# Package 'wavFeatExt'

November 12, 2026

**Type** Package

**Title** Wavelet-based Feature Extraction for Copy-number Alteration Data

**Version** 1.0.0

**Description** Provides tools for simulating copy-number alteration (CNA) profiles, applying a non-decimated Haar wavelet transform to genomic signals, and extracting wavelet-derived features for use in supervised learning. Multiple machine learning methods including lasso and elastic-net regularisation, random forest, partial least squares, neural networks and k-nearest neighbours are implemented to train predictive models from genomic feature vectors. The workflow enables end-to-end analysis from CNA simulation to feature extraction and classification.

**License** GPL-3

**Encoding** UTF-8

**LazyData** false

**Depends** R (>= 4.6)

**Imports** DNAcopy, wavethresh, MASS, randomForest, glmnet, pROC, neuralnet, e1071, class, caret, ica, stats, graphics, utils, pls, matrixStats

**Suggests** BiocStyle, knitr, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**biocViews** CopyNumberVariation, GenomicVariation, FeatureExtraction, Classification,

**URL** https://github.com/maharaniau/wavFeatExt

**BugReports** https://github.com/maharaniau/wavFeatExt/issues

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Config/testthat/edition** 3

**git_url** https://git.bioconductor.org/packages/wavFeatExt

**git_branch** RELEASE_3_23

**git_last_commit** 6656f3d

**git_last_commit_date** 2026-04-28

**Repository** Bioconductor 3.23

**Date/Publication** 2026-11-11

**Author** Maharani Ahsani Ummi [aut, cre],
Arief Gusnanto [aut]

**Maintainer** Maharani Ahsani Ummi <maharaniahsani@itb.ac.id>

# Contents

CBS . . . 2
classifyPcaIca . . . 3
classifyWavFeatExt . . . 6
getIca . . . 8
getPca . . . 10
nhwt . . . 11
plot.nhwt . . . 13
plot.pcaIcaClassifier . . . 15
plot.wavFeatExtClassifier . . . 16
seg . . . 18
simulateCNA . . . 19
wavFeatExt . . . 21

**Index** **24**

---

`CBS` *Circular Binary Segmentation of Copy-number Profiles*

---

## Description

Performs circular binary segmentation (CBS) using the **DNAcopy** package on a numeric vector representing a single copy-number profile.

## Usage

```
CBS(obj, chr = rep(1L, length(obj)), verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `obj` | Numeric vector containing the copy-number (or log-ratio) values along the genome for a single sample. |
| `chr` | Vector of the same length as `obj` giving the chromosome index for each position. By default all positions are assumed to belong to chromosome 1. |
| `verbose` | Logical. If `TRUE`, progress messages from the segmentation procedure are shown. If `FALSE`, they are suppressed. |

## Details

This function is a thin wrapper around `CNA` and `segment` from the **DNAcopy** package. The input vector is first converted to a `"CNA"` object using artificial map locations `1:length(obj)`. Circular binary segmentation is then applied, and the estimated segment means are expanded back to the original resolution to form a segmented profile of the same length as the input.

## Value

A numeric vector of length `length(obj)` containing the segmented copy-number (or log-ratio) values, where each element equals the estimated segment mean for the corresponding genomic position.

## Author(s)

Maharani Ummi <maharaniahsani@gmail.com>

## References

Olshen, A. B., Venkatraman, E. S., Lucito, R. and Wigler, M. (2004). Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics*, **5**(4), 557–572.

## See Also

`wavFeatExt`

## Examples

```
## Generate noisy copy-number data under normal error
set.seed(10)
cn <- rnorm(100, mean = c(rep(2, 50), rep(1, 50)), sd = 0.4)

## Segmentation using CBS
cn.cbs <- CBS(cn)

## Plot raw and segmented profiles
plot(cn, type = "p", xlab = "Genomic position", ylab = "Copy-number")
lines(cn.cbs, col = 2, lwd = 2)
```

---

# classifyPcaIca *Classification on PCA- and ICA-based Feature Sets*

---

## Description

Performs classification using several machine learning methods on feature sets derived from principal component analysis (PCA) and independent component analysis (ICA), as well as on the original (or segmented) data matrix. Supported methods include Lasso, elastic net, random forest, neural network, partial least squares, and k-nearest neighbours.

## Usage

