Package 'alsace'

March 29, 2021

Type Package

Title ALS for the Automatic Chemical Exploration of mixtures

Version 1.26.0

Author Ron Wehrens

Maintainer Ron Wehrens <ron.wehrens@gmail.com>

Description Alternating Least Squares (or Multivariate Curve Resolution) for analytical chemical data, in particular hyphenated data where the first direction is a retention time axis, and the second a spectral axis. Package builds on the basic als function from the ALS package and adds functionality for high-throughput analysis, including definition of time windows, clustering of profiles, retention time correction, etcetera.

URL https://github.com/rwehrens/alsace

License GPL (>= 2)

Depends R (>= 2.10), ALS, ptw (>= 1.0.6)

Suggests lattice, knitr

VignetteBuilder knitr

BiocViews Metabolomics, Preprocessing

git_url https://git.bioconductor.org/packages/alsace

git_branch RELEASE_3_12

git_last_commit 40a7640

git_last_commit_date 2020-10-27

Date/Publication 2021-03-29

R topics documented:

`alsace-package` *Alternating Least Squares in an Analytic Chemistry Environment*

Description

Add-on to the ALS package that implements Alternating Least Squares (or Multivariate Curve Resolution, MCR). This implementation is specifically geared to data from systems like HPLC-DAD where measurements are always positive. In addition, it provides extra functionality to deal with large data sets, and additional postprocessing tools including visualization.

Details

Package:	alsace
Type:	Package
Version:	1.0
Date:	2013-07-08
License:	GPL (>= 2)

The main function of the package is also the one that contributes the least new material: `doALS` is simply a wrapper for `ALS`, provided by the ALS package. More important novel material are the visualization, preprocessing and postprocessing functions: `showALSresult`, `preprocess`, `windows`, `combineComps`,... See the manual pages of these functions for more information.

It should be noted that the examples are only meant to illustrate the use of the functions in the package, and do not constitute the final analysis of the data provided. Indeed, really meaningful results can only be obtained by either careful definition of additional constraints, or the addition of more data.

Author(s)

Ron Wehrens

Maintainer: Ron Wehrens <ron.wehrens@fmach.it>

References

R. Wehrens: Chemometrics with R. Springer Verlag, Heidelberg (2011)

R. Wehrens, E. Carvalho et al.: High-throughput carotenoid profiling using multivariate curve resolution. Anal. Bioanal. Chem., 15:5075-5086 (2013)

components *Functions to assess and refine ALS components*

Description

One of the inherent drawbacks of the MCR-ALS method is that in the vast majority of cases there is no one unique set of components describing the data, a situation known as "rotational ambiguity". This implies that in some cases a spectrum of a chemical compound can be described by a linear combination of two ALS components. This can sometimes be recognised by looking at elution profiles. In addition, in cases where the number of components is too large, some components may only describe noise or very small and irrelevant features. The functions clarified here allow one to find which components only correspond with minor features, to remove components, and to merge components.

Usage

```
smallComps(obj, Ithresh)
removeComps(obj, toRemove, ...)
combineComps(obj, compList, weights, ...)
suggestCompCombis(obj, indices, Ithresh = 0, corthresh = 0.9,
                  clusterHeight = 0.6)
```

Arguments

obj	The R object containing the als model
Ithresh	Intensity cutoff: all components with a maximal intensity (in the elution profiles) below this value will be termed "small".
toRemove	The indices of the components to remove from the ALS model. A new call to doALS will be done with the smaller set of components.
...	Additional arguments to doALS, e.g. maxiter = 1 if no full set of iterations is required.
compList	A list indicating which components need to be combined. Using list(c(1,c(2,3),4)) will lead to a three-component model, where components 1 and 4 are unchanged and components 2 and 3 are combined.
weights	Weights for the components to be combined. If not provided, equal weights will be assumed.
indices	A list indicating in which (groups of) samples correlations will be calculated. See details.
corthresh	Correlation threshold: components with elution profiles showing a higher correlation than this threshold may be candidates for merging.
clusterHeight	Similarity threshold at which to cut the dendrogram (see details).

