description

Denoeses the inputted signal using artificial levels noise variance estimation and bayesian thresholding.

## Usage

```
denoise(x, f, pred, neigh, int, clo, keep, rule = "median")
```

## Arguments

<code>x</code>	A vector of grid values. Can be of any length, not necessarily equally spaced.
<code>f</code>	A vector of function values corresponding to <code>x</code> . Must be of the same length as <code>x</code> .
<code>pred</code>	The type of regression to be performed. Possible options are <a href="#">LinearPred</a> , <a href="#">QuadPred</a> , <a href="#">CubicPred</a> , <a href="#">AdaptPred</a> and <a href="#">AdaptNeigh</a> .
<code>neigh</code>	The number of neighbours over which the regression is performed at each step. If <code>clo</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
<code>int</code>	Indicates whether or not the regression curve includes an intercept.
<code>clo</code>	Refers to the configuration of the chosen neighbours. If <code>clo</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.
<code>keep</code>	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.
<code>rule</code>	The type of bayesian thresholding used in the procedure. Possible values are "mean", "median" (posterior mean or median thresholding) or "hard" or "soft" (hard or soft thresholding).

**Details**

The function uses the transform matrix to normalise the detail coefficients produced from the forward transform according to the correlation structure, so that they can be used in the bayesian thresholding procedure EbayesThresh. The coefficients are divided into artificial levels, and the first (largest) level is used to estimate the noise variance of the coefficients. EbayesThresh is then used to threshold the coefficients. The resulting new coefficients are then unnormalised and the transform inverted to obtain an estimate of the true (unnoisy) signal.

**Value**

out	the output from the forward transform.
w	the matrix associated to the wavelet transform.
indsd	the individual coefficient variances introduced by the transform.
al	the artificial levels used to estimate the noise variance.
sd	the standard deviation of the noise.
fhat	the estimate of the function after denoising.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[denoisehetero](#)

**Examples**

```
x1<-runif(256)
y1<-make.signal2("doppler",x=x1)
n1<-rnorm(256,0,.1)
z1<-y1+n1
#
est1<-denoise(x1,z1,AdaptNeigh,1,TRUE,TRUE,2)
sum(abs(y1-est1$fhat$coeff))
#
#the error between the true signal and the denoised version.
```

---

denoisehetero

*denoisehetero*

---

**Description**

Denoises the inputted signal using artificial levels noise variance estimation and bayesian thresholding, using heteroscedastic (estimated) noise variances.

**Usage**

```
denoisehetero(x, f, pred, neigh, int, clo, keep, rule = "median")
```

**Arguments**

<code>x</code>	A vector of grid values. Can be of any length, not necessarily equally spaced.
<code>f</code>	A vector of function values corresponding to <code>x</code> . Must be of the same length as <code>x</code> .
<code>pred</code>	The type of regression to be performed. Possible options are <a href="#">LinearPred</a> , <a href="#">QuadPred</a> , <a href="#">CubicPred</a> , <a href="#">AdaptPred</a> and <a href="#">AdaptNeigh</a> .
<code>neigh</code>	The number of neighbours over which the regression is performed at each step. If <code>clo</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
<code>int</code>	Indicates whether or not the regression curve includes an intercept.
<code>clo</code>	Refers to the configuration of the chosen neighbours. If <code>clo</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.
<code>keep</code>	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.
<code>rule</code>	The type of bayesian thresholding used in the procedure. Possible values are "mean", "median" (posterior mean or median thresholding) or "hard" or "soft" (hard or soft thresholding).

**Details**

The function uses the transform matrix to normalise the detail coefficients produced from the forward transform, so that they can be used in the bayesian thresholding procedure `EbayesThresh`. The coefficients are divided into artificial levels, and the first (largest) level is used to estimate the noise variances of the coefficients, based on the MAD of those coefficients falling in a sliding window around each gridpoint. `EbayesThresh` is then used to threshold the coefficients. The resulting new coefficients are then unnormalised and the transform inverted to obtain an estimate of the true (unnoisy) signal.

**Value**

<code>out</code>	the output from the forward transform.
<code>w</code>	the matrix associated to the wavelet transform.
<code>indsd</code>	the individual coefficient variances introduced by the transform.
<code>al</code>	the artificial levels used to estimate the noise variance.
<code>sd</code>	the standard deviation of the noise.
<code>fhat</code>	the estimate of the function after denoising.
<code>fhat1</code>	the estimate of the function after denoising, using the alternate variance estimate of MAD, centered at zero.
<code>fhat2</code>	the estimate of the function after denoising, using the alternate variance estimate of the median of the absolute values of the detail coefficients.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[denoise](#), [heterovar](#)

**Examples**

```
x1<-runif(256)
y1<-make.signal2("doppler",x=x1)
n1<-rnorm(256,0,.1)
z1<-y1+n1
#
est1<-denoisehetero(x1,z1,AdaptNeigh,1,TRUE,TRUE,2)
sum(abs(y1-est1$fhat$coeff))
#
#the error between the true signal and the denoised version.
```

---

denoiseheteromp      *denoiseheteromp*

---

**Description**

Denoses the multiple observation inputted signal using artificial levels noise variance estimation and bayesian thresholding, using heteroscedastic (estimated) noise variances.

**Usage**

```
denoiseheteromp(x, f, pred, neigh, int, clo, keep, rule = "median", mpdet="ave")
```

**Arguments**

<code>x</code>	A vector of grid values. Can be of any length, not necessarily equally spaced.
<code>f</code>	A vector of function values corresponding to <code>x</code> . Must be of the same length as <code>x</code> .
<code>pred</code>	The type of regression to be performed. Possible options are <a href="#">LinearPred</a> , <a href="#">QuadPred</a> , <a href="#">CubicPred</a> , <a href="#">AdaptPred</a> and <a href="#">AdaptNeigh</a> .
<code>neigh</code>	The number of neighbours over which the regression is performed at each step. If <code>clo</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
<code>int</code>	Indicates whether or not the regression curve includes an intercept.
<code>clo</code>	Refers to the configuration of the chosen neighbours. If <code>clo</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.

keep	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.
rule	The type of bayesian thresholding used in the procedure. Possible values are "mean", "median" (posterior mean or median thresholding) or "hard" or "soft" (hard or soft thresholding).
mpdet	how the mutiple point detail coefficients are computed. Possible values are "ave", in which the multiple detail coefficients produced when performing the multiple predictions are averaged, or "min", where the overall minimum detail coefficient is taken.

### Details

The function uses the transform matrix to normalise the detail coefficients produced from the forward transform, so that they can be used in the bayesian thresholding procedure `EbayesThresh`. The coefficients are divided into artificial levels, and the first (largest)level is used to estimate the noise variances of the coefficients, based on those coefficients falling in a sliding window around each gridpoint. `EbayesThresh` is then used to threshold the coefficients. The resulting new coefficients are then unnormalised and the transform inverted to obtain an estimate of the true (unnoisy) signal.

### Value

out	the output from the forward transform.
indsd	the individual coefficient variances introduced by the transform.
al	the artificial levels used to estimate the noise variance.
sd	the standard deviation of the noise.
fhat	the estimate of the function after denoising.
fhat1	the estimate of the function after denoising, using the alternate variance estimate of MAD, centered at zero.
fhat2	the estimate of the function after denoising, using the alternate variance estimate of the median of the absolute values of the detail coefficients.

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

### See Also

[denoisehetero](#), [heterovar](#)

### Examples

```
data(motorcycledata)
#
times<-motorcycledata$time
accel<-motorcycledata$accel

est1<-denoiseheteromp(times, accel, AdaptNeighmp, 1, TRUE, TRUE, 2, "median", "ave")
```

```
#
#the estimate of the underlying curve.
```

---

```
denoiseheteroprop denoiseheteroprop
```

---

### Description

Denotes the inputted signal using artificial levels noise variance estimation and bayesian thresholding, assuming noise variances known up to proportionality constants.

### Usage

```
denoiseheteroprop(x, f, pred, neigh, int, clo, keep, rule = "median", gamvec)
```

### Arguments

<code>x</code>	A vector of grid values. Can be of any length, not necessarily equally spaced.
<code>f</code>	A vector of function values corresponding to <code>x</code> . Must be of the same length as <code>x</code> .
<code>pred</code>	The type of regression to be performed. Possible options are <a href="#">LinearPred</a> , <a href="#">QuadPred</a> , <a href="#">CubicPred</a> , <a href="#">AdaptPred</a> and <a href="#">AdaptNeigh</a> .
<code>neigh</code>	The number of neighbours over which the regression is performed at each step. If <code>clo</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
<code>int</code>	Indicates whether or not the regression curve includes an intercept.
<code>clo</code>	Refers to the configuration of the chosen neighbours. If <code>clo</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.
<code>keep</code>	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.
<code>rule</code>	The type of bayesian thresholding used in the procedure. Possible values are "mean", "median" (posterior mean or median thresholding) or "hard" or "soft" (hard or soft thresholding).
<code>gamvec</code>	a vector of proportions of the noise standard deviations (in the order of <code>x</code> ).

### Details

The function uses the transform matrix to normalise the detail coefficients produced from the forward transform, so that they can be used in the bayesian thresholding procedure `EbayesThresh`. The normalising factors are calculated assuming that the noise associated to the  $i$ th gridpoint is  $\gamma_i \sigma$ . The coefficients are divided into artificial levels, and the first (largest) level is used to estimate the noise variance of the coefficients. `EbayesThresh` is then used to threshold the coefficients. The resulting new coefficients are then unnormalised and the transform inverted to obtain an estimate of the true (unnoisy) signal.

