Package 'msProcess'

April 24, 2008

Type Package

Title Protein Mass Spectra Processing

Version 1.0.1

Date Fri Apr 18 14:53:52 PDT 2008

Author Lixin Gong, William Constantine, and Alex Chen

Maintainer Lixin Gong <lgong@insightful.com>

Depends R (>= 2.5.0), RSQLite (>= 0.5-6), wmtsa, robust, XML, stats

DependsSplus RSQLite (>= 0.5-6), wmtsa, robust, SPXML

Description This package provides tools for protein mass spectra processing including data preparation, denoising, noise estimation, baseline correction, intensity normalization, peak detection, peak alignment, peak quantification, and various functionalities for data ingestion/conversion, mass calibration, data quality assessment, and protein mass spectra simulation. It also provides auxiliary tools for data representation, data visualization, and pipeline processing history recording and retrieval.

License GNU General Public License Version 2

URL http://www.insightful.com/services/research/proteome/default.asp

R topics documented:

`apply` *S3 generic apply method for msSet Class*

Description

The `apply` function in S-PLUS is S3 generic, but it is not so in R. For the msProcess package, the `apply` function is overloaded to be an S3 generic function, relying on `UseMethod("apply")` to distribute the call. The `apply.default` function is defined to be `base::apply`, so if the class of the `X` input is not `"msSet"` then the standard R definition will be used.

Usage

```
apply(X, MARGIN, FUN, ..., type="intensity", pre=NULL, covar=NULL)
```

Arguments

`X`	an `msSet` object.
`MARGIN`	an integer denoting the dimension over which the given function is applied. Use `MARGIN=1` for rows and `MARGIN=2` for columns.
`FUN`	a function to be applied to the specified array sections, or a character string giving the name of the function.
`...`	any arguments to `FUN`. They are passed unchanged to each call of `FUN` and include their names.
`type`	a character string specifying the name of the array in the `msSet` object list to operate over. A typical value is `"intensity"` or `"noise"`, but the name of any legitimate matrix attached to the primary `msSet` object list can be used for `type`. Default: `"intensity"` (the intensity matrix).
`pre`	a function that is applied to the matrix prior to processing the data. Typical examples would be `pre=t` (matrix transpose), `pre=log` (log of matrix), etc. Default: `NULL` (no function is applied a priori).
`covar`	a named list of additional matrices to be parsed in the same manner as the primary matrix (specified by `type`). The contents of the `covar` matrices are also sent to the `FUN` function as an input argument with the same name as that supplied in the `covar` list. As an example, assuming `x` is an object of class `msSet` that contains the matrices `x$intensity` and `x$z`, then the call: `apply(x, MARGIN=1, FUN="foo", type="intensity", covar=list(z=z))`, will ultimately result in calls `foo(x$intensity[i,], z=x$z[i,])`, where `i=1:numRows(x$intensity)`. Note that the matrix `x$z` need only have the same number of rows in this case since `MARGIN=1`, but need not necessarily contain the same number of columns, i.e., restrictions on the dimensions of `covar` matrices not specfied by `MARGIN` are controlled by the `FUN` function. Default: `NULL` (no covariate matrices).

Value

a matrix containing the result of the FUN function applied to the matrix of type type found in the original msSet object X.

See Also

msList, msSet.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

# find the means of each spectrum
# and convert the result to a single-row matrix
means <- apply(qcset, MARGIN=2, mean)
nc <- NCOL(qcset$intensity)
means <- matrix(means, ncol=nc)
print(means)

# add the means (single-row) matrix to the original
# msSet object and verify its existence
z <- msSet(qcset, means=means)
names(z)
is.matrix(z$means)

# to illustrate the use of the 'covar' argument in apply,
# create a faux function that finds tha maximum absolute
# difference between each spectrum and its mean value
foo <- function(x,meanvals) max(abs(x-meanvals))
maxdiff <- as.vector(apply(z, MARGIN=2, FUN=foo, covar=list(meanvals=means)))
print(maxdiff)

# verify the results: should get vector of nc zeros
unlist(lapply(seq(along=maxdiff), function(i,z,maxdiff)
  vecnorm(max(abs(z$intensity[,i]-z$means[,i])) - maxdiff[i]),
  z=z, maxdiff=maxdiff))
```

---

argNames *Display the Argument List of a Function*

Description

Displays only the argument names for a function.

Usage

```
argNames(x)
```

Arguments

x	the name of a function. A character string is also accepted, and is useful when the function name would not be interpreted as a name by the parser.

Value

the names of the arguments.

See Also

args.

Examples

```
argNames(lm)
argNames("lm")
```

---

assignEvent *Update a Previously Thrown History Event with New Information*

Description

Updates a history event previously thrown to a specified frame (or envirnoment in R). Typically, a caller function will have thrown the event while the callee(s) update the event information using the assignEvent function. That information can then be retrieved (typically by the caller) using catchEvent.

Usage

```
assignEvent(record, process=NULL, histname="event.history", envir=NULL)
```

Arguments

record	a list containing named character vectors describing the event in the form list(proc2="Descrip 1", proc2="Description 2") and so on. Here, *EventName* is a character string defining the name of the event, and the named list variables *proc1* and *proc2* are character strings that define the specific proceedings for that event. Each of these proceedings is descibed more thoroughly by the assigned character string. If, for a given event, the user wishes only to register the name of the event sans extra proceedings information, set this variable to a blank character string "" and specify only the second event argument.
envir	the frame in S-PLUS (or environment in R) designated for the processing and storage of pipeline history data. Default: msProcessEnv, a global environment impicitly set by a previous call to throwEvent. In general, the user should rely on the specified default value.
histname	a character string defining the name of the history variable stored in the specified frame. Default: "event.history".
process	a character string defining a title for the current information being recorded. This typically serves as a flag for other functions so that the same process is not written more than once. This can be checked using the isProcessRecorded function.

Value

no output is returned directly. Instead, the variable histname in frame frame is updated with the supplied event information.

Note

If assignable, the specified `histname` object is updated in the specified frame with the new process information.

See Also

`throwEvent`, `catchEvent`, `isProcessRecorded`.

Examples

```
## throw an event in the global frame
envir <- msGlobalEnv()
throwEvent("The 2005 British Open Championship", envir=envir)

## assign data to the thrown event
record  <- list(Winner="Tiger Woods")
process <- "champion"
assignEvent(record, "champion", envir=envir)

## verify process has been recorded: TRUE
isProcessRecorded(process, envir=envir)

## catch event
catchEvent(NULL, envir=envir)
```

`calibrants` *Class Representing a Set of Calibrants*

Description

A set of calibrants only contains a small number (typically 5 to 7) of proteins of known mass.

Class slots:

masses A numeric vector of protein masses.

counts An integer vector of protein counts/abundance.

Class extends:

proteins by direct inclusion.

The class `calibrants` inherits all the methods of the class `proteins`.

Usage

```
calibrants(masses, counts)
```

Arguments

`masses`	A positive numeric vector of protein masses, whose elements should be unique.
`counts`	A positive integer vector of protein counts/abundance, which should have the same length as `masses`.

References

Coombes, K.R., Koomen, J.M., Baggerly, K.A., Morris, J.S., Kobayashi, R., "Understanding the characteristics of mass spectrometry data through the use of simulation," *Cancer Informatics*, **2005(1)**:41–52, 2005.

See Also

`proteins`, `spectrometer`.

Examples

```
## generate two protein samples
cal1 <- calibrants(masses=c(1, 95, 190), counts=as.integer(c(500, 3000, 10000)))
cal2 <- calibrants(masses=10000+200*(0:3), counts=as.integer(c(12000, 4000, 2000, 1000)))

## print the synopsis of the protein samples
cal1
cal2

## mix the protein samples
cal <- cal1 + cal2

## visualize the calibrants
plot(cal, type="h")
```

---

calibrator *Class Representing the Calibrator of a Mass Spectrometer*

Description

Class slots:

time.mean a numeric scalar denoting the mean time-of-flight of the calibrants.

model an object of class `lm`.

error.rel a numeric vector denoting the relative calibration error for the calibrants.

References

Coombes, K.R., Koomen, J.M., Baggerly, K.A., Morris, J.S., Kobayashi, R., "Understanding the characteristics of mass spectrometry data through the use of simulation," *Cancer Informatics*, **2005(1)**:41–52, 2005.

See Also

`spectrometer`.

catchEvent *Catch a History Event that has been Thrown*

Description

Catches a history event that has been thrown, extracts the history, and attaches/updates the history in the primary input object.

Usage

```
catchEvent(x, histname="event.history", envir=NULL)
```

Arguments

x	an object of arbitrary class. Optionally, this input may already contain an event history in which case the history is updated after being caught. Otherwise the new history is attached.
envir	the frame in S-PLUS (or environment in R) designated for the processing and storage of pipeline history data. Default: msProcessEnv, a global environment impicitly set by a previous call to throwEvent. In general, the user should rely on the specified default value.
histname	a character string defining the name of the history variable stored in the specified frame. Default: "event.history".

Value

a replication of x with the event history updated/attached.

Note

If available, the last entry of the specified histname object (a list located in the specified frame) is extracted and written to the input x via the msSet constructor function. If no other entries exist after extraction, the histname list is deleted from the specified frame.

See Also

throwEvent, assignEvent, isProcessRecorded, eventHistory.

Examples

```
## throw an event
envir <- msGlobalEnv()
throwEvent("Superbowl XL", envir=envir)

## assign data to the thrown event
record  <- list(NFC="Seattle Seahawks", AFC="Pittsburgh Steelers")
assignEvent(record)

## catch event
catchEvent(NULL)
```

cypherGenXML2Bin *Convert Cyphergen XML files into binary files*

Description

Convert one or a set of Cyphergen XML files into binary files to be imported into SQLite database.

Usage

```
cypherGenXML2Bin(name, pattern, path, mz, tof = FALSE, maxRows = 10000,
        append = TRUE, verbose=TRUE, ...)
cypherGenXMList2BinBlocks(x, pattern, tof = FALSE, maxRows = 10000,
        maxCols = NULL, path = ".", category = NULL, verbose = 1, ...)
```

Arguments

name	A character string specifying the name of a xml file.
pattern	A character string to specify the common prefix for a series of binary files.
path	A character to specify the name of input directory.
mz	A numeric vector specifying mz peaks to be used.
maxRows	An integer defining the maximum number of rows in a binary file.
tof	A logical indicating whether TOF data or the processed data should be read.
append	A logical. If FALSE, then existing files will be overwritten. Otherwise, data will be appended onto the existing file.
x	A vector of characters specifying a set of XML files to be imported.
maxCols	An integer defining the maximum number of columns in a binary file, or NULL, in which case all xml files will be put into one sqliteTable.
category	A data.frame object containing categorical information.
verbose	A logical or a non-negative integer value to control whether or not to print out extra messages during processing. The larger the value, the more information is printed out.
...	Additional optional arguments.

Details

cypherGenXML2Bin converts a single Cyphergen XML file into a series of binary files via function writeBinBlocks. Each file contains at most maxRows mz peaks. Currently we assume calibration has been made by Cyphergen mass spectrometer. We might support customized input of a, b, t0, u in the future.

cypherGenXMList2BinBlocks is a batch version of cypherGenXML2Bin. It converts a list of xml files into a list of binary files. Each binary file contains at most maxRows rows and maxCols columns. If provided, category will be used as group factor to partition XML files into groups, and each binary file contains XML files within the same group.

mz provides a calibration standard to make all mass spectra sharing the same set of mass-charge ratio peaks.

maxRows is used as input to mxRow in function writeBinBlocks.

Value

`cypherGenXML2Bin` returns `NULL`.

`cypherGenXMList2BinBlocks` returns a list containing the following items:

`sampleTable` A data frame that contains table topology.

`categoryTable` A data frame that contains categorical information.

Author(s)

Y. Alex Chen <ychen@insightful.com>

See Also

readBinMatrix, writeBinBlocks, importBin2Sqlite

Examples

`importBin2Sqlite` *Import binary file into SQL*

Description

Functions to import binary files into one or multiple SQL tables. `binblocks2SQLite` reads a series of binary files into a single table. `importBin2Sqlite` is an extension to `binB2SQLite` in order to read a series of binary files into multiple SQLite tables.

Usage

```
binblocks2SQLite(conn, path, tablename, columnID, pattern, what = "double",
  append = FALSE, verbose = TRUE, ...)
importBin2Sqlite(conn, path, tablename, sampleTable, what = "double", categoryTa
  prefix = "", verbose = 1, ...)
```

Arguments

`conn`	Either a `connection` object or a `character`.
`path`	A `character` defining
`tablename`	A `character` string specifying the SQL table name.
`columnID`	A `vector` of characters specifying the column names.
`pattern`	A `character` string specifying patterns in file names.
`what`	A codecharacter to specifying the data type in the binary file. It is possible that a binary file might contain several types of data, but we don't support that currently.
`append`	A logical value. If `FALSE`, then existing tables will be overwritten. Otherwise, data will be appended onto the existing table.

verbose	A logical value or non-negative integer to control whether or not to print out extra information.
sampleTable	A data frame containing three fields: sampleID, tableID and pattern.
categoryTable	A data frame containing categorical information.
prefix	A character string specifying optional prefix on table names.
...	Additional optional arguments.

Details

binB2SQLite reads a series of binary files with names sharing pattern pattern into a single SQLite table name on database conn.

importBin2Sqlite requires a data frame, (sampleTable) that contains table topology, i.e. what table contains what columns, and what table contains what kind of binary files. The sampleTable will be imported to SQL database conn in additon to all binary files. If an optional data frame category is specified, it will also be imported into SQLite. In that case, users can use SQL JOIN to get category related data.

Value

Both functions return a logical indicating whether the importing was successful or not.

Note

Users must make sure that all binary files share the same dimensionality.

Author(s)

Y Alex Chen <ychen@insightful.com>

See Also

readBin, readBinMatrix, writeBinBlocks

Examples

---

importXMLDir *Import a directory of cyphergenXML files into SQLite*

Description

A wrapper function to import a directory of cyphergenXML files into a SQLite database.

Usage

```
importXMLDir(xmldir, dbname, tablename, tof = FALSE, maxRows = 10000,
  maxCols = NULL, tmpdir = tempdir(), splitSubdir = TRUE, verbose = 0,
  ...)
```

Arguments

xmldir	A character specifying the directory that holds xml files.
dbname	A character specifying the full name of a SQLite database file, including path.
tablename	A character string specifying the SQL table name.
tof	A logical determining whether reading in TOF or processed data.
maxRows	An integer specifying the largest number of rows that a intermediate binary file can hold.
maxCols	An integer specifying the largest number of columns that the resulting SQLite table can hold. Or a NULL value, indicating that all xml files to be put into a single table. If the number of cyphergenXML files are too large, it is recommended to specify a maxCols at around 100 so that xml files can be partitioned into several SQLite tables.
tmpdir	A character specifying the name of the temporary directory to store binary files.
splitSubdir	A logical. If TRUE, the subdirectory will be used as grouping factors: all xml under the same subdirectory will be put into the same category. Otherwise, no category structure will be used. See cypherGenXMList2BinBlocks for details.
verbose	A logical or non-negative integer specifying the extend of extra messages to be printed out.
...	Additional optional arguments.

Details

This function will import all cyphergenXML files under a certain directory (including subdirectory) into a SQLite database. Each XML file contains one mass spectra. XML files can be grouped into subdirectories so that each group of XML files will go into the same SQLite table. Otherwise all XML files are treated as the same group. SQLite Tables should not contain too many columns. Therefore a limit is given by maxCols. If the number of XML files in a group is too large, we split the XML files evenly into multiple tables. maxRows determined the size of intermediate binary files. If it is too large, the intermediate file might be out of memory and could not be read in.

Value

It returns a logical indicating whether the importing was successful or not.

Author(s)

Y Alex Chen

See Also

importBin2Sqlite, cypherGenXMList2BinBlocks

Examples

```
## Not run:
        xmldir <- "E:\SQLData\UPCI-2007-06\UPCI AUG WCX"
        dbname <- "e:\mydatabase1.db"
        system.time(p<-importXMLDir(xmldir, dbname, tof=FALSE, split=FALSE,
        maxRows=5000, tablename="nocattable", verbose=3))
        conn <- dbConnect("SQLite", "e:/mydatabase1.db", cache.size=100000)
```

```
        dbListTables(conn)
## End(Not run)
```

`readBinMatrix` *Read a binary file containing dimension information*

Description

`readBinMatrix` reads a binary file containing dimension information, and the result is wrapped into a matrix.

Usage

```
readBinMatrix(name, what = "double", ncol = 1e+06, ...)
```

Arguments

`name`	A `character` string.
`what`	A `character` string.
`ncol`	An `integer` specifying an upper limit of the column number.
`...`	Additional optional arguments.

Details

`readBinMatrix` reads a binary file containing dimension information and wrap the data into a matrix. The first element in the input file `name` must be an integer `n`, determining the length of dimensionality. The following `n` elements must also be integers, specifying the actual dimensionality, followed by data elements. Only the first dimension will used to determine the number of rows. This function needs user-defined ncol (the number of columns). If ncol is unknown, a pre-defined upper limit (1000000) will be used.

Value

returns a logical indicating whether the importing was successful or not.

Author(s)

Y. Alex Chen <ychen@insightful.com>

See Also

readBin, importBin2Sqlite

`writeBinBlocks` *Write a series of binary files containing data blocks*

Description

Functions to write a vector `x` into a series of binary files.

Usage

```
writeBinBlocks(x, pattern, maxRows=10000, path=".", what = c("double", "integer"
        append = FALSE, verbose = TRUE)
```

Arguments

`x`	A `character` string to specify the name of the binary file.
`path`	A `character` string to specify the path to input binary file.
`what`	A `character` string denoting the mode of the data to be read, one of: '"double", "integer", "character"'.
`append`	A `logical` value. If `FALSE`, then existing files will be overwritten. Otherwise, data will be appended onto the existing file.
`pattern`	A `character` string to specify the common prefix for a series of binary files.
`maxRows`	An `integer` to specify the maximum number of rows allowed in a single binary file. See details for more information.
`verbose`	A `logical` value specifying whether or not to print verbose messages.

Details

To facilitate handling large data, function `writeBinBlocks` is provided, in which a long `x` is split into several small parts, each containing less than `maxRows` rows. A series of binary files, like [`pattern`]1, [`pattern`]2, [`pattern`]3, ..., will be thus generated.

Value

returns an invisible `NULL`.

Author(s)

Y. Alex Chen <ychen@insightful.com>

See Also

writeBin, readBinMatrix, importBin2Sqlite

Examples

`eventHistory` *Event History Creation*

Description

The `S+Proteome` module provides a wealth of preprocessing functionality, some or all of which may be used as a preface for subsequent classification investigations. This function allows the user to document the processing pipeline by adding text to a particular attribute of an object that describes the current processing state. Each process in the pipeline is referred to as an *event*. This function will typically not be called directly by the user and will alternatively be called from within various preprocessing functions.

Usage