Details

Function suggestCompCombis checks correlations in elution profiles that could point to a situation where one chemical compound is described by two or more ALS components. For every sample in which this correlation is higher than the threshold, a "hit" will be recorded for these two components. After checking all samples and all combinations, the hit matrix will be used as a similarity measure in a hierarchical clustering. The dendrogram will be cut at a specific height, leading to groups of

components, sometimes containing more than one element. In such a case, these components could be considered for merging.

If injections of pure standards are present, they probably should not be used in isolation to check for coelution; rather, suggestions for combined components can be validated looking at the elution profiles of the standards.

Value

Functions `removeComps` and `combineComps` return `ALS` objects with fewer components than the original object. Function `smallComps` returns a list of two elements:

`smallComps`	the indices of the small components
`maxCvalues`	the maximal values found in the concentration profiles across all samples for each of the components.

Author(s)

Ron Wehrens

Examples

```
data(tea)
new.lambdas <- seq(260, 500, by = 2)
tea <- lapply(tea.raw, preprocess)
tea.split <- splitTimeWindow(tea, c(12, 14), overlap = 10)

Xl <- tea.split[[3]]
Xl.opa <- opa(Xl, 10)

Xl.als <- doALS(Xl, Xl.opa)
smallC <- smallComps(Xl.als, 5)
smallC

Xl.als2 <- removeComps(Xl.als, smallC$smallC)
summary(Xl.als)
summary(Xl.als2)
## smaller models, but with a higher fit error...

## another way to decrease the number of components, this example
## not particularly deep, just to show how it can be done:
Xl.als3 <- combineComps(Xl.als, list(1, 2, 3:4, 5, c(6, 10), 6, 7:9))
summary(Xl.als3)
```

`correctPeaks` *Correct peak positions according to a ptw warping model*

Description

Once an appropriate warping model has been established, corrected retention times can be predicted for each peak. These are stored in a separate column in the list of peak tables.

Usage

```
correctPeaks(peakList, modList)
```

Arguments

peakList	A nested list of peak tables: the first level is the sample, and the second level is the component. Every component is described by a matrix where every row is one peak, and the columns contain information on retention time, full width at half maximum (FWHM), peak width, height, and area.
modList	A list of ptw models.

Value

The input list of peak tables is returned with extra columns containing the corrected retention time.

Author(s)

Ron Wehrens

See Also

correctRT

Examples

```
data(teaMerged)
pks <- getAllPeaks(teaMerged$CList, span = 11)
warping.models <- correctRT(teaMerged$CList, reference = 2,
                            what = "models")
pks.corrected <- correctPeaks(pks, warping.models)

## original profiles and peaks, in black and gray
plot(teaMerged, mat.idx = 3, what = "profiles", comp.idx = 2,
     showWindows = FALSE, col = "gray")
abline(v = pks[[3]][[2]][,"rt"])
## shifted profiles and peaks, in red and pink
CList.corrected <- correctRT(teaMerged$CList, reference = 2)
lines(as.numeric(rownames(CList.corrected[[3]])),
      CList.corrected[[3]][,2], col = "pink")
abline(v = pks.corrected[[3]][[2]][,"rt.cor"], col = "red")
## note that the rightmost peak in the uncorrected data is no longer
## within the range of the data
```

---

correctRT *Retention time correction for ALS chromatographic profiles*

Description

Correction of retention time differences of ALS concentration profiles using parametric time warping.

Usage

```
correctRT(CList, reference,
                 what = c("corrected.values", "models"),
                 init.coef = c(0, 1, 0), ...)
```

Arguments

CList	List of matrices containing concentration profiles.
reference	Index of the sample that is to be considered the reference sample.
what	What to return: either the time-corrected profiles (useful for visual inspection) or the warping models (for further programmatic use).
init.coef	Starting values for the optimisation.
...	Optional arguments for the ptw function. The only argument that cannot be changed is warp.type: this is always equal to "global".

Value

A list of warped concentration profiles, mirroring the CList list element from the ALS object.