```
classifyPcaIca(
  data,
  y,
  pca,
  ica,
  method = c("lasso", "elnet", "RF", "NN", "PLS", "KNN"),
  k = 5,
  ite = length(data)
)
```

## Arguments

| | |
|---|---|
| data | A list of numeric matrices, each with observations in rows and variables in columns, or an object with equivalent structure (e.g. the output from simulateCNA). Each element of the list corresponds to one simulated data set or replicate. |
| y | Response vector for the classification task, of length equal to the number of rows in data[[1]]. It must be either a binary numeric vector coded as 0 and 1 or a binary factor with two levels. |
| pca | Result of applying getPca to data, i.e. a list of PCA-based feature sets for each data set. For each data set, the corresponding element is a list of cumulative principal components. |
| ica | Result of applying getIca to data, i.e. a list of ICA-based feature sets for each data set. For each data set, the corresponding element is a list of cumulative independent components. |
| method | Character string specifying the classification method to use. One of "lasso", "elnet", "RF", "NN", "PLS", or "KNN". |
| k | Number of folds for k-fold cross-validation. Default is 5. The number of observations must be divisible by k. |
| ite | Number of cross-validation replications. Default is length(data), i.e. one replication per data set in the list. |

## Details

For each replication, the function constructs a k-fold cross-validation partition of the observations in data[[1]]. For every feature set considered (each cumulative PCA feature set, each cumulative ICA feature set, and the original/segmented data matrix), the chosen classification method is trained on the training folds and evaluated on the held-out fold.

The cross-validated misclassification error (CE) and area under the ROC curve (AUC) are computed for each fold and then averaged across folds. This procedure is repeated for ite replications, and the resulting averages are stored in matrices for CE and AUC.

The columns of the output matrices correspond to the feature sets used: the first length(pca[[1]]) columns to PCA-based feature sets, the next length(ica[[1]]) columns to ICA-based feature sets, and the final column to the original (or segmented) data matrix (labelled "seg").

## Value

A list with the following components:

**CE** Numeric matrix of cross-validated misclassification errors. Rows correspond to replications (of length `ite`) and columns to feature sets.

**AUC** Numeric matrix of cross-validated areas under the ROC curve. The dimensions and column names match those of `CE`.

**method** Character string giving the classification method used.

## Author(s)

Maharani Ahsani Ummi and Arief Gusnanto

## References

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B* **58**, 267–288.

Breiman, L. (2001). Random forests. *Machine Learning* **45**(1), 5–32.

## See Also

`simulateCNA`, `getPca`, `getIca`, `classifyWavFeatExt`

## Examples

```
set.seed(10)

sim.dat <- simulateCNA(
  n.obs = 20,
  p = 32,
  n.sim = 1,
  n.block = 8,
  verbose = FALSE
)

pca <- getPca(sim.dat, k = 4)
ica <- getIca(sim.dat, k = 4)

y <- factor(rep(c("Group1", "Group2"), each = 10))

res <- classifyPcaIca(
  sim.dat,
  y,
  pca,
  ica,
  method = "KNN",
  k = 5,
  ite = 1
)
```

```
res
```

---

`classifyWavFeatExt` *Classification on Wavelet-based Features*

---

## Description

Performs classification using several machine learning methods on features extracted via non-decimated Haar wavelet transformation, namely detail and scaling coefficients, as well as on the original (or segmented) data matrix. Optionally, all wavelet coefficients can be combined into a single feature set.

## Usage

```
classifyWavFeatExt(
  data,
  y,
  det,
  sca,
  method = c("lasso", "elnet", "RF", "NN", "PLS", "KNN"),
  k = 5,
  ite = length(data),
  all = FALSE,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | A non-empty list of numeric matrices with equal number of rows (rows = observations, columns = variables). Each element of the list corresponds to one data set / replicate. |
| `y` | Response vector for the classification task, of length equal to the number of rows in `data[[1]]`. It must be either a binary numeric vector coded as `0` and `1`, or a binary factor with exactly two levels. |
| `det` | Detail coefficients of `data`, typically obtained by `wavFeatExt(data, type = "detail")`. Must be a list of the same length as `data`; for each data set, the corresponding element is a list of matrices, one per wavelet scale. |
| `sca` | Scaling coefficients of `data`, typically obtained by `wavFeatExt(data, type = "scaling")`. Must be a list of the same length as `data`; for each data set, the corresponding element is a list of matrices, one per wavelet scale. |
| `method` | Character string specifying the classification method to use. One of `"lasso"`, `"elnet"`, `"RF"`, `"NN"`, `"PLS"`, or `"KNN"`. |
| `k` | Number of folds for k-fold cross-validation (default `5`). The number of observations must be divisible by `k`. |