```
eventHistory(x, ..., sub.label="   ", time.stamp=date(), action="append")
```

Arguments

`x`	an object of any class.
`...`	one or more named lists, each containing named character vectors describing an event to register. Each list must be in the form EventName = list(proc2="Description 1", proc2="Description 2") and so on. Here, *EventName* is a character string defining the name of the event, and the named list variables *proc1* and *proc2* are character strings that define the specific proceedings for that event. Each of these proceedings is descibed more thoroughly by the assigned character string. If for a given event, the user wishes only to register the name of the event sans extra proceedings information, it is allowable to make the call ala `eventHistory(x, "Event A", "Event B")` and so on. In this case, each character string is taken to mean the name of the corresponding event.
`action`	a character string defining the action to take with the new history information. Supported values are as follows: **prepend** Prepend new history to the existing event history. **append** Append new history to the existing event history. **replace** Replace the event history with the new history. **merge** Update old events which have a common event name with new events. The uncommon events are appended. Default: `"append"`.
`sub.label`	a character string used to preface the proceedings lines for each event when a history summary is printed on the command line. Default: `" " (3 blanks)`.
`time.stamp`	a character string defining a time stamp for the process(es) being documented. This time stamp will be automatically added to each process in the event. Default: `date()`.

Value

a replication of the original object, with an attached attribute of class `eventHistory` containing a vector of formatted character strings defining the processing history of the input object.

S3 METHODS

@lsb event data access. Input either a character string or an integer defining the registered event in the history.

print pretty-prints the event history. Optional arguments for this method are

pre Character string to preface each event header. Default: `paste("[", attr(x, "index"), "]", sep = "")`.

See Also

`assignEvent`, `isProcessRecorded`, `existHistory`, `getHistory`, `eventHistory`.

Examples

```
## create a list of simple objects
z <- list(dog="chihuahua", vals=1:5, colors=c("red","green","blue"))

## remove mean from vals and document as a history
## event
z$vals <- z$vals - mean(z$vals)
z <- eventHistory(z, "Process A"=list(values="mean removed"))

## now sort the colors and change the dog name,
## and document both actions in the same call
## to msHistory as separate processes
z$colors <- sort(z$colors)
z$dog    <- "pomeranian"
z <- eventHistory(z, "Process B"=list(colors="sorted alphabetically"),
    "Process C"=list(dog="name change", result="more hair"))

## add an event without proceedings
z <- eventHistory(z, "Event A", "Event B")

## print the history
print(getHistory(z))

## replace some of the events with new information
z <- eventHistory(z, "Event A"=list(show="pony"), "Process A",
action="merge")
print(getHistory(z))

## prepend some new events
z <- eventHistory(z, new=list(alpha="first greek letter"),
    action="prepend")
print(getHistory(z))

## replace event history altogether
z <- eventHistory(z, newest=list(omega="final greek letter"),
    action="replace")
print(getHistory(z))

## return an object of class eventHistory, i.e.,
## not attached to any other object
eventHistory(NULL,
    symphony=list(string="violins and cellos", percussion="drums",
       reed="flutes and oboes"))
```

existHistory *Verify Existence of an Embedded History Object*

Description

Checks for the existence of an object of class `eventHistory` attached an as attribute to the input object. Additionally, the user can seek the existence of a particular event within that history.

Usage

```
existHistory(x, event=NULL)
```

Arguments

`x`	an object of arbitrary class.
`event`	a character string defining a registered event to query. If no such event has been registered in a history object (or the history object does not exist) then a `FALSE` is returned. Default: `NULL` (do not search for a specific event).

Value

a logical value. A `TRUE` value is returned if a history object exists as an attribute of the primary input object `x`. If `event` is also specified, the output will be a logical value defining whether or not that event has been registered in the history object.

See Also

`assignEvent`, `isProcessRecorded`, `getHistory`, `eventHistory`.

Examples

```
## check for the existence of a registered history
## in an object without one (FALSE)
z <- 1:5
existHistory(z)

## create a simple history
z <- eventHistory(z, "Event A"=list(number="first", positive="yes"),
    "Event B"=list(horse="mustang"))

## check for a registered history (TRUE)
existHistory(z)

## check to see if "Event A" has been registered
## (TRUE)
existHistory(z,"Event A")

## check to see if "Event D" has been registered
## (FALSE)
existHistory(z,"Event D")
```

`getHistory` *Extract an Event History from an Arbitrary Object*

Description

Extracts a registered event history from an arbitrary object or a specified event within that history.

Usage

```
getHistory(x, event=NULL)
```

Arguments

`x`	an object of any class.
`event`	a character string or integer defining a specific event to query. If no such event has been registered in a history object (or the history object does not exist) then a `NA` is returned. Default: `NULL` (do not extract a specific event).

Value

an object of class `msHistory`. If `event` is specified, only that event is returned in the history.

See Also

`assignEvent`, `isProcessRecorded`, `existHistory`, `eventHistory`.

Examples

```
## create a simple history
z <- 1:5
z <- eventHistory(z, "Event A"=list(number="first", positive="yes"),
    "Event B"=list(horse="mustang"),
    "Event C"=list(string="violins and cellos", percussion="drums",
       reed="flutes and oboes"))

## extract the entire event history
getHistory(z)

## extract only Event B (each method is
## equivalent)
getHistory(z,"Event B")
getHistory(z,2)
getHistory(z)["Event B"]
getHistory(z)[2]

## return entire history except the second event
## (each method is equivalent)
getHistory(z,-2)
getHistory(z)[-2]

## attempt to extract a non-existent event (NA is
## returned)
getHistory(z)["dogs"]
```

ion.focus.delay *Simulating Linear MALDI-TOF with Ion Focus Delay*

Description

Given the mass, initial velocity, and the instrument parameters, it produces the time-of-flight for the protein particles.

Usage

```
ion.focus.delay(mass, v0, setting)
```

Arguments

mass	A vector of masses in daltons.
v0	A matching vector of initial velocities in meters/second.
setting	An object of class setting containing the machine setting.

Value

The time-of-flight for the protein particles.

References

Coombes, K.R., Koomen, J.M., Baggerly, K.A., Morris, J.S., Kobayashi, R., "Understanding the characteristics of mass spectrometry data through the use of simulation," *Cancer Informatics*, **2005(1)**:41–52, 2005.

See Also

spectrometer.

isProcessRecorded *Verifies the Existence of a Recorded Process in a Thrown Event*

Description

Checks for the existence of a previously recorded process in a thrown event history.

Usage

```
isProcessRecorded(process, histname="event.history", envir=NULL)
```

Arguments

process	a character string defining the name of the process.
envir	the frame in S-PLUS (or environment in R) designated for the processing and storage of pipeline history data. Default: msProcessEnv, a global environment impicitly set by a previous call to throwEvent. In general, the user should rely on the specified default value.
histname	a character string defining the name of the history variable stored in the specified frame. Default: "event.history".

Value

a logical value. If TRUE, the process has already been recorded in the thrown event.

See Also

throwEvent, catchEvent, assignEvent, eventHistory.

Examples

```
## throw an event
envir <- msGlobalEnv()
throwEvent("The 2005 British Open Championship", envir=envir)

## assign data to the thrown event
record  <- list(Winner="Tiger Woods")
process <- "champion"
assignEvent(record, process)

## verify process has been recorded: TRUE
isProcessRecorded(process)

## catch event
catchEvent(NULL)

## once event has been caught, isProcessRecorded returns FALSE
isProcessRecorded(process)
```

---

matchObject — *Search for an Object Name Matching a Character String on the Data Directory Search List*

Description

Returns a character string which is the name of an S-PLUS object in a position on the search list.

Usage

```
matchObject(what, ignore.case=TRUE)
```

Arguments

what	a character string to be searched as the name of an S-PLUS object.
ignore.case	if TRUE upper- and lowercase analogs are considered equivalent when matching.

Value

a character string which is the name of an S-PLUS object in a position on the search list is returned.

See Also

ls, objects.

Examples

```
matchObject("max")
matchObject("lm")
matchObject("glm")
```

`msAlign` *Peak Alignment*

Description

Performs cross-spectral alignment of detected peaks.

Usage

```
msAlign(x, FUN="cluster", mz.precision=0.003, snr.thresh=10,...)
```

Arguments

`x`	An object of class `msSet` containing a `"peak.list"` element.
`FUN`	A character string specifying the method to use for alignment. Choices are **`"cluster"`:** clusters peaks using one-dimensional hierarchical clustering and uses distances between peak locations as the similarity measure. **`"gap"`:** analyzes peaks sequentially from low mass to high mass. Two adjacent peaks are classified into the same class if the distances between their locations is smaller than the specified threshold. **`"vote"`:** clusters peaks iteratively. Each peak is associated with a window and the number of peaks that fall within the window across all samples is counted. The peak corresponding to the highest count forms a new peak cluster and all the peaks that have contributed to this peak are removed. The procedure is repeated until all peaks are exhausted from every sample. **`"mrd"`:** clusters peaks by smoothing a histogram of scale-based feature locations for all spectra as identified by a call to `msPeak(x,FUN="mrd", ...)`. The midpoints of the valleys in the smoothed histogram identifies the common peak locations across corresponding spectra. Default: `"cluster"`.
`mz.precision`	A numeric value, used to construct the threshold when performing clustering. The default value is 0.003 because SELDI data is often assumed to have $\pm 0.3\%$ mass drift, i.e., a peak at mass `w` could represent a protein with a mass within the interval $[w(1-0.003), w(1+0.003)]$.
`snr.thresh`	A non-negative numeric value. The peaks with signal-to-noise ratio larger than this value will be used to construct the common set of peak classes. Default: 10.
`...`	Additional arguments passed to the `msAlignMRD` function.

Details

Currently, the mass accuracy of a mass spectrometer is proportional to the mass-to-charge (m/z) values. Thus, for a given set of spectra, the locations of the detected peaks will vary from spectrum to spectrum. In order to perform a comparative analysis of an ensemble of spectra, it is then a prerequisite to perform inter-sample alignment of the detected peaks. This process is normally called peak alignment or clustering.

The basic idea for peak alignment is to group peaks of similar molecular weight across all spectra into peak clusters or classes to form a superset, allowing for slight variations in mass. Each cluster is representative of a particular protein. Various methods have been proposed to align the peaks, which differ in how the superset is constructed.

Value

An object of class `msSet`, which is the input `x` with the added element `"peak.class"`: a matrix with peak classes as rows and some summary statistics of the peak clusters as columns. These statistics include the location, left bound, right bound and peak span of the peak classes in both clock tick (`"tick.loc"`, `"tick.left"`, `"tick.right"`, `"tick.span"`) and mass measure (`"mass.loc"`, `"mass.left"`, `"mass.right"`, `"mass.span"`).

References

Coombes, K.R., Tsavachidis, S., Morris, J.S., Baggerly, K.A., and Kuerer, H.M., "Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform," *Proteomics*, **5**:4107–17, 2005.

Tibshirani, R., Hastie, T., Narasimhan, B., Soltys, S., Shi, G., Koong, A., and Le, Q.T., "Sample classification from protein mass spectrometry, by 'peak probability contrasts'," *Bioinformatics*, **20**(17):3034–44, 2004.

Yasui, Y., McLerran, D., Adam, B.L., Winget, M., Thornquist, M., and Feng, Z., "An automated peak identification/calibration procedure for high-dimensional protein measures from mass spectrometers," *Journal of Biomedicine and Biotechnology*, **2003**(4):242–8, 2003.

Yasui, Y., Pepe, M., Thompson, M.L., Adam, B.L., Wright, Jr., G.L., Qu, Y., Potter, J.D., Winget, M., Thornquist, M., and Feng, Z., "A data-analytic strategy for protein biomarker discovery: Profiling of high-dimensional proteomic data for cancer detection," *Biostatistics*, **4**(3):449–63, 2003.

T.W. Randolph and Y. Yasui, *Multiscale Processing of Mass Spectrometry Data*, *Biometrics*, **62**:589–97, 2006.

See Also

`msPeak`, `msQuantify`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## extract several spectra from the build-in
## dataset
z <- qcset[, 1:8]

## denoising
z <- msDenoise(z, FUN="wavelet", n.level=10, thresh.scale=2)
```

```
## local noise estimation
z <- msNoise(z, FUN="mean")

## baseline subtraction
z <- msDetrend(z, FUN="monotone", attach=TRUE)

## intensity normalization
z <- msNormalize(z)

## peak detection
z <- msPeak(z, FUN="simple", use.mean=FALSE, snr=2)

## peak alignment
z <- msAlign(z, FUN="cluster", snr.thresh=10,
    mz.precision=0.004)

## extract the peak.class
z[["peak.class"]]

## visualize the alignment
plot(z, process="msAlign", subset=1:8, offset=100,
    xlim=c(13000, 17000), lty=c(1,4))
```

`msAssign` *Utility functions for maintaining pipeline processing histories*

Description

For mutual compatibility in S-PLUS and R, these functions switch on `is.R` as an interface to the `assign`, `exists`, `get`, and `remove` functions. In addition, the `msGlobalEnv` function outputs the environment in R and frame in S-PLUS associated with the global work environment and session database, respectively. The `msNewEnv` returns the integer 1 in S-PLUS (for frame 1, the expression frame) or a new environment in R (ala `new.env`).

Usage

```
msAssign(x, value, envir)
msExists(x, envir)
msGet(x, envir)
msGlobalEnv()
msNewEnv()
msRemove(x, envir)
```

Arguments

`x`	a character string denoting the name of the object.
`value`	any S-PLUS or R object; the value to be assigned the name in x. Only used in the `msAssign` function.
`envir`	the `frame` in S-PLUS (or `environment` in R) designated for the processing and storage of pipeline history data. Default: `msProcessEnv`, a global environment impicitly set by a previous call to `throwEvent`. In general, the user should rely on the specified default value.

Value

the standard output of the related functions as described in the *Description* section.

See Also

`assign`, `exists`, `get`, `remove`.

Examples

```
envir <- msGlobalEnv()
x <- "myval"

msAssign(x, 1:10, envir)
msExists(x, envir)
msGet(x, envir)
msRemove(x, envir)
msExists(x, envir)
```

`msCalibrate` *Constructor Function for Objects of Class msCalibrate*

Description

Computes the parameters of the quadratic equation used by Ciphergen mass spectrometers to convert time-of-flight to m/z values.

Usage

```
msCalibrate(mz, tof, u=20000, FUN="lm", digits=4, predict.mz=TRUE)
```

Arguments

`mz`	A vector of mass-to-charge (`m/z`) values (in daltons) of the calibrants used. This input may also be a `list` or named vector whose objects have (at the very least) the names `"u"`, `"t0"`, `"a"`, and `"b"`, each containing a numeric scalar coresponding to the conversion coefficients. In this case, the `predict` method is called to return the predicted `m/z` values for the `tof` input.
`tof`	A numeric vector of corresponding time-of-flight measures (in nanoseconds) of the calibrants used.
`FUN`	A character string specifying the method for quadratic fitting. Possible choices are `"lm"`, `"lmRobMM"`, `"ltsreg"`, `"lmsreg"`, `"l1fit"`, `"rreg"`. Default: `"lm"`.
`digits`	The maximum precision to use in calculating the `m/z` values. Default: 4.
`predict.mz`	A logical. If `TRUE`, predicts the mz from the input tof. Default: `TRUE`.
`u`	A numeric value denoting the voltage (in volts) used. Default: 20000.

Details

Assuming that the mass spectrometry data was recorded by a mass spectrometer using time of flight (TOF) to register the number of ions at each mass/charge (m/z) value, mass calibration means to convert a raw TOF t to m/z. The m/z ratio is usually a direct measure of mass because the protein molecules are almost exclusively singly charged, i.e., z=+1.

Typically, the calibration process involves acquiring a spectrum from a standard sample with at least five proteins or peptides of various molecular weights, spanning the mass range of interest. A quadratic equation relating t to mz is then fit to the t values of the standard peaks in this spectrum: $\frac{m/z}{U} = a(t - t_0)^2 + b$, where U is the preset voltage. The equation (with the fitted a, b, and t_0 coefficients) is then used to convert t to m/z in mass spectra that are collected under the same instrument conditions such as laser intensity, approximate date, and focusing mass or time lag.

Value

An object of class msCalibrate.

S3 METHODS

coef Get regression coefficients.

plot Plot the predicted m/z values versus supplied tof. Optional plot parameters are as follows:

- **type** The plot type. Default: "b".
- **xlab** A character string defining the abscissa label. Default: "tof".
- **ylab** A character string defining the ordinate label. Default: "m/z".
- **add** A logical value. If TRUE, the plot is added using the current par() layout. Otherwise a new plot is produced. Default: FALSE.
- **...** Additional plot arguments, i.e., par() options.

predict Predict m/z values from from tof input. The following optional arguments are supported:

- **newtof** New TOF values to predict over. If missing, the original TOF values will be used. Default: NULL (missing).
- **digits** The maximum precision to use in calculating the m/z values. Default: 4.

print Print the results.

See Also

lm, lmRobMM, ltsreg, lmsreg, l1fit, rreg, msCalibrate.

Examples

```
## set up parameters
u  <- 20000
t0 <- 0.0038
a  <- 0.0002721697
b  <- 0.0

## simulate m/z and time-of-flight
tof <- seq(from=20, to=60, length=7)
mz  <- u*(a*(tof - t0)^2 + b)

## perform quadratic fitting
fit <- msCalibrate(mz=mz, tof=tof, u=u, FUN="lm")
```

```
## check the fitted parameters
print(fit)

## do prediction: convert tof to mass
tof2 <- seq(from=min(tof), to=max(tof), length=60)
mz.predicted <- predict(fit, tof2)

## visualization
plot(fit, type="p", col=1, xlim=range(tof2),
    ylim=range(mz.predicted))
lines(tof2, mz.predicted, col=2)
legend(x=20, y=20000, col=1:2, pch="o ", lty=c(0,1),
    legend=c("true", "predicted"))
```

`msCharge` *Charge Detection*

Description

Find proteins that possibly have multiple charges. This is achieved by detecing proteins whose mz values are nearly exact multiples of others and hence potentially represent the same protein.

Usage

```
msCharge(x, ncharge=2:3, mz.precision=0.003,
    event="Charge Detection", ...)
```

Arguments

`x`	An object of class `msSet` with an existing element `"peak.class"`.
`...`	Not used.
`event`	A character string denoting the name of the event to register with the (embedded) event history object of the input after processing the input data. Default: `"Charge Detection"`.
`mz.precision`	A numeric value, used to construct the threshold when comparing the multiples of mz values. The default value is 0.003 because SELDI data is often assumed to have $\pm 0.3\%$ mass drift, i.e., a peak at mass `w` could represent a protein with a mass within the interval $[w(1-0.003), w(1+0.003)]$.
`ncharge`	A numeric integer vector denoting the multiple charges of interest. All of its elements must be larger than 1.

Value

An object of class `msSet` with charge estimate attached as element `"peak.charge"`.

