Package 'timeSeries'

August 26, 2023

Title Financial Time Series Objects (Rmetrics)

Version 4031.107

Description 'S4' classes and various tools for financial time series:
Basic functions such as scaling and sorting, subsetting,
mathematical operations and statistical functions.

Depends R (>= 2.10), timeDate (>= 2150.95), methods

Imports graphics, grDevices, stats, utils

Suggests RUnit, robustbase, xts, zoo, PerformanceAnalytics, fTrading

LazyData yes

License GPL (>= 2)

URL https://r-forge.r-project.org/scm/viewvc.php/pkg/timeSeries/?root=rmetrics
(devel), https://www.rmetrics.org,
https://geobosh.github.io/timeSeriesDoc/ (doc)

BugReports https://r-forge.r-project.org/projects/rmetrics

NeedsCompilation no

Author Diethelm Wuertz [aut] (original code),
Tobias Setz [aut],
Yohan Chalabi [aut],
Martin Maechler [ctb] (<https://orcid.org/0000-0002-8685-9910>),
Georgi N. Boshnakov [cre, aut]

Maintainer Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

Repository CRAN

Date/Publication 2023-08-26 09:50:02 UTC

R topics documented:

`timeSeries-package` *Utilities and tools package*

Description

Package **timeSeries** is part of the Rmetrics suit of R packages. It provides a class, `timeSeries`, particularly aimed at analysis of financial data, along with many methods, functions, and utilities for statistical and financial computations on time series.

Details

The following sections have not been updated for some time.

timeSeries - S4 'timeSeries' Class

`timeSeries`	Creates a `"timeSeries"` from scratch
`series`, coredata	Extracts the data
`getUnits`	Extracts the time serie units
`time`	Extracts the positions of timestamps
`x@format`	Extracts the format of the timestamp
`finCenter`	Extracts the financial center
`x@recordIDs`	Extracts the record IDs
`x@title`	Extracts the title
`x@documentation`	Extracts the documentation

Base Time Series Functions

`apply`	Applies a function to blocks of a `"timeSeries"`
`cbind`	Combines columns of two `"timeSeries"` objects
`rbind`	Combines rows of two `"timeSeries"` objects
`diff`	Returns differences of a `"timeSeries"` object
`dim`	returns dimensions of a `"timeSeries"` object

merge	Merges two "timeSeries" objects
rank	Returns sample ranks of a "timeSeries" object
rev	Reverts a "timeSeries" object
sample	Resamples a "timeSeries" object
scale	Scales a "timeSeries" object
sort	Sorts a "timeSeries" object
start	Returns start date/time of a "timeSeries"
end	Returns end date/time of a "timeSeries"
t	Returns the transpose of a "timeSeries" object
attach	Attaches a "timeSeries" to the search path

Subsetting 'timeSeries' Objects

[	Subsets a "timeSeries" object
[<-	Assigns values to a subset
$	Subsets a "timeSeries" by column names
$<-	Replaces subset by column names
head	Returns the head of a "timeSeries"
tail	Returns the tail of a time Series
na.omit	Handles NAs in a "timeSeries" object
removeNA	removes NAs from a matrix object
substituteNA	substitutes NAs by zero, column mean or median
interpNA	interpolates NAs using R's "approx" function

Mathematical Operation

Ops	S4: Arith method for a "timeSeries" object
Math	S4: Math method for a "timeSeries" object
Math2	S4: Maths method for a "timeSeries" object
abs	Returns absolute values of a "timeSeries" object
sqrt	Returns square root of a "timeSeries" object
exp	Returns the exponential values of a "timeSeries" object
log	Returns the logarithm of a "timeSeries" object
sign	Returns the signs of a "timeSeries" object
diff	Differences a "timeSeries" object
scale	Centers and/or scales a "timeSeries" object
quantile	Returns quantiles of an univariate "timeSeries"

Methods

as.timeSeries	Defines method for a "timeSeries"

as.*.default	Returns the input
as.*.ts	Transforma a 'ts' object into a "timeSeries"
as.*.data.frame	Transforms a 'data.frame' intp a 'timeSeries
as.*.character	Loads and transforms from a demo file
as.*.zoo	Transforms a 'zoo' object into a "timeSeries"
as.vector.*	Converts univariate "timeSeries" to vector
as.matrix.*	Converts "timeSeries" to matrix
as.numeric.*	Converts "timeSeries" to numeric
as.data.frame.*	Converts "timeSeries" to data.frame
as.ts.*	Converts "timeSeries" to ts
as.logical.*	Converts "timeSeries" to logical
is.timeSeries	Tests for a "timeSeries" object
plot	Displays a X-Y "timeSeries" Plot
lines	Adds connected line segments to a plot
points	Adds Points to a plot
show	Prints a 'timeSeries oobject

Financial time series functions

align	Aligns a "timeSeries" to time stamps
cumulated	Computes cumulated series from a returns
alignDailySeries	Aligns a "timeSeries" to calendarical dates
rollDailySeries	Rolls a 'timeSeries daily
drawdowns	Computes series of drawdowns from financial returns
drawdownsStats	Computes drawdowns statistics
durations	Computes durations from a financial time series
countMonthlyRecords	Counts monthly records in a "timeSeries"
rollMonthlyWindows	Rolls Monthly windows
rollMonthlySeries	Rolls a "timeSeries" monthly
endOfPeriodSeries	Returns end of periodical series
endOfPeriodStats	Returns end of period statistics
endOfPeriodBenchmarks	Returns period benchmarks
returns	Computes returns from prices or indexes
returns0	Computes untrimmed returns from prices or indexes
runlengths	Computes run lenghts of a "timeSeries"
smoothLowess	Smoothes a "timeSeries"
smoothSpline	Smoothes a "timeSeries"
smoothSupsmu	Smoothes a "timeSeries"
splits	Detects "timeSeries" splits by outlier detection
spreads	Computes spreads from a price/index stream
turns	Computes turning points in a "timeSeries" object
turnsStats	Computes turning points statistics

Statistics Time Series functions

colCumsums	Computes cumulated column sums of a "timeSeries"
colCummaxs	Computes cumulated maximum of a "timeSeries"
colCummins	Computes cumulated minimum of a "timeSeries"
colCumprods	Computes cumulated pruduct values by column
colCumreturns	Computes cumulated returns by column
colSums	Computes sums of all values in each column
colMeans	Computes means of all values in each column
colSds	Computes standard deviations of all values in each column
colVars	Computes variances of all values in each column
colSkewness	Computes skewness of all values in each column
colKurtosis	Computes kurtosis of all values in each column
colMaxs	Computes maxima of all values in each column
colMins	Computes minima of all values in each column
colProds	Computes products of all values in each column
colStats	Computes statistics of all values in each column
orderColnames	Returns ordered column names of a "timeSeries"
sortColnames	Returns alphabetically sorted column names
sampleColnames	Returns sampled column names of a "timeSeries"
pcaColnames	Returns PCA correlation ordered column names
hclustColnames	Returns hierarchically clustered columnames
statsColnames	Returns statisticall rearrange columnames
orderStatistics	Computes order statistics of a "timeSeries" object
rollMean	Computes rolling means of a "timeSeries" object
rollMin	Computes rolling minima of a "timeSeries" object
rollMax	Computes rolling maxima of a "timeSeries" object
rollMedian	Computes rolling medians of a "timeSeries" object
rollStats	Computes rolling statistics of a "timeSeries" objectcr
rowCumsums	Computes cumulated column sums of a "timeSeries"
smoothLowess	Smoothes a series with lowess function
smoothSupsmu	Smoothes a series with supsmu function
smoothSpline	Smoothes a series with smooth.spline function

Misc Functions

dummyDailySeries	Creates a dummy daily "timeSeries" object
isMonthly	Decides if the series consists of monthly records
isDaily	Decides if the series consists of daily records
isQuarterly	Decides if the series consists of Quarterly records
description	Creates default description string

Author(s)

Diethelm Wuertz [aut] (original code), Tobias Setz [aut], Yohan Chalabi [aut], Martin Maechler [ctb] (<https://orcid.org/0000-0002-8685-9910>), Georgi N. Boshnakov [cre, aut]

Maintainer: Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

`aggregate-methods` *Aggregate time series*

Description

Aggregate a `"timeSeries"` object over general periods.

Usage

```
## S4 method for signature 'timeSeries'
aggregate(x, by, FUN, ...)
```

Arguments

`x`	an object of class `"timeSeries"`.
`by`	a sequence of `"timeDate"` objects denoting the aggregation periods, see section 'Details'.
`FUN`	the function to be applied.
`...`	arguments passed to other methods.