| | |
|---|---|
| `ite` | Number of cross-validation replications (default `length(data)`). |
| `all` | Logical; if `FALSE` (default), each wavelet scale is evaluated separately plus the original/segmented matrix. If `TRUE`, all wavelet coefficients are concatenated into a single feature set. |
| `verbose` | Logical; if `TRUE`, progress messages are printed. |

## Details

For each replication, the function randomly assigns observations to `k` folds of equal size. For every feature set considered, the chosen classification method is trained on the training folds and evaluated on the held-out fold.

The misclassification error (CE) is computed as the proportion of incorrect predicted classes on the test fold. In addition, the area under the ROC curve (AUC) is computed from the predicted class probabilities using **pROC**.

Fold-wise CE and AUC are averaged across the `k` folds to obtain one CE and one AUC per feature set and replication. This is repeated for `ite` replications, producing two matrices (CE and AUC).

**Feature sets and column names:**

- If `all = FALSE`: columns correspond to detail scales (`D1, D2, ..., Dq`), scaling scales (`S1, S2, ..., Sp`), and the original/segmented matrix (`"seg"`).
- If `all = TRUE`: columns are `"ALL"` (all wavelet coefficients combined) and `"seg"`.

## Value

An object of class `"wavFeatExtClassifier"`, a list with:

**CE** Numeric matrix of cross-validated misclassification errors.

**AUC** Numeric matrix of cross-validated AUC values.

**method** Character string giving the classification method used.

**all** Logical indicating whether `all = TRUE` was used.

## Author(s)

Maharani Ahsani Ummi and Arief Gusnanto

## References

Nason, G. P. (2008). *Wavelet Methods in Statistics with R*. Springer.

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B* **58**, 267–288.

## See Also

`simulateCNA`, `wavFeatExt`, `classifyPcaIca`, `nhwt`

## Examples

```
set.seed(10)

sim.dat <- simulateCNA(
  n.obs = 20,
  p = 32,
  n.sim = 1,
  n.block = 8,
  verbose = FALSE
)

det.coef <- wavFeatExt(sim.dat, type = "detail")
sca.coef <- wavFeatExt(sim.dat, type = "scaling")

y <- factor(rep(c("Group1", "Group2"), each = 10))

res <- classifyWavFeatExt(
  sim.dat,
  y,
  det.coef,
  sca.coef,
  method = "KNN",
  k = 5,
  ite = 1
)

res
```

---

# getIca *Independent Component Analysis (ICA) for a List of Matrices*

---

## Description

Applies independent component analysis (ICA) to each matrix in a collection of data sets and returns cumulative independent components for use in downstream classification or feature extraction.

## Usage

```
getIca(x, k, ...)
```

## Arguments

| | |
|---|---|
| x | Either a numeric matrix, a list of numeric matrices (with rows corresponding to observations and columns to variables), or the output from simulateCNA. Each matrix will undergo ICA separately. |
| k | Integer specifying the number of independent components to extract from each matrix. Must be at least 2. |
| ... | Additional arguments passed to icafast from the **ica** package. |

## Details

For each matrix in x, the function calls icafast to perform independent component analysis, requesting k independent components. Let $S$ denote the resulting matrix of independent components (rows = observations, columns = components).

The function then constructs a list of k - 1 matrices per data set, where the $j$-th element (for $j = 2, \ldots, k$) contains the first $j$ independent components, i.e. S[, 1:j]. This cumulative structure mirrors the behaviour of getPca and provides feature sets with increasing dimensionality.

When x is a single matrix or data frame, it is internally wrapped into a list of length one, and the result is still a list of lists for consistency.

## Value

A list of length equal to the number of matrices in x. For each input matrix, an element is returned which is itself a list of length k - 1. The $j$-th element (for $j = 2, \ldots, k$) of the inner list is a numeric matrix containing the first $j$ independent components for that data set.