See Also

`msSet`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## extract several spectra from the build-in
## dataset
z <- qcset[, 1:8]

## denoising
z <- msDenoise(z, FUN="wavelet", n.level=10, thresh.scale=2)

## local noise estimation
z <- msNoise(z, FUN="mean")

## baseline subtraction
z <- msDetrend(z, FUN="monotone", attach=TRUE)

## intensity normalization
z <- msNormalize(z)

## peak detection
z <- msPeak(z, FUN="simple", use.mean=FALSE, snr=2)

## peak alignment
z <- msAlign(z, FUN="cluster", snr.thresh=10,
    mz.precision=0.004)

## charge detection
z <- msCharge(z, ncharge=2:5, mz.precision=0.003)

## extract the peak.charge
z[["peak.charge"]]
```

---

msDenoise *Mother Function for Mass Spectra Denoising*

Description

Denoise spectra with various functions.

Usage

```
msDenoise(x, FUN="wavelet", MARGIN=2, type="intensity",
    attach.noise=TRUE, event="Denoising", ...)
```

Arguments

x	An object of class msSet.
...	Additional arguments to FUN. They are passed unchanged to each call of FUN and include their names. See the help documentation of the specified FUN for details.

FUN	Either an object of class "character" or of class "function". **character:** A character string denoting the method to use in denoising the data. Supported choices are "wavelet" for wavelet shrinkage, "mrd" for partial summation of a wavelet-based multiresolution decomposition, and "smooth" for robust running medians. Default: "wavelet". **function:** A user-defined function with an argument list of the form (x, ...) where x is a required argument corresponding to a numeric vector (typically these values will be the intensity values of a mass spectrum). In either case, the additional arguments ... will be passed directly to the specified FUN.
MARGIN	The subscripts over which the function is to be applied to the data defined by type. For example, if type="intensity", the data is defined by the intensity matrix of x and MARGIN=1 indicates rows while MARGIN=2 indicates columns. Default: 2 (FUN will operate over each column of the data).
attach.noise	A logical indicating if the noise removed should be attached or not. Default: TRUE.
event	A character string denoting the name of the event to register with the (embedded) event history object of the input after processing the input data. Default: "Denoising".
type	A character string defining the type of data matrix to extract from x and operate over. The operative data will be x[[type]] if it exists or, if it does not, attr(x, type) will be used. If neither exists, an error is returned. Default: "intensity".

Value

An object of class msSet, optionally, with the estimated noise attached as element "noise".

Note

If FUN="mrd", an mrd object containing meta information regarding the multiresolution decomposition is attached to the msSet output object for subsequent use by other MRD-based function calls such as msPeak(x, FUN="mrd", ...

See Also

msDenoiseSmooth, msDenoiseWavelet, msDenoiseMRD, matchObject.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## denoise a spectrum portion via waveshrink and a
## smoothing function
mz   <- (qcset$mz > 3000 & qcset$mz < 5000)
data <- qcset[mz, 1, drop=FALSE]

## add a little Gaussian noise for illustration
noise  <- rnorm(length(data$intensity), sd=stdev(data$intensity)/3)
xnoise <- data
xnoise$intensity <- data$intensity +
    matrix(noise, ncol=1)
```

```
## denoise using the supported routines
z1 <- msDenoise(xnoise, FUN="wavelet")
z2 <- msDenoise(xnoise, FUN="smooth")
z3 <- msDenoise(xnoise, FUN="mrd", levels=4:6)

## create a user-defined (albeit naive) denoising
## function
my.fun <- function(x, wavelet="d4"){
    filt <- wavDaubechies(wavelet=wavelet, norm=FALSE)$scaling
    return(filter(x, filt))
}
z4 <- msDenoise(xnoise, FUN=my.fun, wavelet="s12")

## create a stackplot of the results
z <- list(original=data$intensity[,1],
    noisy=xnoise$intensity[,1],
    waveshrink=z1$intensity[,1],
    smooth=z2$intensity[,1],
    mrd=z3$intensity[,1],
    "my function"=z4$intensity[,1])
wavStackPlot(z, col=seq(along=z), same.scale=TRUE)
```

`msDenoiseMRD` *Denoising a Mass Spectrum via Partial Summation of an MRD*

Description

Forms a multiresolution decomposition (MRD) by taking a specified discrete wavelet transform of the input spectrum and subsequently inverting each level of the transform back to the "time" domain. The resulting components of the MRD form an octave-band decomposition of the original spectrum, and can be summed together to reconstruct the original spectrum. Summing only a subset of these components can be viewed as a denoising operation if the "noisy" components are excluded from the summation.

Usage

```
msDenoiseMRD(x, wavelet="s8",
    levels=1, xform="modwt", reflect=TRUE,
    keep.smooth=TRUE, keep.details=TRUE,
    process="msDenoiseMRD")
```

Arguments

`x`	A vector containing a uniformly-sampled real-valued time series.
`keep.details`	A logical value. If `TRUE`, the details corresponding to the specified levels are included in the partial summation over the MRD components. The user also has the choice to exclude the smooth in the summation via the `keep.smooth` option, but one of `keep.details` and `keep.smooth` must be `TRUE`. Default: `TRUE`.
`keep.smooth`	A logical value. If `TRUE`, the smooth at the last decomposition level is added to the partial summation over specified details. The smooth typically contains low-frequency trends present in a spectrum, so removing the smooth (`keep.smooth=FALSE`)

will result in removing the trend in such cases. The user also has the choice to exclude the details in the summation via the `keep.details` option, but one of `keep.details` and `keep.smooth` must be `TRUE`. Default: `TRUE`.

`levels` An integer vector of integers denoting the MRD detail(s) to sum over in forming a denoised approximation to the orginal spectrum (the summation is performed across scale and nto across time). All values must be positive integers, and cannot exceed `floor(logb(length(x),2))` if `reflect=FALSE` and, if `reflect=TRUE`, cannot exceed `floor(logb((length(x)-1)/(L-1) + 1, b=2))` where L is the length of the wavelet filter. Use the `keep.smooth` option to also include the last level's smooth in the summation. Default: 1.

`process` A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. This process is not updated if it already exists in the event history. Default: `"msDenoiseMRD"`.

`reflect` A logical value. If `TRUE`, the last $L_J = (2^{\text{n.level}} - 1)(L-1)+1$ coefficients of the series are reflected (reversed and appended to the end of the series) in order to attenuate the adverse effect of circular filter operations on wavelet transform coefficients for series whose endpoint levels are (highly) mismatched. The variable L_J represents the effective filter length at decomposition level `n.level`, where L is the length of the wavelet (or scaling) filter. A similar operation is performed at the beginning of the series. After synthesis and (partial) summation of the resulting details and smooth, the middle N points of the result are returned, where N is the length of the original time series. Default: `TRUE`.

`wavelet` A character string denoting the filter type. See `wavDaubechies` for details. Default: "s8".

`xform` A character string denoting the wavelet transform type. Choices are `"dwt"` and `"modwt"` for the discrete wavelet transform (DWT) and maximal overlap DWT (MODWT), respectively. The DWT is a decimated transform where (at each level) the number of transform coefficients is halved. Given N is the length of the original time series, the total number of DWT transform coefficients is N. The MODWT is a non-decimated transform where the number of coefficients at each level is N and the total number of transform coefficients is N*`n.level`. Unlike the DWT, the MODWT is shift-invariant and is seen as a weighted average of all possible non-redundant shifts of the DWT. See the references for details. Default: `"modwt"`.

Details

Performs a level J decimated or undecimated discrete wavelet transform on the input series and inverts the transform at each level separately to produce details $D_1, \ldots, D_J$ and smooth S_J. The decomposition is additive such that the original series X may be reconstructed ala $X = S_J + \sum_{j=1}^{J} D_j$. As the effective wavelet filters at level j are nominally associated with approximate band pass filters, the details D_j correspond approximately to normalized frequencies on the interval $[1/2^{j+1}, 1/2^j]$, while the content of the smooth S_J corresponds approximately to normalized frequencies $[0, 1/2^{J+1}]$. The collection of details and smooth form a multiresolution decomposition (MRD).

With the intent of removing unwanted noise events, a summation over a subset of MRD components may be calculated yielding a smooth approximation to the original spectrum. For example, summing all MRD components beyond D_1 is tantamount to a low-pass filtering of the original spectrum (whether or not this is a relevant and sufficient noise removal technique is left to the discretion of the practitioner). This function allows the user to specify the decomposition levels they wish to sum

over in order to form a multiresolution approximation. The inclusion of the last level's smooth in the summation is controlled by the optional `keep.smooth` argument.

The user may also select either a decimated wavelet transform (DWT) or an undecimated wavelet transform (MODWT). However, we recommend that the user stick with the MODWT for the following reasons:

Translation invariance Unlike the DWT, the MODWT is translation invariant, meaning that a (circular) shift of the input spectrum will result in a corresponding (circular) shift of the transform coefficients.

Smoothness The MODWT coefficients are a result of *cycle-spinning*, where averages are taken over all unique DWTs resulting from various circular shifts of the original spectrum. The resulting MODWT MRD is relatively more smooth than the corresponding DWT MRD.

Zero phase aligment Unlike the DWT MRD, the MODWT MRD produces components that are associated with **exactly** zero phase filter operations such that events (such as peaks) in the details and smooth line up exactly with those of the original spectrum in TOF (or m/z).

Computational speed The DWT is faster than the MODWT, but the MODWT is still quite fast, requiring multiplication and summation operations on the same order as the popular Fast Fourier Transform.

Value

A vector containing the denoised series.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

T.W. Randolph and Y. Yasui, *Multiscale Processing of Mass Spectrometry Data*, *Biometrics*, **62**:589–97, 2006.

T.W. Randolph, *Scale-based normalization of spectral data*, *Disease Biomarkers*, **2**:135–144, 2006.

See Also

`msDenoise`, `msDenoiseWaveletThreshold`, `msNoise`, `wavDaubechies`, `wavDWT`, `wavMODWT`, `wavMRD`, `msSmoothLoess`, `msSmoothSpline`, `msSmoothKsmooth`, `msSmoothSupsmu`, `msSmoothApprox`, `msDenoiseSmooth`, `eventHistory`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## obtain a subset of a mass spectrum and add some
## noise
x <- qcset$intensity[5000:7000,1]
sd.noise <- 2
set.seed(100)
xnoise <- x + rnorm(length(x), sd=sd.noise)
mz <- as.matrix(as.numeric(names(x)))

## define two different ranges of summation levels
lev1 <- 6:8
lev2 <- 4:8
```

```
## calculate MODWT MRDs over these levels
z1 <- msDenoiseMRD(xnoise, levels=lev1)
z2 <- msDenoiseMRD(xnoise, levels=lev2)

## plot the results
Slab <- "S8"
lab1 <- paste(paste("D", lev1, sep="", collapse="+"), Slab, sep="+")
lab2 <- paste(paste("D", lev2, sep="", collapse="+"), Slab, sep="+")

msPlot(matlines=list(
    list(x=mz, y=cbind(z1, z2), lty=1, lwd=3),
    list(x=mz, y=cbind(xnoise, xnoise), type="p", pch="o", cex=0.15)),
    yref=FALSE, xlab="m/z", ylab="MODWT MRD",
    text=list(x=rep(8300,2), y=c(2300,1650),
    labels=c(lab2,lab1), adj=0, col=2:1))
```

`msDenoiseSmooth` *Denoising Mass Spectra via Smoothing*

Description

Mass spectra are denoised via running medians.

Usage

```
msDenoiseSmooth(x, twiceit=TRUE, process="msDenoiseSmooth")
```

Arguments

`x`	A numeric vector.
`process`	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. This process is not updated if it already exists in the event history. Default: `"msDenoiseSmooth"`.
`twiceit`	A logical flag. If `TRUE`, smooth performs twicing. Twicing is the process of smoothing, computing the residuals from the smooth, smoothing these, and then adding the two smoothed series together. Default: `TRUE`.

Value

A vector of the same length of `x` with noise removed.

See Also

`smooth`.

`msDenoiseWavelet` *Denoising a Mass Spectrum via Wavelet Shrinkage*

Description

Performs a decimated or undecimated discrete wavelet transform on the input series and "shrinks" (decreases the amplitude towards zero) the wavelet coefficients based on a calculated noise threshold and specified shrinkage function. The resulting shrunken set of wavelet transform coefficients is inverted in a synthesis operation, resulting in a denoised version of the original series.

Usage

```
msDenoiseWavelet(x, wavelet="s8",
    n.level=as.integer(floor(logb(length(x), 2))),
    shrink.fun="hard",
    thresh.fun="universal", thresh.scale=1,
    xform="modwt", noise.variance=NULL,
    reflect=TRUE, process="msDenoiseWavelet",
    assign.attributes=FALSE)
```

Arguments

`x`	A vector containing a uniformly-sampled real-valued time series.
`assign.attributes`	A logical value. If `TRUE`, the argument values to the function call will be attached as attributes to the output vector. Default: `FALSE`.
`n.level`	The number of decomposition levels, limited to `floor(logb(length(x),2))`. Default:`as.integer(floor(logb(length(x), 2)))`.
`noise.variance`	A numeric scalar representing (an estimate of) the additive Gaussian white noise variance. If unknown, setting this value to 0.0 (or less) will prompt the function to automatically estimate the noise variance based on the median absolute deviation (MAD) of the scale one wavelet coefficients. Default: NA (MAD estimate will be used).
`process`	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. This process is not updated if it already exists in the event history. Default: `"msDenoiseWavelet"`.
`reflect`	A logical value. If `TRUE`, the last $L_J = (2^{\text{n.level}} - 1)(L-1)+1$ coefficients of the series are reflected (reversed and appended to the end of the series) in order to attenuate the adverse effect of circular filter operations on wavelet transform coefficients for series whose endpoint levels are (highly) mismatched. The variable L_J represents the effective filter length at decomposition level `n.level`, where L is the length of the wavelet (or scaling) filter. After waveshrinking and reconstructing, the first N points of the result are returned, where N is the length of the original time series. Default: `TRUE`.
`shrink.fun`	A character string denoting the shrinkage function. Choices are `"hard"`, `"soft"`, and `"mid"`. Default: `"hard"`.
`thresh.fun`	A character string denoting the threshold function to use in calculating the waveshrink thresholds.

character string Choices are `"universal"`, `"minimax"`, and `"adaptive"`.

numeric values Either a single threshold value or a vector of values containing `n.levels` thresholds (one threshold per decomposition level).

Note: if `xform == "modwt"`, then only the `"universal"` threshold function is (currently) supported. Default: `"universal"`.

`thresh.scale` A positive valued numeric scalar which is used to amplify or attenuate the threshold values at each decomposition level. The use of this argument signifies a departure from a model driven estimate of the thresholds and can be used to tweak the levels to obtain a smoother or rougher result. Default: 1.

`wavelet` A character string denoting the filter type. See `wavDaubechies` for details. Default: "s8".

`xform` A character string denoting the wavelet transform type. Choices are `"dwt"` and `"modwt"` for the discrete wavelet transform (DWT) and maximal overlap DWT (MODWT), respectively. The DWT is a decimated transform where (at each level) the number of transform coefficients is halved. Given N is the length of the original time series, the total number of DWT transform coefficients is N. The MODWT is a non-decimated transform where the number of coefficients at each level is N and the total number of transform coefficients is N*`n.level`. Unlike the DWT, the MODWT is shift-invariant and is seen as a weighted average of all possible non-redundant shifts of the DWT. See the references for details. Default: `"modwt"`.

Details

Assume that an appropriate model for our time series is $\mathbf{X} = \mathbf{D} + \epsilon$ where $\mathbf{D}$ represents an unknown deterministic signal of interest and ϵ is some undesired stochastic noise that is independent and identically distributed and has a process mean of zero. Waveshrink seeks to eliminate the noise component ϵ of $\mathbf{X}$ in hopes of obtaining (a close approximation to) $\mathbf{D}$. The basic algorithm works as follows:

1 Calculate the DWT of X.

2 Shrink (reduce towards zero) the wavelet coefficients based on a selected thresholding scheme.

3 Invert the DWT.

This function support different shrinkage methods and threshold estimation schemes. Let W represent an arbitrary DWT coefficient and $W^{(\mathrm{t})}$ the correpsonding thresholded coefficient using a threshold of δ. The supported shrinkage methods are

hard thresholding

$$W^{(\mathrm{t})} = \left\{ \begin{array}{l} 0, \& \text{if } |W| \leq \delta; \backslash W, \& \text{otherwise} \end{array} \right.$$

soft thresholding

$$W^{(\mathrm{t})} = \text{sign}(W)\ f(|W| - \delta)$$

where

$$\text{sign}(W) \equiv \left\{ \begin{array}{l} +1, \& \text{if } W > 0; \backslash 0, \& \text{if } W = 0; \backslash -1, \& \text{if } W < 0. \end{array} \right.$$

and

$$f(x) \equiv \left\{ \begin{array}{l} x, \& \text{if } x \geq 0; \backslash 0, \& \text{if } x < 0. \end{array} \right.$$

mid thresholding

$$W^{(\mathrm{t})} = \mathrm{sign}(W)\ g(|W| - \delta)$$

where

$$g(|W| - \delta) \equiv \left\{ \begin{array}{l} 2f(|W| - \delta), \& \mathrm{if}\ |W| < 2\delta; \backslash |W|, \& \mathrm{otherwise}. \end{array} \right.$$

Hard thresholding reduces to zero all coefficients that do not exceed the threshold. *Soft* thresholding pushes toward zero any coefficient whose magnitude exceeds the threshold, and zeros the coefficient otherwise. *Mid* thresholding represents a compromise between hard and soft thresholding such that coefficients whose magnitude exceeds twice the threshold are not adjusted, those between the threshold and twice the trhreshold are shrunk, and those below the threshold are zeroed.

The supported threshold functions are

universal The universal threshold function is dependent on the type of wavelet transform used to decompose the time series.

DWT transform: $\delta = \sqrt{2\sigma_\varepsilon^2 \log(N)}$ where σ_ε^2 is the noise variance and N is the number of samples in the time series. If the optional input argument `noise.variance` is non-positive, it signifies that the additive noise variance is unknown and (in this case) the standard deviation of the noise is estimated by

$$\hat{\sigma}_{\mathrm{MAD}} \equiv \frac{\mathrm{median}\{|\mathbf{W}_{1,t}|\}}{0.6745}$$

where the $\mathbf{W}_{1,t}$ are the set of level 1 DWT wavelet coefficients. The MAD estimate is normalized by 0.6745 to ensure to return the proper result if the input series were solely comprised of Gaussian white noise.

MODWT transform: $\delta_j = \sqrt{2\sigma_\varepsilon^2 \log(N)/2^j}$ where σ_ε^2 is the noise variance, N is the number of samples in the time series, and j is the decomposition level. Note that, unlike the DWT case, the threshold levels for the MODWT are a function of decomposition level. If the optional input argument `noise.variance` is non-positive, it signifies that the additive noise variance is unknown and (in this case) the standard deviation of the noise is estimated by

$$\widetilde{\sigma}_{\mathrm{MAD}} \equiv \frac{2^{1/2}\mathrm{median}\{|\widetilde{\mathbf{W}}_{1,t}|\}}{0.6745}$$

where the $\widetilde{\mathbf{W}}_{1,t}$ are the set of level 1 MODWT wavelet coefficients. The MAD estimate is normalized by 0.6745 to ensure to return the proper result if the input series were solely comprised of Gaussian white noise.

In either case, the universal threshold is defined so that if the original time series was solely comprised of Gaussian noise, then all the wavelet coefficients would be (correctly) set to zero using a hard thresholding scheme. Inasmuch, the universal threshold results in highly smoothed output.

minimax These thresholds are used with soft and hard thresholding, and are precomputed based on a minimization of a theoretical upperbound on the asymptotic risk. The minimax thresholds are always smaller than the universal threshold for a given sample size, thus resulting in relatively less smoothing.

adaptive These are scale-adaptive thresholds, based on the minimization of Stein's Unbiased Risk Estimator for each level of the DWT. This method is only available with soft shrinkage. As a caveat, this threshold can produce poor results if the data is too sparse (see the references for details).

Finally, the user has the choice of using either a decimated (standard) form of the discrete wavelet transform (DWT) or an undecimated version of the DWT (known as the Maximal Overlap DWT

(MODWT)). Unlike the DWT, the MODWT is a (circular) shift-invariant transform so that a circular shift in the original time series produces an equivalent shift of the MODWT coefficients. In addition, the MODWT can be interpreted as a *cycle-spun* version of the DWT, which is achieved by averaging over all non-redundant DWTs of shifted versions of the original series. The result is a smoother version of the DWT at the cost of an increase in computational complexity (for an N-point series, the DWT requires $O(N)$ multiplications while the MODWT requires $O(N \log_2 N$ multiplications.

Value

A vector containing the denoised series and optionally with the argument values of the function call detached.

References

Donoho, D. and Johnstone, I. *Ideal Spatial Adaptation by Wavelet Shrinkage*. Technical report, Department of Statistics, Stanford University, 1992.

Donoho, D. and Johnstone, I. *Adapting to Unknown Smoothness via Wavelet Shrinkage*. Technical report, Department of Statistics, Stanford University, 1992.

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

msDenoise, msDenoiseWaveletThreshold, msNoise, wavDaubechies, wavDWT, wavMODWT, msSmoothLoess, msSmoothSpline, msSmoothKsmooth, msSmoothSupsmu, msSmoothApprox, msDenoiseSmooth, eventHistory.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## create plot layout
old.par <- par()
par(mfrow=c(3,2))

## grab portion of a mass spectrum and plot
x <- qcset$intensity[5000:7000,1]
plot(x,type="l");title("original")

## create noise and add it to the spectrum portion
sd.noise <- 2
xnoise <- x + rnorm(length(x), sd=sd.noise)
plot(xnoise,type="l")
title(paste("original + noise (sd=", round(sd.noise,3),")",sep=""))

## plot MODWT waveshrink results after scaling the
## estimated threshold values
for ( k in seq(0.5,2,length=4)){

    y <- msDenoiseWavelet(xnoise, wavelet="s8",
       shrink.fun="hard", thresh.fun="universal",
       thresh.scale=k, xform="modwt")
    plot(y,type="l")
    title(paste("WS: thresh.scale =", round(k,2)))
}
```

```
## DWT waveshrink using different threshold
## functions
plot(x,type="l")
title("original")
plot(xnoise,type="l")
title(paste("original + noise (sd=", round(sd.noise,3),")",sep=""))

thresh.funs <- c("universal", "minimax", "adaptive")
for (k in thresh.funs){
    plot(msDenoiseWavelet(xnoise, thresh.fun=k, xform="dwt"), type="l", ylab="y")
    title(paste("WS: thresh.fun =", k))
}

## restore original plot layout
par(old.par)
```

`msDenoiseWaveletThreshold`
Wavelet Shrinkage Threshold Test

Description

Performs waveshrink on a given spectrum over a (wide) range of thresholds. For each threshold, a waveshrunk version of the input spectrum is calculated and a separation statistic is formed based on the absolute difference between the waveshrunk output and an *infinitely smooth* reference series. The reference series is a the result of waveshrinking the input spectrum with a very large threshold, where (nearly) all the wavelet coefficients are (shrunk toward) zero in the shrinkage process. The `plot` method can be used to display an image of the separation statistics over the specified range of thresholds and corresponding m/z values. This technique eliminates adaptive thresholding, where a unique threshold is used to shrink the wavelet coefficients at different scales, since only a single threshold is supplied.