Author(s)

Ron Wehrens

See Also

ptw, correctPeaks

Examples

```
data(teaMerged)
CList.corrected <- correctRT(teaMerged$CList, reference = 2)

original.profiles <- sapply(teaMerged$CList, identity, simplify = "array")
corrected.profiles <- sapply(CList.corrected, identity, simplify = "array")

def.par <- par(no.readonly = TRUE)
par(mfrow = c(2,4))
for (i in 1:4)
    matplot(dimnames(original.profiles)[[1]],
            original.profiles[,i,], type = "l", lty = 1,
            xlab = "Time (min.)", ylab = "Response",
            main = paste("Component", i))
for (i in 1:4)
    matplot(dimnames(original.profiles)[[1]],
            corrected.profiles[,i,], type = "l", lty = 1,
            xlab = "Time (min.)", ylab = "Response",
            main = paste("Component", i, "- warped"))
par(def.par) ## reset defaults
```

---

doALS *Wrapper function for als, plus some support functions*

Description

Wrapper function for the als function in the ALS package, providing a simple interface with sensible defaults for hyphenated data.

Usage

```
doALS(Xl, PureS, maxiter = 100, verbose)
## S3 method for class 'ALS'
print(x, ...)
## S3 method for class 'ALS'
summary(object, ...)
## S3 method for class 'ALS'
plot(x, what = c("spectra", "profiles"), showWindows = TRUE,
                   mat.idx, comp.idx, xlab, ylab, main, ...)
getTime(x)
getWavelength(x)
```

Arguments

Xl	a list of (numerical) data matrices. Missing values are not allowed.
x, object	an object of class ALS.
PureS	Initial estimates of pure spectral components.
maxiter	maximum number of iterations in ALS.
verbose	show als feedback during optimisation.
what	Show spectra or elution profiles
showWindows	If showing elution profiles, the window borders and the overlap areas between the windows can be shown (by default). Simply set this parameter to FALSE if this is undesired.
mat.idx	If showing elution profiles, one can provide the index of the sample(s) that should be shown. For every sample one plot will be made. Default is to show all.
comp.idx	Indices of components to be shown. Default is to show all components.
xlab, ylab, main, ...	self-explanatory optional arguments

Details

The plot method can be used to plot the spectral components (one plot for the model) or the elution profiles (one plot for each data matrix, so usually several plots). The summary method also returns fit statistics like LOF, R2 and RMS. Extractor functions getTime and getWavelength provide the vectors of time points and wavelengths from the ALS object.

Value

Function doALS returns an object of class "ALS", a list with the following fields:

CList	a list of matrices with the elution profiles in the columns. Every matrix in this list corresponds with a matrix in the input.
S	a matrix with the spectral components in the columns. These are common for all data matrices.
rss	residual sum of squares.
resid	a list of residual matrices.
iter	number of iterations.
summ.stats	summary statistics, providing more information about the fit quality.

See the als function for more details; only the summ.stats field is not part of the original als output.

Author(s)

Ron Wehrens

See Also

als,showALSresult

Examples

```
data(tea)
new.lambdas <- seq(260, 500, by = 2)
tea <- lapply(tea.raw, preprocess, dim2 = new.lambdas)
tea.split <- splitTimeWindow(tea, c(12, 14), overlap = 10)

Xl <- tea.split[[2]]
Xl.opa <- opa(Xl, 4)

Xl.als <- doALS(Xl, Xl.opa)
Xl.als
summary(Xl.als)
plot(Xl.als, "spectra")
par(mfrow = c(1, 3))
plot(Xl.als, "profiles", ylim = c(0, 600), mat.idx = 1:3)
```

---

filterPeaks *Filter peak lists*

Description

Utility function to remove peaks from a peak list, e.g. because their intensity is too low. Currently one can filter on peak height, peak area, and width at half maximum.