## Note

The cumulative component representation (2, 3, ..., k components) is designed to be used in combination with classifyPcaIca for comparing classification performance across different feature dimensions.

## Author(s)

Maharani Ahsani Ummi and Arief Gusnanto

## References

Hyv\"arinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks* **13**(4–5), 411–430.

## See Also

getPca, classifyPcaIca, simulateCNA, icafast

## Examples

```
set.seed(10)

## Small simulated data for fast execution
sim.dat <- simulateCNA(
  n.obs = 20,
  p = 32,
  n.sim = 1,
  n.block = 8,
  verbose = FALSE
)

## Obtain ICA-based features
ica <- getIca(sim.dat, k = 5)
```

```
length(ica)
length(ica[[1]])
sapply(ica[[1]], dim)
```

---

`getPca` *Principal Component Analysis (PCA) for a List of Matrices*

---

## Description

Applies principal component analysis (PCA) to each matrix in a collection of data sets and returns cumulative principal components for use in downstream classification or feature extraction.

## Usage

```
getPca(x, k, ...)
```

## Arguments

| | |
|---|---|
| `x` | Either a numeric matrix, a list of numeric matrices (with rows corresponding to observations and columns to variables), or the output from `simulateCNA`. Each matrix will undergo PCA separately. |
| `k` | Integer specifying the number of principal components to extract from each matrix. Must be at least 2. |
| `...` | Additional arguments passed to `prcomp`. |

## Details

For each matrix in `x`, the function calls `prcomp` to compute principal component scores. Let $Z$ denote the resulting score matrix (rows = observations, columns = principal components).

The function then constructs a list of `k - 1` matrices per data set, where the $j$-th element (for $j = 2, \ldots, k$) contains the first $j$ principal components, i.e. `Z[, 1:j]`. This cumulative structure mirrors the behaviour of `getIca` and provides feature sets with increasing dimensionality.

## Value

A list of length equal to the number of matrices in `x`. For each input matrix, an element is returned which is itself a list of length `k - 1`. The $j$-th element (for $j = 2, \ldots, k$) of the inner list is a numeric matrix containing the first $j$ principal components for that data set.

## Note

The cumulative component representation is designed to be used in combination with `classifyPcaIca` for comparing classification performance across different feature dimensions.

## Author(s)

Maharani Ahsani Ummi and Arief Gusnanto

## References

Jolliffe, I. T. (2002). *Principal Component Analysis*. 2nd edition. Springer.

## See Also

getIca, classifyPcaIca, simulateCNA, prcomp

## Examples

```
set.seed(10)

## Small simulated data for fast execution
sim.dat <- simulateCNA(
  n.obs = 20,
  p = 32,
  n.sim = 1,
  n.block = 8,
  verbose = FALSE
)

## Obtain PCA-based features
pca <- getPca(sim.dat, k = 5)

length(pca)
length(pca[[1]])
sapply(pca[[1]], dim)
```

---

# nhwt *Non-decimated Haar Wavelet Transform (Legacy)*

---

## Description

Performs a non-decimated Haar wavelet transform (stationary wavelet transform) on one or more one-dimensional signals. This function is kept for compatibility with older code and is not used in the main workflow of the package.

## Usage

```
nhwt(data, type = c("detail", "scaling"))
```

## Arguments

| | |
|---|---|
| `data` | Numeric vector, matrix, or data frame containing the data to be decomposed. If a matrix or data frame is supplied, rows are interpreted as observations and columns as ordered locations. |
| `type` | Character string indicating which coefficients to extract from the transform. Use `"detail"` for detail coefficients or `"scaling"` for scaling coefficients. |

## Details

The function applies a non-decimated (stationary) Haar wavelet transform to each row of the input. If the number of columns is not a power of two, the series is extended to the next power of two by constant padding with value 1 on both sides before computing the transform. After the transform, the resulting coefficients are re-aligned and cropped back to the original length.

The underlying transform is computed using `wd` with `family = "DaubExPhase"`, `filter.number = 1`, and `type = "station"`. For each scale, either detail or scaling coefficients can be extracted via `accessD` or `accessC`, respectively.