**Value**

out	the output from the forward transform.
w	the matrix associated to the wavelet transform.
indsd	the individual coefficient variances introduced by the transform.
al	the artificial levels used to estimate the noise variance.
sd	the standard deviation of the noise.
fhat	the estimate of the function after denoising.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[denoise](#)

**Examples**

```
x1<-runif(256)
y1<-make.signal2("doppler",x=x1)
n1<-rnorm(256,0,.1)
z1<-y1+n1
gvec<-c(rep(.4,times=100),rep(.7,times=100),rep(.3,times=56))
#
est1<-denoiseheteroprop(x1,z1,AdaptNeigh,1,TRUE,TRUE,2,"median",gvec)
sum(abs(y1-est1$fhat$coeff))
#
#the error between the true signal and the denoised version.
```

---

dojitter

*dojitter*

---

**Description**

This function adds a random uniform vector of the same length as the input to modify the input.

**Usage**

```
dojitter(x, amount = 0)
```

**Arguments**

x	a vector to be jittered (e.g. a gridpoint vector).
amount	a value of how much to jitter the vector x.

**Details**

The function creates `length(x)` samples from a uniform`[-amount,amount]`, and adds these to the original vector `x`. If `amount=0`, the new vector `jx` is the same as the original vector.

**Value**

`jx`                    the jittered version of `x`

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[modjitter](#)

**Examples**

```
#create grid vector
#
xgrid<-seq(0,1,length=51)
#
xgrid
#
#a regularly-spaced grid
#
dojitter(xgrid,.01)
#
#a jittered grid.
#
```

---

fwtnp

*fwtnp*

---

**Description**

Performs the lifting transform on a signal with grid `input` and corresponding function values `f`. There is a unique correspondence between the grid values and the function values. Can also cope with length vector input instead of gridpoint vector input.

**Usage**

```
fwtnp(input, f, inputtype = "points", nkeep = 2, intercept = TRUE,
initboundhandl = "reflect", updateboundhandl = "add", neighbours = 1,
closest = FALSE, LocalPred = LinearPred)
```

**Arguments**

<code>input</code>	A vector of grid values. Can be of any length, not necessarily equally spaced.
<code>f</code>	A vector of function values corresponding to <code>input</code> . Must be of the same length as <code>input</code> .
<code>inputtype</code>	A character string, either "points" to specify that a gridpoint vector is inputted, or "lengths", when a vector of interval lengths is used as input.
<code>nkeep</code>	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.
<code>intercept</code>	Indicates whether or not the regression curve includes an intercept.
<code>initboundhandl</code>	variable specifying how to handle the boundary at the start of the transform. Possible values are "reflect" - the intervals corresponding to the first and last datapoints are taken to have the respective grid values as midpoints; and "stop" - the first and last intervals have the first and last grid values (respectively) as outer endpoints.
<code>updateboundhandl</code>	variable specifying how the intervals are changed during the update step of the transform ( <a href="#">PointsUpdate</a> ), when the removed point is on the boundary: "reflect" makes the neighbouring intervals symmetric about the corresponding gridpoints; "stop" has the neighbouring intervals ending at the gridpoints; "add" has the neighbouring intervals having outer endpoints where the boundary intervals did.
<code>neighbours</code>	The number of neighbours over which the regression is performed at each step. If <code>closest</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
<code>closest</code>	Refers to the configuration of the chosen neighbours. If <code>closest</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.
<code>LocalPred</code>	The type of regression to be performed. Possible options are <code>LinearPred</code> , <code>QuadPred</code> , <code>CubicPred</code> , <code>AdaptPred</code> and <code>AdaptNeigh</code> .

**Details**

Given  $n$  points on a line, `input`, each with a corresponding `f` value this algorithm computes a lifting transform of the  $(input, f)$  data. If lengths are inputted (`inputtype="lengths"`), then the gridpoints are taken to be the left endpoints of the intervals defined by the lengths inputted. Step One. Order the grid values so that corresponding intervals can be constructed.

Step Two. Compute "integrals" for each point. For each point its integral is the length of the interval associated to the gridpoint.

Step Three. Identify the point to remove as that with the smallest integral. Generally, we remove points in order of smallest to largest integral. The integrals of neighbours of removed points change at each step.

Step Four(a). The neighbours of the removed point are identified using the specified neighbour configuration. The value of `f` at the removed point is predicted using the specified regression curve over the neighbours, unless an adaptive procedure is chosen. In this case, the algorithm adjusts

itself. The difference between the removed point's  $f$  value and the prediction is computed: this is the wavelet coefficient for the removed point. The difference replaces the function value in the vector `coeff` at the removed point's location. In this way wavelet coefficients gradually overwrite (scaling) function values in `coeff`.

Step Four(b). The integrals and the scaling function values (other `coeff` values) of neighbours of the removed point are updated. The values of the rest of the scaling coefficients are unaffected.

Step Five. Return to step 3 but in the identification of a point to remove the updated integrals are used.

The algorithm continues until as many points as desired are removed.

### Value

<code>X</code>	data vector of the grid used in the transform.
<code>coeff</code>	vector of detail and scaling coefficients in the wavelet decomposition of the signal.
<code>origlengths</code>	vector of initial interval lengths corresponding to the gridpoints.
<code>lengths</code>	vector of (updated) interval lengths at the end of the transform. This is of length <code>nkeep</code> .
<code>lengthsremove</code>	vector of interval lengths corresponding to the points removed during the transform (in <code>removelist</code> ).
<code>pointsin</code>	indices into <code>X</code> of the scaling coefficients in the wavelet decomposition. These are the indices of the <code>X</code> values which remain after all points in <code>removelist</code> have been predicted and removed. This has length <code>nkeep</code> .
<code>removelist</code>	a vector of indices into <code>X</code> of the lifted coefficients during the transform (in the order of removal).
<code>neighbrs</code>	a list of indices into <code>X</code> . Each list entry gives the indices of the neighbours of the removed point used at that particular step of the transform.
<code>neighbours</code>	the user-specified number of neighbours used in the prediction step of the transform.
<code>gamlist</code>	a list of all the prediction weights used at each step of the transform.
<code>alphalist</code>	a list of the update coefficients used in the update step of the decomposition.
<code>schemehist</code>	a vector of character strings indicating the type of regression used at each step of the transform.
<code>interhist</code>	a boolean vector indicating whether or not an intercept was used in the regression curve at each step.
<code>clolist</code>	a boolean vector showing whether or not the neighbours were symmetrical (FALSE) about the removed point during the transform. This is NULL except when <code>LocalPred=AdaptNeigh</code> .

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina.Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[AdaptNeigh](#), [AdaptPred](#), [CubicPred](#), [fwtntpmp](#), [invtnp](#), [LinearPred](#), [QuadPred](#)

**Examples**

```
#
# Generate some one-dimensional data: 100 observations.
#
input <- runif(100)
f <- input^2 - 3*input
#
# Compute fwtntp function on this data
#
out <- fwtntp(input,f,LocalPred=AdaptPred,neighbours=2,closest=TRUE)
#
# That's it.
#
```

---

fwtntpmp

*fwtntpmp*

---

**Description**

Performs the lifting transform on a signal with grid `input` and corresponding function values `f`, where `f` has multiple points, that is, more than one function value for (some of) the grid values.

**Usage**

```
fwtntpmp(input, f, inputtype = "points", nkeep = 2, intercept = TRUE,
  initboundhandl = "reflect", updateboundhandl = "add", neighbours = 1,
  closest = FALSE, LocalPredmp = LinearPredmp, mpdet="ave")
```

**Arguments**

<code>input</code>	A vector of grid values. Can be of any length, not necessarily equally spaced.
<code>f</code>	A vector of function values corresponding to <code>input</code> . Must be of the same length as <code>input</code> .
<code>inputtype</code>	A character string, either "points" to specify that a gridpoint vector is inputted, or "lengths", when a vector of interval lengths is used as input.
<code>nkeep</code>	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.
<code>intercept</code>	Indicates whether or not the regression curve includes an intercept.
<code>initboundhandl</code>	variable specifying how to handle the boundary at the start of the transform. Possible values are "reflect" - the intervals corresponding to the first and last datapoints are taken to have the respective grid values as midpoints; and "stop" - the first and last intervals have the first and last grid values (respectively) as outer endpoints.

<code>updateboundhandl</code>	variable specifying how the intervals are changed during the update step of the transform ( <code>PointsUpdate</code> ), when the removed point is on the boundary: <code>"reflect"</code> makes the neighbouring intervals symmetric about the corresponding gridpoints; <code>"stop"</code> has the neighbouring intervals ending at the gridpoints; <code>"add"</code> has the neighbouring intervals having outer endpoints where the boundary intervals did.
<code>neighbours</code>	The number of neighbours over which the regression is performed at each step. If <code>closest</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
<code>closest</code>	Refers to the configuration of the chosen neighbours. If <code>closest</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.
<code>LocalPredmp</code>	The type of regression to be performed. Possible options are <code>LinearPredmp</code> , <code>QuadPredmp</code> , <code>CubicPredmp</code> , <code>AdaptPredmp</code> and <code>AdaptNeighmp</code> .
<code>mpdet</code>	how the mutiple point detail coefficients are computed. Possible values are <code>"ave"</code> , in which the multiple detail coefficients produced when performing the multiple predictions are averaged, or <code>"min"</code> , where the overall minimum detail coefficient is taken.