Details

`aggregate` aggregates `x` by applying `FUN` on the values of the time series in each of the aggregation periods, specified by argument `by`.

Argument `by` should be of the same class as `time(x)`. `by` is sorted and duplicated values are removed from it. Each pair of consecutive values in `by` then determines a period over which to apply the aggregation function `FUN`, see `findInterval`.

Value

an object of class `"timeSeries"`

See Also

`apply`, `align`

Examples

```
## Load Microsoft Data Set -
   x <- MSFT

## Aggregate by Weeks -
   by <- timeSequence(from = start(x),  to = end(x), by = "week")
   aggregate(x, by, mean)

## Aggregate to Last Friday of Month -
   by <- unique(timeLastNdayInMonth(time(x), 5))
   X <- aggregate(x, by, mean)
   X
   dayOfWeek(time(X))
   isMonthly(X)

## Aggregate to Last Day of Quarter -
   by <- unique(timeLastDayInQuarter(time(x)))
   X <-  aggregate(x, by, mean)
   X
   isQuarterly(X)
```

---

align-methods *Align a 'timeSeries' object to equidistant time stamps*

Description

Aligns a "timeSeries" object to equidistant time stamps. There are also functions for the common cases of changing daily to weekly and daily to monthly.

Usage

```
## S4 method for signature 'timeSeries'
align(x, by = "1d", offset = "0s",
    method = c("before", "after", "interp", "fillNA",
               "fmm", "periodic", "natural", "monoH.FC"),
    include.weekends = FALSE, ...)

alignDailySeries(x, method = c("before", "after", "interp", "fillNA",
    "fmm", "periodic", "natural", "monoH.FC"),
    include.weekends = FALSE, units = NULL, zone = "",
    FinCenter = "", ...)

daily2monthly(x, init = FALSE)
daily2weekly(x, startOn = "Tue", init = FALSE)
```

Arguments

x	an object of class "timeSeries".
by	a character string denoting the period.
offset	a character string denoting the offset.
method	the method to be used for the alignment. A character string, one of "before", use the data from the row whose position is just before the unmatched position, or "after", use the data from the row whose position is just after the unmatched position, or "linear", interpolate linearly between "before" and "after".
include.weekends	a logical value. Should weekend dates be included or removed from the series?
units	an optional character string, which allows to overwrite the current column names of a timeSeries object. By default NULL which means that the column names are selected automatically.
zone	the time zone or financial center where the data were recorded.
FinCenter	a character with the the location of the financial center named as "continent/city".
startOn	a character string, specifying the day of week as a three letter abbreviation. Weekly aggregated data records are then fixed to the weekdays given by the argument startOn.
init	a logical value, if set to TRUE then the time series will be indexed to 1 for its first value. By default init is set to FALSE.
...	further arguments to be passed to the interpolating function.

Details

TODO: complete.

alignDailySeries aligns a daily 'timeSeries' to new positions, Effectively, it is a frontend to the "timeSeries" method for align with by = "1d", and offset = "0s".

In addition, there are two tailored functions for common cases: daily2monthly and daily2weekly which aggregate "timeSeries" objects from daily to monthly or weekly levels, respectively.

In the case of the function daily2weekly one can explicitly set the starting day of the week, the default value is Tuesday, startOn = "Tue".

Value

a "timeSeries" object,

for alignDailySeries, a weekly aligned daily "timeSeries" object from a daily time series with missing holidays.

See Also

aggregate, apply

Examples

```
## Use Microsofts' OHLCV Price Series -
   head(MSFT)
   end(MSFT)

## Use MSFT and Compute Sample Size -
   dim(MSFT)

## Align the Series -
   MSFT.AL <- align(MSFT)

## Show the Size of the Aligned Series -
   dim(MSFT.AL)


## alignDailySeries

## Cut out April Data from 2001 -
   Close <- MSFT[, "Close"]
   tsApril01 <- window(Close, start="2001-04-01", end="2001-04-30")
   tsApril01

## Align Daily Series with NA -
   tsRet <- returns(tsApril01, trim = TRUE)
   GoodFriday(2001)
   EasterMonday(2001)
   alignDailySeries(tsRet, method = "fillNA", include.weekends = FALSE)
   alignDailySeries(tsRet, method = "fillNA", include.weekends = TRUE)

## Align Daily Series by Interpolated Values -
   alignDailySeries(tsRet, method = "interp", include.weekend = FALSE)
   alignDailySeries(tsRet, method = "interp", include.weekend = TRUE)


## Load Microsoft Data Set -
   x <- MSFT

## Aggregate daily records to end of month records -
   X <- daily2monthly(x)
   X
   isMonthly(X)

## Aggregate daily records to end of week records -
   X <- daily2weekly(x, startOn="Fri")
   X
   dayOfWeek(time(X))
```

---

apply *Apply functions over time windows*

Description

Applies a function to a "timeSeries" object over regular or irregular time windows, possibly overlapping.

Usage

```
## S4 method for signature 'timeSeries'
apply(X, MARGIN, FUN, ..., simplify = TRUE)

fapply(x, from, to, FUN, ...)

applySeries(x, from = NULL, to = NULL, by = c("monthly", "quarterly"),
            FUN = colMeans, units = NULL, format = x@format,
            zone = x@FinCenter, FinCenter = x@FinCenter,
            recordIDs = data.frame(), title = x@title,
            documentation = x@documentation, ...)

rollDailySeries(x, period = "7d", FUN, ...)
```

Arguments

x,X	an object of class timeSeries.
MARGIN	a vector giving the subscripts which the function will be applied over, see base R's apply.
FUN	the function to be applied. For the function applySeries the default setting is FUN = colMeans.
simplify	simplify the result?
from, to	starting date and end date as "timeDate" objects. Note, to must be time ordered after from. If from and to are missing in function fapply they are set by default to from=start(x), and to=end(x).
by	a character value either "monthly" or "quarterly" used in the function applySeries. The default value is "monthly". Only operative when both arguments from and to have their default values NULL. In this case the function FUN will be applied to monthly or quarterly periods.
units	an optional character string, which allows to overwrite the current column names of a timeSeries object. By default NULL which means that the column names are selected automatically.
format	the format specification of the input character vector in POSIX notation.
zone	the time zone or financial center where the data were recorded.
FinCenter	a character value with the the location of the financial center named as "continent/city", or "city".
recordIDs	a data frame which can be used for record identification information. Note, this is not yet handled by the apply functions, an empty data.frame will be returned.
title	an optional title string, if not specified the input's data name is deparsed.
documentation	optional documentation string, or a vector of character strings.

period	a character string specifying the rollling period composed by the length of the period and its unit, e.g. "7d" represents one week.
...	arguments passed to other methods.

Details

The "timeSeries" method for apply extracts the core data (a matrix) from X and calls apply, passing on all the remaining arguments. If the result is suitable, it converts it to "timeSeries", otherwise returns it as is. 'Suitable' here means that it is a matrix or a vector (which is converted to a matrix) and the number of observations is the same as X.

Like apply applies a function to the margins of an array, the function fapply applies a function to the time stamps or signal counts of a financial (therefore the "f" in front of the function name) time series of class "timeSeries".

applySeries takes a "timeSeries" object as input and applies FUN to windows of x. The windows are specified by from and to, which need to have the same length. Then from[i], to[i] specifies the i-th window. If time(x) is a "timeDate" object, then from and to are converted to "timeDate" (if they are not already such objects), otherwise they are converted to integers.

An alternative way to specify the window(s) on which applySeries operates is with argument by. It is used only if from and to are missing or NULL. by = "monthly" or by = "quarterly" applies FUN to the data for each year-month or year-quarter, respectively. By year-month we mean that there are separate windows for the months in different years.

The resulting time stamps are the time stamps of the to vector. The periods can be regular or irregular, and they can even overlap.

If from = start(x) and to = end(x), then the function behaves like apply on the column margin.

fapply is the same as applySeries (in fact, the former calls the latter), except that the defaults for from and to are start(x) and end(x), respectively. (GNB: in addition, fapply throws error if x is a 'signal series'.)

rollDailySeries rolls a daily 'timeSeries' on a given period.

Value

for rollDailySeries, an object of class "timeSeries" with rolling values, computed from the function FUN.