Usage

```
msDenoiseWaveletThreshold(x, wavelet="s8",
    n.level=as.integer(floor(logb(length(x), 2))),
    shrink.fun="hard", thresh.scale=NULL,
    xform="modwt", reflect=TRUE, n.threshold=500,
    thresh.fun="universal", noise.variance=NULL,
    min.thresh=NULL, max.thresh=NULL)
```

Arguments

`x`	A vector containing a uniformly-sampled real-valued time series, typically a mass spectrum.
`min.thresh, max.thresh`	Numeric scalars defining the threshold range. These arguments are only used if the default value of the `thresh.scale` argument is used, i.e., `thresh.scale=NULL`. Default: range(abs(wavelet coefficients)) using the specified transform.
`n.level`	The number of decomposition levels, limited to `floor(logb(length(x),2))`. Default: `as.integer(floor(logb(length(x), 2)))`.

`n.threshold` The number of thresholds. This argument is only used if the default value of the `thresh.scale` argument is used, i.e., `thresh.scale=NULL`. This argument must be a positive integer. Default: 500.

`noise.variance`
A numeric scalar representing (an estimate of) the additive Gaussian white noise variance. If unknown, setting this value to 0.0 (or less) will prompt the function to automatically estimate the noise variance based on the median absolute deviation (MAD) of the scale one wavelet coefficients. Default: NA (MAD estimate will be used).

`reflect` A logical value. If `TRUE`, the last $L_J = (2^{\text{n.level}} - 1)(L-1)+1$ coefficients of the series are reflected (reversed and appended to the end of the series) in order to attenuate the adverse effect of circular filter operations on wavelet transform coefficients for series whose endpoint levels are (highly) mismatched. The variable L_J represents the effective filter length at decomposition level `n.level`, where L is the length of the wavelet (or scaling) filter. After waveshrinking and reconstructing, the first N points of the result are returned, where N is the length of the original time series. Default: `TRUE`.

`shrink.fun` A character string denoting the shrinkage function. Choices are `"hard"`, `"soft"`, and `"mid"`. Default: `"hard"`.

`thresh.fun` A character string denoting the threshold function to use in calculating the waveshrink thresholds.

character string Choices are `"universal"`, `"minimax"`, and `"adaptive"`.

numeric values Either a single threshold value or a vector of values containing `n.levels` thresholds (one threshold per decomposition level).

Note: if `xform == "modwt"`, then only the `"universal"` threshold function is (currently) supported. Default: `"universal"`.

`thresh.scale` A numeric vector containing the threshold values to use in denoising the wavelet coefficients. This vector must contain at least two numeric values. Default: `seq(min(abs(wavelet coefficients)), max(abs(wavelet coefficients)), length=n.threshold)` where `wavelet coefficients` are defined by the specified transform. This range of thresholds allows the user to explore values over all the effective threshold levels.

`wavelet` A character string denoting the filter type. See `wavDaubechies` for details. Default: "s8".

`xform` A character string denoting the wavelet transform type. Choices are `"dwt"` and `"modwt"` for the discrete wavelet transform (DWT) and maximal overlap DWT (MODWT), respectively. The DWT is a decimated transform where (at each level) the number of transform coefficients is halved. Given N is the length of the original time series, the total number of DWT transform coefficients is N. The MODWT is a non-decimated transform where the number of coefficients at each level is N and the total number of transform coefficients is N*`n.level`. Unlike the DWT, the MODWT is shift-invariant and is seen as a weighted average of all possible non-redundant shifts of the DWT. See the references for details. Default: `"modwt"`.

Value

An object of class `msDenoiseWaveletThreshold`.

S3 METHODS

plot Plots an image of the separation statistics as a function of threshold (ordinate) and m/z value (abscissa). For reference, a line plot of the original and *infinitely smoothed* spectra are overlaid on the image. Available options are:

- **xlab** character string defining x-axis label. Default: `"m/z"`.
- **ylab** character string defining y-axis label. Default: `"Waveshrink Threshold"`.
- **lty** an integer denoting the line type ala the `par` function. Default: 1.
- **lwd** an integer denoting the line width ala the `par` function. Default: 2.
- **...** additional argument sent directly to the `lines` function used to overlay the image with the original and infiniteluyu smooth spectra.

print Prints a summary of the returned object. Available options are:

- **justify** text justification ala the `format` function. Default: `"left"`.
- **sep** header separator. Default: `":"`.

References

Donoho, D. and Johnstone, I. *Ideal Spatial Adaptation by Wavelet Shrinkage*. Technical report, Department of Statistics, Stanford University, 1992.

Donoho, D. and Johnstone, I. *Adapting to Unknown Smoothness via Wavelet Shrinkage*. Technical report, Department of Statistics, Stanford University, 1992.

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

http://bioinformatics.mdanderson.org/sizer.html.

See Also

`msDenoiseWavelet`, `msDenoise`, `rescale`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## grab portion of a mass spectrum and plot
x <- qcset$intensity[5000:7000,1]

## create noise and add it to the spectrum portion
sd.noise <- 2
set.seed(100)
xnoise <- x + rnorm(length(x), sd=sd.noise)

## calculate the waveshrink separation statistics
z <- msDenoiseWaveletThreshold(xnoise)
print(z)
plot(z)
```

`msDetrend` *Baseline Correction*

Description

Estimate and subsequently subtract the baselines from mass spectra. The basic technique for baseline estimation is to fit a curve locally to the intensity minima.

Usage

```
msDetrend(x, FUN="loess",
    type="intensity", pre=NULL, MARGIN=2, attach.base=TRUE,
    event="Baseline Correction", ...)
```

Arguments

`x`	An object of class `msSet`.
`...`	Additional arguments for the `FUN` specified. See the specific underlying function for details.
`FUN`	Either an object of class `"character"` or of class `"function"`. **character:** A character string denoting the method to use in smoothing the data to estimate the baseline. Supported choices are **`"loess"`** uses the function `loess.smooth` to fit a local regression model to the local minima. **`"spline"`** uses the function `spline` to interpolate through the local minima by means of a cubic spline. **`"supsmu"`** uses the function `supsmu` to fit a smooth curve to the local minima. **`"approx"`** uses the function `approx` to linearly interpolate the local minima. **`"monotone"`** uses the function `cummin` to fit a non-increasing curve to all the intensity values. **`"mrd"`** uses the function `msSmoothMRD` to extract wavelet-based multiresolution decomposition components relevant to baseline trends. Default: `"loess"`. **function:** A user-defined function with an argument list of the form `(x, ...)` where `x` is a required argument corresponding to a numeric vector (typically these values will be the noise estimates of a mass spectrum). In either case, the additional arguments `...` will be passed directly to the specified routine.
`MARGIN`	The subscripts over which the function is to be applied to the data defined by `type`. For example, if `type="intensity"`, the data is defined by the intensity matrix of `x` and `MARGIN=1` indicates rows while `MARGIN=2` indicates columns. Default: 2 (FUN will operate over each column of the data).
`attach.base`	A logical value specifying if the estimated baseline needs to be attached as an element to the output, and the default is `T`. It has to be `T` if you want to visualize the baseline.

event	A character string denoting the name of the event to register with the (embedded) event history object of the input after processing the input data. Default: "Baseline Correction".
pre	A function that is applied to the data defined by type prior to processing it with the function defined by FUN. Default: NULL (no function is applied a priori).
type	A character string defining the type of data matrix to extract from x and operate over. The operative data will be x[[type]] if it exists. Default: "intensity".

Value

An object of class msSet, optionally, with the estimated baseline attached as element "baseline".

Note

If FUN="mrd", an mrd object containing meta information regarding the multiresolution decomposition is attached to the msSet output object for subsequent use by other MRD-based function calls such as msPeak(x, FUN="mrd", ...).

See Also

msSmoothLoess, msSmoothSpline, msSmoothSupsmu, msSmoothApprox, msSmoothMonotone, msSmoothMRD.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## extract several spectra from the build-in
## dataset
z <- qcset[, 1:8]

## denoising
z <- msDenoise(z, FUN="wavelet", n.level=10, thresh.scale=2)

## baseline subtraction
z <- msDetrend(z, FUN="monotone", attach=TRUE)

## visualize the baseline
plot(z, process="msDetrend", subset=1:3,
    xlim=c(5000, 8500), lty=1, lwd=c(1,3))
```

---

msExtrema *Find both Local Maxima and Local Minima*

Description

Finds both local extrema in a vector, time series, or in each column of a matrix.

Usage

```
msExtrema(x, span=3)
```

Arguments

x	a vector, time series, or matrix. If x is a matrix, msExtrema finds both local maxima and local minima in each column of x.
span	a numeric value indicating the span of the local extreme. A local maximum (minimum) is defined as an element in a sequence which is greater (smaller) than all other elements within a window of width span centered at that element. The default value is 3, meaning that a maximum (minimum) is bigger (smaller) than both of its neighbors.

Value

a list with two elements:

index.max	an object like x of logical values. Values that are TRUE correspond to local maxima in the data.
index.min	an object like x of logical values. Values that are TRUE correspond to local minima in the data.

See Also

zeroCross.

Examples

```
## create a synthetic sequence
a<-c(3,3,2,1,1,2,1,2,3,3,1,1,1,3,3,3,2,2,2,1,1,1,1,2,3)

## detect local maxima and minima
maxmin <- msExtrema(a)

## visualize the result
par(mfrow=c(1,1))
plot(a, type="l")
points((1:length(a))[maxmin$index.max],
    a[maxmin$index.max], col=2, pch=1)
points((1:length(a))[maxmin$index.min],
    a[maxmin$index.min], col=3, pch=2)
if (!is.R()){
    legend(x=18, y=3, col=2:3, marks=1:2, legend=c("maxima", "minima"))
} else {
    legend(x=18, y=3, col=2:3, pch=1:2, legend=c("maxima", "minima"))
}
```

---

msHelp *Open msProcess Help Files in S-PLUS*

Description

Used to quickly open help files related to the msProcess package.

Usage

```
msHelp(keyword="", section="msProcess")
```

Arguments

keyword	A character string specifying a particular S+Proteome function or object. Default: "".
section	A character string giving the name of a module or a library associated with the S+Proteome module. Default: "proteome".

See Also

help.

Examples

```
## open the S+Proteome help file
msHelp()
```

---

msImport *Mass Spectra Data Import*

Description

Imports data from (multiple) mass spectrum file(s) and compiles them into an object of msList, i.e., a list of two-column matrices.

Usage

```
msImport(path, label="unclassified", type="ASCII",
    pattern="", ...)
```

Arguments

path	A single character string or a vector of character strings. Each character string can either be a path to the directory containing the mass spectra files (i.e., directory path) or a path to a mass spectra file (i.e., file path). In the case of a vector of character strings, the paths should be either all directory paths or all file paths.
...	Additional arguments. See importData for details.
label	A single character string or a vector of character strings defining the classification label(s) for the spectra to be imported. The length of label must be equal to the number of elements in the path argument. The labels will be used in mass spectra classification but not in processing. Default: "unclassified".
pattern	A character string denoting the pattern to use in filtering the list of files in the path directory. Default: "" (all files in path).
type	A single character string or a vector of character strings specifying the input data file type(s). The length of type must be equal to the number of elements in the path variable, otherwise the first entry is replicated accordingly. This argument may be any of the supported types in the importData function or type "CiphergenXML", which represents a Ciphergen XML mass spectrometry data file. Default: "ASCII".

Details

Each data file has two columns defined by `m/z` and `intensity` values. This function also checks if the following conditions hold:

1 the m/z values are distinct;

2 the m/z values are positive;

3 the m/z values are the same across spectra;

4 the lengths of each spectra are the same.

and issues warning messages if not.

Value

An object of `msList`, which is a list of two-column matrices, one matrix for each imported file. The first column is named `"mz"` and contains m/z values. The second column is named `"intensity"` and contains the intensity values. The list has an attribute named `"type"`, which is a factor and contains the classification labels for the imported spectra.

See Also

`importData`, `msImportCiphergenXML`.

Examples

```
## create faux MS data files
n.file <- 10
files <- file.path(getwd(), paste("ms", seq(n.file), ".csv", sep=""))

for (i in seq(n.file)) {
    ms <- data.frame("m/z"=(1:5), intensity=(6:10)+i)
    if (is.R())
      write.table(ms, file=files[i])
    else
      write.table(ms, file=files[i], dimnames="colnames")
}

## load the mass spectra files into a list of
## two-column matrices
msImport(path=files)

## do the same except use the path and pattern
## arguments
msImport(path=getwd(), pattern=".csv" )

## remove the files
unlink(files)
```

msImportCiphergenXML *Import Mass Spectrometry Data from a Ciphergen XML File*

Description

Imports either time-of-flight (TOF) or intensity values from a mass spectrum written in Ciphergen's XML format. The corresponding m/z data are either imported directly from the file or calculated (to a specified precision) based on mass calibration factors extracted from the XML file.

Usage

```
msImportCiphergenXML(x, tof=FALSE, mz.calc=TRUE, digits=3)
```

Arguments

x	A character string defining the path to the Ciphergen XML file.
digits	An integer defining the precision of the m/z if mz.calc is TRUE. Default: 3.
mz.calc	A logical value. If TRUE, the m/z values are calculated based on the mass calibration parameters given in the file. If any of the required parameters are missing in the file, then the pre-calculated m/z values are returned instead (if they do not exist, then an error is returned). The precision of the m/z values is controlled through the digits argument. Default: TRUE.
tof	A logical value. If TRUE, the TOF data are returned. Otherwise, the intensity values (processed TOF data) are returned. In the case that the tof is FALSE but the intensity values do not exist in the file, then the TOF data are returned instead. Default: FALSE.

See Also

msImport.

Examples

```
## create a faux Ciphergen file with basic MS data
## information
xmlFileName <- file.path(getwd(), "ciphergen_example.xml")
cat(paste(
"<spectrum>",
   "<processingParameters>",
      "<massCalibration>",
         "<massCalibrationA>264659356.3912175</massCalibrationA>",
         "<massCalibrationB>0.00055173104994636O4</massCalibrationB>",
         "<massCalibrationT0>2.053450999111159e-007</massCalibrationT0>",
      "</massCalibration>",
   "</processingParameters>",
   "<tofData>",
      "<tofDataNumSamples>5</tofDataNumSamples>",
      "<tofDataTimeZero>0</tofDataTimeZero>",
      "<tofDataSamples>2172 2163 2114 2061 2107</tofDataSamples>",
   "</tofData>",
   "<acquisitionInfo>",
```

```
      "<setting>",
         "<ionSourceVoltage>20000</ionSourceVoltage>",
         "<digitizerRate>2.5e+008</digitizerRate>",
      "</setting>",
   "</acquisitionInfo>",
   "<processedData>",
      "<processedDataSamples>",
         "10.811,4.4363 10.820,4.4179 10.828,4.3178",
         "10.837,4.2096 10.845,4.3035",
      "</processedDataSamples>",
   "</processedData>",
"</spectrum>",
sep="\n"), file=xmlFileName)

## read in the data, comparing calculated and
## preset m/z values
msImportCiphergenXML(xmlFileName, mz.calc=TRUE, digits=3)$mz
msImportCiphergenXML(xmlFileName, mz.calc=FALSE)$mz

## read in TOF and then the intensity data
msImportCiphergenXML(xmlFileName, tof=TRUE, digits=3)$tof
msImportCiphergenXML(xmlFileName, tof=FALSE)$intensity
```

`msLaunchExample` *Open msProcess Example/Demo Files in S-PLUS*

Description

Used to quickly open help files related to the msProcess package.

Usage

```
msLaunchExample(x, open=TRUE, run=TRUE, type="R-ex")
```

Arguments

`x`	A character string specifying the name of a particular example or demo script.
`open`	A logical indicating open the script or not. Default: `TRUE`.
`run`	A logical indicating run the script or not. Default: `TRUE`.
`type`	A character string giving the name of a subdirectory where the example or demo script is located. Default: `"R-ex"`.

See Also

`example`.

Examples

```
## Not run:
if (!is.R() && is.ui.app("s+gui")) {
## open a msProcess example file
msLaunchExample("msDenoise", open=TRUE, run=FALSE, type="R-ex")
}
## End(Not run)
```

msList *S3 Class Representing a List of Spectra with Possibly Different m/z Values*

Description

An msList object is a list of matrices with each matrix representing a spectrum. Each matrix in the list has two columns. The first column is named "mz" and contains the m/z values. The second column is named "intensity" and contains the intensity values. The list has also an attribute named "type", which is of type factor and contains the classification labels for the spectra. An object of this class is usually generated from the function msImport() in package proteome.

S3 METHODS

[extract or replace parts of an msList object.

Usage: x[i]

x an msList object.

i a subscript expression used to identify the spectra to extract or replace.

plot plot a single spectrum from an msList object.

Usage: plot(x, index=1, type="l", add=FALSE, ...)

x an msList object.

index a single numeric value or character string specifying the spectrum to be plotted. The default is 1.

type a single character specifying the type of plot. see fucntion par for details. The default is "l"

add A logical value. If TRUE, the plot is added using the current par() layout. Otherwise a new plot is produced. Default: FALSE.

... other graphical parameters passed to the plot function.

print prints an msList object.

Usage: print(x, justify="left", sep=":", ...) or x

x an msList object.

justify a character string giving the justification of the numbers relative to each other. The choices are "none", "left", "right" and "decimal". Only the first letter needs to be given.

sep a character string to be inserted between text and values. The default is a colon.

summary provides a synopsis of an msList object.