## Details

Given  $n$  points on a line, `input`, with multiple  $f$  values, this algorithm computes a lifting transform of the  $(input, f)$  data.

Step One. Order the grid values so that corresponding intervals can be constructed, using the average function value at multiple points.

Step Two. Compute "integrals" for each point. For each point its integral is the length of the interval associated to the gridpoint.

Step Three. Identify the point to remove as that with the smallest integral. Generally, we remove points in order of smallest to largest integral. The integrals of neighbours of removed points change at each step.

Step Four(a). The neighbours of the removed point are identified using the specified neighbour configuration. The values of  $f$  at the removed point are predicted using the specified regression curve over the neighbours, unless an adaptive procedure is chosen. In this case, the algorithm adjusts itself. If the removed point has multiple point neighbours, the extra points are used in the regression. The difference between the removed point(s)  $f$  value and the prediction is computed: these are the wavelet coefficient for the removed point. When the removed point is itself a multiple point, this will produce multiple detail coefficients at that point. `mpdet` says how the final detail coefficient for that point is recorded (either averaged or the minimum). The detail replaces the function value in the vector `coeff` at the removed point's location. In this way wavelet coefficients gradually overwrite (scaling) function values in `coeff`.

Step Four(b). The integrals and the scaling function values (other `coeff` and `coefflist` values) of neighbours of the removed point are updated. The values of the rest of the scaling coefficients are unaffected.

Step Five. Return to step 3 but in the identification of a point to remove the updated integrals are used.

The algorithm continues until as many points as desired are removed.

### Value

<code>X</code>	data vector of the grid used in the transform.
<code>coeff</code>	vector of detail and scaling coefficients in the wavelet decomposition of the signal.
<code>coefflist</code>	list of detail and scaling coefficients. Should be the same as <code>coeff</code> , apart from possible multiple points at the scaling function values.
<code>origlengths</code>	vector of initial interval lengths corresponding to the gridpoints.
<code>lengths</code>	vector of (updated) interval lengths at the end of the transform. This is of length <code>nkeep</code> .
<code>lengthsremove</code>	vector of interval lengths corresponding to the points removed during the transform (in <code>removelist</code> ).
<code>pointsin</code>	indices into <code>X</code> of the scaling coefficients in the wavelet decomposition. These are the indices of the <code>X</code> values which remain after all points in <code>removelist</code> have been predicted and removed. This has length <code>nkeep</code> .
<code>removelist</code>	a vector of indices into <code>X</code> of the lifted coefficients during the transform (in the order of removal).
<code>neighbrs</code>	a list of indices into <code>X</code> . Each list entry gives the indices of the neighbours of the removed point used at that particular step of the transform.
<code>neighbours</code>	the user-specified number of neighbours used in the prediction step of the transform.
<code>gamlist</code>	a list of all the prediction weights used at each step of the transform.
<code>alphalist</code>	a list of the update coefficients used in the update step of the decomposition.
<code>schemehist</code>	a vector of character strings indicating the type of regression used at each step of the transform.
<code>interhist</code>	a boolean vector indicating whether or not an intercept was used in the regression curve at each step.
<code>clolist</code>	a boolean vector showing whether or not the neighbours were symmetrical (FALSE) about the removed point during the transform. This is NULL except when <code>LocalPred=AdaptNeigh</code> .
<code>g</code>	a list describing the group structure (indices) of the initial function values.
<code>mp</code>	a boolean vector of which of the groups are actually multiple points.

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina.Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

### See Also

[AdaptNeighmp](#), [AdaptPredmp](#), [CubicPredmp](#), [fwtntp](#), [invtnpmp](#), [LinearPredmp](#), [QuadPredmp](#)

**Examples**

```
#read in multiple point data...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel

out<-fwtnpmp(times,accel,LocalPredmp=AdaptPredmp,neighbours=2)
out$coeff

#these are the detail coefficients of the transform.
```

---

getnbrs

*getnbrs*


---

**Description**

This function uses the user's neighbourhood configuration input to find the neighbours of the lifted datapoint to be used in the prediction step of the transform.

**Usage**

```
getnbrs(X, remove, pointsin, neighbours, closest)
```

**Arguments**

X	The vector of gridpoints.
remove	the index (into X) of the point to be removed.
pointsin	The indices of gridpoints still to be removed.
neighbours	the number of neighbours to find for use in prediction.
closest	Boolean argument: If FALSE, the neighbours selected are the ones on both sides of the removed point.

**Details**

The function uses the value of `neighbours` and `closest` to choose the neighbours to return. If `closest` is FALSE, `pointsin` is used to find `neighbours` indices on both sides of the index of the removed point (`remove`). If `closest` is TRUE, then the function uses the gridpoint vector (X) to calculate distances from the removed point to `neighbours` neighbours on each side of the removed point (if they exist) and then uses this information to choose the closest `neighbours` ones, recording where they lie in relation to the removed point, and accordingly their index can be obtained. If the removed point is on the boundary, then by choice, we take only one neighbour.

**Value**

nbrs	the indices of the neighbours corresponding to the specified configuration.
index	the indices into <code>pointsin</code> of the neighbours

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[fwtnp](#)

**Examples**

```
x1<-runif(20)
#
x1
#
y1<-make.signal2("bumps",x=x1)
#
y1
#
order(x1)
#
# shows where the points lie in relation to each other.
#
neigh<-getnbrs(x1,3,order(x1),4,TRUE)
#
neigh$nbrs
#
# these are the indices of the 4 closest neighbours to point 3.
#
```

---

heterovar

*heterovar*

---

**Description**

Estimates individual wavelet coefficient variances using a sliding window approach.

**Usage**

```
heterovar(y, detail, al)
```

**Arguments**

<code>y</code>	a vector of the gridpoints of <code>removelist</code> after executing the forward transform, in the order of the gridpoint vector.
<code>detail</code>	the vector of detail coefficients after the forward transform has been performed, in the order of the gridpoint vector.
<code>al</code>	The list of indices into <code>removelist</code> divided into artificial levels.

## Details

The function works out the interval endpoints for each gridpoint in `removelist`, based on an initial window length of one fifth of the range of `y`, and then adjusts them so that they lie within the range of `y`. The indices of the `removelist` points inside these intervals are then compared against the indices of the first artificial level for the data. These new indices are then used to compute the individual coefficient variances, based on the detail values of the new indices. If any of the window indices list entries contains less than four values, then the initial window length is increased by 5% and the process redone, until each window contains at least four coefficients.

## Value

<code>ep1</code>	a two-column matrix with the (true) endpoints of the windows from which to calculate the coefficient variances (according to the specified window length).
<code>ep2</code>	a two-column matrix with the endpoints of the windows from which to calculate the coefficient variances (adjusted to be of the window length and within the range of <code>y</code> ).
<code>idlist</code>	a list of indices into <code>y</code> showing the points each interval contains.
<code>newidlist</code>	a list of indices into <code>y</code> showing the points each interval contains, which are also in the first artificial level.
<code>dlist</code>	a list of detail coefficients which correspond to the indices in <code>newidlist</code> .
<code>varvec</code>	a vector of median absolute deviation values (from the median) for the coefficients in <code>dlist</code> .
<code>varvec1</code>	a vector of median absolute deviation values (from the median), centered at zero, for the coefficients in <code>dlist</code> .
<code>varvec2</code>	a vector of median absolute deviation values (from the median), centered at zero, for the coefficients in <code>dlist</code> .

## Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

## See Also

[denoisehetero](#)

## Examples

```
x1<-runif(256)
#
y1<-make.signal2("doppler",x=x1)
#
fwd<-fwtnp(x1,y1,LocalPred=AdaptNeigh,neighbours=2)
#
y<-fwd$lengthsremove
rem<-fwd$removelist
al<-artlev1(y,rem)
#
yrem<-x1[sort(rem)]
```

```

detail<-fwd$coeff[sort(rem)]
#
h<-heterovar(yrem,detail,a1)
#
h$varvec[1:10]
#
#the first ten coefficient variances to be used in the normalisation of the detail
#coefficients

```

---

intervals

*intervals*


---

### Description

This function constructs the intervals around the grid values to be used as scaling integrals during the transform

### Usage

```
intervals(X, initboundhandl)
```

### Arguments

X	The vector of gridpoints.
initboundhandl	the interval construction at the boundary. Takes the value "reflect" for intervals symmetric about the endpoints or "stop" if the endpoint intervals are limited to the edges of the dataset, i.e. the intervals end at the first and last gridpoints respectively.

### Details

The function constructs the intervals by sorting the observed gridpoints. The endpoints of the intervals are found as the midpoints between consecutive (sorted) gridpoints. In this way the intervals are not necessarily centered around the gridpoints. The first and last intervals are then modified according to `initboundhandl` (see above). These intervals represent the support of the initial scaling functions associated to each gridpoint.

### Value

intervals	a vector of length $(\text{length}(X) + 1)$ with the X values of the endpoints of the intervals (including the edges).
order	<code>order(X)</code> (the sorted observation order).

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**[lengthintervals](#)**Examples**

```
x2<-runif(50)
x2
#
intervals(x2,"reflect")
#
#check that the gridpoints are between the interval vector entries...
#
```

---

*invtnp**invtnp*

---

**Description**

Performs the inverse lifting transform on a detail and scaling coefficient vector with grid  $X$  and corresponding coefficients `coeff`. There is a unique correspondence between the grid values and the function values.