Examples

```
## Percentual Returns of Swiss Bond Index and Performance Index -
   LPP <- 100 * LPP2005REC[, c("SBI", "SPI")]
   head(LPP, 20)

## Aggregate Quarterly Returns -
   applySeries(LPP, by = "quarterly", FUN = colSums)

## Aggregate Quarterly every last Friday in Quarter -
   oneDay <- 24*3600
   from <- unique(timeFirstDayInQuarter(time(LPP))) - oneDay
   from <- timeLastNdayInMonth(from, nday = 5)
   to <- unique(timeLastDayInQuarter(time(LPP)))
```

```
   to <- timeLastNdayInMonth(to, nday = 5)
   data.frame(from = as.character(from), to = as.character(to))

   applySeries(LPP, from, to, FUN = colSums)
## Alternative Use -
   fapply(LPP, from, to, FUN = colSums)

## Count Trading Days per Month -
   colCounts <- function(x) rep(NROW(x), times = NCOL(x))
   applySeries(LPP, FUN = colCounts, by = "monthly")


## TODO: examples for rollDailySeries()
```

---

as *Convert objects to/from class 'timeSeries'*

Description

Functions and methods dealing with the coercion between "timeSeries" and other classes.

Usage

```
## convert to 'timeSeries'
as.timeSeries(x, ...)

## convert from 'timeSeries' to other classes
## S4 method for signature 'timeSeries'
as.matrix(x, ...)
## S4 method for signature 'timeSeries'
as.ts(x, ...)
## S4 method for signature 'timeSeries'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
## S4 method for signature 'timeSeries'
as.list(x, ...)
```

Arguments

x	the object to be converted, see Section 'Details' for the special case when class(x) is "character".
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	a logical value. If TRUE, setting row names and converting column names (to syntactic names) is optional.
...	arguments passed to other methods.

Details

Functions to create "timeSeries" objects from other objects and to convert "timeSeries" objects to other classes.

as.timeSeries is a generic function to convert an object to "timeSeries". There are specialised methods for the following classes: "ts", "data.frame", "character", and "zoo". The default method is equivalent to calling "timeSeries()", so x can be of any type that "timeSeries()" accepts.

The character method of as.timeSeries is special, in that its contents are parsed and evaluated, then as.timeSeries is called on the returned value (passing also the "..." arguments. Care is needed to avoid infinite recursion here since currently the code doesn't guard against it.

Value

for as.timeSeries, an object of class "timeSeries".

for as.numeric, as.data.frame, as.matrix, as.ts, as.list - a numeric vector, a data frame, a matrix, an object of class ts, or a "list", respectively.

See Also

timeSeries, class timeSeries

Examples

```
## Create an Artificial 'timeSeries' Object
setRmetricsOptions(myFinCenter = "GMT")
charvec <- timeCalendar()
data <- matrix(rnorm(12))
TS <- timeSeries(data, charvec, units = "RAND")
TS

## Coerce to Vector
as.vector(TS)

## Coerce to Matrix
as.matrix(TS)

## Coerce to Data Frame
as.data.frame(TS)
```

---

attach *Attach a 'timeSeries' to the search path*

Description

Attaches a "timeSeries" object to the search path.

Usage

```
## S4 method for signature 'timeSeries'
attach(what, pos = 2, name = deparse(substitute(what)),
    warn.conflicts = TRUE)
```

Arguments

name	alternative way to specify the database to be attached. See for details help(attach, package = base).
pos	an integer specifying position in search() where to attach the database. See for details help(attach, package = base).
warn.conflicts	a logical value. If TRUE, warnings are printed about conflicts from attaching the database, unless that database contains an object .conflicts.OK. A conflict is a function masking a function, or a non-function masking a non-function. See for details help(attach, package = base).
what	database to be attached. This may currently be a "timeSeries" object, a data.frame, a list, an R data file created with save, NULL, or an environment. See for details help(attach, package = base).

Value

the environment, invisibly, with a name attribute

Note

The function detach from the base package can be used to detach the attached objects.

Examples

```
## Load Microsoft Data Set -
   x <- MSFT[1:10, ]
   colnames(x)

## Attach the Series and Compute the Range -
   attach(x)
   range <- High - Low
   range

## Convert Vector to a \code{"timeSeries"} Object -
   timeSeries(data=range, charvec=time(x), units="Range")

## Detach the series from the search path -
   detach("x")
   ans <- try(High, silent=TRUE)
   cat(ans[1])
```

attributes *Get and set optional attributes of a 'timeSeries'*

Description

Extracts or assigns optional attributes from or to a "timeSeries" object.

Usage

```
getAttributes(obj)
setAttributes(obj) <- value
```

Arguments

obj	a timeSeries object whose optional attributes are to be accessed.
value	an object, the new value of the attribute, or NULL to remove the attribute.

Details

Each timeSeries object is documented. By default a time series object holds in the documentation slot a string with creation time and the user who has defined it. But this is not all. Optionally the whole creation process and history can be recorded. For this the @documentation slot may have an optional "Attributes" element. This attribute is tracked over the whole life time of the object whenever the time series is changed. Whenever you like to be informed about the optional attributes, or you like to recover them you can dot it, and evenmore, whenever you like to add information as an addiitonal attribute you can also do it.

The two functions getAttributes and setAttributes provide access to and allow to modify the optional attributes of a timeSeries object.

Examples

```
## Create an artificial 'timeSeries' Object -
   tS <- dummyMonthlySeries()
   tS

## Get Optional Attributes -
   getAttributes(tS)
   tS@documentation

## Set a new Optional Attribute -
   setAttributes(tS) <- list(what="A dummy Series")
   tS
   getAttributes(tS)
   tS@documentation
```

cbind *Bind 'timeSeries' objects by column or row*

Description

Binds "timeSeries" objects either by column or by row.

Usage

```
## S3 method for class 'timeSeries'
cbind(..., deparse.level = 1)
## S3 method for class 'timeSeries'
rbind(..., deparse.level = 1)

## S4 method for signature 'timeSeries,ANY'
cbind2(x, y)
## other methods for 'cbind2' with the same arguments, see Details

## S4 method for signature 'timeSeries,ANY'
rbind2(x, y)
## other methods for 'rbind2' with the same arguments, see Details
```

Arguments

x, y	objects, at least one of whom is of class "timeSeries".
...	further arguments to bind.
deparse.level	see the documentation of base::cbind.

Details

These functions bind the objects by row rXXX or column (cXXX.

cbind and rbind are S3 generics, so the "timeSeries" methods describe here are called only when the first argument is "timeSeries".

cbind2 and rbind2 are S4 generics which dispatch on the first two arguments. The "timeSeries" methods for these are invoked whenever at least one of the first two arguments is of class "timeSeries".

All functions can be called with more than two arguments. After the first two are merged, the result is merged with the third, and so on.

Value

an object of class "timeSeries"

See Also

merge for another way to merge "timeSeries" object column-wise.

rbind and cbind from base R,

rbind2 and cbind2 from package "methods",

Examples

```
## Load Microsoft Data Set -
   x <- MSFT[1:12, ]
   x

## Bind Columnwise -
   X <- cbind(x[, "Open"], returns(x[, "Open"]))
   colnames(X) <- c("Open", "Return")
   X

## Bind Rowwise -
   Y <- rbind(x[1:3, "Open"], x[10:12, "Open"])
   Y
```

---

colCum *Cumulated column statistics*

Description

Functions to compute cumulative column statistics.

Usage

```
## S4 method for signature 'timeSeries'
colCumsums(x, na.rm = FALSE, ...)

## S4 method for signature 'timeSeries'
colCummaxs(x, na.rm = FALSE, ...)

## S4 method for signature 'timeSeries'
colCummins(x, na.rm = FALSE, ...)

## S4 method for signature 'timeSeries'
colCumprods(x, na.rm = FALSE, ...)

## S4 method for signature 'timeSeries'
colCumreturns(x, method = c("geometric", "simple"),
                                    na.rm = FALSE, ...)
```

Arguments

x	a time series, may be an object of class "matrix", or "timeSeries".
na.rm	a logical. Should missing values be removed?
method	a character string to indicate if geometric (TRUE) or simple (FALSE) returns should be computed.
...	arguments to be passed.

Details

These functions compute the requested cumulative quantities columnwise to obtain a matrix of the same dimension as the data. The "timeSeries" methods replace the data part of the original object with the resulting matrix.

The "timeSeries" methods for the Math group functions cummin, cummax, cumsum, and cumprod, work similarly but don't have the na.rm argument.