Usage

```
filterPeaks(peakList, minHeight, minArea, minWHM, maxWHM)
```

Arguments

peakList	A nested list of peak tables: the first level is the sample, and the second level is the component. Every component is described by a matrix where every row is one peak, and the columns contain information on retention time, full width at half maximum (FWHM), peak width, height, and area.
minHeight	Minimum peak height.
minArea	Minimum peak area.
minWHM	Minimal width at half maximum.
maxWHM	Maximum width at half maximum.

Value

A peak list similar to the input peakList, but with all rows removed from the peak tables that are not satisfying the criteria.

Author(s)

Ron Wehrens

See Also

getAllPeaks

Examples

```
data(teaMerged)
pks <- getAllPeaks(teaMerged$CList, span = 11)
## only retain peaks with a peak height of at least 2
pks.filtered <- filterPeaks(pks, minHeight = 2)
plot(teaMerged, mat.idx = 3, what = "profiles", comp.idx = 2,
     showWindows = FALSE, col = "blue")
abline(v = pks[[3]][[2]][,"rt"], col = "gray", lty = 2)
abline(v = pks.filtered[[3]][[2]][,"rt"])
```

---

fitpeaks *Fit chromatographic peaks with a gaussian profile*

Description

Find chromatographic peaks, and fit peak parameters using a gaussian profile. The algorithm is extremely simple and could be replaced by a more sophisticated algorithm. In particular one can expect bad fits if peaks are overlapping significantly.

Usage

```
findpeaks(y, span = NULL)
fitpeaks(y, pos)
```

Arguments

y	response (numerical vector)
span	number of points used in the definition of what constitutes a "local" maximum. If not given, a default value of 20 percent of the number of time points is used.
pos	locations of local maxima in vector y

Details

Finding peaks with function findpeaks is based on the position of local maxima within a window of width span.

Peak parameters are calculated using fitpeaks, assuming a normal distribution. Peak width is given as a standard deviation, calculated from the full width at half maximum (FWHM); the peak area is given by the ratio of the peak height and the density.

Value

Function `findpeaks` simply returns the locations of the local maxima, expressed as indices.

Function `fitpeaks` returns a matrix, whose columns contain the following information:

`rt`	location of the maximum of the peak (x)
`sd`	width of the peak (x)
`FWHM`	full width at half maximum (x)
`height`	height of the peak (y)
`area`	peak area

Again, the first three elements (rt, sd and FWHM) are expressed as indices, so not in terms of the real retention times. The transformation to "real" time is done in function `getAllPeaks`.

Note

Function `findpeaks` was modelled after code suggested by Brian Ripley on the R help list.

Author(s)

Ron Wehrens

See Also

`getAllPeaks`

Examples

```
data(tea)
new.lambdas <- seq(260, 500, by = 2)
tea <- lapply(tea.raw, preprocess, dim2 = new.lambdas)
tea.split <- splitTimeWindow(tea, c(12, 14), overlap = 10)

Xl <- tea.split[[2]]
Xl.opa <- opa(Xl, 4)

Xl.als <- doALS(Xl, Xl.opa)

tpoints <- getTime(Xl.als)
plot(tpoints, Xl.als$CList[[2]][,2], type = "l", col = "gray")
pk.pos <- findpeaks(Xl.als$CList[[2]][,2], span = 11)
abline(v = tpoints[pk.pos], col = 4)

pks <- fitpeaks(Xl.als$CList[[2]][,2], pk.pos)
apply(pks, 1,
      function(pkmodel) {
        lines(tpoints,
              dnorm(1:length(tpoints), pkmodel["rt"], pkmodel["sd"]) *
              pkmodel["area"],
              col = 2)
        invisible()
      })
## reasonably close fit, apart from the small peak in the middle...
```

getAllPeaks *Extract all peaks from the chromatographic profiles of an ALS object*

Description

Extractor function to find all peaks in the chromatographic profiles of an ALS object. Peaks are located as local maxima within the given span (function findpeaks) and at the given positions a gaussian curve is fit (function fitpeaks).

Usage

```
getAllPeaks(CList, span = NULL, eps = 1e-01)
```

Arguments

CList	A list of profile matrices, each of the same dimensions (timepoints times components).
span	The span used for identifying local maxima in the individual components. If not given, the default of findpeaks is used.
eps	Minimal value for the peak width, basically used to eliminate peaks with zero width.