**Usage**

```
invtnp(X, coeff, lengths, lengthsremove, pointsin, removelist, neighbors, schemehist,
interhist, nadd = length(X) - 2, intercept = TRUE, neighbours = 1, closest = FALSE,
LocalPred = LinearPred)
```

**Arguments**

<code>X</code>	data vector of the grid used in the transform.
<code>coeff</code>	vector of detail and scaling coefficients in the wavelet decomposition of the signal.
<code>lengths</code>	vector of interval lengths to be used in the update step of the transform. This is of length <code>pointsin</code> .
<code>lengthsremove</code>	vector of interval lengths corresponding to the points removed during the forward transform.
<code>pointsin</code>	indices into <code>X</code> of the scaling coefficients in the wavelet decomposition.
<code>removelist</code>	a vector of indices into <code>X</code> of the lifted coefficients during the transform (in the order of removal).
<code>neighbors</code>	a list of indices into <code>X</code> . Each list entry gives the indices of the neighbours of the removed point used at that particular step of the forward transform.
<code>schemehist</code>	a vector of character strings indicating the type of regression used at each step of the forward transform. This is <code>NULL</code> apart from when <code>AdaptNeigh</code> is to be used in the transform.

<code>interhist</code>	a boolean vector indicating whether or not an intercept was used in the regression curve at each step of the forward transform. This is NULL apart from when <code>AdaptNeigh</code> is to be used in the transform.
<code>nadd</code>	The number of steps to perform of the inverse transform. This corresponds to $(\text{length}(X) - n_{\text{keep}})$ in the forward transform.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform.
<code>neighbours</code>	the number of neighbours in the computation of the predicted value.
<code>closest</code>	a boolean value showing whether or not the neighbours were symmetrical (FALSE) about the removed point during the transform.
<code>LocalPred</code>	The type of regression to be performed. Possible options are <code>LinearPred</code> , <code>QuadPred</code> , <code>CubicPred</code> , <code>AdaptPred</code> and <code>AdaptNeigh</code> .

### Details

This algorithm reconstructs an estimate of a function/signal from information about detail and scaling coefficients in its wavelet decomposition. **Step One.** Extract information about the first point to be added in the transform from the **last** entries in `removelist`, `lengthsremove` and `neighbrs`. Use this information to discover the correct placement of this point in relation to the indices in `pointsin`.

**Step Two.** Using the information about the prediction scheme used in the "forward" transform, use the corresponding version of `LocalPred` to obtain prediction weights and value for the lifted point.

**Step Three.** "Undo" the update step of the transform, and then the prediction step of the transform. The vector of scaling and detail coefficients, as well as the interval lengths are modified accordingly.

**Step Four.** Remove the added point from `removelist`. Update `pointsin` and `lengths` to contain the added point.

**Step Five.** Return to step 1 but in the identification of the next point to add, the second to last entries in (the original) `removelist`, `lengthsremove` and `neighbrs` are used to indentify the point and then place it in `pointsin`.

The algorithm continues like this until as many points as desired are added.

### Value

<code>X</code>	data vector of the grid used in the transform.
<code>coeff</code>	vector of signal function values after inversion.
<code>lengths</code>	vector of interval lengths at the start of the transform. This should be the same as <code>intervals(X)</code> .
<code>lengthsremove</code>	vector of interval lengths corresponding to the points added during the transform.
<code>pointsin</code>	indices into <code>X</code> of the scaling coefficients in the wavelet decomposition. These are the indices of the <code>X</code> values which remain after all points in <code>removelist</code> have been added. For a straight forward-inverse transform implementation, this should be <code>order(X)</code> .

`removelist` a vector of indices into  $X$  of the lifted coefficients during the transform (in the reverse order of how they were added).

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

### See Also

[AdaptNeigh](#), [AdaptPred](#), [CubicPred](#), [fwtnp](#), [invtnpmp](#), [LinearPred](#), [QuadPred](#), [UndoPointsUpdate](#)

### Examples

```
#
# Generate some one-dimensional data: 500 observations.
x2<-runif(500)
f2<-make.signal2("bumps",x=x2)
#
# perform the forward transform...
out<-fwtnp(x2,f2,LocalPred=AdaptPred)
#
# and now invert using the information from out...
#
fhat<-invtnp(x2,out$coeff,out$lengths,out$lengthsremove,out$pointsin,out$removelist,
  out$neighbrs,out$schemehist,out$interhist,LocalPred=AdaptPred)
#
# Now compare the original signal with the reconstruction.
sum(abs(f2-fhat$coeff))
#
```

---

invtnpmp

*invtnpmp*

---

### Description

Performs the inverse lifting transform on a detail and scaling coefficient vector with grid  $X$  and corresponding coefficients `coeff`, based on multiple point information.

### Usage

```
invtnpmp(X, coefflist, coeff, lengths, lengthsremove, pointsin, removelist,
  neighbrs, newneighbrs, schemehist, interhist, nadd = length(X) - 2,
  intercept = TRUE, neighbours = 1, closest = FALSE, LocalPredmp = LinearPredmp, mpc
```

**Arguments**

<code>X</code>	data vector of the grid used in the transform.
<code>coefflist</code>	list of detail and multiple scaling coefficients.
<code>coeff</code>	vector of detail and scaling coefficients in the wavelet decomposition of the signal.
<code>lengths</code>	vector of interval lengths to be used in the update step of the transform. This is of length <code>pointsin</code> .
<code>lengthsremove</code>	vector of interval lengths corresponding to the points removed during the forward transform.
<code>pointsin</code>	indices into <code>X</code> of the scaling coefficients in the wavelet decomposition.
<code>removelist</code>	a vector of indices into <code>X</code> of the lifted coefficients during the transform (in the order of removal).
<code>neighbrs</code>	a list of indices into <code>X</code> . Each list entry gives the indices of the neighbours of the removed point used at that particular step of the forward transform.
<code>newneighbrs</code>	a list of indices into <code>X</code> . Each list entry gives the indices of the multiple neighbours of the removed point used at that particular step of the forward transform.
<code>schemehist</code>	a vector of character strings indicating the type of regression used at each step of the forward transform. This is <code>NULL</code> apart from when <code>AdaptNeigh</code> is to be used in the transform.
<code>interhist</code>	a boolean vector indicating whether or not an intercept was used in the regression curve at each step of the forward transform. This is <code>NULL</code> apart from when <code>AdaptNeigh</code> is to be used in the transform.
<code>nadd</code>	The number of steps to perform of the inverse transform. This corresponds to $(\text{length}(X) - n_{\text{keep}})$ in the forward transform.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform.
<code>neighbours</code>	the number of neighbours in the computation of the predicted value.
<code>closest</code>	a boolean value showing whether or not the neighbours were symmetrical ( <code>FALSE</code> ) about the removed point during the transform.
<code>LocalPredmp</code>	The type of regression to be performed. Possible options are <code>LinearPredmp</code> , <code>QuadPredmp</code> , <code>CubicPredmp</code> , <code>AdaptPredmp</code> and <code>AdaptNeighmp</code> .
<code>mpdet</code>	how the mutiple point detail coefficients are computed. Possible values are "ave", in which the multiple detail coefficients produced when performing the multiple predictions are averaged, or "min", where the overall minimum detail coefficient is taken.

**Details**

This algorithm reconstructs an estimate of a function/signal from information about detail and scaling coefficients in its wavelet decomposition, using the multiple point structure information to estimate the spread of original points. Step One. Extract information about the first point to be added in the transform from the **last** entries in `removelist`, `lengthsremove` and `neighbrs`. Use this information to discover the correct placement of this point in relation to the indices in `pointsin`.

Step Two. Using the information about the prediction scheme used in the "forward" transform, use the corresponding version of `LocalPred` to obtain prediction weights and value for the lifted point.

Step Three. "Undo" the update step of the transform, and then the prediction step of the transform. The vector of scaling and detail coefficients, as well as the interval lengths are modified accordingly.

Step Four. Remove the added point from `removelist`. Update `pointsin` and `lengths` to contain the added point.

Step Five. Return to step 1 but in the identification of the next point to add, the second to last entries in (the original) `removelist`, `lengthsremove` and `neighbrs` are used to indentify the point and then place it in `pointsin`.

The algorithm continues like this until as many points as desired are added.

### Value

<code>X</code>	data vector of the grid used in the transform.
<code>coeff</code>	vector of signal function values after inversion.
<code>lengths</code>	vector of interval lengths at the start of the transform. This should be the same as <code>intervals(X)</code> .
<code>lengthsremove</code>	vector of interval lengths corresponding to the points added during the transform.
<code>pointsin</code>	indices into <code>X</code> of the scaling coefficients in the wavelet decomposition. These are the indices of the <code>X</code> values which remain after all points in <code>removelist</code> have been added. For a straight forward-inverse transform implementation, this should be <code>order(X)</code> .
<code>removelist</code>	a vector of indices into <code>X</code> of the lifted coefficients during the transform (in the reverse order of how they were added).

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

### See Also

[AdaptNeighmp](#), [AdaptPredmp](#), [CubicPredmp](#), [fwtnpmp](#), [invtnp](#), [LinearPredmp](#), [QuadPredmp](#), [UndoPointsUpdatemp](#)

### Examples

```
#read in multiple point data...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel
shortf<-adjustx(times,accel)$sepf

out<-fwtnpmp(times,accel,LocalPredmp=CubicPredmp,neighbours=2)
```

```

inv<-invtnpmp(times, out$coefflist, out$coeff, out$lengths, out$lengthsremove, out$pointsin,
out$removelist, out$neighbrs, out$newneighbrs, out$schemehist, out$interhist, neighbours = 2,
LocalPredmp = CubicPredmp)

sum(abs(shortf-inv$coeff))

```

---

lengthintervals      *lengthintervals*

---

### Description

This function constructs the vector of interval lengths from a vector of interval endpoints.