Value

"matrix" for the default methods of all functions,

"timeSeries" for the "timeSeries" methods

See Also

Math,timeSeries-method, rowCumsums

Examples

```
## simulate return data
x <- matrix(rnorm(24), ncol = 2)
X <- as.timeSeries(x)

## cumulative sums  by column  -
class(colCumsums(x))  # "matrix"
class(colCumsums(X))  # "timeSeries"

colCumsums(X)
```

---

colStats *Column statistics*

Description

A collection of functions to compute column statistical properties of financial and economic time series data.

Usage

```
colStats(x, FUN, ...)

colSds(x, ...)
colVars(x, ...)
colSkewness(x, ...)
colKurtosis(x, ...)
colMaxs(x, ...)
colMins(x, ...)
colProds(x, ...)
colQuantiles(x, prob = 0.05, ...)
```

Arguments

x	a rectangular object which can be transformed into a matrix by the function as.matrix.
FUN	a function name, the statistical function to be applied.
prob	a numeric value in [0,1].
...	arguments to be passed.

Details

colStats	calculates column statistics,
colSums	calculates column sums,
colMeans	calculates column means,
colSds	calculates column standard deviations,
colVars	calculates column variances,
colSkewness	calculates column skewness,
colKurtosis	calculates column kurtosis,
colMaxs	calculates maximum values in each column,
colMins	calculates minimum values in each column,
colProds	computes product of all values in each column,
colQuantiles	computes quantiles of each column.

Value

each function returns a numeric vector of the statistics, one for each column

See Also

rollStats

Examples

```
## Simulated Return Data in Matrix Form -
   x = matrix(rnorm(252), ncol = 2)

## Mean Columnwise Statistics -
   colStats(x, FUN = mean)

## Quantiles Column by Column -
   colQuantiles(x, prob = 0.10, type = 1)
```

comment *Get and set comments for 'timeSeries' objects*

Description

Get or assign new comment to a `timeSeries` object.

Usage

```
## S4 method for signature 'timeSeries'
comment(x)
## S4 replacement method for signature 'timeSeries'
comment(x) <- value
```

Arguments

`x`	a `timeSeries` object.
`value`	a character vector, the comment.

Details

Objects from class `"timeSeries"` have a slot for documentation. These functions get and change its contents.

Examples

```
## Get description from a 'timeSeries' -
   comment(LPP2005REC)

## Add User to comment -
   comment(LPP2005REC) <- paste(comment(LPP2005REC), "by User Rmetrics")
   comment(LPP2005REC)
```

cumulated *Cumulated time series from returns*

Description

Computes a cumulated financial `"timeSeries"`, e.g. prices or indexes, from financial returns.

Usage

```
cumulated(x, ...)

## Default S3 method:
cumulated(x, method = c("continuous", "discrete",
    "compound", "simple"), percentage = FALSE, ...)
```

Arguments

x	an object of class timeSeries.
method	a character string, the method for computation of returns.
percentage	a logical value. By default FALSE, if TRUE the series will be expressed in percentage changes.
...	ignored by the default method.

Details

Note, the function cumulated assumes as input discrete returns from a price or index series. Only then the cumulated series agrees with the original price or index series. The first values of the cumulated series cannot be computed, it is assumed that the series is indexed to 1.

Value

a "timeSeries" object

See Also

returns,

drawdowns, splits,

midquotes, index2wealth

Examples

```
## Use the Microsofts' Close Prices Indexed to 1 -
   MSFT.CL <- MSFT[, "Close"]
   MSFT.CL <- MSFT.CL/MSFT[[1, "Close"]]
   head(MSFT.CL)

## Compute Discrete Return -
   MSFT.RET <- returns(MSFT.CL, method = "discrete")

## Cumulated Series and Compare -
   MSFT.CUM <- cumulated(MSFT.RET, method = "discrete")
   head(cbind(MSFT.CL, MSFT.CUM))
```

DataPart,timeSeries-method

DataPart,timeSeries-method

Description

Utilities called to implement object@.Data of timeSeries objects.

Examples

```
## Load Microsoft Data -
   X <- MSFT[1:10, 1:4]

## Get Data Part -
   DATA <- getDataPart(X)
   class(DATA)
```

---

description *Creates date and user information*

Description

Creates and returns a string containing the user, the current datetime and the user name.

Usage

```
description()
```

Examples

```
## Show Default Description String -
   description()
```

---

diff *Difference a 'timeSeries' object*

Description

Difference a "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
diff(x, lag = 1, diff = 1, trim = FALSE, pad = NA, ...)
```

Arguments

x	an object of class "timeSeries".
lag	an integer indicating which lag to use.
diff	an integer indicating the order of the difference.
trim	a logical flag. Should NAs at the beginning of the series be removed?
pad	a numeric value with which NAs should be replaced at the beginning of the series.
...	currently not used.

Value

the differenced "timeSeries" object

See Also

diff for base::diff, lag

Examples

```
## Load Microsoft Data Set -
   x <- MSFT[1:12, ]
   x

## Compute Differences -
   diff(x)

## Trimmed Differences -
   diff(x, trim = TRUE)

## Padded Differences -
   diff(x, trim = FALSE, pad = 0)
```

---

dimnames *Dimension and their names for 'timeSeries' objects*

Description

Get and assign names, row names, column names, and dim names of "timeSeries" objects.

Details

"timeSeries" methods are available for base R functions working on dimension names, including dim, dimnames, colnames, rownames, names and their assignment variants.

dim is the dimension of the underlying data matrix.

rownames gives the datetime stamps as a character vector. rownames<- sets them.

colnames gives the values of x@units. These are conceptually the column names of the data matrix. colnames<- sets slot units of x.

dimnames gives list(rownames(x), colnames(x). dimnames<- calls rownames and colnames on value[[1]] and value[[2]], respectively.

Note

(GNB; todo) The "dim<-", currently converts x to a vector if value is NULL, otherwise it ignores value, does nothing and returns x unchanged. This behaviour should not be relied upon and may be changed in the future, e.g. by issuing warning when value is not NULL. Or throwing error altogether if assignment with "dim<-" is attempted.

Examples

```
## Load Swiss Pension Fund Benchmark Data -
   X <- LPP2005REC[1:10, 1:3]

## Get Dimension -
   dim(X)

## Get Column and Row Names -
   dimnames(X)

## Get Column / Row Names -
   colnames(X)
   rownames(X)

## Try your own DIM -
   DIM <- function(x) {c(NROW(x), NCOL(x))}
   DIM(X)
   DIM(X[, 1])

## Try length / LENGTH -
   length(X)
   length(X[, 1])
   LENGTH <- function(X) NROW(X)
   LENGTH(X)

## Columns / Rows -
   ncol(X); NCOL(X)
   nrow(X); NROW(X)

## See also -
   isUnivariate(X)
   isMultivariate(X)
```

---

drawdowns *Calculations of drawdowns*

Description

Compute series of drawdowns from financial returns and calculate drawdown statisitcs.

Usage

```
drawdowns(x, ...)

drawdownsStats(x, ...)
```

Arguments

x	a "timeSeries" object of financial returns. Note, drawdowns can be calculated from an uni- or multivariate time deries object, statistics can only be computed from an univariate time series object.
...	optional arguments passed to na.omit.

Details

The code in the core of the function drawdownsStats was borrowed from the package PerformanceAnalytics authored by Peter Carl and Sankalp Upadhyay.

Value

for drawdowns, an object of class timeSeries.

for drawdownsStats an object of class "data.frame" with the following components:

drawdown	the depth of the drawdown,
from	the start date,
trough	the trough period,
to	the end date,
length	the length in number of records,
peaktrough	the peak trough, and
recovery	the recovery length in number of records.

Author(s)

Peter Carl and Sankalp Upadhyay for code from the contributed R package PerformanceAnalytics used in the function drawdownsStats.

See Also

returns, cumulated,

splits,

midquotes, index2wealth

Examples

```
## Use Swiss Pension Fund Data Set of Returns -
   head(LPP2005REC)
   SPI <- LPP2005REC[, "SPI"]
   head(SPI)

## Plot Drawdowns -
   dd = drawdowns(LPP2005REC[, "SPI"], main = "Drawdowns")
   plot(dd)
   dd = drawdowns(LPP2005REC[, 1:6], main = "Drawdowns")
   plot(dd)
```

```
## Compute Drawdowns Statistics -
   ddStats <- drawdownsStats(SPI)
   class(ddStats)
   ddStats