Value

The result is a list, with each element corresponding to one data file, and containing data for the fitted peaks for each of the ALS components. Note that this function presents the "rt", "sd" and "FWHM" fields in real time units.

Author(s)

Ron Wehrens

Examples

```
data(teaMerged)
pks <- getAllPeaks(teaMerged$CList, span = 11)
## show component 2 from the second file
par(mfrow = c(2,1))
plot(teaMerged, what = "profiles", showWindows = FALSE,
     mat.idx = 2, comp.idx = 2)
## and show where the peaks are picked
abline(v = pks[[2]][[2]][,"rt"], col = "gray")

## same for component 6
plot(teaMerged, what = "profiles", showWindows = FALSE,
     mat.idx = 2, comp.idx = 6, col = "red")
abline(v = pks[[2]][[6]][,"rt"], col = "pink")
```

`getPeakTable` *Convert MCR results into an ordered peak table*

Description

Function returns a matrix of intensities, where rows correspond to (aligned) features and columns to objects (samples, injections, ...). The function performs a complete linkage clustering of retention times across all samples, and cuts at a height given by the user (which can be interpreted as the maximal inter-cluster retention time difference). If two peaks from the same sample are assigned to the same cluster, and error message is given.

Usage

```
getPeakTable(peakList, response = c("area", "height"),
             use.cor = TRUE, maxdiff = 0.2, plotIt = FALSE, ask = plotIt)
```

Arguments

`peakList`	A nested list of peak tables: the first level is the sample, and the second level is the component. Every component is described by a matrix where every row is one peak, and the columns contain information on retention time, full width at half maximum (FWHM), peak width, height, and area.
`response`	An indicator whether peak area or peak height is to be used as intensity measure. Default is peak area.
`use.cor`	Logical, indicating whether to use corrected retention times (by default) or raw retention times (not advised!).
`maxdiff`	Height at which the complete linkage dendrogram will be cut. Can be interpreted as the maximal inter-cluster retention time difference.
`plotIt`	Logical. If TRUE, for every component a stripplot will be shown indicating the clustering.
`ask`	Logical. Ask before showing new plot?

Details

If one sees warnings about peaks from the same sample sharing a cluster label, one option is to reduce the `maxdiff` variable - this, however, will increase the number of clusters. Another option is to filter the peaks on intensity: perhaps one of the two peaks in the cluster is only a very small feature.

Value

The function returns a data frame where the first couple of columns contain meta-information on the features (component, peak, retention time) and the other columns contain the intensities of the features in the individual injections.

Author(s)

Ron Wehrens

Examples

```
data(teaMerged)
pks <- getAllPeaks(teaMerged$CList, span = 11)
warping.models <- correctRT(teaMerged$CList, reference = 2,
                            what = "models")
pks.corrected <- correctPeaks(pks, warping.models)
pkTab <- getPeakTable(pks.corrected, response = "area")
```

---

opa *Finding the most dissimilar variables in a data matrix: the Orthogonal Projection Approach*

Description

This function finds the set of most dissimilar rows in a data matrix. If no initial selection is presented, the first object is selected by comparison with the vector of column means. As a distance function the determinant of the crossproduct matrix is used.

Usage

```
opa(x, ncomp, initXref = NULL)
```

Arguments

x	Data matrix (numerical). May not contain missing values.
ncomp	Number of rows to be selected.
initXref	Optional matrix to be expanded - should be a subset of the rows to select.

Value

The function returns a submatrix of X, where the columns contain the (unit-length scaled) spectra from the input data that are most dissimilar.