### Usage

```
lengthintervals(X, I, type = "midpoints", neighbours, closest)
```

### Arguments

<code>X</code>	The vector of gridpoints.
<code>I</code>	a vector of interval endpoints. This is of length $\text{length}(X)+1$ .
<code>type</code>	a character string, either "midpoints" or "average", denoting the way of computing the interval lengths, if <code>closest=TRUE</code> . If "average", then the average neighbour distance is associated as the interval lengths to the gridpoints; otherwise the lengths are associated from the interval vector, <code>I</code> in the obvious way : right endpoint - left endpoint.
<code>neighbours</code>	the number of neighbours to be used in the prediction step of the transform. This is only used if <code>closest=TRUE</code> , since it specifies how many distances to average over when <code>type="average"</code> .
<code>closest</code>	indicates whether the neighbourhood structure to be used in the transform is symmetrical or not. When combined with <code>type="average"</code> , enables the option of average closest neighbour distance as the associated interval lengths to the gridpoints.

### Details

The function computes the interval lengths by finding the differences between the consecutive entries of the supplied interval vector `I`. In the case of the choice of average closest neighbour distance interval association, the method uses the function `getnbrs` to find the initial neighbours of each gridpoint to compute the average distances.

**Value**

`lengths` a vector of length  $(X)$  with the intervals `lengths` associated to the gridpoints.  
`initialnbrs` a matrix with columns `order(X)`, possibly together with the neighbour indices into  $X$  of each gridpoint, if `type="average"`.  
`initialindex` If `closest=TRUE` and `type="average"`, a matrix of dimension `length(X) x neighbours`, showing the indices into `order(X)` of the neighbours of each gridpoint. Otherwise is `NULL`.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[getnbrs](#), [intervals](#)

**Examples**

```
input<-runif(10)
#gridpoint vector
#
I<-intervals(input,"reflect")
#create the interval endpoint vector using the input
#
lengthintervals(input,I,"average",3,TRUE)
#
#computes 'intervals' based on 3 closest neighbour distance averages
#
```

---

LinearPred

*LinearPred*

---

**Description**

This function performs the prediction lifting step using a linear regression curve given a configuration of neighbours.

**Usage**

```
LinearPred(pointsin, X, coeff, nbrs, remove, intercept, neighbours)
```

**Arguments**

`pointsin` The indices of gridpoints still to be removed.  
`X` the vector of grid values.  
`coeff` the vector of detail and scaling coefficients at that step of the transform.  
`nbrs` the indices (into  $X$ ) of the neighbours to be used in the prediction step.

<code>remove</code>	the index (into X) of the point to be removed.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform.
<code>neighbours</code>	the number of neighbours in the computation of the predicted value. This is not actually used specifically in <code>LinearPred</code> , since this is known already from <code>nbrs</code> .

### Details

The procedure performs linear regression using the given neighbours using an intercept if chosen. The regression coefficients (`weights`) are used to predict the new function value at the removed point.

### Value

<code>Xneigh</code>	matrix of X values corresponding to the neighbours of the removed point. The matrix consists of the column <code>X[nbrs]</code> augmented with a column of ones if an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
<code>bhat</code>	The regression coefficients used in prediction.
<code>weights</code>	the prediction weights for the neighbours.
<code>pred</code>	the predicted function value obtained from the regression.
<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

### See Also

[CubicPred](#), [fwtnp](#), [QuadPred](#)

### Examples

```
#
# Generate some doppler data: 500 observations.
#
tx <- runif(500)
ty<-make.signal2("doppler",x=tx)
#
# Compute the neighbours of point 173 (2 neighbours on each side)
#
out<-getnbrs(tx,173,order(tx),2,FALSE)
#
# Perform linear regression based on the neighbours (without intercept)
#
lp<-LinearPred(order(tx),tx,ty,out$nbrs,173,FALSE,2)
#
```

```
#
lp
#
#the regression curve details
```

---

 LinearPredmp

*LinearPredmp*


---

### Description

This function performs the prediction lifting step using a linear regression curve given a configuration of neighbours, for multiple point data.

### Usage

```
LinearPredmp(pointsin, X, coefflist, coeff, nbrs, newnbrs, remove, intercept,
neighbours, mpdet, g)
```

### Arguments

<code>pointsin</code>	The indices of gridpoints still to be removed.
<code>X</code>	the vector of grid values.
<code>coeff</code>	the vector of detail and scaling coefficients at that step of the transform.
<code>coefflist</code>	the list of detail and multiple scaling coefficients at that step of the transform.
<code>nbrs</code>	the indices (into X) of the neighbours to be used in the prediction step.
<code>newnbrs</code>	as <code>nbrs</code> , but repeated according to the multiple point structure of the grid.
<code>remove</code>	the index (into X) of the point to be removed.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform.
<code>neighbours</code>	the number of neighbours in the computation of the predicted value. This is not actually used specifically in <code>LinearPredmp</code> , since this is known already from <code>nbrs</code> .
<code>mpdet</code>	how the mutiple point detail coefficients are computed. Possible values are "ave", in which the multiple detail coefficients produced when performing the multiple predictions are averaged, or "min", where the overall minimum detail coefficient is taken. Note that this is taken to standardise the input when <code>LocalPredmp</code> is called.
<code>g</code>	the group structure of the multiple point data. Note that this is taken to standardise the input when <code>LocalPredmp</code> is called.

### Details

The procedure performs linear regression using the given neighbours using an intercept if chosen. The regression coefficients (`weights`) are used to predict the new function value at the removed point.

**Value**

Xneigh	matrix of X values corresponding to the neighbours of the removed point. The matrix consists of the column X[newnbrs] augmented with a column of ones if an intercept is used. Refer to any reference on linear regression for more details.
mm	the matrix from which the prediction is made. In terms of Xneigh, it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
bhat	The regression coefficients used in prediction.
weights	the prediction weights for the neighbours.
pred	the predicted function value obtained from the regression.
coeff	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.

**Note**

The **Matrix** is needed for this function.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[CubicPredmp](#), [fwtnpmp](#), [QuadPredmp](#)

**Examples**

```
#read in data with multiple values...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel
short<-adjustx(times,accel,"mean")
X<-short$sepx
coeff<-short$ssepx
g<-short$g

coefflist<-list()
for (i in 1:length(g)){
  coefflist[[i]]<-accel[g[[i]]]
}

#work out neighbours of point to be removed (31)

out<-getnbrs(X,31,order(X),2,TRUE)
nbrs<-out$n

nbrs

newnbrs<-NULL
for (i in 1:length(nbrs)){
```

```
newnbrs<-c(newnbrs,rep(nbrs[i],times=length(g[[nbrs[i]]])))
}

#work out repeated neighbours using g...
newnbrs

LinearPredmp(order(X),X,coefflist,coeff,nbrs,newnbrs,31,TRUE,2,"ave",g)
```

---

```
make.signal2      make.signal2
```

---

### Description

This function computes signal function values based on a grid input.

### Usage

```
make.signal2(name, x, snr = Inf, ...)
```

### Arguments

name	a character string of the test signal to create.
x	a vector of gridpoints.
snr	
...	

### Details

This function is based on the `make.signal` function included in the **S-Plus wavelets** module, except that the `x` vector can be irregular. As well as the signals included for the original version (e.g. the Donoho/Johnstone test signals), a piecewise polynomial can be sampled.

### Value

z	the signal function values.
---	-----------------------------

### Note

The test signals have domain  $[0,1]$ , so the grid vector `x` must have values within this interval.

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**Examples**

```
#create grid vector
#
xgrid<-rnorm(50)
xgrid
#
pp<-make.signal2("ppoly",x=xgrid)
#
#piecewise polynomial data vector
#
plot(sort(xgrid),pp[order(xgrid)],type="l")
#
```

matcond

*matcond***Description**

Works out two alternative condition numbers for the transform associated to the prediction scheme given in the arguments to the function.

**Usage**

```
matcond(x, f, Pred, neigh, int, clo, keep)
```

**Arguments**

x	A vector of grid values. Can be of any length, not necessarily equally spaced.
f	A vector of function values corresponding to x. Must be of the same length as x.
Pred	The type of regression to be performed. Possible options are <a href="#">LinearPred</a> , <a href="#">QuadPred</a> , <a href="#">CubicPred</a> , <a href="#">AdaptPred</a> and <a href="#">AdaptNeigh</a> .
neigh	The number of neighbours over which the regression is performed at each step. If <code>clo</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
int	Indicates whether or not the regression curve includes an intercept.
clo	Refers to the configuration of the chosen neighbours. If <code>clo</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.
keep	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.

**Details**

The function uses the transform matrices to work out their norms and singular value decompositions. Condition numbers are calculated by  $\|T_j\| * \|T_j^{-1}\|$  and  $\text{svd}\$d[1] / \text{svd}\$d[\text{nrow}(T\_j)]$  respectively.

**Value**

cno	the condition numbers for the augmented transform matrices, calculated using the Frobenius norm (see condno).
v	the condition numbers for the augmented transform matrices, calculated using the ratio between the largest to the smallest singular values in the singular value decomposition of the augmented matrices.
a	the transform matrix information for the transform (output from transmatdual).