## Value

A list of length $J$, where $J = \lfloor \log_2(n) \rfloor$ and $n$ is the number of columns in `data`. Each element is a numeric matrix of dimension `nrow(data) x n` containing the detail or scaling coefficients at a given scale.

## Note

This function is considered legacy and is not used by the main feature extraction and classification functions in the package. It is provided for backward compatibility with older analysis pipelines.

## Author(s)

Maharani Ahsani Ummi and Arief Gusnanto

## References

Nason, G. P. (2008). *Wavelet Methods in Statistics with R*. Springer.

## See Also

`wavFeatExt`, `plot.nhwt`, `wd`, `accessD`, `accessC`

## Examples

```
## Simple example: non-decimated Haar wavelet coefficients
## for a piecewise constant signal
obj <- c(rep(1, 10), rep(3, 20))

## Detail coefficients at all scales
obj.nhwt.det <- nhwt(obj, type = "detail")
```

```
length(obj.nhwt.det)
sapply(obj.nhwt.det, dim)
```

---

`plot.nhwt` *Plot non-decimated Haar wavelet transform coefficients*

---

## Description

Plot the non-decimated Haar wavelet transform (NHWT) coefficients produced by `nhwt()` at one or more resolution scales. The function can display either detail or scaling coefficients, and supports global or per-level rescaling for visual comparison.

## Usage

```
## S3 method for class 'nhwt'
plot(
  x,
  coef = c("detail", "scaling"),
  type = c("global", "by.level"),
  scale = "all",
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | The output from `nhwt()`, i.e. a list of matrices with one element per resolution scale. Each matrix has rows corresponding to observations and columns to genomic locations (or data positions). Alternatively, a matrix can be supplied directly, in which case rows are interpreted as scales and columns as positions. For multi-observation inputs, only the first observation is plotted. |
| `coef` | Character string indicating the type of coefficients to be plotted. One of `"detail"` for detail coefficients or `"scaling"` for scaling coefficients. |
| `type` | Character string indicating how the coefficients are rescaled for plotting when `scale = "all"`. The options are:<br>`"global"` A single scale factor is chosen for the entire plot, based on the coefficient (across all scales) with the largest absolute value. This option is useful for comparing coefficients across different resolution scales.<br>`"by.level"` A separate scale factor is chosen for each resolution scale, based on the coefficient within that scale that has the largest absolute value. This option is useful for comparing coefficients within a given resolution scale. |
| `scale` | Either `"all"` (the default) to plot all resolution scales in a stacked display, or a single integer specifying the index of the scale to plot. In the latter case, the chosen scale is shown on its own. |
| `...` | Further graphical arguments (ignored). |

## Details

The function produces a vertical stacked line plot where each resolution scale is shown on a separate horizontal band. Within a scale, each coefficient is drawn as a vertical segment whose height is proportional to its (rescaled) value.

If `obj.nhwt` originates from `nhwt()` applied to a data matrix with multiple observations, only the coefficients for the first observation (row) are used. This behaviour is intended for visualising a single time series or copy-number profile.

## Value

This function is called for its side effects of producing a plot. It returns `invisible(NULL)`.

## Note

This plotting function is primarily intended for exploratory visual inspection of non-decimated Haar wavelet coefficients, for example when assessing multi-scale structure in simulated copy number alteration (CNA) profiles.

## Author(s)

Maharani Ahsani Ummi

## References

Nason, G. P. (2008). *Wavelet Methods in Statistics with R*. Springer.

## See Also

`nhwt()`, `wavFeatExt()`

## Examples

```
## Simple example: single synthetic profile
set.seed(1)
obj <- c(rep(1, 10), rep(3, 20))

## Non-decimated Haar wavelet transform (detail coefficients)
obj.nhwt.det <- nhwt(obj, type = "detail")

## Plot NHWT detail coefficients (all scales, global scaling)
plot(obj.nhwt.det, coef = "detail", type = "global")

## Scaling coefficients
obj.nhwt.sca <- nhwt(obj, type = "scaling")
plot(obj.nhwt.sca, coef = "scaling", type = "global")

## Plot a single scale (scale 1) for detail and scaling coefficients
plot(obj.nhwt.det, coef = "detail",  scale = 1)
plot(obj.nhwt.sca, coef = "scaling", scale = 1)
```

plot.pcaIcaClassifier *Plot Classification Results from* classifyPcaIca

## Description

Create boxplots of cross-validated performance measures (classification error or AUC) for the feature sets used in classifyPcaIca, namely PCA-based, ICA-based, and original/segmented data features.

## Usage