Author(s)

Ron Wehrens

References

F. Questa Sanchez et al.: Algorithm for the assessment of peak purity in liquid chromatography with photodiode-array detection. Analytica Chimica Acta 285:181-192 (1994)

R. Wehrens: Chemometrics with R. Springer Verlag, Heidelberg (2011)

Examples

```
data(tea)

tea <- lapply(tea.raw, preprocess, maxI = 100)

ncomp <- 7
spectra <- opa(tea, ncomp)
```

```
myPalette <- colorRampPalette(c("black", "red", "blue", "green"))
mycols <- myPalette(ncomp)
matplot(as.numeric(rownames(spectra)), spectra, type = "l", lty = 1,
        xlab = expression(lambda), ylab = "", col = mycols)
legend("topright", legend = paste("Comp.", 1:ncomp), col = mycols,
       lty = 1, ncol = 2, bty = "n")
```

`preprocess` *Preprocessing smooth time-wavelength data*

Description

Standard preprocessing of response matrices where the first axis is a time axis, and the second a spectral axis. An example is HPLC-DAD data. For smooth data, like UV-VIS data, there is the option to decrease the size of the matrix by interpolation. By default, the data are baseline-corrected in the time direction and smoothed in the spectral dimension.

Usage

```
preprocess(X, dim1, dim2, remove.time.baseline = TRUE,
           spec.smooth = TRUE, maxI, ...)
```

Arguments

`X`	A numerical data matrix, missing values are not allowed. If rownames or colnames attributes are used, they should be numerical and signify time points and wavelengths, respectively.
`dim1`	A new, usually shorter, set of time points (numerical). The range of these should not be outside the range of the original time points, otherwise the function stops with an error message.
`dim2`	A new, usually shorter, set of wavelengths (numerical). The range of these should not be outside the range of the original wavelengths, otherwise the function stops with an error message.
`remove.time.baseline`	logical, indicating whether baseline correction should be done in the time direction. Default is TRUE.
`spec.smooth`	logical, indicating whether smoothing should be done in the spectral direction. Default is TRUE.
`maxI`	if given, the maximum intensity in the matrix is set to this value.
`...`	further optional arguments to the `baseline.corr` function.

Value

The function returns the preprocessed data matrix, with rownames and colnames indicating the time points and wavelengths, respectively.

Author(s)

Ron Wehrens

Examples

```
data(tea)
tpoints <- as.numeric(rownames(tea.raw[[1]]))
lambdas <- as.numeric(colnames(tea.raw[[1]]))

## limit retention time and wavelength ranges, and do smoothing and
## baseline correction
new.time <- seq(13, 14.1, by = .05)
new.wavelengths <- seq(400, 500, by = 2)
tea.raw1.processed <-
  preprocess(tea.raw[[1]], dim1 = new.time, dim2 = new.wavelengths)

plot(tpoints, tea.raw[[1]][,lambdas == 470],
     xlim = range(new.time), type = "l", col = "gray",
     main = "Chromatogram at 470 nm", xlab = "Time (min.)",
     ylab = "")
lines(new.time, tea.raw1.processed[,new.wavelengths == 470], col = "red")
legend("topleft", lty = 1, col = c("gray", "red"), bty = "n",
       legend = c("Original data", "Preprocessed data"))

plot(lambdas, tea.raw[[1]][tpoints == 13.7,],
     xlim = range(new.wavelengths),
     ylim = c(0, max(tea.raw[[1]][tpoints == 13.7,])),
     type = "l", col = "gray",
     main = "Spectrum at 13.7 min.", xlab = expression(lambda),
     ylab = "")
lines(new.wavelengths, tea.raw1.processed[new.time == 13.7,], col = "red")
legend("topleft", lty = 1, col = c("gray", "red"), bty = "n",
       legend = c("Original data", "Preprocessed data"))
```

`showALSresult` *Plot ALS results in a more elaborate way*

Description

Simultaneous visualization of pure components (spectra and time profiles) and either raw data, fitted data or residuals.