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[condno](#), [transmatdual](#)

**Examples**

```
x1<-runif(256)
y1<-make.signal2("doppler",x=x1)
#
m<-matcond(x1,y1,AdaptNeigh,2,TRUE,TRUE,2)
#
m$cno
#
m$v
# shows the two different condition number measures for the matrix associated
# to the transform performed.
#
```

---

modjitter

*modjitter*

---

**Description**

This function jitters grid values by a proportion of the regular distance between consecutive grid-points and then alters it to lie in [0,1].

**Usage**

```
modjitter(x, amount)
```

**Arguments**

x	a vector to be jittered (e.g. a gridpoint vector).
amount	a value of how much to jitter the vector (expressed as a proportion of the regular gridpoint distance, d).

**Details**

The function uses `dojitter` to jitter the gridpoint vector by  $(\text{amount} * d)$ . The endpoints are fixed to be zero and one, and the corresponding `jx` values to `x[2]` and `x[length(x)-1]` are randomised again in the intervals  $[0, x[2] + \text{amount} * d]$  and  $[x[length(x)-1] - \text{amount} * d, 1]$  respectively.

**Value**

`jx` the jittered version of `x`

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[dojitter](#), [make.signal2](#)

**Examples**

```
#create grid vector
#
xgrid<-seq(0,1,length=51)
#
xgrid
#
#a regularly-spaced grid on [0,1]
#
modjitter(xgrid,1)
#
#jitters xgrid with a maximum change of .02, keeping endpoints of zero and one
```

---

motorcycledata      *Motorcycle data.*

---

**Description**

This table gives the results of 133 simulations showing the effects of motorcycle crashes on victims heads: time after a simulated impact with motorcycles and head acceleration of a PTMO (post mortem human test object) were recorded.

**Usage**

```
data(motorcycledata)
```

**Format**

A 133 by 2 data frame.

## References

Hardle, W. (1990) *Applied Nonparametric Regression*. Cambridge University Press.

---

PointsUpdate	<i>PointsUpdate</i>
--------------	---------------------

---

## Description

This function performs the update lifting step using a given configuration of neighbours and boundary handling.

## Usage

```
PointsUpdate(X, coeff, nbrs, index, remove, pointsin, weights, lengths,
             updateboundhandl)
```

## Arguments

X	the vector of grid values.
coeff	the vector of detail and scaling coefficients at that step of the transform.
nbrs	the indices (into X) of the neighbours to be used in the lifting step.
index	the indices into <code>pointsin</code> of <code>nbrs</code> , the neighbours of <code>remove</code> .
remove	the index (into X) of the point to be removed.
pointsin	The indices of gridpoints still to be removed.
weights	the prediction weights obtained from the regression in the prediction step of the transform.
lengths	the vector of interval lengths at the present step of the transform (to be updated).
updateboundhandl	boundary handling in the update step. Possible values are "reflect", "stop" and "add". If the point to be removed is at the boundary, "reflect" updates the neighbour interval to be symmetrical about its gridpoint; "stop" extends its length up until the boundary gridpoint; and "add" increases its interval length by the interval length associated to the removed boundary point.

## Details

The procedure performs a minimum norm update lifting step. Firstly the interval lengths are updated using the coefficients obtained. Secondly, the scaling and detail coefficient vector is modified using the new interval lengths.

**Value**

<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the next step of the transform.
<code>lengths</code>	the vector of interval lengths after the update step of the transform.
<code>r</code>	the index into <code>pointsin</code> of <code>remove</code> .
<code>N</code>	<code>length(pointsin)</code> .
<code>weights</code>	The regression coefficients used in prediction.
<code>alpha</code>	the update weights used to update <code>lengths</code> and <code>coeff</code> .

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[AdaptNeigh](#), [AdaptPred](#), [CubicPred](#), [fwtnp](#), [LinearPred](#), [QuadPred](#), [UndoPointsUpdate](#)

**Examples**

```
#
# Generate some blocks data: 100 observations.
#
x <- runif(100)
y <- make.signal2("blocks", x=x)
#
# find initial interval lengths...
#
I <- intervals(x, "reflect")
l <- lengthintervals(x, I$intervals, neighbours=2, closest=FALSE)
lengths <- l$lengths
#
# perform prediction step...
p <- AdaptNeigh(order(x), x, y, 32, 5, TRUE, 2)
#
#
u <- PointsUpdate(x, p$results[[6]], p$newinfo[[3]], p$newinfo[[4]], 5, order(x), p$results[[4]],
lengths, "add")
#
# and here are the updated coefficients...
u$coeff
#
```

---

PointsUpdatemp      *PointsUpdatemp*

---

### Description

This function performs the update lifting step using a given configuration of neighbours and boundary handling.

### Usage

```
PointsUpdatemp(X, coeff, nbrs, newnbrs, index, remove, pointsin, weights, lengths,
               updateboundhandl)
```

### Arguments

X	the vector of grid values.
coeff	the vector of detail and scaling coefficients at that step of the transform.
nbrs	the indices (into X) of the neighbours to be used in the lifting step.
newnbrs	as nbrs, but repeated according to the multiple point structure of the grid.
index	the indices into pointsin of nbrs, the neighbours of remove.
remove	the index (into X) of the point to be removed.
pointsin	The indices of gridpoints still to be removed.
weights	the prediction weights obtained from the regression in the prediction step of the transform.
lengths	the vector of interval lengths at the present step of the transform (to be updated).
updateboundhandl	boundary handling in the update step. Possible values are "reflect", "stop" and "add". If the point to be removed is at the boundary, "reflect" updates the neighbour interval to be symmetrical about its gridpoint; "stop" extends its length up until the boundary gridpoint; and "add" increases its interval length by the interval length associated to the removed boundary point.

### Details

The procedure performs a minimum norm update lifting step. Firstly the interval lengths are updated using the coefficients obtained. Secondly, the scaling and detail coefficient list is modified using the new interval lengths.

### Value

coeff	vector of (modified) detail and scaling coefficients to be used in the next step of the transform.
lengths	the vector of interval lengths after the update step of the transform.
r	the index into pointsin of remove.

N	length(pointsin).
weights	The regression coefficients used in prediction.
alpha	the update weights used to update lengths and coeff.

**Author(s)**

Matt Nunes (([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk))), Marina Popa (([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk)))

**See Also**

[AdaptNeighmp](#), [AdaptPredmp](#), [CubicPredmp](#), [fwtntpmp](#), [LinearPredmp](#), [QuadPredmp](#), [UndoPointsUpdatemp](#)

**Examples**

```
#read in data with multiple values...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel

short<-adjustx(times,accel,"mean")
X<-short$sepx
coeff<-short$ssepx
g<-short$g

coefflist<-list()
for (i in 1:length(g)){
coefflist[[i]]<-accel[g[[i]]]
}

I<-intervals(X,"reflect")
l<-lengthintervals(X,I$intervals,neighbours=2,closest=TRUE)
lengths<-l$lengths

#work out neighbours of point to be removed (31)

out<-getnbrs(X,31,order(X),2,TRUE)
nbrs<-out$n

nbrs

newnbrs<-NULL
for (i in 1:length(nbrs)){
newnbrs<-c(newnbrs,rep(nbrs[i],times=length(g[[nbrs[i]]])))
}

#work out repeated neighbours using g...
newnbrs

p<-AdaptNeighmp(order(X),X,coefflist,coeff,nbrs,newnbrs,31,TRUE,2,"ave",g)
```

```

nbrs<-p$newinfo[[3]]
nbrs
newnbrs<-NULL
for (i in 1:length(nbrs)){
newnbrs<-c(newnbrs,rep(nbrs[i],times=length(g[[nbrs[i]]])))
}
newnbrs

coefflist[[31]]<-p$results[[6]][31]

u<-PointsUpdatemp(X,coefflist,p$newinfo[[3]],newnbrs,p$newinfo[[4]],31,order(X),p$results[[4],lengths,"add")
#
#and here is the updated coefficient list...
u$coeff

```

---

postmean.cauchy      *postmean.cauchy*

---

### Description

Posterior mean calculations for Bayesian thresholding.

### Details

This function replaces one in the EbayesThresh package, which perform Bayesian thresholding. For more information, see help by Silverman (see references below).

### References

Johnstone, I.M. and Silverman, B.W. (2002) EbayesThresh: R and S-PLUS software for Empirical Bayes thresholding (Submitted for publication).

Johnstone, I.M. and Silverman, B.W. (2004) Needles and hay in haystacks: Empirical Bayes estimates of possibly sparse sequences. *Ann. Statist.*, **32**, 1594–1649.

Bernard Silverman's website: <http://www.stats.ox.ac.uk/~silverma/ebayesthresh/>.

### See Also

[denoise](#), [denoisehetero](#), [denoiseheteromp](#), [denoiseheteroprop](#)

---

pts	<i>pts</i>
-----	------------

---

### Description

This function constructs the gridpoints (X values) from a vector of lengths, with optional starting point.

### Usage

```
pts(input)
```

### Arguments

`input` a vector containing the X starting point in the first entry, and the distances between observed values. `input[1]` can be NA.

### Details

The function constructs the grid values using the first entry in `input` as the first gridpoint, consecutively adding the given lengths in the rest of `input` to find the other gridpoints. If `input[1]` is NA, then the starting point of the gridpoint vector X is taken to be  $\text{sum}(\text{lengths}) / (\text{length}(\text{lengths}) + 1)$ .