```
## S3 method for class 'pcaIcaClassifier'
plot(
  x,
  type = c("CE", "AUC"),
  d.type = c("both", "PCA", "ICA"),
  adjust = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| x | The result object returned by classifyPcaIca, containing the components CE, AUC, and method. |
| type | Character string specifying which performance measure to plot. One of "CE" or "AUC". |
| d.type | Character string indicating which feature types to include in the plot: "both", "PCA", or "ICA". |
| adjust | Logical flag reserved for future extensions of the plotting function. Currently not used. |
| ... | Additional graphical parameters passed to graphics::boxplot. |

## Details

This function assumes that classifyPcaIca assigns column names to its CE and AUC matrices, with prefixes "PC" for PCA-based features, "I" for ICA-based features, and the column name "seg" for the original (or segmented) data matrix.

## Value

This function is used for its side effect of producing a plot and returns invisible(NULL).

## See Also

classifyPcaIca, classifyWavFeatExt, simulateCNA, getPca, getIca

## Examples

```
set.seed(10)
sim.dat <- simulateCNA(
  n.obs = 20,
  p = 32,
  n.sim = 1,
  n.block = 8,
  verbose = FALSE
)
pca <- getPca(sim.dat, k = 4)
ica <- getIca(sim.dat, k = 4)
y <- factor(rep(c("Group1", "Group2"), each = 10))
res <- classifyPcaIca(
  sim.dat,
  y,
  pca,
  ica,
  method = "KNN",
  k = 5,
  ite = 1
)
plot(res, type = "CE")
plot(res, type = "AUC", d.type = "PCA")
```

---

# plot.wavFeatExtClassifier

*Plot Classification Results from* `classifyWavFeatExt`

---

## Description

Create boxplots of cross-validated performance measures (classification error or AUC) for wavelet-based feature sets used in `classifyWavFeatExt()`, namely detail and scaling coefficients, and the original/segmented data.

## Usage

```
## S3 method for class 'wavFeatExtClassifier'
plot(x, type = c("CE", "AUC"), adjust = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `x` | The result object returned by `classifyWavFeatExt()`, containing at least the components `CE`, `AUC`, and `method`. |
| `type` | Character string specifying which performance measure to plot. One of `"CE"` for misclassification error, or `"AUC"` for the area under the ROC curve. |
| `adjust` | Logical flag reserved for future extensions of the plotting function. Currently not used. |
| `...` | Additional graphical parameters passed to `graphics::boxplot()`. |

## Details

It is assumed that `classifyWavFeatExt()` assigns column names to its `CE` and `AUC` matrices as follows:

- Columns beginning with `"D"` correspond to wavelet detail coefficients.
- Columns beginning with `"S"` correspond to wavelet scaling coefficients.
- The column `"seg"` corresponds to the original (or segmented) data.

The corresponding columns are displayed as boxplots summarising cross-validated performance across replications.

Two horizontal reference lines are added:

- Red dashed line: median performance of `"seg"`
- Blue dotted line: best median performance among all features

## Value

Invisibly returns `NULL`.

## See Also

`classifyWavFeatExt()`, `wavFeatExt()`, `simulateCNA()`

## Examples

```
set.seed(10)

## Small simulated data for fast execution
sim.dat <- simulateCNA(
  n.obs = 20,
  p = 32,
  n.sim = 1,
  n.block = 8,
  verbose = FALSE
)

## Extract wavelet features
det.coef <- wavFeatExt(sim.dat, type = "detail")
sca.coef <- wavFeatExt(sim.dat, type = "scaling")

## Binary response
y <- factor(rep(c("Group1", "Group2"), each = 10))

## Classification (fast method)
res <- classifyWavFeatExt(
  sim.dat,
  y,
  det.coef,
  sca.coef,
  method = "KNN",
  k = 5,
```

```
  ite = 1
)

## Plot results
plot(res, type = "CE")
plot(res, type = "AUC")
```

---

seg *Segmentation Per Observation Using CBS*

---

## Description

Applies circular binary segmentation (CBS) to each row of a data matrix. This function is a convenience wrapper around CBS for performing segmentation per observation.

## Usage