Usage: summary(x)

x an msList object.

See Also

msSet.

Examples

```
if (!exists("qclist")) data("qclist", package="msProcess")

## print an this-is-escaped-code{ object
qclist

## print the synopsis of an this-is-escaped-codenormal-bracket77bracket-normal object
summary(qclist)

## plot the first spectrum from an this-is-escaped-codenormal-bracket78bracket-normal
## object
plot(qclist, index=1)
```

---

msNoise *Local Noise Estimation*

Description

Estimates the local noise level by applying a specified smoother function to the the noise data attached to the primary input variable.

Usage

```
msNoise(x, FUN="spline", MARGIN=2, type="noise",
    pre=abs, detach.noise=FALSE,
    event="Local Noise Estimation", ...)
```

Arguments

x	An object of class msSet with an existing element "noise". The noise element should be a matrix with the same dimensions as the intensity data, such as that returned by the msDenoise function.
...	Additional arguments to FUN. They are passed unchanged to each call of FUN and include their names. See the help documentation of the specified routine for details.
FUN	Either an object of class "character" or of class "function". **character:** A character string denoting the method to use in smoothing the noise data. Supported choices are "spline", "supsmu", "ksmooth", "loess.smooth", and "mean". Default: "spline". **function:** A user-defined function with an argument list of the form (x, ...) where x is a required argument corresponding to a numeric vector (typically these values will be the noise estimates of a mass spectrum). In either case, the additional arguments ... will be passed directly to the specified routine.
MARGIN	The subscripts over which the function is to be applied to the data defined by type. For example, if type="intensity", the data is defined by the intensity matrix of x and MARGIN=1 indicates rows while MARGIN=2 indicates columns. Default: 2 (FUN will operate over each column of the data).
detach.noise	A logical indicating if the noise removed previously should be detached or not. Default: FALSE.

event	A character string denoting the name of the event to register with the (embedded) event history object of the input after processing the input data. Default: "Local Noise Estimation".
pre	A function that is applied to the data defined by type prior to processing it with the function defined by FUN. Default: abs.
type	A character string defining the type of data matrix to extract from x and operate over. The operative data will be x[[type]] if it exists. Default: "noise".

Value

An object of class msSet with local noise estimate attached as element "noise.local" and optionally with the element "noise" detached.

See Also

msDenoise, msSet, properCase.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## denoise a noise contaminated spectrum portion
## via waveshrink
mz     <- (qcset$mz > 3000 & qcset$mz < 5000)
data   <- qcset[mz, 1, drop=FALSE]
noise  <- rnorm(length(data$intensity), sd=stdev(data$intensity)/3)
xnoise <- data

xnoise$intensity <- data$intensity + matrix(noise, ncol=1)
z <- vector("list", length=4)
z[[1]] <- msDenoise(xnoise, FUN="wavelet")

## smooth the resulting noise estimates to form a
## localized estimate of the noise using
## various supported methods

z[[2]] <- msNoise(z[[1]], FUN="spline")
z[[3]] <- msNoise(z[[1]], FUN="loess")

## create a user-defined smoothing function
my.fun <- function(x, wavelet="d4"){
    filt <- wavDaubechies(wavelet=wavelet, norm=FALSE)$scaling
    return(filter(x, filt))
}
z[[4]] <- msNoise(z[[1]], FUN=my.fun, wavelet="s12")

## create a stackplot of the results
type <- c("noise", rep("noise.local", 3))
for (i in 1:4){
    z[[i]] <- as.vector(z[[i]][[type[i]]])
}
names(z) <- c("noise","spline", "loess", "my function")
wavStackPlot(z, col=seq(along=z), same.scale=TRUE)
```

msNormalize *Mother Function for Intensity Normalization*

Description

Normalizes the intensity matrix of an msSet object.

Usage

```
msNormalize(x, FUN="tic", MARGIN=2,
    event="Intensity Normalization", ...)
```

Arguments

x	An object of class msSet.
...	Additional arguments to FUN. They are passed unchanged to each call of FUN and include their names. See the help documentation of the specified FUN for details.
FUN	Either an object of class "character" or of class "function". **character:** A character string denoting the method to use in denoising the data. Supported choices are "snv" for standard normal variate transformation or "tic" for normalization based on the median total ion current estimate for all spectra. Default: "tic". **function:** A user-defined function with an argument list of the form (x, ...) where x is a required argument corresponding to a numeric vector (typically these values will be the intensity values of a mass spectrum). In either case, the additional arguments ... will be passed directly to the specified FUN.
MARGIN	The subscripts over which the function is to be applied to the intensity matrix. Default: 2 (FUN will operate over each column of the intensity matrix).
event	A character string denoting the name of the event to register with the (embedded) event history object of the input after processing the input data. Default: "Intensity Normalization".

Value

An msSet object with the intensity matrix object replaced by its normalized form.

See Also

msNormalizeTIC, msNormalizeSNV, msSet.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## normalize a subset of spectra in the
## qcset object using total ion current.
data <- qcset[,1:8]
zion <- msNormalize(data, FUN="tic")
```

```
plot(zion, process="msNormalize", subset=1:8,
xlim=c(13000, 17000), lty=c(1,4), lwd=1:2)

## normalize a subset of spectra in the
## qcset object using a standard normal
## variate transformation.
zsnv <- msNormalize(data, FUN="snv")
plot(zsnv, process="msNormalize", subset=1:8,
xlim=c(13000, 17000), lty=c(1,4), lwd=1:2)

## perform a multiresolution decomposition of each
## spectrum in the intensity matrix, sum over
## levels 6-8, then normalize using the SNV
## transformation (this process is equivalent
## to scale-based normalization (SBN)).
data <- msDenoise(data, FUN="mrd", levels=6:8, keep.smooth=FALSE)
zsnv <- msNormalize(data, FUN="snv")
plot(zsnv, process="msNormalize", subset=1:8,
xlim=c(13000, 17000), lty=c(1,4), lwd=1:2)
```

`msNormalizeSNV` *Standard Normal Variate Intensity Normalization*

Description

Normalizes the input spectrum via the Standard Normal Variate (SNV) transformation defined as $X_t \equiv (X_t - \bar{X})/\sqrt{\text{var}(X)}$, where X is the spectrum.

Usage

```
msNormalizeSNV(x, process="msNormalizeSNV")
```

Arguments

`x`	A vector containing a uniformly-sampled real-valued time series.
`process`	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. This process is not updated if it already exists in the event history. Default: `"msNormalizeSNV"`.

Details

This function can also produce scale-based normalization transformations if the input is a (partial) sum over multiresolution decomposition (MRD) components formed by taking a discrete wavelet transform of the input spectrum and subsequently inverting each level of the transform back to the "time" domain. The resulting components of the MRD form an octave-band decomposition of the original spectrum, and can be summed together to reconstruct the original spectrum. Summing only a subset of these components can be viewed as a denoising operation if the "noisy" components are excluded from the summation. The result is then normalized by the standard deviation of the sum of the details. As this function merely calls the `msDenoiseMRD` function with the argument `normalize=TRUE`, see that function for more details.

Value

A vector containing the scale-based normalization of the input spectrum.

References

I.S. Helland, T. Naes and T. Isaksson, Related versions of the multiplicative scatter correction method for preprocessing spectroscopic data, *Chemometrics and Intelligent Laboratory Systems*, **29**:233–241, 1995.

T.W. Randolph, Scale-based normalization of spectral data, *Cancer Biomarkers*, **2**:135–144, 2006.

T.W. Randolph and Y. Yasui, *Multiscale Processing of Mass Spectrometry Data*, *Biometrics*, **62**:589–97, 2006.

See Also

msNormalize, msNormalizeTIC, msDenoiseWavelet, wavDaubechies, wavDWT, wavMODWT, wavMRD, eventHistory.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## obtain a subset of a mass spectrum and add some
## noise
x <- qcset[5000:7000,1]
sd.noise <- 2
set.seed(100)
x$intensity <- x$intensity + rnorm(length(x), sd=sd.noise)
mz <- x$mz

## sum over specified MODWT MRD details and
## normalize
y <- msDenoise(x, FUN="mrd", levels=6:8, keep.smooth=FALSE)
z <- msNormalizeSNV(y)

## plot the results
old.plt <- par("plt")
par(plt=c(0.08,1,0.5,0.95))
plot(mz, x$intensity, type="l", xaxt="n", xlab="", ylab="xnoise")
par(plt=c(0.08,1,0.12,0.5), new=TRUE)
plot(mz, z$intensity, type="l", xlab="m/z", ylab="Normalized (D6+D7+D8)")
par(plt=old.plt)
```

---

msNormalizeTIC *Intensity Normalization Using Total Ion Current*

Description

Normalizes a set of spectra using the total ion current (TIC). Each TIC is calculated as the sum of the intensities. The intensity of a spectrum is divided by its TIC and then multiplied by the median TIC of the set of spectra.

Usage

```
msNormalizeTIC(x, process="msNormalizeTIC")
```

Arguments

x	An object of class msSet.
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. This process is not updated if it already exists in the event history. Default: "msNormalizeTIC".

Value

An object of class msSet such that the normalized spectra have the same AUC (the median of the AUCs of spectra). The TIC's are attached as element "tic".

References

E.T. Fung and C. Enderwick, ProteinChip clinical proteomics: computational challenges and solutions, *Biotechniques*, **Supplement 32**:S34-S41, 2002.

See Also

msNormalize, msNormalizeSNV, colSums.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## extract several spectra from the build-in
## dataset
z <- qcset[, 1:8]

## denoising
z <- msDenoise(z, FUN="wavelet", n.level=10, thresh.scale=2)

## baseline subtraction
z <- msDetrend(z, FUN="monotone", attach=TRUE)

## intensity normalization
z <- msNormalizeTIC(z)

## visualize the normalization
plot(z, process="msNormalize", subset=1:8,
    xlim=c(13000, 17000), lty=c(1,4), lwd=1:2)
```

---

msPeak *Mother Function for Peak Detection*

Description

This function detects peaks in a set of mass spectra.

Usage

```
msPeak(x, FUN="simple", MARGIN=2, type="intensity",
    use.mean=FALSE, event="Peak Detection", ...)
```

Arguments

x	An object of class msSet.
...	Additional arguments for the FUN specified. See the specific routine for details.
FUN	A character string specifying the method for peak detection. Possible choices are "simple", "search", "cwt" and "mrd". In the MRD case, the input msSet object is expected to contain an attached mrd object containing meta information regarding the wavelet-based multiresolution decomposition as output by msDenoise(x, FUN="mrd", ...) as an example. See the msPeakMRD function for more details. Default: "simple".
MARGIN	The subscripts over which the function is to be applied to the data defined by type. Default: 2 (FUN will operate over each column of the data).
event	A character string denoting the name of the event to register with the (embedded) event history object of the input after processing the input data. Default: "Peak Detection".
type	A character string defining the type of data matrix to extract from x and operate over. The operative data will be x[[type]] if it exists. Default: "intensity".
use.mean	A logical value specifying if to detect peaks in the mean spectum. Default: FALSE.

Value

An object of class msSet with elements depending on the value of use.mean:

this-is-escaped-codenormal-bracket39bracket-normal
: the mean spectrum is attached as element "intensity.mean" (along with "noise.mean" and "noise.local.mean") the peak info is attached as element "peak.class", and the argument "use.mean" is attached as element "use.mean". The element "peak.class" is a matrix with peak classes as rows and some summary statistics as columns. These statistics include the location, left bound, right bound, and span of each peak class in both clock tick ("tick.loc", "tick.left", "tick.right", "tick.span") and mass measure ("mass.loc", "mass.left", "mass.right", "mass.span").

this-is-escaped-codenormal-bracket57bracket-normal
: the peak info is attached as element "peak.list". The element "peak.list" is a list with one element for each spectrum. Each element is a data.frame with 10 columns: the location, left bound, right bound, and span of each peak in both clock tick ("tick.loc", "tick.left", "tick.right", "tick.span") and mass measure ("mass.loc", "mass.left", "mass.right", "mass.span"), and also peak signal-to-noise ratio and intensity ("snr", "intensity").

See Also

msPeakSimple, msPeakSearch, msPeakMRD, msNormalize.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## extract several spectra from the build-in
## dataset
z <- qcset[, 1:8]

## denoising
z <- msDenoise(z, FUN="wavelet", n.level=10, thresh.scale=2)

## local noise estimation
z <- msNoise(z, FUN="mean")

## baseline subtraction
z <- msDetrend(z, FUN="monotone", attach=TRUE)

## intensity normalization based on total ion
## current
z <- msNormalize(z, FUN="tic")

## peak detection
z <- msPeak(z, FUN="simple", use.mean=FALSE, snr=2)

## visualize the detected peaks
plot(z, process="msPeak", subset=1:8, offset=100,
    xlim=c(13000, 17000))

## perform a similar analysis using a
## multiresolution decomposition approach
z <- qcset[, 1:8]
z <- msDenoise(z, FUN="mrd", levels=6, keep.smooth=FALSE)
z <- msPeak(z, FUN="mrd")
plot(z, process="msPeak", subset=1:8, offset=100,
    xlim=c(13000, 17000))

## perform a similar analysis using a CWT approach
# z <- qcset[, 1:8]
# z <- msPeak(z, FUN="cwt", scale.min=8)
# plot(z, process="msPeak", subset=1:8, offset=100,
#           xlim=c(13000, 17000))
```

`msPeakCWT` *Peak Detection via the Continuous Wavelet Transform*

Description

This function isolates peaks of interest in the input mass spectrum by way of a continuous wavelet transform (CWT). The method is scale-selective, i.e., the user can specify the scale range of interest in defining peaks. The basic algorithm works as follows:

CWT a continuous wavelet transform of the input spectrum `y` is calculated over a broad range of scales.

CWT Tree a CWT tree is formed by dividing the set of local maxima in the CWT time-scale plane into *branches*. Each branch contains a list of maxima whose neighbors are close in time as scales are traversed from coarse to fine. The `m/z` value at the smallest scale identifies the peak location in the original input spectrum.

Pruning The collection of branches (and corresponding peak locations) is pruned in numerous ways (see the `wavCWTPeaks` function for details.)

In addition to peak identification, this function also calculates an estimate of the Holder exponent associated with each peak. Qualitatively speaking, the magnitude (of the modulus) of the Holder exponent is proportional to the the sharpness of the corresponding peak. See the `holderSpectrum` function for more information. The Holder exponents are packed into the output `data.frame` along with other peak-related information.

Usage

```
msPeakCWT(x, y, n.scale = 100, snr.min = 3, scale.min = 4,
          length.min = 10, noise.span = NULL, noise.fun =
          "quantile", noise.min = NULL, n.octave.min = 1,
          tolerance = 0, holder = TRUE, process = "msPeakCWT")
```

Arguments

`x`	A numeric vector representing the `m/z` values of a spectrum.
`y`	A numeric vector representing the intensity values of the spectrum.
`length.min`	The minimum number of points along a CWT tree branch and within the specified `scale.range` needed in order for that branches peak to be considered a peak candidate. See the `wavCWTPeaks` function for more details. Default: `10`.
`n.octave.min`	A pruning factor for excluding non-persistent branches. If a branch of connected extrema does not span this number of octaves, it is excluded from the tree. See the `wavCWTTree` function for more details. Default: 1.
`n.scale`	The (maximum) number of logarithmically distributed scales over which to evaluate the CWT. For a uniformly sampled time series (`x`), the supported range of CWT scales is `deltat(x) * c(1, length(x))`. where `deltat(x)` is the sampling interval. However, a mass spectrum is generally viewed as a non-uniformly sampled time series because the difference in successive `m/z` values is non-constant due to the quadratic mapping of the TOF values to the m/z domain. For the purpose of peak detection, however, the sampling interval can be set to unity. Thus, the possible range of CWT scales is `1` to `length(x)` and the scale values are integers. See the `wavCWT` function for more details on CWT scales. Default: `100`.
`holder`	a logical value. If TRUE, the holder exponents corresponding to the peaks are also calculated.
`noise.fun`	A character string defining the function to apply to the local noise estimates in order to sumarize and quantify the local noise level into a scalar value. See the **DETAILS** section for more information. Supported values are **"quantile"** quantile(x, probs=0.95) **"sd"** sd(x) **"mad"** mad(x, center=0) where `x` is a vector of smallest-scale CWT coefficients whose time indices are near that of the branch termination time. See the `wavCWTPeaks` function for more details. Default: `"quantile"`.

noise.min — The minimum allowed estimated local noise level. See the wavCWTPeaks function for more details. Default: quantile(attr(x,"noise"), prob=0.05), where x is the input wavCWTTree object.

noise.span — The span in time surrounding each branche's temrination point to use in forming local noise estimates and (ultimately) peak SNR estimates. See the wavCWTPeaks function for more details. Default: NULL,max(0.01 * diff(range(times)), 5*sampling.interval), where times and sampling.interval are attributes of the input wavCWTTree object.

process — A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msPeakCWT".

scale.min — The minimum allowable value of the scale associated with a CWT peak. See the scale.range argument of the wavCWTPeaks function for more details. Default: 4.

snr.min — The minimum allowed peak signal-to-noise ratio. See the wavCWTPeaks function for more details. Default: 3.

tolerance — A tolerance vector used to find CWT extrema. This vector must be as long as there are scales in the CWT such that the j^{th} element defines the tolerance to use in finding modulus maxima at the j^{th} scale of the CWT. If not, the last value is replicated appropriately. See the wavCWTTree function for more details. Default: 0.

Value

A data.frame with 11 columns: peak class location, left bound, right bound and peak span in both clock tick ("tick.loc", "tick.left", "tick.right", "tick.span") and mass measure ("mass.loc", "mass.left", "mass.right", "mass.span"), and peak signal-to-noise ratio and intensity ("snr", "intensity"). The final column is "holder", representing the estimated Holder exponent asscoiated with each peak. Since noise.local is NULL, "snr" is the same as ("intensity").

References

Pan Du, Warren A. Kibbe, and Simon M. Lin, "Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching", *Bioinformatics*, **22**, 2059–2065 (2006).

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavCWT, wavCWTTree, msPeak, msPeakInfo.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## extract a subset of a single spectrum
mz  <- qcset$mz
imz <- mz > 3000 & mz < 5000
z   <- as.vector(qcset[imz, 1]$intensity)
mz  <- mz[imz]
```

```
plot(mz, z, cex=0.5, ylab="Intensity", xlab="m/z", type="l")

## estimate the peak locations using various
## minimum CWT scales and overlay the plot
## locations
scale.min <- c(2,4,8)
col <- c("green","blue","red")

for (i in seq(along=scale.min)){
    p <- msPeakCWT(mz, z, scale.min=scale.min[i])
    ipeak <- p[["tick.loc"]]
    points(mz[ipeak], z[ipeak], cex=i, col=col[i], pch=1)
}

## add a legend
if (is.R()){
  legend(3000,10000,pch=1,col=col,legend=paste("scale.min =", scale.min))
} else {
  legend(3000,10000,marks=1,col=col,legend=paste("scale.min =", scale.min))
}

## plot the Holder exponents corresponding to each
## peak
hx <- p[["mass.loc"]]
hy <- abs(p[["holder"]])
plot(hx, hy, type="h", xlim=range(mz), lty="dashed", col="blue", xlab="m/z", ylab="|Holde
points(hx, hy, col="red", cex=1.2)
lines(mz, rescale(z,c(0,0.5)), lwd=3)
abline(h=0, lty="dotted")
legend(3000,1,col=c("blue","black"), lwd=c(1,3), lty=c("dashed","solid"),
    legend=c("|Holder exponent|", "Scaled spectrum"))
```

`msPeakInfo` *Peak Detection Constructor Function*

Description

Provides a common interface for packing detected mass spectrometry peaks into an appropriate output object.