Usage

```
showALSresult(xals, xlst,
              tp = getTime(xals), wl = getWavelength(xals),
              mat.idx = 1:length(xlst),
              img.col = terrain.colors(10), zlim, xlab, ylab,
              compound.col = 1:ncol(xals$S), logsc = TRUE,
              plotPureC = c("both", "spec", "conc", "none"),
              titles, annotation = show.img,
              PureChght = 0.33, PureCwdth = min(nplot, 5)/5 - 0.1,
              show.img = TRUE)
```

Arguments

xals	The fitted ALS object. Not needed if plotPureC equals "none": in that case only the data are shown.
xlst	The data: the list of matrices on which the ALS object is based.
tp	Optional vector of time points. If missing, will be determined using function getTime.
wl	Optional vector of wavelengths. If missing, will be determined using function getWavelength.
mat.idx	Indices of the samples to be visualized.
img.col	Color vector for image.
zlim	Range of the image colors.
xlab, ylab	Axis annotation strings.
compound.col	Colors to be used for components in the pure specta/profile plots.
logsc	Logical, indicating whether the images should be logscaled vefore visualization. Default: TRUE.
plotPureC	Determines which part (if any) of the pure components is shown.
titles	Titles for the plots for the individual samples. If not given, the names of the xlst elements will be used.
annotation	if a text string, this will be shown in the top right corner panel. If anything else but FALSE, a color bar will be drawn. The default is to show the color bar for images and not to show it for contour plots.
PureChght	Height, relative to the height of the data panels, of the top row of pure concentration profiles
PureCwdth	Width, relative to the width of the data panels, of the right column of pure spectra
show.img	logical, indicating whether image is used (the default) or contour

Author(s)

Ron Wehrens

See Also

plot.ALS

Examples

```
data(teaMerged)

ncomp <- ncol(teaMerged$S)
myPalette <- colorRampPalette(c("black", "red", "blue", "green"))
mycols <- myPalette(ncomp)

maxResid <- max(abs(range(teaMerged$resid)))
showALSresult(teaMerged, teaMerged$resid,
              compound.col = mycols, logsc = FALSE, img.col = cm.colors(9),
              mat.idx = 2:4, zlim =c(-maxResid, maxResid))
```

tea *HLPC-DAD data for grape extracts conserved with TEA*

Description

Five (very much compressed) HPLC-DAD data matrices of grape extracts after several storage times. All extracts come from the same pooled sample. Since the raw data are given (no smoothing or baseline subtraction has been done, only subsetting of the time and wavelength axes), the object is called `tea.raw`.

Usage

```
data(tea)
```

Format

The UV-Vis data (`tea.raw`) are given as a list of five matrices, each of dimension 97 times 209 (time x wavelength). The names of the list indicate the day of measurement - day 0 is represented by two measurements.

Source

Provided by Elisabete Carvalho.

References

This is part of the data that have been used in: R. Wehrens, E. Carvalho, D. Masuero, A. de Juan and S. Martens: High-throughput carotenoid profiling using multivariate curve resolution. Anal. Bioanal. Chem. 15:5057-5086 (2013)

Examples

```
data(tea)
tpoints <- as.numeric(rownames(tea.raw[[1]]))
lambdas <- as.numeric(colnames(tea.raw[[1]]))
contour(tpoints, lambdas, tea.raw[[1]], col = terrain.colors(15),
        xlab = "Retention time (min.)", ylab = "Wavelength (nm)")
```

teaMerged *Results of an ALS analysis on individual windows*

Description

Object of class `ALS`: the result of the analysis of the `tea` data, using three time windows with an overlap parameter of 10. The three ALS models have been merged into one ALS object, which can be inspected and used for further analysis.

Usage

```
data(teaMerged)
```

Examples

```
## generation of the data
data(tea)
new.lambdas <- seq(260, 500, by = 2)
tea <- lapply(tea.raw,
              preprocess,
              dim2 = new.lambdas)
tea.split <- splitTimeWindow(tea, c(12, 14), overlap = 10)
tea.alslist <- lapply(tea.split,
                      function(Xl) {
                        Xl.opa <- opa(Xl, 4)
                        doALS(Xl, Xl.opa)
                      })
teaMerged <- mergeTimeWindows(tea.alslist)
## This is the object saved in teaMerged.RData

ncomp <- ncol(teaMerged$S)
myPalette <- colorRampPalette(c("black", "red", "blue", "green"))
mycols <- myPalette(ncomp)

plot(teaMerged, what = "spectra", col = mycols)
legend("top", col = mycols, lty = 1, bty = "n", ncol = 2,
       legend = paste("C", 1:ncol(teaMerged$S)))
```

`windows` *Splitting and merging of data across the time axis.*

Description

Often MCR data sets can be analysed much more quickly and efficiently when split into several smaller time windows. For interpretation purposes, the results after analysis can be merged again.