```
seg(test.noise, denoise = "CBS", verbose = FALSE)
```

## Arguments

| | |
|---|---|
| test.noise | Numeric matrix or data frame containing the noisy data to be segmented. Rows correspond to observations (e.g. samples or patients) and columns to genomic locations or predictor variables. |
| denoise | Character string specifying the segmentation method. Currently only "CBS" is supported. |
| verbose | Logical. If TRUE, progress messages from the segmentation procedure are shown. If FALSE, they are suppressed. |

## Details

For each row of test.noise, the function calls CBS to perform circular binary segmentation using the **DNAcopy** infrastructure. The result is a matrix of segmented values with the same dimensions as the input, where each row represents the segmented copy-number (or intensity) profile for that observation.

The denoise argument is retained for future extensions, where alternative segmentation methods might be implemented.

## Value

A numeric matrix of the same dimension as test.noise, containing the segmented profiles for all observations. Each row corresponds to one observation and each column to one position.

## Author(s)

Maharani Ummi <maharaniahsani@gmail.com>

## References

Olshen, A. B., Venkatraman, E. S., Lucito, R. and Wigler, M. (2004). Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics* **5**(4), 557–572.

## See Also

CBS, wavFeatExt

## Examples

```
set.seed(10)

cn <- replicate(
  10,
  rnorm(100, mean = c(rep(2, 50), rep(1, 50)), sd = 0.4)
)
cn <- t(cn)

cn.seg <- seg(cn, denoise = "CBS", verbose = FALSE)

dim(cn.seg)

par(mfrow = c(1, 2))
plot(cn[1, ], type = "p", main = "Observation 1",
     xlab = "Position", ylab = "Copy-number")
lines(cn.seg[1, ], col = 2, lwd = 2)

plot(cn[2, ], type = "p", main = "Observation 2",
     xlab = "Position", ylab = "Copy-number")
lines(cn.seg[2, ], col = 2, lwd = 2)
```

---

# simulateCNA *Simulate Copy-number Alteration (CNA) Data in a Two-group Setting*

---

## Description

Generates simulated copy-number alteration (CNA) data for a two-group setting, under a multivariate normal model with block correlation structure. Mean differences between the two groups are introduced in selected blocks, and each simulated profile is subsequently segmented using circular binary segmentation (CBS).

## Usage

```
simulateCNA(
  n.obs = 100,
  p = 1024,
  n.sim = 1,
  effect.diff = 1,
```

```
  n.block = 32,
  true.rho = 0.9,
  block.cor = 0.4,
  block.diff = "A",
  true.mu = NULL,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| n.obs | Number of observations in each simulated data set. Must be even, so that the first half belong to group 1 and the second half to group 2. |
| p | Number of variables (genomic locations) in each simulated data set. It is recommended to use a power of two for compatibility with subsequent wavelet-based analyses. |
| n.sim | Number of data sets to generate. Each data set has n.obs rows and p columns. |
| effect.diff | Magnitude of the mean difference between the two groups in the blocks where differences are present. |
| n.block | Number of correlation blocks along the genome. Must be a divisor of p, so that the block size p / n.block is an integer. |
| true.rho | Correlation between genomic regions within a block. |
| block.cor | Correlation between adjacent blocks. A value of zero implies that blocks are independent from each other. |
| block.diff | Pattern of blocks that carry mean differences between the two groups. Currently one of "A" or "B". |
| true.mu | Optional numeric vector of length p giving the baseline mean level of CNA across genomic locations, common to both groups. If NULL, a simple four-block mean pattern is used and repeated along the genome. |
| verbose | Logical. If TRUE, progress messages are shown during simulation and segmentation. If FALSE, they are suppressed. |

## Details

The function first constructs a block-structured covariance matrix $\Sigma$ of dimension p x p, with within-block correlations given by true.rho and between-block correlations given by block.cor. Copy-number data are then simulated from a multivariate normal distribution with mean vector true.mu and covariance $\Sigma$.

To create a two-group setting, a mean shift vector is added to the first half of the observations (group 1), while the second half (group 2) remains at the baseline mean. The pattern of mean shifts depends on block.diff.