## Note, Only Univariate Series are allowd -
   ddStats <- try(drawdownsStats(LPP2005REC))
   class(ddStats)
```

---

dummyTimeSeries *Create dummy time series*

Description

Create dummy daily and monthly time series for examples and exploration.

Usage

```
dummyDailySeries(x = rnorm(365), units = NULL, zone = "",
                 FinCenter = "")

dummyMonthlySeries(...)
```

Arguments

x	an object of class timeSeries.
units	an optional character string, which allows to overwrite the current column names of a timeSeries object. By default NULL which means that the column names are selected automatically.
FinCenter	a character with the the location of the financial center named as "continent/city".
zone	the time zone or financial center where the data were recorded.
...	optional arguments passed to timeSeries.

Details

dummyDailySeries creates a timeSeries object with dummy daily dates from a numeric matrix with daily records of unknown dates.

dummyMonthlySeries creates a dummy monthly "timeSeries" object.

Value

a "timeSeries" object

Examples

```
dd <- dummyDailySeries()
head(dd)
tail(dd)

dummyMonthlySeries(y = 2022)
```

`durations` *Durations from a 'timeSeries'*

Description

Computes durations from an object of class `"timeSeries"`.

Usage

```
durations(x, trim = FALSE, units = c("secs", "mins", "hours", "days"))
```

Arguments

`x`	an object of class `"timeSeries"`.
`trim`	a logical value. By default `TRUE`, the first missing observation in the return series will be removed.
`units`	a character value or vector which allows to set the units in which the durations are measured. By default durations are measured in seconds.

Details

Durations measure how long it takes until we get the next record in a `timesSeries` object. We return a time series in which for each time stamp we get the length of the period from when we got the last record. This period is measured in length specified by the argument `units`, for daily data use `units="days"`.

Value

an object of class `"timeSeries"`

Examples

```
## Compute Durations in days for the MSFT Sereries -
   head(durations(MSFT, units = "days"))
   head(durations(MSFT, trim = TRUE, units = "days"))

## The same in hours -
   head(durations(MSFT, trim = TRUE, units = "hours"))
```

filter *Linear filtering on a time series*

Description

Applies linear filtering to a univariate "timeSeries".

Usage

```
## S4 method for signature 'timeSeries'
filter(x, filter, method = c("convolution", "recursive"), sides = 2,
       circular = FALSE, init = NULL)
```

Arguments

x	an object from class "timeSeries".
filter	coefficients of the filter.
method	"convolution" or "recursive".
sides,circular	for convolution filters only. Onesided if sides = 1, centred around lag 0 if sides = 2. Circular if circular = TRUE.
init	for recursive filters only. Values before the start of the time series.

Details

filter is a generic function with default method stats::filter. The method for "timeSeries" is a wrapper for the latter.

See ?stats::filter for details about the arguments.

Value

a "timeSeries" object

See Also

base R function filter

Examples

```
## Creata a dummy signal 'timeSeries' -
   data <- matrix(rnorm(100), ncol = 2)
   s <- timeSeries(data, units=c("A", "B"))
   head(s)

## Filter the series -
   f <- filter(s, rep(1, 3))
   head(f)
```

```
## Plot and compare the first series -
   plot(cbind(s[, 1], f[, 1]), plot.type="s")
```

`finCenter` *Get and set Financial center of a 'timeSeries'*

Description

Get or assign a financial center to a "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
finCenter(x)
## S4 replacement method for signature 'timeSeries'
finCenter(x) <- value

getFinCenter(x)
setFinCenter(x) <- value
```

Arguments

x	a "timeSeries" object.
value	a character with the the location of the financial center named as "continent/city".

See Also

listFinCenter and finCenter in package "timeDate"

Examples

```
## An artificial 'timeSeries' Object -
   tS <- dummyMonthlySeries()
   tS

## Print Financial Center -
   finCenter(tS)
   getFinCenter(tS)

## Assign New Financial Center -
   finCenter(tS) <- "Zurich"
   tS
   setFinCenter(tS) <- "New_York"
   tS
```

is.timeSeries *Check if an object is from class 'timeSeries'*

Description

is.timeSeries tests if its argument is a timeSeries. is.signalSeries tests if series has no timestamps.

Usage

```
is.timeSeries(x)
is.signalSeries(x)
```

Arguments

x	an object.

Value

a logical value, TRUE or FALSE.

Examples

```
## Create an artificial 'timeSeries' object -
   setRmetricsOptions(myFinCenter = "GMT")
   charvec <- timeCalendar()
   data <- matrix(rnorm(12))
   TS <- timeSeries(data, charvec, units = "RAND")
   TS

## Test for 'timeSeries' -
   is.timeSeries(TS)
```

isRegular *Checks if a time series is regular*

Description

Checks if a time series is regular.

Usage

```
## S4 method for signature 'timeSeries'
isDaily(x)
## S4 method for signature 'timeSeries'
isMonthly(x)
## S4 method for signature 'timeSeries'
isQuarterly(x)

## S4 method for signature 'timeSeries'
isRegular(x)

## S4 method for signature 'timeSeries'
frequency(x, ...)
```

Arguments

x	an R object of class 'timeSeries'.
...	arguments to be passed.

Details

What is a regular time series? If a time series is daily, monthly, or weekly, then we speak of a regular series. This can be tested calling the functions `isDaily`, `isMonthly`, `isQuarterly`, or in general `isRegular`. If the series is regular then its frequency can be determined by calling `frequency`.

Here are the definitions of daily, monthly, and quarterly time series:

daily if the series has no more than one date/time stamp per day.

monthly if the series has no more than one date/time stamp per month.

quarterly if the series has no more than one date/time stamp per quarter.

A regular series is either a monthly or a quarterly series.

Note that with the above definitions a monthly series is also a daily series, a quarterly series is also a monthly series. On the other hand, a daily series is not regular!

NOT yet implemented is the case of weekly series.

Value

The `is*` functions return `TRUE` or `FALSE` depending on whether the series fulfills the condition or not.

`frequency` returns in general 1, for quarterly series 4, and for monthly series 12.

Examples

```
data(MSFT)
isRegular(MSFT)

## a monthly ts
```

```
ap <- as.timeSeries(AirPassengers)
isRegular(ap)

## a quarterly ts
pres <- as.timeSeries(presidents)
isRegular(pres)
```

`isUnivariate` *Checks if a time series is univariate*

Description

Checks if a time series object or any other rectangular object is univariate or multivariate.

Usage

```
isUnivariate(x)
isMultivariate(x)
```

Arguments

`x`	an object of class `"timeSeries"` or any other rectangular object.

Details

A rectangular object `x` is considered to be univariate if the function `NCOL(x)` returns one, and is considered to be multivariate if `NCOL(x)` returns a value bigger than one.

Value

a logical value

Examples

```
## Load Microsoft Data -
   setRmetricsOptions(myFinCenter = "GMT")
   data(MSFT)
   Open = MSFT[, "Open"]

## Is the 'timeSeries' Univariate -
   isUnivariate(MSFT)
   isUnivariate(Open)

## Is the 'timeSeries' Multivariate -
   isMultivariate(MSFT)
   isMultivariate(Open)
```

lag *Lag a 'timeSeries' object*

Description

Compute a lagged version of a "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
lag(x, k = 1, trim = FALSE, units = NULL, ...)
```

Arguments

x	an object of class timeSeries.
k	an integer number, the number of lags (in units of observations). By default 1. Can also be a vector, in which case the result is a multivariate "timeSeries" in which column i contains the series lagged by k[i], see the examples.
trim	a logical value. By default TRUE, the first missing observation in the return series will be removed.
units	an optional character string, which allows to overwrite the current column names of a "timeSeries" object. By default NULL which means that the column names are selected automatically.
...	arguments passed to other methods.

Value

an object of class "timeSeries"

See Also

lag for stats::lag, diff

Examples

```
## Load Micsrosoft Data Set -
   x = MSFT[1:20, "Open"]

## Lag the 'timeSeries' Object:
   lag(x, k = -1:1)
```

math *Mathematical operations on 'timeSeries'*

Description

Functions and methods for mathematical operations on "timeSeries".

Usage

```
## S4 method for signature 'timeSeries,timeSeries'
Ops(e1, e2)
## S4 method for signature 'timeSeries'
Math(x)
## S4 method for signature 'timeSeries'
Math2(x, digits)