Usage

```
splitTimeWindow(datalist, splitpoints, overlap = 0)
mergeTimeWindows(obj, simSThreshold = .9, simCThreshold = .9, verbose = FALSE)
```

Arguments

`datalist`	A list of (numerical) data matrices
`splitpoints`	A numerical vector of cut points. In case the time axis extends beyond the range of the cut points, additional cut points are added at the beginning or at the end of the time axis to ensure that all time points are taken into account.
`overlap`	Number of points in the overlap region between two consecutive windows. Default: 0 (non-overlapping windows).
`obj`	Either experimental data that have been split up in different time windows (a list of matrices), or a list of ALS objects. See details section.
`simSThreshold, simCThreshold`	similarity thresholds to determine whether two patterns are the same (correlation). The two thresholds are checking the spectral and chromatographic components, respectively. If no overlap is present between time windows, `simCThreshold` is not used.
`verbose`	logical: print additional information?

Details

When splitting data files, the non-overlapping areas should be at least as big as the overlap areas. If not, the function stops with an error message. Note that the example below is only meant to show the use of the function: the data do not have enough time resolution to allow for a big overlap.

Value

Function `splitTimeWindows` splits every matrix in a list of data matrices into submatrices corresponding to time windows. This is represented as a list of lists, where each top level element is one time window. Such a time window can then be presented to the ALS algorithm.

Function `mergeTimeWindows` can be used to merge data matrices as well as ALS result objects. In the first case, for each series of data matrices corresponding to different time windows, one big concatenated matrix will be returned. In the second case, exactly the same will be done for the residual matrices and concentration profiles in the ALS object. Spectral components are assumed to be different in different time windows, unless they have a correlation higher than `simSThreshold`, in which case they are merged. If overlapping time windows are used, an additional requirement is that the similarity between the concentration profiles in the overlap area must be at least `simCThreshold`. This similarity again is measured as a correlation.

Author(s)

Ron Wehrens

Examples

```
## splitting and merging of data files
data(tea)
tea.split <- splitTimeWindow(tea.raw, c(12, 14))
names(tea.split)
sapply(tea.split, length)
lapply(tea.split, function(x) sapply(x, dim))
rownames(tea.split[[1]][[1]])[1:10]
rownames(tea.split[[2]][[1]])[1:10]

tea.merge <- mergeTimeWindows(tea.split)
all.equal(tea.merge, tea.raw)                    ## should be TRUE

tea.split2 <- splitTimeWindow(tea.raw, c(12, 14), overlap = 10)
lapply(tea.split2, function(x) sapply(x, dim))
tea.merge2 <- mergeTimeWindows(tea.split2)
all.equal(tea.merge2, tea.raw)                   ## should be TRUE

## merging of ALS results
data(teaMerged)
ncomp <- ncol(teaMerged$S)
myPalette <- colorRampPalette(c("black", "red", "blue", "green"))
mycols <- myPalette(ncomp)

## show spectra - plotting only a few of them is much more clear...
plot(teaMerged, what = "spectra", col = mycols, comp.idx = c(2, 6))
legend("top", col = mycols[c(2, 6)], lty = 1, bty = "n",
       legend = paste("C", c(2, 6)))

## show concentration profiles - all six files
plot(teaMerged, what = "profiles", col = mycols)
```

```
## only the second file
plot(teaMerged, what = "profiles", mat.idx = 2, col = mycols)
legend("topleft", col = mycols, lty = 1, bty = "n",
       legend = paste("C", 1:ncol(teaMerged$S)))
## Note that components 2 and 6 are continuous across the window borders
## - these are found in all three windows
```

Index