### Value

`lengths` a vector of length  $(\text{length}(\text{input}) - 1)$  with the distances between the constructed gridpoints (this is `input[2:length(input)]`).

`X` the constructed gridpoint vector.

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

### See Also

[fwtnp](#), [fwtnpmp](#)

### Examples

```
y<-runif(20)
y
#
y1<-c(0,y)
y2<-c(NA,y)
#
#
pts(y1)$X
#the gridpoints, with specified startpoint of zero.
#
```

```
#
pts(y2)$X
#the grid vector with unspecified startpoint.
#
```

---

QuadPred

*QuadPred*


---

### Description

This function performs the prediction lifting step using a quadratic regression curve given a configuration of neighbours.

### Usage

```
QuadPred(pointsin, X, coeff, nbrs, remove, intercept, neighbours)
```

### Arguments

<code>pointsin</code>	The indices of gridpoints still to be removed.
<code>X</code>	the vector of grid values.
<code>coeff</code>	the vector of detail and scaling coefficients at that step of the transform.
<code>nbrs</code>	the indices (into X) of the neighbours to be used in the prediction step.
<code>remove</code>	the index (into X) of the point to be removed.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform.
<code>neighbours</code>	the number of neighbours in the computation of the predicted value. This is not actually used specifically in <code>QuadPred</code> , since this is known already from <code>nbrs</code> .

### Details

The procedure performs quadratic regression using the given neighbours using an intercept if chosen. The regression coefficients (`weights`) are used to predict the new function value at the removed point. If there are not enough neighbours to generate a quadratic regression curve, the order of prediction is decreased down to `LinearPred`.

### Value

<code>Xneigh</code>	matrix of X values corresponding to the neighbours of the removed point. The matrix consists of columns $X[nbrs]$ , $X[nbrs]^2$ , augmented with a column of ones if an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .

bhat	The regression coefficients used in prediction.
weights	the prediction weights for the neighbours.
pred	the predicted function value obtained from the regression.
coeff	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[CubicPred](#), [fwtnp](#), [LinearPred](#)

**Examples**

```
#
# Generate some doppler data: 500 observations.
#
tx <- runif(500)
ty<-make.signal2("doppler",x=tx)
#
# Compute the neighbours of point 173 (2 neighbours on each side)
#
out<-getnbrs(tx,173,order(tx),2,FALSE)
#
# Perform quadratic prediction based on the neighbours (without intercept)
#
qp<-QuadPred(order(tx),tx,ty,out$nbrs,173,FALSE,2)
#
#
qp[3:5]
#
#the regression curve details
```

---

QuadPredmp

*QuadPredmp*

---

**Description**

This function performs the prediction lifting step using a quadratic regression curve given a configuration of neighbours, for multiple point data.

**Usage**

```
QuadPredmp(pointsin, X, coefflist, coeff, nbrs, newnbrs, remove, intercept,
  neighbours, mpdet, g)
```

**Arguments**

<code>pointsin</code>	The indices of gridpoints still to be removed.
<code>X</code>	the vector of grid values.
<code>coeff</code>	the vector of detail and scaling coefficients at that step of the transform.
<code>coefflist</code>	the list of detail and multiple scaling coefficients at that step of the transform.
<code>nbrs</code>	the indices (into <code>X</code> ) of the neighbours to be used in the prediction step.
<code>newnbrs</code>	as <code>nbrs</code> , but repeated according to the multiple point structure of the grid.
<code>remove</code>	the index (into <code>X</code> ) of the point to be removed.
<code>intercept</code>	Boolean value for whether or not an intercept is used in the prediction step of the transform.
<code>neighbours</code>	the number of neighbours in the computation of the predicted value. This is not actually used specifically in <code>QuadPredmp</code> , since this is known already from <code>nbrs</code> .
<code>mpdet</code>	how the mutple point detail coefficients are computed. Possible values are "ave", in which the multiple detail coefficients produced when performing the multiple predictions are averaged, or "min", where the overall minimum detail coefficient is taken. Note that this is taken to standardise the input when <code>LocalPredmp</code> is called.
<code>g</code>	the group structure of the multiple point data. Note that this is taken to standardise the input when <code>LocalPredmp</code> is called.

**Details**

The procedure performs quadratic regression using the given neighbours using an intercept if chosen. The regression coefficients (`weights`) are used to predict the new function value at the removed point.

**Value**

<code>Xneigh</code>	matrix of <code>X</code> values corresponding to the neighbours of the removed point. The matrix consists of the column <code>X[newnbrs]</code> augmented with a column of ones if an intercept is used. Refer to any reference on linear regression for more details.
<code>mm</code>	the matrix from which the prediction is made. In terms of <code>Xneigh</code> , it is $(Xneigh^T Xneigh)^{-1} Xneigh^T$ .
<code>bhat</code>	The regression coefficients used in prediction.
<code>weights</code>	the prediction weights for the neighbours.
<code>pred</code>	the predicted function value obtained from the regression.
<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used in the update step of the transform.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[CubicPredmp](#), [fwtnpmp](#), [LinearPredmp](#)

**Examples**

```
#read in data with multiple values...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel
short<-adjustx(times,accel,"mean")
X<-short$sepx
coeff<-short$sepx
g<-short$g

coefflist<-list()
for (i in 1:length(g)){
coefflist[[i]]<-accel[g[[i]]]
}

#work out neighbours of point to be removed (31)

out<-getnbrs(X,31,order(X),2,TRUE)
nbrs<-out$n

nbrs

newnbrs<-NULL
for (i in 1:length(nbrs)){
newnbrs<-c(newnbrs,rep(nbrs[i],times=length(g[[nbrs[i]]])))
}

#work out repeated neighbours using g...
newnbrs

QuadPredmp(order(X),X,coefflist,coeff,nbrs,newnbrs,31,TRUE,2,"ave",g)
```

---

Rmatsolve

*Rmatsolve*


---

**Description**

This function calculates matrix inverses for symmetric matrices.

**Usage**

```
Rmatsolve(m)
```

**Arguments**

`m` a (symmetric) matrix.

**Details**

This function uses the eigenvalue decomposition of a matrix `m` to work out its inverse. The function is used here since standard matrix inverse algorithms do not cope well with matrices which are near singular (this often happens in the regression stages of the forward transforms).

**Value**

`inv` the matrix inverse of `m`.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**Examples**

```
#
#create a 4x4 matrix
m<-matrix(runif(16),4,4)

temp<-crossprod(m)

#i.e. temp is t(m)

inv<-Rmatsolve(temp)
```

---

transmatdual

*transmatdual*

---

**Description**

Works out the transform matrix for a particular prediction scheme and neighbourhood structure.

**Usage**

```
transmatdual(x, f, Pred = AdaptNeigh, neigh = 1, int = TRUE, clo = TRUE,
  keep = 2)
```

**Arguments**

<code>x</code>	A vector of grid values. Can be of any length, not necessarily equally spaced.
<code>f</code>	A vector of function values corresponding to <code>x</code> . Must be of the same length as <code>x</code> .
<code>Pred</code>	The type of regression to be performed. Possible options are <a href="#">LinearPred</a> , <a href="#">QuadPred</a> , <a href="#">CubicPred</a> , <a href="#">AdaptPred</a> and <a href="#">AdaptNeigh</a> .
<code>neigh</code>	The number of neighbours over which the regression is performed at each step. If <code>clo</code> is false, then this in fact denotes the number of neighbours on each side of the removed point.
<code>int</code>	Indicates whether or not the regression curve includes an intercept.
<code>clo</code>	Refers to the configuration of the chosen neighbours. If <code>clo</code> is false, the neighbours will be chosen symmetrically around the removed point. Otherwise, the closest neighbours will be chosen.
<code>keep</code>	The number of scaling coefficients to be kept in the final representation of the initial signal. This must be at least two.

**Details**

The function uses `Amatdual` to form the refinement matrices  $A_j$ , from which the augmented matrices  $T_j$  are constructed. This process is iterated, to find the transform matrix (the top level augmented matrix). The rows and columns of this matrix are then reordered to be in the order of `out$coeff`, i.e. so that the columns correspond to  $f_1 \dots f_n$ .

**Value**

<code>out</code>	the output from the forward transform.
<code>Wnew</code>	the matrix associated to the wavelet transform.
<code>x</code>	the original gridpoint vector.

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[Amatdual](#)

**Examples**

```
x1<-runif(10)
y1<-make.signal2("doppler",x=x1)
#
a<-transmatdual(x1,y1,AdaptNeigh,2,TRUE,TRUE,2)
#
a$Wnew
#
#the transform matrix for this adaptive lifting scheme
```

---

UndoPointsUpdate     *UndoPointsUpdate*

---

### Description

This function undoes the update lifting step in the inverse transform.

### Usage

```
UndoPointsUpdate(X, coeff, nbrs, index, remove, r, N, pointsin, gamweights,
                 lengths, lengthrem)
```

### Arguments

X	the vector of grid values.
coeff	the vector of detail and scaling coefficients at that step of the transform.
nbrs	the indices (into X) of the neighbours to be used in the lifting step.
index	the indices into pointsin of nbrs, the neighbours of remove, the point to be added.
remove	the index (into X) of the point to be added.
r	the index into pointsin of the added point, remove.
N	length(pointsin).
pointsin	The indices of gridpoints still to be added.
gamweights	the prediction weights obtained from the regression in the prediction step of the transform.
lengths	the vector of interval lengths at the present step of the transform.
lengthrem	the interval length associated to the point to be added.