## S4 method for signature 'timeSeries'
quantile(x, ...)
## S4 method for signature 'timeSeries'
median(x, na.rm = FALSE, ...)
```

Arguments

x	an object of class timeSeries.
digits	number of digits to be used in 'round' or 'signif'.
e1, e2	at least one of the two objects is from class "timeSeries" (for the methods described on this page).
na.rm	a logical value: should missing values be removed?
...	arguments to be passed.

Details

The methods for the Math and Math2 groups of mathematical functions return 'timeSeries' objects. Most of them work element-wise on the data part of the time series with the exception of cummin, cummax, cumsum, and cumprod which work columnwise.

The Ops group includes mathematical operators. For the binary operators methods are defined for pairs of at least one 'timeSeries' object. These work as expected on the data parts of the arguments. If the operation gives a value of the same dimension as the data part of the 'timeSeries' object, it replaces the original data in the object.

There are also methods for quantile and median.

Value

the value from a mathematical or logical operation operating on objects of class "timeSeries" or the value computed by a mathematical function.

See Also

colCumXXX

Examples

```
## Create an Artificial 'timeSeries' Object -
   setRmetricsOptions(myFinCenter = "GMT")
   charvec = timeCalendar()
   set.seed(4711)
   data = matrix(exp(cumsum(rnorm(12, sd = 0.1))))
   TS = timeSeries(data, charvec, units = "TS")
   TS

## Mathematical Operations: | +/- * ^ ... -
   TS^2
   TS[2:4]
   OR = returns(TS)
   OR
   OR > 0
```

---

merge *Merge 'timeSeries' objects*

Description

Merges several object types with "timeSeries" objects. The number of rows must match.

Usage

```
merge(x, y, ...)
```

Arguments

x,y	objects to merge, at least one of class "timeSeries".
...	further objects to merge.

Value

a "timeSeries" object

Methods

```
signature(x = "timeSeries", y = "missing")
signature(x = "timeSeries", y = "ANY")
signature(x = "timeSeries", y = "matrix")
signature(x = "timeSeries", y = "numeric")
signature(x = "timeSeries", y = "timeSeries")
```

```
signature(x = "ANY", y = "ANY")
signature(x = "ANY", y = "timeSeries")
signature(x = "matrix", y = "timeSeries")
signature(x = "numeric", y = "timeSeries")
```

See Also

cbind

Examples

```
## Load Series -
   x <- MSFT[1:12, ]

## Merge 'timeSeries' with missing Object -
   merge(x)


## Merge 'timeSeries' with numeric Object -
   y <- rnorm(12)
   class(y)
   merge(x, y)

## Merge 'timeSeries' with matrix Object -
   y <- matrix(rnorm(24), ncol=2)
   class(y)
   merge(x, y)

## Merge 'timeSeries' with matrix Object -
   y <- timeSeries(data=rnorm(12), charvec=time(x))
   class(y)
   merge(x, y)
```

---

monthly *Special monthly series*

Description

Functions and methods dealing with special monthly "timeSeries" objects.

Usage

```
rollMonthlyWindows(x, period = "12m", by = "1m")

rollMonthlySeries(x, period = "12m", by = "1m", FUN, ...)
countMonthlyRecords(x)
```

Arguments

x	a "timeSeries" object.
period,by	character strings specifying the rollling period composed by the length of the period and its unit. Examples: "3m" "6m", "12m", and "24m" represent quarterly, semi-annual, annual and bi-annual shifts, respectively. It is the responsibility of the user to determine proper start of the series.
FUN	the function for the statistic to be applied. For example, colMean in the case of aggregation.
...	arguments passed to the function FUN.

Details

rollMonthlySeries computes the statistics defined by the function FUN over rolling windows, internally computed by the function rollMonthlyWindows. Note, the periods may be overlapping, may be dense, or even may have gaps.

countMonthlyRecords computes a "timeSeries" that holds the number of records for each month, see examples. The dates are set to the end of the month.

rollMonthlyWindows computes start and end dates for rolling time windows. Argument period specifies the length of the periods over which FUN is applied, while by gives the amount by which the window is shifted. Non-overlapping windows correspond to by >= period.

Value

for countMonthlyRecords and rollMonthlySeries, a "timeSeries" object.

for rollMonthlyWindows, a list with attribute "control" keeping the start and end dates of the series. The components of the list are:

from	an object from class "timeDate".
to	an object from class "timeDate".

Examples

```
## Load Microsoft Daily Data Set
x <- MSFT

## count monthly records
head(x)   # 3 obs. for Sep 2000
counts <- countMonthlyRecords(x)
counts

## diy computation of the counts
diy <- rollMonthlySeries(x[ , 1], period = "1m", by = "1m", FUN = NROW)

## difference is only in some attributes (e.g. column names)
all.equal(diy,  counts)


## quaterly non-overlapping time periods -
```

```
windows <- rollMonthlyWindows(counts[-1, ], period = "3m", by = "3m")
windows
## nicely print results as a data.frame, each row is a time window
data.frame(cbind(FROM = format(windows$from), TO = format(windows$to)))

## Compute the average number of monthly trading days per quarter -
rollMonthlySeries(counts[-1, ], period = "3m", by = "3m", FUN = mean)
```

---

na *Handle missing values in 'timeSeries' objects*

Description

Functions for handling missing values in "timeSeries" objects.

Usage

```
## S4 method for signature 'timeSeries'
na.omit(object, method = c("r", "s", "z", "ir", "iz", "ie"),
    interp = c("before", "linear", "after"), FUN, ...)

## Deprecated:
removeNA(x, ...)
substituteNA(x, type = c("zeros", "mean", "median"), ...)
interpNA(x, method = c("linear", "before", "after"), ...)
```

Arguments

object	an object of class "timeSeries".
method	for na.omit, the method of handling NAs; for interpNA, how to interpolate the matrix column by column, see Section 'Details'.
interp, type	Three alternative methods are provided to remove NAs from the data: type="zeros" replaces the missing values with zeros, type="mean" replaces the missing values with the column mean, type="median" replaces the missing values with the column median.
FUN	a function or a name of a function, such as "mean" or median. FUN is applied to the non-NA values in each column to determine the replacement value. The call looks like FUN(coli, na.rm = TRUE), so FUN should have argument na.rm. All arguments except object are ignored if FUN is specified.
x	a numeric matrix, or any other object which can be transformed into a matrix through x = as.matrix(x, ...). If x is a vector, it will be transformed into a one-dimensional matrix.
...	arguments to be passed to the function as.matrix.

Details

Functions for handling missing values in "timeSeries" objects and in objects which can be transformed into a vector or a two dimensional matrix.

For na.omit argument method specifies the method how to handle NAs. Can be one of the following strings:

method="s" na.rm = FALSE, skip, i.e. do nothing,

method="r" remove NAs,

method="z" substitute NAs by zeros,

method="ir" interpolate NAs and remove NAs at the beginning and end of the series,

method="iz" interpolate NAs and substitute NAs at the beginning and end of the series,

method="ie" interpolate NAs and extrapolate NAs at the beginning and end of theseries.

For interpNA argument method specifies how to interpolate the matrix column by column. One of the following character strings: "linear", "before", "after". For interpolation the function approx is used.

The functions are listed by topic.

na.omit	Handles NAs,
removeNA	Removes NAs from a matrix object,
substituteNA	substitute NAs by zero, the column mean or median,
interpNA	interpolates NAs using R's "approx" function.

Missing Values in Price and Index Series:

Applied to timeSeries objects the function removeNA just removes rows with NAs from the series. For an interpolation of time series points one can use the function interpNA. Three different methods of interpolation are offered: "linear" does a linear interpolation, "before" uses the previous value, and "after" uses the following value. Note, that the interpolation is done on the index scale and not on the time scale.

Missing Values in Return Series:

For return series the function substituteNA may be useful. The function allows to fill missing values either by method="zeros", the method="mean" or the method="median" value of the appropriate columns.

Note

When dealing with daily data sets, there exists another function alignDailySeries which can handle missing data in un-aligned calendrical "timeSeries" objects.

The functions removeNA, substituteNA and interpNA are older implementations. Please use in all cases if possible the new function na.omit.

Additional remarks by GNB:

removeNA(x) is equivalent to na.omit(x) or na.omit(x), methods = "r".

interpNA can be replaced by a call to na.omit with argument method equal to ir, iz, or ie, and argument "interp" equal to the "method" argument for interpNA (note that the defaults are not the same).

substituteNA(x, type = "zeros") is equivalent to na.omit(x, method = "z"). For other values of type one can use argument FUN, as in na.omit(x, FUN = "mean").

A final remark: the three deprecated functions are non-generic. removeNA(x) is completely redundant as it simply calls na.omit. The other two however may be useful for matrix-like objects. Please inform the maintainer of the package if you use them on objects other than from class "timeSeries" and wish them kept in the future.

References

Troyanskaya O., Cantor M., Sherlock G., Brown P., Hastie T., Tibshirani R., Botstein D., Altman R.B., (2001); *Missing Value Estimation Methods for DNA microarrays* Bioinformatics 17, 520–525.

See Also

alignDailySeries

Examples