Finally, for each simulated data set, circular binary segmentation (CBS) is applied to every observation (row) via seg and CBS. The returned data therefore correspond to segmented CNA profiles rather than raw multivariate normal values.

## Value

A list of length `n.sim`. Each element is a numeric matrix of dimension `n.obs` x `p`, containing the segmented CNA profiles for one simulated data set.

## Author(s)

Arief Gusnanto

## See Also

`seg`, `CBS`, `wavFeatExt`, `classifyPcaIca`, `classifyWavFeatExt`

## Examples

```
set.seed(10)

sim.dat <- simulateCNA(
  n.obs = 20,
  p = 32,
  n.sim = 1,
  n.block = 8,
  verbose = FALSE
)

length(sim.dat)
dim(sim.dat[[1]])

plot(sim.dat[[1]][1, ], type = "l",
     xlab = "Genomic location", ylab = "Segmented CNA",
     main = "Simulated segmented CNA profile")
```

---

# wavFeatExt *Feature Extraction Using Non-decimated Haar Wavelet Transform*

---

## Description

Performs feature extraction from copy-number (or other one-dimensional) profiles using the non-decimated (stationary) Haar wavelet transform. The function computes detail or scaling coefficients at multiple scales for each observation and organises them into a structured list.

## Usage

```
wavFeatExt(data, type = c("detail", "scaling"))
```

## Arguments

| | |
|---|---|
| data | Either a numeric matrix, a list of numeric matrices (with rows corresponding to observations and columns to variables or genomic locations), or the output from simulateCNA. Each matrix represents a set of profiles for which wavelet-based features are to be extracted. |
| type | Character string indicating which type of coefficients to extract from the non-decimated wavelet transform. One of "detail" or "scaling". |

## Details

For each matrix in data, the function applies a non-decimated Haar wavelet transform (via nhwt) to each row. This yields, for each scale, a matrix of wavelet coefficients of the same dimension as the original data matrix.

The input data is first normalised to a list of matrices. If a single matrix is provided, it is internally wrapped into a list of length one. When the input is the output of simulateCNA, each element corresponds to one simulated data set.

The resulting object is a list of length equal to the number of matrices in the input. Each element of this list is itself a list of length equal to the number of available scales. At each scale, a numeric matrix contains the wavelet coefficients (detail or scaling) for all observations.

## Value

A list of length equal to the number of matrices in data. For each data matrix, an element is returned which is a list of length $J$, where $J$ is the number of scales determined by the wavelet transform. Each sub-list element is a numeric matrix of the same dimension as the original data matrix.

## Author(s)

Maharani Ahsani Ummi and Arief Gusnanto

## References

Nason, G. P. (2008). *Wavelet Methods in Statistics with R*. Springer.

## See Also

nhwt, simulateCNA, classifyWavFeatExt, classifyPcaIca

## Examples

```
set.seed(10)

sim.dat <- simulateCNA(
  n.obs = 20,
  p = 32,
  n.sim = 1,
  n.block = 8,
  verbose = FALSE
)
```

```
det.coef <- wavFeatExt(sim.dat, type = "detail")
sca.coef <- wavFeatExt(sim.dat, type = "scaling")

length(det.coef)
length(sca.coef)
```

# Index

∗ **hplot**
plot.nhwt, 13
∗ **wavelets**
plot.nhwt, 13

accessC, *12*
accessD, *12*

CBS, 2, *18–21*
classifyPcaIca, 3, *7*, *9–11*, *15*, *21, 22*
classifyWavFeatExt, *5*, 6, *15*, *21, 22*
classifyWavFeatExt(), *16, 17*
CNA, *3*

getIca, *4, 5*, 8, *10, 11*, *15*
getPca, *4, 5*, *9*, 10, *15*
graphics::boxplot, *15*
graphics::boxplot(), *16*

icafast, *8, 9*

nhwt, *7*, 11, *22*
nhwt(), *13, 14*

plot.nhwt, *12*, 13
plot.pcaIcaClassifier, 15
plot.wavFeatExtClassifier, 16
prcomp, *10, 11*

seg, 18, *20, 21*
segment, *3*
simulateCNA, *4, 5*, *7–11*, *15*, 19, *22*
simulateCNA(), *17*

wavFeatExt, *3*, *6, 7*, *12*, *19*, *21*, 21
wavFeatExt(), *14*, *17*
wd, *12*