Usage

```
msPeakInfo(x, y, index.min, index.max, noise.local = NULL,
 snr.thresh = 2)
```

Arguments

`x`	A numeric vector representing the `m/z` values of a spectrum.
`y`	A numeric vector representing the intensity values of the spectrum.
`index.max`	A logical vector the size of the original mass spectrum. If the kth element is `TRUE`, it indicates that the corresonding kth element of the original mass spectrum is a local maxima.

`index.min` A logical vector the size of the original mass spectrum. If the kth element is `TRUE`, it indicates that the corresonding kth element of the original mass spectrum is a local minima. It is assumed that for each peak (identified as a `TRUE` element in `index.max`) there exists two minima which encompass the peak in time. Hence, it is expected that `length(which(index.min))` will be one greater than `length(which(index.max))`.

`noise.local` A numeric vector representing the estimated instantaneous noise level, i.e., one noise element for each m/z value. This argument is used to form signal-to-noise ratio (SNR) estimates that are subsequently compared to the `snr.thresh` argument in the pruning process. Default: `NULL` (no SNR pruning is performed).

`snr.thresh` A numeric value representing the signal intensity threshold. Only the local maxima whose signal intensity is above this value will be recorded as peaks. Default: `2`.

Value

A data.frame with 10 columns: peak class location, left bound, right bound and peak span in both clock tick (`"tick.loc"`, `"tick.left"`, `"tick.right"`, `"tick.span"`) and mass measure (`"mass.loc"`, `"mass.left"`, `"mass.right"`, `"mass.span"`), and peak signal-to-noise ratio and intensity (`"snr"`, `"intensity"`). Since `noise.local` is `NULL`, `"snr"` is the same as (`"intensity"`).

See Also

`msPeak`, `msPeakSimple`, `msPeakMRD`, `msPeakCWT`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## create faux MS peak data
z <- qcset[seq(500), 1]
x <- z$mz
y <- z$intensity
noise <- as.vector(wavCWT(y, n.scale=1))
index.min <- peaks(-y, span=31)
index.max <- peaks(y, span=31)

## pack data using the constructor
msPeakInfo(x, y, index.min, index.max, noise.local=noise, snr.thresh=1.3)
```

---

msPeakMRD *Peak Detection via Multiresolution Decomposition*

Description

A multiresolution decomposition (MRD) of the input time is formed using the maximal overlap discrete wavelet transform (MODWT). The sum of the MRD details (D) over the user-specified decomposition level(s) is formed.

If the number of specified decomposition levels is unity, first and second derivative approximations of D are approximated via (approximate) zero phase shifted versions of the MODWT(D) using the

Haar and D4 (Daubechies extremal phase 4-tap) wavelet filter, respectively. The index locations of the original series are "marked" where the the first derivative is approximately zero and the second derivative exceeds a user defined threshold, providing an estimate of the local extrema locations in D. Here, we note that a *positive* convavity threshold is used (as opposed to a negative value) due to a natural negation of the second derivative approximation using the wavelet scheme described above.

If the number of decomposition levels is greater than one, a simple peak detection scheme (`msExtrema`) is used to return the locations of the local maxima in D.

Usage

```
msPeakMRD(x, y, n.level=floor(log2(length(y))),
    concavity.threshold=0, snr.thresh=2, process="msPeakMRD")
```

Arguments

`x`	A numeric vector representing the `m/z` values of a spectrum.
`y`	A numeric vector representing the intensity values of the spectrum.
`concavity.threshold`	A non-negative concavity threshold. All points in the second derivative approximation to the calculated detail series that do no exceed this threshold are removed a potential local extrema candidates. Only used if `length(levels) == 1`. Default: 0.
`n.level`	The decomposition level in which the analysis is to be carried out. limited to `floor(log2(length(y)))`. Default: `floor(log2(length(y)))`.
`process`	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: `"msPeakMRD"`.
`snr.thresh`	A numeric value representing the signal intensity threshold. Only the local maxima whose signal intensity is above this value will be recorded as peaks. Default: 2.

Value

A data.frame with 10 columns: peak class location, left bound, right bound and peak span in both clock tick (`"tick.loc"`, `"tick.left"`, `"tick.right"`, `"tick.span"`) and mass measure (`"mass.loc"`, `"mass.left"`, `"mass.right"`, `"mass.span"`), and peak signal-to-noise ratio and intensity (`"snr"`, `"intensity"`). Since `noise.local` is `NULL`, `"snr"` is the same as (`"intensity"`).

References

T. W. Randolph and Y. Yasui, Multiscale Processing of Mass Spectrometry Data, *Biometrics*, **62**, pp. 589–597, 2006.

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

`wavMODWT`, `wavDaubechies`, `wavIndex`, `msPeak`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## extract a subset of a single spectrum
mz  <- qcset$mz
imz <- mz > 3000 & mz < 5000
z   <- as.vector(qcset[imz, 1]$intensity)
mz  <- mz[imz]

## specify the decomposition levels of interest
level <- 6

## calculate the MRD detail
D <- wavMRDSum(z, wavelet="haar",
    levels=level, xform="modwt",
    keep.smooth=FALSE, keep.details=TRUE,
    reflect=TRUE)

## locate MODWT MRD detail features via
## wavelet-based first and second derivative
## approximations
ipeak1 <- msPeakMRD(mz, D, n.level=level)[["tick.loc"]]

## plot the results
plot(D, cex=0.5, ylab=paste("D",level,sep=""), type="b")
abline(v=ipeak1, col="blue", lty="dashed")
```

---

msPeakSearch *Peak Detection via Elevated Intensity*

Description

This method seeks intensities that are higher than those in a local area and are higher than an estimated average background at the sites.

Usage

```
msPeakSearch(x, y, noise.local=NULL, span=41, span.supsmu=0.05,
    snr.thresh=2, process="msPeakSearch")
```

Arguments

x	A numeric vector representing the m/z values of a spectrum.
y	A numeric vector representing the intensity values of the spectrum.
noise.local	A numeric vector representing the estimated local noise level. Default: NULL.
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msPeakSearch".
snr.thresh	A numeric value representing the signal to noise threshold. Only the local maxima whose signal to noise level is above this value will be recorded as peaks. Default: 2.

span	A peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. Default: 41.
span.supsmu	The fraction of observations in the smoothing window. If span="cv", then automatic (variable) span selection is done by means of cross validation. Default: 0.05.

Value

A data.frame with 10 columns: peak class location, left bound, right bound and peak span in both clock tick ("tick.loc", "tick.left", "tick.right", "tick.span") and mass measure ("mass.loc", "mass.left", "mass.right", "mass.span"), and peak signal-to-noise ratio and intensity ("snr", "intensity"). If noise.local is NULL, "snr" is the same as ("intensity").

References

Tibshirani, R., Hastie, T., Narasimhan, B., Soltys, S., Shi, G., Koong, A., and Le, Q.T., "Sample classification from protein mass spectrometry, by peak probability contrasts," *Bioinformatics*, **20**(17):3034–44, 2004.

Yasui, Y., McLerran, D., Adam, B.L., Winget, M., Thornquist, M., Feng, Z., "An automated peak identification/calibration procedure for high-dimensional protein measures from mass spectrometers," *Journal of Biomedicine and Biotechnology*, **2003**(4):242–8, 2003.

Yasui, Y., Pepe, M., Thompson, M.L., Adam, B.L., Wright, Jr., G.L., Qu, Y., Potter, J.D., Winget, M., Thornquist, M., and Feng, Z., "A data-analytic strategy for protein biomarker discovery: Profiling of high-dimensional proteomic data for cancer detection," *Biostatistics*, **4**(3):449–63, 2003.

See Also

msPeak, msPeakSimple, msExtrema, peaks.

---

msPeakSimple *Peak Detection via Local Maxima*

Description

Performs peak detection via a simple local maxima search.

Usage

```
msPeakSimple(x, y, noise.local=NULL, span=3,
    snr.thresh=2, process="msPeakSimple")
```

Arguments

x	A vector representing the m/z values of a spectrum.
y	A vector representing the intensity values of the spectrum.
noise.local	A vector representing the estimated local noise level. Default: NULL.
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msPeakSimple".

snr.thresh — A value representing the signal to noise threshold. Only the local maxima whose signal to noise level is above this value will be recorded as peaks. Default: 2.

span — A peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. The default value is span=3, meaning that a peak is bigger than both of its neighbors.

Value

A data.frame with 10 columns: peak class location, left bound, right bound and peak span in both clock tick ("tick.loc", "tick.left", "tick.right", "tick.span") and mass measure ("mass.loc", "mass.left", "mass.right", "mass.span"), and peak signal-to-noise ratio and intensity ("snr", "intensity"). If noise.local is NULL, "snr" is the same as ("intensity").

References

Coombes, K.R., Tsavachidis, S., Morris, J.S., Baggerly, K.A., Kuerer, H.M., "Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform," *Proteomics*, **5**:4107–17, 2005.

See Also

msPeak, msPeakSearch, msExtrema, peaks.

---

msPlot *General Plotting Utility*

Description

This function facilitates a wide array of customized plots, enabling the user to easily combine matlines, matpoints, lines, points, image, abline, and text elements. An advantage of this technique is that the plot boundaries defined by xlim and ylim are automatically and correctly set using all sources of relevant input data to avoid "plot out of bounds" warnings. A second advantage is with the matlines and matpoints plots in that the optional offset function can be used to visually separate each series (column) of the supplied data matrix in the display, creating a waterfall-like plot. There are restrictions, however, in that the number of columns of the matlines and matpoints matrices must be the same if supplied. Also, currently a mix of matlines-matpoints with lines-points is not aupported.

Usage

```
msPlot(matlines.=NULL, matpoints.=NULL, lines.=NULL, points.=NULL,
    image.=NULL, abline.=NULL, text.=NULL, offset=NULL, recenter=FALSE,
    offset.fun=function(x){3 * stdev(x, na.rm=TRUE)},
    xlab="", ylab="", yref=TRUE, main="", add=FALSE, ...)
```

Arguments

`...`	additional arguments for `plot()`.
`abline.`	a list of named objects. This list must contain either of the objects `h` or `v`, containing the vector of coordinates at which to plot at the specifed ablines. Default: `NULL` (no ablines).
`add`	a logical value. If `TRUE`, the plot is added using the current `par()` layout. Otherwise a new plot is produced. Default: `FALSE`.
`image.`	a list of named objects. At the very least, the objects `x`, `y`, and `z` must exist, containing the x and y vectors and z matrix, respectively. Default: `NULL` (no image).
`lines.`	a list of named objects. At the very least, the objects `x` and `y` must exist, containing the x and y matrices, respectively. Default: `NULL` (no lines).
`main`	a character string representing the title label. Default: `""` (no label).
`matlines.`	a list of named objects. At the very least, the objects `x` and `y` must exist, containing the x and y matrices, respectively. Default: `NULL` (no matlines).
`matpoints.`	a list of named objects. At the very least, the objects `x` and `y` must exist, containing the x and y matrices, respectively. Default: `NULL` (no matpoints).
`offset`	a numeric scalar representing the vertical offset to apply between each line in `matlines` and each point in `matpoints`. If `NULL`, the offset is automatically calculated. Default: `NULL` (the offset between plots is set to one standard deviation calculated over the entire set of `matlines`-`matpoints` y data).
`offset.fun`	the function to use in calculating the `offset` if not supplied. This function should have a single input argument which operates over a vector assumed to contain a vectorized collection of all `matlines` and `matpoints` `y` matrices. Default: `function(x){3 * stdev(x)}`.
`points.`	a list of named objects. At the very least, the objects `x` and `y` must exist, containing the x and y matrices, respectively. Default: `NULL` (no points).
`recenter`	a logical value specifying whether or not to recenter the plot. Default: `FALSE`.
`text.`	a list of named objects. At the very least, the objects `x` and `y` must exist, containing the x and y coordinates, respectively, used to place the text. Default: `NULL` (no text).
`xlab`	a character string representing the x-axis label. Default: `""` (no label).
`ylab`	a character string representing the y-axis label. Default: `""` (no label).
`yref`	a logical value. If `TRUE`, a line connecting each y-offset value on the ordinate to the starting point of the corresponding series is drawn for `matlines` and `lines` plots. Default: `TRUE`.

Value

the offsets applied to each column of existing `matlines`-`matpoints` y data. This vector is returned invisibly.

See Also

`matlines`, `matpoints`, `lines`, `points`, `image`, `abline`, `text`, `par`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## define range variables
iz  <- seq(7500)
pts <- c(1500,2500,4500)

## plot matrix style objects
offsets <- msPlot(
    matlines=list(x=qcset$mz[iz], y=qcset$intensity[iz,1:4],
       lty = 1),
    matpoints=list(x=qcset$mz[pts], y=qcset$intensity[pts,1:4],
       pch=1,cex=2),
    text=list(x=c(3,7)*1000, y=c(15,18)*1000,
       labels=c("cow","moon"), cex=2, adj=1, col=1:2),
    abline=list(v=qcset$mz[pts], h=10000),
    xlab="X Label", ylab="Y Label", main="Title",
    yref=TRUE)

## plot non-matrix style objects
x <- y <- seq(-4*pi, 4*pi, len=27)
r <- sqrt(outer(x^2, y^2, "+"))
z <- cos(r^2)*exp(-r/6)

msPlot(
    lines=list(x=-60:-55, y=40:45, lwd=5, col=4),
    points=list(x=rnorm(30)-65, y=sin(1:30)+35, col=3, pch=2, cex=2),
    text=list(x=-65,y=39.5,"Sample text", cex=4, col=2),
    image=list(x=x/5-70, y=y/5+45, z=z),
    yref=FALSE)

## multiple matlines in a single call
x <- matrix(seq(0,10,by=0.1))
y <- sin(x %*% t(1:4))
msPlot(matlines=list(
    list(x=x, y=y, lty=1, lwd=1),
    list(x=x, y=y + 0.2, lty=1, lwd=2),
    list(x=x, y=y - 0.2, lty=1, lwd=4)),
    yref=TRUE)
```

`msPrepare` *Convert an msList Object to an msSet Object*

Description

Converts an `msList` object to an `msSet` object by truncating the spectra to the mass range of interest, interpolating the spectra to a common set of `m/z` values, and transforming the intensity values if specified.

Usage

```
msPrepare(x, mass.min=1500, transform=NULL, data.name = NULL)
```

Arguments

x	An object of class msList.
mass.min	A numeric value denoting the lowest mass (in Dalton) of interest. The portion of the spectra below this value will be removed. The low mass region is generally considered suspicious as matrix contamination may be a problem. Default: 1500.
transform	A function to be used to transform the intensity values. The main purpose of the transformation is to reduce the variance of the intensity values and to stabilize the noise component of the spectra. Some possible choices are logarithm (log), square root (sqrt) and cube root. The default is not to perform any tranformation. Default: NULL (no transform).
data.name	name for the data. Default: deparseText(substitute(x)).

Value

An object of class msSet that has the same number of spectra as in the input msList object x.

See Also

msList, msSet, msNormalize.

Examples

```
if (!exists("qclist")) data("qclist", package="msProcess")

## extract several spectra from the build-in
## dataset qclist
zList <- qclist[1:8]

## convert the subset to an msSet object
cbrt <- function(x) x^(1/3)
zSet <- msPrepare(zList, mass.min=950, transform=cbrt)

## visualize a portion of the spectra
plot(zSet, subset=NULL, xlim=c(13000, 17000), offset=0.5)
```

---

msProcess-package *Protein Mass Spectra Processing*

Description

This package provides tools for protein mass spectra processing including data preparation, denoising, noise estimation, baseline correction, intensity normalization, peak detection, peak alignment, peak quantification, and various functionalities for data ingestion/conversion, mass calibration, data quality assessment, and protein mass spectra simulation. It also provides auxiliary tools for data representation, data visualization, and pipeline processing history recording and retrieval.

Details

Basic class structures:

msList	Spectra List
msSet	Spectra Set

Top level functions:

`msImport`	Data Import
`msPrepare`	Data Conversion
`msDenoise`	Spectra Denoising
`msDetrend`	Baseline Correction
`msNormalize`	Intensity Normalization
`msPeak`	Peak Detection
`msAlign`	Peak Alignment
`msQuantify`	Peak Quantification
`msQualify`	Quality Assessment

Run `library(help="msProcess")` or `packageDescription("msProcess")` from S-PLUS command line to get more general information about this package.

See Also

`dimeR`, `MLearn`

`msQualify` *Quality Assessment of Mass Spectra Data*

Description

Creates a data quality object using a set of spectra from a quality control (QC) sample. The result can be used to assess the quality of other spectra generated from the same QC sample.

Usage

```
msQualify(x, FUN="princomp", ...)
```

Arguments

`x`	A matrix of peak intensity values with spectra as rows and peak classes as columns. The peak intensity matrix can be estimated via the `msAlign` function (with `measure="intensity"`) whose output contains (in part) a `peak.matrix` object.
`...`	Additional arguments for the specified principal component analysis `FUN`. See the specific function for details.
`FUN`	A character string specifying the method for principal component analysis. Possible choices are `"princomp"`, `"princompRob"`. Default: `"princomp"`.

Details

The user is expected to provide a (training) peak intensity matrix that has been derived from a set of pooled quality control samples. The output of `msQualify` contains the projection of this matrix onto its principal components (PCs) via the `princomp` or `princompRob` function. The user can subsequently assess the quality of another (test) peak intensity matrix generated from the same QC sample via the `predict` method, which compares the training PCs to the test PCs.

Value

An object of class `msQualify`.

S3 METHODS

predict Predict the quality of a set of spectra. This method supports the following optional arguments.

object An object of class `msQualify`.

newdata A matrix of peak intensities. It must have the same number of columns as the peak intensity matrix used to compute the `msQualify` object.

criterion A character string indicating the criterion to be use. Possible choices are "Cattell" and "Kaiser". Default: "Cattell".

threshold A numeric value representing the threshold to be used. Default: 0.9.

References

Coombes KR, Fritsche HA Jr., Clarke C, Chen JN, Baggerly KA, Morris JS, Xiao LC, Hung MC, and Kuerer HM, "Quality control and peak finding for proteomics data collected from nipple aspirate fluid by surface-enhanced laser desorption and ionization," *Clinical Chemistry*, **49**(10), pp. 1615–23, 2003.

See Also

`princomp`, `princompRob`.

Examples

```
## create multiple reference samples with multiple
## peaks
set.seed(10)
nrs <- 240
nv <- 35
my.mean <- 10
my.sd <- rnorm(nv)
my.sd <- my.sd - min(my.sd) + 1
rsam <- rmvnorm(n=nrs, d=nv, mean=rep(my.mean, nv),
    cov=diag(nv), sd=my.sd)

## run msQualify
pca <- msQualify(rsam, FUN="princompRob", estim="auto")

## create multiple reference samples with multiple
## peaks from the same distribution
nts <- 72
tsam <- rmvnorm(n=nts, d=nv, mean=rep(my.mean, nv),
    cov=diag(nv), sd=my.sd)

## predict the quality of the test samples

quality <- predict(pca, tsam)
quality$pass
if (!is.R()) assign("quality", quality, frame=1)
## check if the distances truly follow
## chisq(nkeep) distribution
```

```
qqmath(~quality$dist,
    distribution=function(p, df=quality$df) qchisq(p, df),
    panel = function(x, y) {
    panel.grid()
    panel.abline(0, 1)
    panel.qqmath(x, y)
    },
    aspect=1,
    xlab=paste("Chisq(", quality$df, ") Quantile"),
    ylab="mahalanobis distance")
```

`msQuantify` *Mother Function for Peak Quantification*

Description

Given an `msSet` object containing a `peak.class` element defining a common set of peak classes, this function returns either (i) a matrix of peak intensities or (ii) a count of the peaks that are associated with each peak class. The `measure` argument is used to specify the output type.

Usage