```
X <- matrix(rnorm(100), ncol = 5)  # Create a Matrix X
X[3, 5] <- NA                      # Replace a Single NA Inside
X[17, 2:4] <- c(NA, NA, NA)        # Replace Three in a Row Inside
X[13:15, 4] <- c(NA, NA, NA)       # Replace Three in a Column Inside
X[11:12, 5] <- c(NA, NA)           # Replace Two at the Right Border
X[20, 1] <- NA                     # Replace One in the Lower Left Corner
X
Xts <- timeSeries(X)  # convert X to timeSeries Xts

## Remove Rows with NAs
na.omit(Xts)

## Subsitute NA's with zeros or column means (formerly substituteNA())
na.omit(Xts, method = "z")
na.omit(Xts, FUN = "mean")
na.omit(Xts, FUN = "median")

## Subsitute NA's with a trimmed mean
na.omit(Xts, FUN = function(x, na.rm) mean(x, trim = 0.10, na.rm = na.rm))

## Interpolate NA's Linearily (formerly interpNA())
na.omit(X, method = "ir", interp = "linear")
na.omit(X, method = "iz", interp = "linear")
na.omit(X, method = "ie", interp = "linear")

## Take Previous Values in a Column
na.omit(X, method = "ir", interp = "before")
na.omit(X, method = "iz", interp = "before")
na.omit(X, method = "ie", interp = "before")
```

```
## examples with X (which is a matrix, not "timeSeries"
## (these examples are not run automatically as these functions are
## deprecated.)
if(FALSE){
## Remove Rows with NAs -
   removeNA(X)

## Subsitute NA's by Zeros or Column Means -
   substituteNA(X, type = "zeros")
   substituteNA(X, type = "mean")

## Interpolate NA's Linearily -
   interpNA(X, method = "linear")
   # Note the corner missing value cannot be interpolated!

## Take Previous Values in a Column -
   interpNA(X, method = "before")
   # Also here, the corner value is excluded
}
```

`na.contiguous` *Find longest contiguous stretch of non-NAs or check for NAs*

Description

Find the longest consecutive stretch of non-missing values in a `"timeSeries"` object. In the event of a tie, the first such stretch. Also, `"timeSeries"` method for `is.na`.

Usage

```
## S4 method for signature 'timeSeries'
na.contiguous(object, ...)
```

Arguments

`object`	a `"timeSeries"` object.
`...`	further arguments passed to other methods.

Value

for the `na.contiguous` method, a `"timeSeries"` object without missing values,

for the `is.na` method, a `"timeSeries"` object whose data part is a logical matrix of the same dimension as in `x` indicating if the corresponding values are `NA` or not.

Examples

```
## Dummy 'timeSeries' with NAs entries

   data <- matrix(sample(c(1:20, rep(NA,4))), ncol = 2)
   s <- timeSeries(data, timeCalendar(2023))
   is.na(s)
## Find the longest consecutive non-missing values
   na.contiguous(s)

## tied longest stretches: 1:3, 6:9 and 10:12
x <- c(1:3, NA, NA, 6:8, NA, 10:12)
## should return the 1st one
na.contiguous(x)             # correct for R > 4.3.0
na.contiguous(timeSeries(x)) # correct for timeSeries version > 4030.106
```

`orderColnames` *Reorder column names of a time series*

Description

Functions and methods dealing with the rearrangement of column names of 'timeSeries' objects.

`orderColnames`	Returns ordered column names of a time Series,
`sortColnames`	Returns sorted column names of a time Series,
`sampleColnames`	Returns sampled column names of a time Series,
`statsColnames`	Returns statistically rearranged column names,
`pcaColnames`	Returns PCA correlation ordered column names,
`hclustColnames`	Returns hierarchical clustered column names.

Usage

```
orderColnames(x, ...)
sortColnames(x, ...)
sampleColnames(x, ...)
statsColnames(x, FUN = colMeans, ...)
pcaColnames(x, robust = FALSE, ...)
hclustColnames(x, method = c("euclidean", "complete"), ...)
```

Arguments

`x`	an object of class `timesSeries` or any other rectangular object which can be transformed by the function `as.matrix` into a numeric matrix.
`FUN`	a character string indicating which statistical function should be applied. By default statistical ordering operates on the column means of the time series.
`method`	a character string with two elements. The first determines the choice of the

	distance measure, see dist, and the second determines the choice of the agglomeration method, see hclust.
robust	a logical flag which indicates if robust correlations should be used.
...	further arguments to be passed to the underlying functions doing the main work, see section 'Details'.

Details

These functions reorder the column names of a "timeSeries" object according to some statistical measure.

Statistically Motivated Rearrangement

The function statsColnames rearranges the column names according to a statical measure. These measure must operate on the columns of the time series and return a vector of values which can be sorted. Typical functions ar those listed in help page colStats but custom functions can be used that compute for example risk or any other statistical measure. The ... argument allows to pass additional arguments to the underlying function FUN.

PCA Ordering of the Correlation Matrix

The function pcaColnames rearranges the column names according to the PCA ordered correlation matrix. The argument robust allsows to select between the use of the standard cor and computation of robust correlations using the function covMcd from contributed R package robustbase. The ... argument allows to pass additional arguments to the two underlying functions cor or covMcd. E.g., adding method="kendall" to the argument list calculates Kendall's rank correlations instead the default which calculates Person's correlations.

Ordering by Hierarchical Clustering

The function pcaColnames uses the hierarchical clustering approach hclust to rearrange the column names of the time series.

Value

for orderColnames, an integer vector representing the permutaion that will sort the column names,

for the other functions, a character vector giving the rearranged column names

Examples

```
## Load Swiss Pension Fund Benchmark Data -
   data <- LPP2005REC[,1:6]

## Abbreviate Column Names -
   colnames(data)

## Sort Alphabetically -
   sortColnames(data)

## Sort by Column Names by Hierarchical Clustering -
   hclustColnames(data)
   head(data[, hclustColnames(data)])
```

orderStatistics *Order statistics*

Description

Computes the order statistics of a "timeSeries" object.

Usage

```
orderStatistics(x)
```

Arguments

x	a "timeSeries" object.

Details

orderStatistics computes the order statistics for each column of a "timeSeries" object. The output is a named list with the order statistics for each column in a separate component.

Value

a named list, in which each component is an univariate "timeSeries" containing the order statistics of the corresponding column of the input time series.

Examples

```
## Load Swiss Pension Fund Benchmark Data -
   setRmetricsOptions(myFinCenter = "GMT")
   X <- LPP2005REC[, "SPI"]
   colnames(X)

## Compute 1% Order Statistics -
   N <- round(0.01*nrow(X))
   N
   OS <- orderStatistics(X)[[1]]
   OS[1:N, ]
```

periodical *End-of-Period series, stats, and benchmarks*

Description

Computes periodical statistics back to a given period.

Usage

```
endOfPeriodSeries(x,
    nYearsBack = c("1y", "2y", "3y", "5y", "10y", "YTD"))

endOfPeriodStats(x,
    nYearsBack = c("1y", "2y", "3y", "5y", "10y", "YTD"))

endOfPeriodBenchmarks(x, benchmark = ncol(x),
    nYearsBack = c("1y", "2y", "3y", "5y", "10y", "YTD"))
```

Arguments

x	an end-of-month recorded multivariate "timeSeries" object. One of the columns holds the benchmark series specified by argument benchmark,
nYearsBack	a period string. How long back should the series be treated? Options include values from 1 year to 10 years, and year-to-date: "1y", "2y", "3y", "5y", "10y", "YTD".
benchmark	an integer giving the position of the benchmark series in x. By default this is the last column of x.

Details

endOfPeriodSeries extract the data for the last few years, as specified by argument nYearsBack.

endOfPeriodStats computes basic exploratory statistics for the last few years in the data.

endOfPeriodBenchmarks returns benchmarks back to a given period.

x must be end of month data. Such series can be created using functions like align, alignDailySeries, daily2monthly.

Value

for endOfPeriodSeries, a "timeSeries",

for endOfPeriodStats, a data frame,

for endOfPeriodBenchmarks - currently NULL (invisibly), the function is unfinished.

Examples