### Details

This procedure uses minimum norm update coefficients to invert the update step of the transform. The prediction weights are used to change the interval lengths before the update weights are used to modify `coeff`.

### Value

coeff	vector of (modified) detail and scaling coefficients to be used later in the transform.
lengths	vector of interval lengths after inverting the update step of the transform.
alpha	the weights used to modify lengths and coeff.

### Author(s)

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[AdaptNeigh](#), [AdaptPred](#), [CubicPred](#), [invtnp](#), [LinearPred](#), [PointsUpdate](#), [QuadPred](#)

**Examples**

```
#
# Generate some blocks data: 100 observations.
#
x <- runif(100)
y <-make.signal2("blocks",x=x)
#
#find initial interval lengths...
#
I<-intervals(x,"reflect")
l<-lengthintervals(x,I$intervals,neighbours=2,closest=FALSE)
lengths<-l$lengths
#
#perform prediction step...
p<-AdaptNeigh(order(x),x,y,32,5,TRUE,2)
#
#
u<-PointsUpdate(x,p$results[[6]],p$newinfo[[3]],p$newinfo[[4]],5,order(x),p$results[[4]],
lengths,"add")
#
p2<-setdiff(order(x),5)
a<-which(order(x)==5)
l2<-lengths[setdiff(1:100, a)]
#
#remove the lifted coefficient
#
#now undo the update step...
#
undo<-UndoPointsUpdate(x,u$coeff,p$newinfo[[3]],p$newinfo[[4]],5,a,99,p2,p$results[[4]],l2,
lengths[a])
#
```

---

UndoPointsUpdatemp *UndoPointsUpdatemp*

---

**Description**

This function undoes the update lifting step in the multiple observation inverse transform.

**Usage**

```
UndoPointsUpdatemp(X, coeff, nbrs, newnbrs, index, remove, r, N, pointsin,
gamweights, lengths, lengthrem)
```

**Arguments**

<code>X</code>	the vector of grid values.
<code>coeff</code>	the vector of detail and scaling coefficients at that step of the transform.
<code>nbrs</code>	the indices (into <code>X</code> ) of the neighbours to be used in the lifting step.
<code>newnbrs</code>	as <code>nbrs</code> , but repeated according to the multiple point structure of the grid.
<code>index</code>	the indices into <code>pointsin</code> of <code>nbrs</code> , the neighbours of <code>remove</code> , the point to be added.
<code>remove</code>	the index (into <code>X</code> ) of the point to be added.
<code>r</code>	the index into <code>pointsin</code> of the added point, <code>remove</code> .
<code>N</code>	<code>length(pointsin)</code> .
<code>pointsin</code>	The indices of gridpoints still to be added.
<code>gamweights</code>	the prediction weights obtained from the regression in the prediction step of the transform.
<code>lengths</code>	the vector of interval lengths at the present step of the transform.
<code>lengthrem</code>	the interval length associated to the point to be added.

**Details**

This procedure uses minimum norm update coefficients to invert the update step of the transform. The prediction weights are used to change the interval lengths before the update weights are used to modify `coefflist`.

**Value**

<code>coeff</code>	vector of (modified) detail and scaling coefficients to be used later in the transform.
<code>lengths</code>	vector of interval lengths after inverting the update step of the transform.
<code>alpha</code>	the weights used to modify <code>lengths</code> and <code>coeff</code> .

**Author(s)**

Matt Nunes ([matt.nunes@bristol.ac.uk](mailto:matt.nunes@bristol.ac.uk)), Marina Popa ([Marina.Popa@bristol.ac.uk](mailto:Marina.Popa@bristol.ac.uk))

**See Also**

[AdaptNeighmp](#), [AdaptPredmp](#), [CubicPredmp](#), [invtnpmp](#), [LinearPredmp](#), [PointsUpdatemp](#), [QuadPredmp](#)

**Examples**

```
#read in data with multiple values...

data(motorcycledata)
times<-motorcycledata$time
accel<-motorcycledata$accel
```

```

short<-adjustx(times, accel, "mean")
X<-short$sepx
coeff<-short$sepx
g<-short$g

coefflist<-list()
for (i in 1:length(g)){
coefflist[[i]]<-accel[g[[i]]]
}

I<-intervals(X, "reflect")
l<-lengthintervals(X, I$intervals, neighbours=2, closest=TRUE)
lengths<-l$lengths

#work out neighbours of point to be removed (31)

out<-getnbrs(X, 31, order(X), 2, TRUE)
nbrs<-out$n

nbrs

newnbrs<-NULL
for (i in 1:length(nbrs)){
newnbrs<-c(newnbrs, rep(nbrs[i], times=length(g[[nbrs[i]]])))
}

#work out repeated neighbours using g...
newnbrs

p<-AdaptNeighmp(order(X), X, coefflist, coeff, nbrs, newnbrs, 31, TRUE, 2, "ave", g)

nbrs<-p$newinfo[[3]]
newnbrs<-NULL
for (i in 1:length(nbrs)){
newnbrs<-c(newnbrs, rep(nbrs[i], times=length(g[[nbrs[i]]])))
}
coefflist[[31]]<-p$results[[6]][31]

u<-PointsUpdatemp(X, coefflist, p$newinfo[[3]], newnbrs, p$newinfo[[4]], 31, order(X), p$results[[4], lengths, "add")

p2<-setdiff(order(X), 31)
a<-which(order(X)==31)
l2<-lengths[setdiff(1:length(X), a)]
#
#remove the lifted coefficient
#
#now undo the update step...
#
undo<-UndoPointsUpdatemp(X, coeff, newnbrs, p$newinfo[[3]], p$newinfo[[4]], 31, a, length(X)-1, p2, p$results[[4], l2, lengths[a])
#

```

# Index

## \*Topic **algebra**

condno, 19  
matcond, 52

## \*Topic **arith**

adjustx, 11  
artlev1, 13  
dojitter, 30  
getnbrs, 37  
heterovar, 38  
intervals, 40  
lengthintervals, 46  
modjitter, 53  
PointsUpdate, 55  
PointsUpdatemp, 57  
pts, 60  
UndoPointsUpdate, 67  
UndoPointsUpdatemp, 68

## \*Topic **array**

Amatdual, 12  
condno, 19  
matcond, 52  
Rmatsolve, 64  
transmatdual, 65

## \*Topic **datagen**

make.signal2, 51

## \*Topic **datasets**

motorcycledata, 54

## \*Topic **graphs**

basisfns, 17

## \*Topic **manip**

adjustx, 11  
as.column, 15  
as.row, 16  
dojitter, 30  
modjitter, 53

## \*Topic **methods**

fwtnp, 31  
fwtnpmp, 34  
invtnp, 41

invtnpmp, 43

postmean.cauchy, 59

## \*Topic **regression**

AdaptNeigh, 1  
AdaptNeighmp, 4  
AdaptPred, 6  
AdaptPredmp, 8  
CubicPred, 20  
CubicPredmp, 22  
denoise, 24  
denoisehetero, 25  
denoiseheteromp, 27  
denoiseheteroprop, 29  
LinearPred, 47  
LinearPredmp, 49  
QuadPred, 61  
QuadPredmp, 62

## \*Topic **smooth**

denoise, 24  
denoisehetero, 25  
denoiseheteromp, 27  
denoiseheteroprop, 29

AdaptNeigh, 1, 8, 17, 24, 26, 27, 29, 34, 43,  
52, 56, 66, 68

AdaptNeighmp, 4, 10, 36, 45, 58, 69

AdaptPred, 3, 6, 17, 24, 26, 27, 29, 34, 43,  
52, 56, 66, 68

AdaptPredmp, 5, 8, 36, 45, 58, 69

adjustx, 11

Amatdual, 12, 66

artlev1, 13

as.column, 15, 16

as.row, 15, 16

basisfns, 17

condno, 19, 53

CubicPred, 8, 17, 20, 24, 26, 27, 29, 34, 43,  
48, 52, 56, 62, 66, 68

CubicPredmp, 10, 22, 36, 45, 50, 58, 64, 69

denoise, 14, 24, 27, 30, 59  
denoisehetero, 25, 25, 28, 39, 59  
denoiseheteromp, 27, 59  
denoiseheteroprop, 29, 59  
dojitter, 30, 54

fwtnp, 3, 8, 11, 18, 21, 31, 36, 38, 43, 48, 56,  
60, 62  
fwtnpmp, 5, 10, 23, 34, 34, 45, 50, 58, 60, 64

getnbrs, 37, 47

heterovar, 27, 28, 38

intervals, 40, 47  
invtnp, 34, 41, 45, 68  
invtnpmp, 36, 43, 43, 69

lengthintervals, 41, 46  
LinearPred, 8, 17, 21, 24, 26, 27, 29, 34,  
43, 47, 52, 56, 62, 66, 68  
LinearPredmp, 10, 23, 36, 45, 49, 58, 64,  
69

make.signal2, 51, 54  
matcond, 52  
modjitter, 31, 53  
motorcycledata, 54

PointsUpdate, 32, 35, 55, 68  
PointsUpdatemp, 57, 69  
postmean.cauchy, 59  
pts, 60

QuadPred, 8, 17, 21, 24, 26, 27, 29, 34, 43,  
48, 52, 56, 61, 66, 68  
QuadPredmp, 10, 23, 36, 45, 50, 58, 62, 69

Rmatsolve, 64

transmatdual, 13, 18, 53, 65

UndoPointsUpdate, 43, 56, 67  
UndoPointsUpdatemp, 45, 58, 68