```
msQuantify(x, xnew=NULL, measure="intensity")
```

Arguments

`x`	An object of class `msSet` containing a `peak.class` element.
`measure`	A character string specifying the measure to be used for quantification. Choices are **"intensity"** quantifies a peak class using the maximum intensity in the corrected spectra within the span of the peak class. **"count"** quantifies a peak class using the number of peaks found in the corrected spectra within the span of the peak class. Default: `"intensity"`.
`xnew`	An object of class `msSet`. This object may contain a set of spectra that were not used to originally generate the peak classes. If the user wishes to quantify the original spectra, set `xnew=NULL`. Default: `NULL`.

Value

The same input `msSet` object (`x` if `xnew=NULL`, `xnew` otherwise) with an updated/new `peak.matrix` element. The rows and columns of the `peak.matrix` are the peak class measures and peak classes, respectively. If `measure="count"`, the element `"peak.list"` is also updated with a class ID for each peak.

References

Morris, J.S., Coombes, K.R., Koomen, J., Baggerly, K.A., Kobayashi, R., "Feature extraction and quantification for mass spectrometry in biomedical applications using the mean spectrum," *Bioinformatics*, **21**(9):1764–75, 2005.

Tibshirani, R., Hastie, T., Narasimhan, B., Soltys, S., Shi, G., Koong, A., and Le, Q.T., "Sample classification from protein mass spectrometry, by peak probability contrasts," *Bioinformatics*, **20**(17):3034–44, 2004.

Yasui, Y., McLerran, D., Adam, B.L., Winget, M., Thornquist, M., Feng, Z., "An automated peak identification/calibration procedure for high-dimensional protein measures from mass spectrometers," *Journal of Biomedicine and Biotechnology*, **2003**(4):242–8, 2003.

Yasui, Y., Pepe, M., Thompson, M.L., Adam, B.L., Wright, Jr., G.L., Qu, Y., Potter, J.D., Winget, M., Thornquist, M., and Feng, Z., "A data-analytic strategy for protein biomarker discovery: Profiling of high-dimensional proteomic data for cancer detection," *Biostatistics*, **4**(3):449–63, 2003.

See Also

`msQuantifyIntensity`, `msQuantifyCount`, `msAlign`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## extract several spectra from the build-in
## dataset
z <- qcset[, 1:8]

## denoising
z <- msDenoise(z, FUN="wavelet", n.level=10, thresh.scale=2)

## local noise estimation
z <- msNoise(z, FUN="mean")

## baseline subtraction
z <- msDetrend(z, FUN="monotone", attach=TRUE)

## intensity normalization
z <- msNormalize(z)

## peak detection
z <- msPeak(z, FUN="simple", use.mean=FALSE, snr=2)

## peak alignment
z <- msAlign(z, FUN="cluster", snr.thresh=10, mz.precision=0.004)

## peak quantification using intensity
z <- msQuantify(z, measure="intensity")

## extract peak.matrix
z[["peak.matrix"]]

## visualize the peak.matrix
image(z, what="peak.matrix")
```

`msQuantifyCount` *Count of Spectral Peaks Associated with a Peak Class*

Description

Using the `peak.list` element of a given `msSet` object, this function returns a count of peaks associated with a common set of peak classes.

Usage

```
msQuantifyCount(x, xnew=NULL)
```

Arguments

`x`	An object of class `msSet` containing a `peak.class` element.
`xnew`	An object of class `msSet`. This object may contain a set of spectra that were not used to originally generate the peak classes. If the user wishes to quantify the original spectra, set `xnew=NULL`. Default: `NULL`.

Value

The same input `msSet` object (`x` if `xnew=NULL`, `xnew` otherwise) with an updated/new `peak.matrix` element. The rows and columns of the `peak.matrix` are the peak class measures and peak classes, respectively. If `measure="count"`, the element `"peak.list"` is also updated with a class ID for each peak.

References

Morris, J.S., Coombes, K.R., Koomen, J., Baggerly, K.A., Kobayashi, R., "Feature extraction and quantification for mass spectrometry in biomedical applications using the mean spectrum," *Bioinformatics*, **21**(9):1764–75, 2005.

Tibshirani, R., Hastie, T., Narasimhan, B., Soltys, S., Shi, G., Koong, A., and Le, Q.T., "Sample classification from protein mass spectrometry, by peak probability contrasts," *Bioinformatics*, **20**(17):3034–44, 2004.

Yasui, Y., McLerran, D., Adam, B.L., Winget, M., Thornquist, M., Feng, Z., "An automated peak identification/calibration procedure for high-dimensional protein measures from mass spectrometers," *Journal of Biomedicine and Biotechnology*, **2003**(4):242–8, 2003.

Yasui, Y., Pepe, M., Thompson, M.L., Adam, B.L., Wright, Jr., G.L., Qu, Y., Potter, J.D., Winget, M., Thornquist, M., and Feng, Z., "A data-analytic strategy for protein biomarker discovery: Profiling of high-dimensional proteomic data for cancer detection," *Biostatistics*, **4**(3):449–63, 2003.

See Also

`msQuantify`, `msQuantifyIntensity`.

`msQuantifyIntensity` *Spectral Peak Intensities Associated with a Peak Class*

Description

Given an `msSet` object, this function quantifies the spectral peak intensities associated with a common set of peak classes. The measure used to identify candidate peaks is the maximum intensity value found over a pre-defined span of the corresponding peak class.

Usage

```
msQuantifyIntensity(x, xnew=NULL)
```

Arguments

`x`	An object of class `msSet` containing the `peak.class` element.
`xnew`	An object of class `msSet`. This object may contain a set of spectra that were not used to originally generate the peak classes. If the user wishes to quantify the original spectra, set `xnew=NULL`. Default: `NULL`.

Value

The same input `msSet` object (`x` if `xnew=NULL`, `xnew` otherwise) with an updated/new `peak.matrix` element. The rows and columns of the `peak.matrix` are the peak class measures and peak classes, respectively.

References

Morris, J.S., Coombes, K.R., Koomen, J., Baggerly, K.A., Kobayashi, R., "Feature extraction and quantification for mass spectrometry in biomedical applications using the mean spectrum," *Bioinformatics*, **21**(9):1764–75, 2005.

Tibshirani, R., Hastie, T., Narasimhan, B., Soltys, S., Shi, G., Koong, A., and Le, Q.T., "Sample classification from protein mass spectrometry, by peak probability contrasts," *Bioinformatics*, **20**(17):3034–44, 2004.

Yasui, Y., McLerran, D., Adam, B.L., Winget, M., Thornquist, M., Feng, Z., "An automated peak identification/calibration procedure for high-dimensional protein measures from mass spectrometers," *Journal of Biomedicine and Biotechnology*, **2003**(4):242–8, 2003.

Yasui, Y., Pepe, M., Thompson, M.L., Adam, B.L., Wright, Jr., G.L., Qu, Y., Potter, J.D., Winget, M., Thornquist, M., and Feng, Z., "A data-analytic strategy for protein biomarker discovery: Profiling of high-dimensional proteomic data for cancer detection," *Biostatistics*, **4**(3):449–63, 2003.

See Also

`msQuantify`, `msQuantifyCount`.

`msSet` *S3 Class Representing a Set of Spectra with Common m/z Values*

Description

An object of class `msSet` is a list containing three required components:

mz A vector of mass/charge values.

type A factor denoting the disease label for each spectrum.

intensity A matrix of intensity values. Each column corresponds to a spectrum and each row corresponds to an m/z value. The number of rows should be the same as the length of `mz`. The number of columns should be the same as the length of `type`.

It can also have additional elements such as `"noise"`, `"noise.local"`, `"baseline"`, `"tic"`, `"peak.list"`, `"peak.class"`, `"peak.matrix"`, `"intensity.mean"`, `"noise.mean"`, and `"noise.local.mean"`.

Usage

```
msSet(x, mz=NULL, type=NULL, data.name=NULL, ...)
```

Arguments

`x`	a `vector`, a `matrix`, a `data.frame`, or an object of class `msSet`.
`...`	additional elements to be added.
`data.name`	a character string denoting the name for the dataset. Default: NULL, i.e., using the variable name of the `x` argument.
`mz`	a vector of m/z values. It must be of the same length as `x` if `x` is a vector, or its length must be the same as the number of rows of `x` if `x` is a `matrix` or `data.frame`.
`type`	a factor denoting the disease label for each spectrum. This argument must be a scalar (length=1) if `x` is a `vector`, or a vector whose length is the same as the number of rows of `x` in the case that `x` is a `matrix` or `data.frame`.

Value

an object of class `msSet`.

S3 METHODS

[spectra extraction from an `msSet` object.

Usage: x[i, j]

x an `msSet` object.

i a subscript expression used to identify the m/z elements to extract or replace.

j a subscript expression used to identify the spectra to extract or replace.

image displays a set of spectra as an image.

Usage: image(x, what="spectra", subset=NULL, xaxis="mass", xlim=NULL, add=FALSE, xaxs="i", yaxs="i", ...)

x an `msSet` object.

what a character string specifying what to be imaged. The options are "spectra", "noise", "noise.local", "baseline", "peak.list", and "peak.matrix". Default: "spectra". If what=="peak.matrix", the arguments subset, xaxis, and xlim are ignored.

subset a numeric vector indicating the spectra numbers to be included. All spectra are included by default.

xaxis a character string specifying what to be used as the x-axis. The options are "mass" and "time". Default: "time".

xlim a numeric vector with two elements specifying the range for the x axis. Its first element should be less than its second element. If it is NULL (by default), the entire range of the data is plotted. If one of its element is NA, it is replaced with the corresponding limit.

add a logical value. If TRUE, the plot is added using the current par() layout. Otherwise a new plot is produced. Default: FALSE.

xaxs a character specifying the style of x axis interval calculation. Default: "i". See par for details.

yaxs a character specifying the style of y axis interval calculation. Default: "i". See par for details.

... Other graphical parameters passed to the image function.

plot visualizes a particular processing step.

Usage: plot(x, process="msPrepare", subset=1, offset=0, xaxis="mass", xlim=NULL, pch=1, lty=1:2, col=1:8, lwd=1, add=FALSE, ...)

x an msSet object.

process a character string specifying the process to be visualized. The options are "msPrepare", "msDenoise", "msNoise", "msDetrend", "msIntensityNormalize", "msPeak", "msAlign". Default: "msPrepare".

subset a numeric vector or NULL indicating the index of the spectra to be plotted. Default: 1, i.e., the first spectrum. If NULL, the entire set of spectra are plotted.

offset a numeric scalar representing the vertical offset to apply between each spectrum. If NULL, the offset is automatically calculated. Default: NULL.

xaxis a character string specifying what to be used as the x-axis. The options are "mass" and "time". Default: "time".

xlim a numeric vector with two elements specifying the range for the x axis. Its first element should be less than its second element. If it is NULL (by default), the entire range of the data is plotted. If one of its element is NA, it is replaced with the corresponding limit.

pch a single character or integer denoting the plotting character for the peaks. Default: 1, i.e., a circle.

lty an integer vector of length 2 denoting the line type for the two lines associated with each spectrum to be plotted. If it is a single integer, it will be used cyclically. If its length is more than 2, only the first two elements will be used.

col a vector of integers denoting the colors for the spectra to be plotted. Colors are used cyclically.

lwd a integer vector of length 2 denoting the line width for the two lines associated with each spectrum to be plotted, device dependent. If it is a single integer, it will be used cyclically. If its length is more than 2, only the first two elements will be used. Width 1 is the standard width for the device. Many devices cannot change line width. Default: 1.

add a logical value. If TRUE, the plot is added using the current par() layout. Otherwise a new plot is produced. Default: FALSE.

... Other graphical parameters passed to the underlying plotting functions.

print prints an `msSet` object.

Usage: print(x, justify="left", sep=":", ...) or x

x an `msSet` object.

justify a character string giving the justification of the numbers relative to each other. The choices are `"none"`, `"left"`, `"right"` and `"decimal"`. Only the first letter needs to be given.

sep a character string to be inserted between text and values. The default is a colon.

summary provides a synopsis of an `msSet` object.

Usage: summary(x)

x an `msSet` object.

See Also

`msList`, `msPlot`, `apply.msSet`.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## plot a few spectra from qcset
plot(qcset, subset=1:5)

## image of all spectra in qcset
image(qcset)
```

---

msSmoothApprox *Piecewise Linear Baseline Estimation*

Description

Estimates the baseline of a spectrum as a linear or constant interpolation of the local minima of a spectrum.

Usage

```
msSmoothApprox(x, y, method="linear", rule=2, f=0.5,
    index=rep(TRUE, length(x)), process="msSmoothApprox")
```

Arguments

`x`	A numeric vector representing the `m/z` values of a spectrum.
`y`	A numeric vector representing the intensity values of the spectrum corresponding to the specified `m/z` values.
`f`	A numeric scalar used when `method="constant"`, which determines a blend of the left and right side `y` values. e.g., suppose we want an interpolated value between x1 and x2 (with corresponding y values y1 and y2). Then the interpolated value is $(1 - f) * y1 + f * y2$. Thus, if $f = 0$, the left y-value is used, if $f = 1$, the right y-value, and if f is between 0 and 1, an intermediate value is used. Default: 0.5.

index	A logical vector indicating the local minima to be used to approximate the baseline. Default: rep(TRUE, length(x)).
method	A character string describing the method to be used in approximating the baseline. This must be either "linear" or "constant". Default: "linear".
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msSmoothApprox".
rule	An integer describing the rule to be used for values that are outside the range of the minima of x. If rule=1, NAs will be supplied for any such points. If rule=2, the y values corresponding to the extreme x values will be used. If rule=3, linear extrapolation is used. Default: 2.

Value

A numeric vector representing the estimated piece-wise linear baseline.

See Also

msSmoothKsmooth, msSmoothLoess, msSmoothMean, msSmoothMonotone, msSmoothSpline, msSmoothSupsmu.

---

msSmoothKsmooth *Fit a Smooth Curve Using Kernel Smoothers*

Description

Fits a smooth curve to a set of data points using kernel smoothers and returns a vector of fitted smooth curve values evaluated at the original locations.

Usage

```
msSmoothKsmooth(x, y, kernel="box", bandwidth=100, process="msSmoothKsmooth")
```

Arguments

x	A numeric vector of abscissa values.
y	A numeric vector of ordinate values, which must be of the same length as x.
kernel, bandwidth	See fucntion ksmooth for descriptions.
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msSmoothKsmooth".

Value

A numeric vector of fitted smooth curve values evaluated at the original locations.

See Also

msSmoothApprox, msSmoothLoess, msSmoothMean, msSmoothMonotone, msSmoothSpline, msSmoothSupsmu.

msSmoothLoess *Fit a Smooth Curve Using Loess*

Description

Fits a local regression model to a subset or set of data points and returns a vector of fitted smooth curve values evaluated at the original locations.

Usage

```
msSmoothLoess(x, y, span=0.1, degree=1, family="symmetric",
    index=rep(TRUE, length(x)), process="msSmoothLoess")
```

Arguments

x	A numeric vector of abscissa coordinates.
y	A numeric vector of ordinate coordinates, which must be of the same length as x.
index	A logical vector of the same length as x, indicating the indices of x and y to use in fitting the data. Default: rep(TRUE, length(x)).
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msSmoothLoess".
span, degree, family	See function loess.smooth for descriptions.

Value

A numeric vector of fitted smooth curve values evaluated at the original locations.

See Also

msSmoothApprox, msSmoothKsmooth, msSmoothMean, msSmoothMonotone, msSmoothSpline, msSmoothSupsmu.

msSmoothMRD *Baseline Estimation via Partial Summation of an MRD*

Description

Forms a multiresolution decomposition (MRD) by taking a specified discrete wavelet transform of the input spectrum and subsequently inverting each level of the transform back to the "time" domain. The resulting components of the MRD form an octave-band decomposition of the original spectrum, and can be summed together to reconstruct the original spectrum. In many real-world observations, the trend of the data is caught up in the last decomposition level's so-called *smooth*, which corresponds to residual low frequency content. This function allows the user to approximate the baseline trend in a mass spectrum by calculating the MRD smooth. Additionally, the user has the option to include relatively higher frequency content in the approximation by including various MRD details (see the DETAILS section for a definition of MRD details and MRD smooth). As this function primarily calls the wavMRDSum function with appropriate arguments, see the corresponding help documentation for more details.

Usage

```
msSmoothMRD(x, wavelet="s8", levels=1,
    xform="modwt", reflect=TRUE,
    keep.smooth=TRUE, keep.details=FALSE,
    process="msSmoothMRD")
```

Arguments

x	A vector containing a uniformly-sampled real-valued time series.
keep.details	A logical value. If TRUE, the details corresponding to the specified levels are included in the partial summation over the MRD components. The user also has the choice to include the smooth in the summation via the keep.smooth option, but one of keep.details and keep.smooth must be TRUE. Default: FALSE.
keep.smooth	A logical value. If TRUE, the smooth at the last decomposition level is added to the partial summation over specified details. The smooth typically contains low-frequency trends present in a spectrum. The user also has the choice to include the details in the summation via the keep.details option, but one of keep.details and keep.smooth must be TRUE. Default: TRUE.
levels	An integer vector of integers denoting the MRD detail(s) to sum over in forming a denoised approximation to the orginal spectrum (the summation is performed across scale and nto across time). All values must be positive integers, and cannot exceed floor(logb(length(x),2)) if reflect=FALSE and, if reflect=TRUE, cannot exceed floor(logb((length(x)-1)/(L-1) + 1, b=2)) where L is the length of the wavelet filter. Use the keep.smooth option to also include the last level's smooth in the summation. Default: 1.
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. This process is not updated if it already exists in the event history. Default: "msSmoothMRD".
reflect	A logical value. If TRUE, the last $L_J = (2^{\text{n.level}} - 1)(L-1)+1$ coefficients of the series are reflected (reversed and appended to the end of the series) in order to attenuate the adverse effect of circular filter operations on wavelet transform coefficients for series whose endpoint levels are (highly) mismatched. The variable L_J represents the effective filter length at decomposition level n.level, where L is the length of the wavelet (or scaling) filter. A similar operation is performed at the beginning of the series. After synthesis and (partial) summation of the resulting details and smooth, the middle N points of the result are returned, where N is the length of the original time series. Default: TRUE.
wavelet	A character string denoting the filter type. See wavDaubechies for details. Default: "s8".
xform	A character string denoting the wavelet transform type. Choices are "dwt" and "modwt" for the discrete wavelet transform (DWT) and maximal overlap DWT (MODWT), respectively. The DWT is a decimated transform where (at each level) the number of transform coefficients is halved. Given N is the length of the original time series, the total number of DWT transform coefficients is N. The MODWT is a non-decimated transform where the number of coefficients at each level is N and the total number of transform coefficients is N*n.level. Unlike the DWT, the MODWT is shift-invariant and is seen as a weighted average of all possible non-redundant shifts of the DWT. See the references for details. Default: "modwt".

Value

A vector containing the baseline trend of the input spectrum.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

T.W. Randolph and Y. Yasui, *Multiscale Processing of Mass Spectrometry Data*, *Biometrics*, **62**:589–97, 2006.

See Also

wavMRDSum, msDetrend, wavDaubechies, wavDWT, wavMODWT, wavMRD, eventHistory.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## obtain a subset of a mass spectrum and add some
## noise
x <- qcset$intensity[5000:10000,1]
sd.noise <- 2
set.seed(100)
xnoise <- x + rnorm(length(x), sd=sd.noise)
mz <- as.matrix(as.numeric(names(x)))

## calculate the smooth at decomposition level 9
## and use that as an approximation of the
## spectrum baseline
z <- msSmoothMRD(xnoise, levels=9)

## plot the results
plot(range(mz), range(xnoise), type="n",
    xlab="m/z", ylab="Spectrum and Baseline Estimation")
lines(mz, xnoise, type="l", lty=2, col=1)
lines(mz, z, lty=1, lwd=3, col=2)
```

---

msSmoothMean *Fit a Smooth Curve Using Moving Average*

Description

Fits a smooth curve to a vector using moving average and returns a vector of fitted smooth curve values evaluated at the original locations.

Usage

```
msSmoothMean(y, half.span=250, process="msSmoothMean")
```

Arguments

y	A numeric vector.
half.span	A numeric value denoting half of the window width when performing average. Default: 250.
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msSmoothMean".

Value

A vector of fitted smooth curve values evaluated at the original locations.

See Also

msSmoothApprox, msSmoothKsmooth, msSmoothLoess, msSmoothMonotone, msSmoothSpline, msSmoothSupsmu.

---

msSmoothMonotone *Monotonic Baseline Estimation*

Description

Estimates the baseline of a spectrum as a decreasing function that always takes on a monotonic minimum value.

Usage

```
msSmoothMonotone(x, process="msSmoothMonotone")
```

Arguments

x	A numeric vector representing the intensity values of a spectrum.
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msSmoothMonotone".

Details

Since this function estimates the baseline of a spectrum as a monotone minimum, it works only if the portion before the maximum of the saturation period has been removed from the spectrum.

Value

A vector representing the estimated monotonically decreasing baseline.

References

Coombes, K.R., Tsavachidis, S., Morris, J.S., Baggerly, K.A., Kuerer, H.M., "Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform," *Proteomics*, **5**:4107–17, 2005.

See Also

msSmoothApprox, msSmoothKsmooth, msSmoothLoess, msSmoothMean, msSmoothSpline, msSmoothSupsmu.

`msSmoothSpline` *Fit a Smooth Curve Using Cubic Spline*

Description

Fits a cubic spline curve to a subset or set of data points and returns a vector of fitted smooth curve values evaluated at the original locations.

Usage

```
msSmoothSpline(x, y, df=30, spar=0, cv=FALSE, all.knots=FALSE,
    df.offset=0, penalty=1, index=rep(TRUE, length(x)),
    process="msSmoothSpline")
```

Arguments

`x`	A numeric vector of abscissa values.
`y`	A numeric vector of ordinate values, which must be of the same length of `x`.
`df, spar, cv, all.knots, df.offset, penalty`	See function `smooth.spline` for descriptions.
`index`	A logical vector of the same length of `x` indicating the elements to be used in the fitting. Deafult: `rep(TRUE, length(x))`.
`process`	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: `"msSmoothSpline"`.

Value

A numeric vector of fitted smooth curve values evaluated at the original locations.

See Also

msSmoothApprox, msSmoothKsmooth, msSmoothLoess, msSmoothMean, msSmoothMonotone, msSmoothSupsmu.

msSmoothSupsmu *Fit a Smooth Curve Using Super Smoother*

Description

Fits a smooth curve to a subset or set of data points using a super smoother and returns a vector of fitted smooth curve values evaluated at the original locations.

Usage

```
msSmoothSupsmu(x, y, span="cv", periodic=FALSE, bass=0,
    index=rep(TRUE, length(x)), process="msSmoothSupsmu")
```

Arguments

x	A numeric vector of abscissa values.
y	A numeric vector of ordinate values, which must be of the same length of x.
index	A logical vector of the same length of x indicating the elements to be used in the fitting. Deafult: rep(TRUE, length(x)).
process	A character string denoting the name of the process to register with the (embedded) event history object of the input after processing the input data. Default: "msSmoothSupsmu".
span, periodic, bass	See function supsmu for descriptions.

Value

A vector of fitted smooth curve values evaluated at the original locations.

See Also

msSmoothApprox, msSmoothKsmooth, msSmoothLoess, msSmoothMean, msSmoothMonotone, msSmoothSpline.

proteins *Class Representing a Protein Mixture or Sample*

Description

Class slots:

masses A numeric vector of protein masses in daltons.

counts An integer vector of protein counts/abundance.

Usage

```
proteins(masses, counts)
```

Arguments

`masses`	A positive numeric vector of protein masses in daltons, whose elements should be unique.
`counts`	A positive integer vector of protein counts/abundance, which should have the same length as `masses`.

S3 METHODS

Arith Arith group generic functions.

Usage: FUN(x, y) or x op y

FUN a member of the `Arith` group generic functions, which include +, −, ⋆,^, %%, %/%, and /.

x a `proteins` object or a numeric vector.

y a `proteins` object or an integer vector.

If both `x` and `y` are objects of `proteins`, only + and − apply as mixing two protein samples and taking part of a protein sample away, respectively. If `x` is a numeric and `y` is an objects of `proteins`, the `masses` of `y` will be modified according to the operation. If `x` is an objects of `proteins` and `y` is an integer, the `counts` of `x` will be modified according to the operation.

Compare Compare group generic functions.

Usage: FUN(x, y)

FUN a member of the `Compare` group generic functions, which include ==, >, <, ! =, <=, >=, and *compare*.

x a `proteins` object or a numeric vector.

y a `proteins` object or a numeric vector.

If both `x` and `y` are objects of `proteins`, only == and ! = apply . If `x` is a numeric and `y` is an objects of `proteins`, the `masses` of `y` will be compared according to the operation. If `x` is an objects of `proteins` and `y` is a numeric, the `counts` of `x` will be compared according to the operation.

Math, Math2, Logic Math, Math2, Logic group generic functions are not defined for class `proteins`.

Summary Summary group generic functions.

Usage: FUN(x)

FUN a member of the `Summary` group generic functions, which include `max`, `min`, `range`, `prod`, `sum`, `any`, and `all`.

x a `proteins` object.

[Extract parts of a `proteins` object.

Usage: x[i]

x a `proteins` object.

i a subscript expression identifying the proteins to extract.

[@mt<- Replace the counts of parts of a `proteins` object.

Usage: x[i]< −value

x a `proteins` object.

i a subscript expression identifying the protein counts to replace.

value a `proteins` object or an integer vector.

show Display a `proteins` object.

Usage: show(object) or object

object a `proteins` object.

xyCall Make all functions that naturally relate to points in the plane work with objects of class `proteins`, including `plot`, `lines`, `points`, and etc. This generic function is not meant to be called directly.

Usage: xyCall(x, y, FUN, ..., xexpr, yexpr)

x a `proteins` object.

y missing.

FUN a function to be called that have arguments `x`, `y`, ...

xexpr the S object representing the `x` argument to `FUN` unevaluated.

yexpr the S object representing the `y` argument to `FUN` unevaluated.

References

Coombes, K.R., Koomen, J.M., Baggerly, K.A., Morris, J.S., Kobayashi, R., "Understanding the characteristics of mass spectrometry data through the use of simulation," *Cancer Informatics*, **2005(1)**:41–52, 2005.

See Also

`calibrants`, `spectrometer`.

Examples

```
## generate two protein samples
sam1 <- proteins(masses=c(1, 95, 190), counts=as.integer(c(500, 3000, 10000)))
sam2 <- proteins(masses=10000+200*(0:3), counts=as.integer(c(12000, 4000, 2000, 1000)))

## print the synopsis of the protein samples
sam1
sam2

## mix the protein samples
sam <- sam1 + sam2

## visualize the protein mixture
plot(sam, type="h")
```

---

qclist *Mass Spectra from a Breast Cancer Quality Control Sample*

Description

A data object of class `msList`, consisting of 8 mass spectra generated from a pooled sample of nipple aspirate fluid (NAF) from healthy breasts and breasts with cancer. (see references for details).

References

Coombes, K.R., Tsavachidis, S., Morris, J.S., Baggerly, K.A., and Kuerer, H.M., "Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform," *Proteomics*, **5**:4107–17, 2005.

Pawlik, T.M., Fritsche, H., Coombes, K.R., Xiao, L., Krishnamurthy, S., Hunt, K.K., Pusztai, L., Chen, J.N., Clarke, C.H., Arun, B., Hung, M.C., and Kuerer, H.M., "Significant differences in nipple aspirate fluid protein expression between healthy women and those with breast cancer demonstrated by time-of-flight mass spectrometry," *Breast Cancer Research and Treatment*, **89**(2):149–57, 2005.

Kuerer, H.M., Coombes, K.R., Chen, J.N., Xiao, L., Clarke, C., Fritsche, H., Krishnamurthy, S., Marcy, S., Hung, M.C., and Hunt, K.K., "Association between ductal fluid proteomic expression profiles and the presence of lymph node metastases in women with breast cancer," *Surgery*, **136**(5):1061–9, 2004.

Coombes, K.R., Fritsche, Jr., H.A., Clarke, C., Chen, J.N., Baggerly, K.A., Morris, J.S., Xiao, L.C., Hung, M.C., and Kuerer, H.M., "Quality control and peak finding for proteomics data collected from nipple aspirate fluid by surface-enhanced laser desorption and ionization," *Clinical Chemistry*, **49**(10):1615–23, 2003.

See Also

qcset.

Examples

```
if (!exists("qclist")) data("qclist", package="msProcess")

## print an this-is-escaped-code{ object
qclist

## print the synopsis of an this-is-escaped-codenormal-bracket21bracket-normal object
summary(qclist)

## plot the first spectrum from an this-is-escaped-codenormal-bracket22bracket-normal
## object
plot(qclist, index=1)
```

---

qcset *Mass Spectra from a Breast Cancer Quality Control Sample*

Description

A data object of class msSet, consisting of 8 mass spectra generated from a pooled sample of nipple aspirate fluid (NAF) from healthy breasts and breasts with cancer. The data set was derived from qclist by eliminating the mass region below 950 Da/charge. Please see the references for more details.

References

Coombes, K.R., Tsavachidis, S., Morris, J.S., Baggerly, K.A., and Kuerer, H.M., "Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform," *Proteomics*, **5**:4107–17, 2005.

Pawlik, T.M., Fritsche, H., Coombes, K.R., Xiao, L., Krishnamurthy, S., Hunt, K.K., Pusztai, L., Chen, J.N., Clarke, C.H., Arun, B., Hung, M.C., and Kuerer, H.M., "Significant differences in nipple aspirate fluid protein expression between healthy women and those with breast cancer demonstrated by time-of-flight mass spectrometry," *Breast Cancer Research and Treatment*, **89**(2):149–57, 2005.

Kuerer, H.M., Coombes, K.R., Chen, J.N., Xiao, L., Clarke, C., Fritsche, H., Krishnamurthy, S., Marcy, S., Hung, M.C., and Hunt, K.K., "Association between ductal fluid proteomic expression profiles and the presence of lymph node metastases in women with breast cancer," *Surgery*, **136**(5):1061–9, 2004.

Coombes, K.R., Fritsche, Jr., H.A., Clarke, C., Chen, J.N., Baggerly, K.A., Morris, J.S., Xiao, L.C., Hung, M.C., and Kuerer, H.M., "Quality control and peak finding for proteomics data collected from nipple aspirate fluid by surface-enhanced laser desorption and ionization," *Clinical Chemistry*, **49**(10):1615–23, 2003.

See Also

qclist.

Examples

```
if (!exists("qcset")) data("qcset", package="msProcess")

## plot a few spectra
plot(qcset, subset=1:5)

## image of all spectra
image(qcset)
```

---

rescale *Rescale a Vector or a Matrix to the Specified Limits*

Description

Linear scales a vector or matrix to the specified range.

Usage

```
rescale(x, range.=c(0,1), ...)
```

Arguments

x	a vector as defined by isVectorAtomic, a signalSeries object, or a matrix.
...	additional data to be rescaled. Each additional input is scaled in the exact same way as the x input. This provides relative scaling functionality to the user.
range.	a two-element numeric vector containing the upper and lower bounds for the scaled data: Default: c(0,1).

See Also

scale.

Examples

```
rescale(1:10,c(-3,-5))
rescale(-1:4, c(0,1), 1:3, 0:1)
```

---

setting *Class Representing the Setting of a Mass Spectrometer*

Description

Class slots:

dist.drift a numeric scalar denoting the length of drift tube in meters.

dist.focus a numeric scalar denoting the distance between charged grids in millimeters.

dist.accel a numeric scalar denoting the distance from sample plate to first grid in millimeters.

volt.accel a numeric scalar denoting the voltage between charged grids in volts.

volt.focus a numeric scalar denoting the voltage used in ion focusing phase in volts.

time.delay a numeric scalar denoting the delay time before focus voltage is applied in nanoseconds.

time.resol a numeric scalar denoting the time between detector records in seconds.

vel0.mean a numeric scalar denoting the mean initial velocity in meters/second.

vel0.std a numeric scalar denoting the standard deviation of initial velocity.

The model of a linear MALDI-TOF instrument with time-lag focusing depends on the nine parameters. In a real instrument, the three distance parameters (`dist.drift`, `dist.focus`, and `dist.accel`) are unchanging characteristics of the design. The user has direct control over `volt.accel`, `volt.focus`, `time.delay`, and `time.resol`. The parameters that determine the normal distribution of initial velocities, i.e., `vel0.mean` and `vel0.std`, are controlled indirectly by the choice of EAM and by the laser intensity.

S3 METHODS

show Display a `setting` object.

Usage: show(object) or object

object a `setting` object.

References

Coombes, K.R., Koomen, J.M., Baggerly, K.A., Morris, J.S., Kobayashi, R., "Understanding the characteristics of mass spectrometry data through the use of simulation," *Cancer Informatics*, **2005(1)**:41–52, 2005.

See Also

spectrometer.

`spectrometer` *Class Representing a Mass Spectrometer*

Description

Class slots:

setting an object of class `setting`.

calibrator an object of class `calibrator`.

Usage

```
spectrometer(dist.drift=1, dist.focus=17, dist.accel=8,
    volt.accel=20000, volt.focus=2000,
    time.delay=600, time.resol=4e-9,
    vel0.mean=350, vel0.std=50,
    time.mean=numeric(0), model=structure(NULL, class="lm"), error.rel=numeric(0
```

Arguments

`dist.accel`	A numeric scalar denoting the distance from sample plate to first grid in millimeters.
`dist.drift`	A numeric scalar denoting the length of drift tube in meters.
`dist.focus`	A numeric scalar denoting the distance between charged grids in millimeters.
`error.rel`	A numeric vector denoting the relative calibration error for the calibrants.
`model`	An object of class `lm`.
`time.delay`	A numeric scalar denoting the delay time before focus voltage is applied in nanoseconds.
`time.mean`	A numeric scalar denoting the mean time-of-flight of the calibrants.
`time.resol`	A numeric scalar denoting the time between detector records in seconds.
`vel0.mean`	A numeric scalar denoting the mean initial velocity in meters/second.
`vel0.std`	A numeric scalar denoting the standard deviation of initial velocity.
`volt.accel`	A numeric scalar denoting the voltage between charged grids in volts.
`volt.focus`	A numeric scalar denoting the voltage used in ion focusing phase in volts.

S3 METHODS

run Run a mass spectrometer.

Usage: run(simObj, proObj, isotope)

simObj a `spectrometer` object.

proObj a `proteins` object or a `calibrants` object. If `proObj` is a `calibrants` object, then a calibration run is performed and a calibrated `spectrometer` object is returned. If `proObj` is a `proteins` object, then a real run is performed and a `spectrum` object is returned. If the `spectrometer` object has been calibrated, the returned `spectrum` will have values for both `mz` slot and `tof` slot. Otherwise, the returned `spectrum` will have values only for `tof` slot.

isotope a logical value indicating if isotope distribution should be simulated.

References

Coombes, K.R., Koomen, J.M., Baggerly, K.A., Morris, J.S., Kobayashi, R., "Understanding the characteristics of mass spectrometry data through the use of simulation," *Cancer Informatics*, **2005(1)**:41–52, 2005.

See Also

`setting`, `calibrator`, `spectrum`, `ion.focus.delay`.

Examples

```
## run a uncalibrated mass spectrometer
sam <- proteins(masses=c(1, 95, 190), counts=as.integer(c(500, 3000, 10000)))
sim <- spectrometer(vel0.mean=350, vel0.std=75, time.resol=4e-9)
x <- run(sim, sam)
plot(x)

## run a calibrated mass spectrometer
cal <- calibrants(masses=c(1000, 2000, 5000, 10000, 20000), counts=as.integer(rep(1000, 5
sim.cal <- run(sim, cal)
y <- run(sim.cal, sam)
plot(y)
```

`spectrum` *Class Representing a Spectrum*

Description

Class slots:

tof a numeric vector denoting the time-of-flight value of the proteins.

mz a numeric vector denoting the mass-to-charge ratio of the proteins.

intensity a integer vector denoting the aboundance/counts of the proteins.

S3 METHODS

xyCall Make all functions that naturally relate to points in the plane work with objects of class `spectrum`, including `plot`, `lines`, `points`, and etc. This generic function is not meant to be called directly.

Usage: xyCall(x, y, FUN, ..., xexpr, yexpr)

x a `spectrum` object.

y missing.

FUN a function to be called that have arguments `x`, `y`, ...

xexpr the S object representing the `x` argument to `FUN` unevaluated.

yexpr the S object representing the `y` argument to `FUN` unevaluated.

References

Coombes, K.R., Koomen, J.M., Baggerly, K.A., Morris, J.S., Kobayashi, R., "Understanding the characteristics of mass spectrometry data through the use of simulation," *Cancer Informatics*, **2005(1)**:41–52, 2005.

See Also

spectrometer.

`throwEvent` *Throw a History Event to a Specified Frame*

Description

Throws a history event with a specified name to a specified frame in S-PLUS (or environment in R). Typically, this function will be issued within a caller with the expectation that the thrown history will be added to by the callee(s). The history can then be caught by the caller using the `catchHistory` function.

Usage

```
throwEvent(x, histname="event.history", envir=msNewEnv())
```

Arguments

`x`	a character string defining the event.
`envir`	the `frame` in S-PLUS (or `environment` in R) designated for the processing and storage of pipeline history data. Default: `msNewEnv()`.
`histname`	a character string defining the name of the history variable stored in the specified frame. Default: `"event.history"`.

Value

a list named `histname` containing the following objects is written to the specified frame.

`name`	A character string defining the name of the event.
`history`	The actual history object.
`recorded`	A vector of chacter strings maintaining a record of all processes that have been recorded for the event.

Typically, these list objects will not be accessed directly by the user. Rather, this information is updated and retrieved via the `throwEvent` and `catchEvent` function.

See Also

`catchEvent`, `assignEvent`, `isProcessRecorded`, `eventHistory`.

Examples

```
## throw an event
envir <- msGlobalEnv()
throwEvent("Wimbledon 2005", envir=envir)

## assign data to the thrown event
record  <- list(Men="Roger Federer", Women="Venus Williams")
assignEvent(record)

## catch event
catchEvent(NULL)
```

`zeroCross` *Find the Zero Crossing of a Time Series*

Description

Finds positive-negative of negative-positive zero crossings of the input time series.

Usage

```
zeroCross(x, slope="positive")
```

Arguments

`x`	a numeric vector representing a time series.
`slope`	a character string defining the typeo of zero-crossings to find: specifying a `"negative"` slope identifies all positive- to negative-valued zero-crossings, while a `"positive"` slope identifies all negative- to positive-valued zero crossings. Default: `"positive"`.

See Also

`msExtrema`.

Examples

```
x <- sin(seq(0,10,length=1000))
pos <- zeroCross(x, "positive")
neg <- zeroCross(x, "negative")
plot(x, cex=0.5, type="b")
abline(v=pos, col="red", lty="solid")
abline(v=neg, col="blue", lty="dashed")
```

Index