```
## Load Series: Column 1:3 Swiss Market, Column 8 (4) Benchmark
   x <- 100 * LPP2005REC[, c(1:3, 8)]
   colnames(x)
   x <- daily2monthly(x)
   x

## Get the Monthly Series -
   endOfPeriodSeries(x, nYearsBack="1y")
```

```
## Compute the Monthly Statistics -
   endOfPeriodStats(x, nYearsBack="1y")

## Compute the Benchmark -
   endOfPeriodBenchmarks(x, benchmark=4)
```

---

plot-methods *Plot 'timeSeries' objects*

Description

"timeSeries" methods for plot, lines and points.

Usage

```
## S4 method for signature 'timeSeries'
plot(x, y, FinCenter = NULL,
    plot.type = c("multiple", "single"), format = "auto",
    at = pretty(x), widths = 1, heights = 1, xy.labels,
    xy.lines, panel = lines, nc, yax.flip = FALSE,
    mar.multi = c(0, 5.1, 0, if (yax.flip) 5.1 else 2.1),
    oma.multi = c(6, 0, 5, 0), axes = TRUE, ...)

## S4 method for signature 'timeSeries'
lines(x, FinCenter = NULL, ...)
## S4 method for signature 'timeSeries'
points(x, FinCenter = NULL, ...)

## S3 method for class 'timeSeries'
pretty(x, n=5, min.n=n%/%3, shrink.sml=0.75,
    high.u.bias=1.5, u5.bias=0.5+1.5*high.u.bias, eps.correct=0, ...)
```

Arguments

x, y	objects of class timeSeries.
FinCenter	a character with the the location of the financial center named as "continent/city".
plot.type	for multivariate time series, should the series by plotted separately (with a common time axis) or on a single plot?
format	POSIX label format, e.g. "%Y-%m-%d" or "%F" for ISO-8601 standard date format.
at	a timeDate object setting the plot label positions. If at=pretty(x), the postitions are generated automatized calling the function pretty. Default option at="auto" selects 6 equal spaced time label positions. For the new plot themes set at="pretty" or at="chic". In this case additional arguments can be passed through the ... arguments, see details.

`widths, heights`
: widths and heights for individual graphs, see `layout`.

`xy.labels`
: logical, indicating if `text()` labels should be used for an x-y plot, _or_ character, supplying a vector of labels to be used. The default is to label for up to 150 points, and not for more.

`xy.lines`
: logical, indicating if `lines` should be drawn for an x-y plot. Defaults to the value of `xy.labels` if that is logical, otherwise to `TRUE`

`panel`
: a `function(x, col, bg, pch, type, ...)` which gives the action to be carried out in each panel of the display for `plot.type="multiple"`. The default is `lines`.

`nc`
: the number of columns to use when `type="multiple"`. Defaults to 1 for up to 4 series, otherwise to 2.

`yax.flip`
: logical indicating if the y-axis (ticks and numbering) should flip from side 2 (left) to 4 (right) from series to series when `type="multiple"`.

`mar.multi, oma.multi`
: the (default) `par` settings for `plot.type="multiple"`.

`axes`
: logical indicating if x- and y- axes should be drawn.

`n`
: an integer giving the desired number of intervals.

`min.n`
: a nonnegative integer giving the minimal number of intervals.

`shrink.sml`
: a positive numeric by a which a default scale is shrunk in the case when range(x) is very small.

`high.u.bias`
: a non-negative numeric, typically > 1. Larger high.u.bias values favor larger units.

`u5.bias`
: a non-negative numeric multiplier favoring factor 5 over 2.

`eps.correct`
: an integer code, one of 0,1,2. If non-0, a correction is made at the boundaries.

`...`
: additional graphical arguments, see `plot`, `plot.default` and `par`.

Details

Our original method `plot` was build along R's plotting function `plot.ts` with an additional argument to tailor the position marks at user defined position specified by the argument `at`. We call this style or theme `"ts"`.

With verson R 3.1 we have inroduced two new additional plotting themes called `"pretty"` and `"chick"`. They are becoming active when we set `at = "pretty"` or `at = "chic"`.

Plot style or theme `"pretty"` is an extension of our original plotting method.

Plot style or theme `"chic"` is an implementation along the contributed packages `xts` and `PerformanceAnalytics` from the Chicago finance group members (`"chic"` is an abbreviation of Chicago.

For both themes, `"pretty"` and `"chic"`, additional arguments are passed through the `...` arguments. These are:

Argument	Default	Description
type	"l"	types pf plot
col	1	colors for lines and points
pch	20	plot symbol

cex	1	character and symbol scales
lty	1	line types
lwd	2	line widths
cex.axes	1	scale of axes
cex.lab	1	scale of labels
cex.pch	1	scale of plot symbols
grid	TRUE	should grid lines plotted?
frame.plot	TRUE	should b box around the plot?
axes	TRUE	should be axes drawn on the plot?
ann	TRUE	should default annotations appear?

Concerning the plot elements, the length of these vectors has to be the same as the number of columns in the time series to be plotted. If their length is only one, then they are repeated.

There is an almost 70 pages vignette added to the package, with dozens of examples of tailored plots. Have a look in it.

Value

NULL (invisibly), the functions are called for the side effect of producing plots

See Also

vignette("timeSeriesPlot", package="timeSeries"), which provides extensive plot examples.

Examples

```
## Load Swiss Pension Fund Benchmark Data -
   LPP <- LPP2005REC[1:12, 1:4]
   colnames(LPP) <- abbreviate(colnames(LPP), 2)
   finCenter(LPP) <- "GMT"

## Example Plot 1 -
   plot(LPP[, 1], type = "o", col = "steelblue",
     main = "LPP", xlab = "2005", ylab = "Return")
   plot(LPP[, 1], at="auto", type = "o", col = "steelblue",
     main = "LPP", xlab = "2005", ylab = "Return")

## Example Plot 2 -
   plot(LPP[, 1:2], type = "o", col = "steelblue",
     main = "LPP", xlab = "2005", ylab = "Return")

## Example Plot 3 -
   plot(LPP[, 1], LPP[, 2], type = "p", col = "steelblue",
     main = "LPP", xlab = "Return 1", ylab = "Return 2")

## Example Plot 4a, The Wrong Way to do it! -
   LPP <- as.timeSeries(data(LPP2005REC))
   ZRH <- as.timeSeries(LPP[,"SPI"], zone = "Zurich", FinCenter = "Zurich")
```

```
   NYC <- as.timeSeries(LPP[,"LMI"], zone = "NewYork", FinCenter = "NewYork")
   finCenter(ZRH)
   finCenter(NYC)
   plot(ZRH, at="auto", type = "p", pch = 19, col = "blue")
   points(NYC, pch = 19, col = "red")

## Example Plot 4b, Convert NYC to Zurich Time -
   finCenter(ZRH) <- "Zurich"
   finCenter(NYC) <- "Zurich"
   at <- unique(round(time(ZRH)))
   plot(ZRH, type = "p", pch = 19, col = "blue", format = "%b %d", at = at,
       xlab = paste(ZRH@FinCenter, "local Time"), main = ZRH@FinCenter)
   points(NYC, pch = 19, col = "red")

## Example 4c, Force Everything to GMT Using "FinCenter" Argument -
   finCenter(ZRH) <- "Zurich"
   finCenter(NYC) <- "NewYork"
   at <- unique(round(time(ZRH)))
   plot(ZRH, type = "p", pch = 19, col = "blue", format = "%b %d", at = at,
       FinCenter = "GMT", xlab = "GMT", main = "ZRH - GMT")
   points(NYC, FinCenter = "GMT", pch = 19, col = "red")
```

`print-methods` *Print 'timeSeries' objects*

Description

Print `"timeSeries"` objects.

Usage

```
## S4 method for signature 'timeSeries'
show(object)

## S4 method for signature 'timeSeries'
print(x, FinCenter = NULL, format = NULL,
          style = c("tS", "h", "ts"), by = c("month", "quarter"), ...)
```

Arguments

`object,x`	an object of class `"timeSeries"`.
`FinCenter`	a character with the the location of the financial center named as "continent/city".
`format`	the format specification of the input character vector, a character string with the format in POSIX notation.
`style`	a character string, one of `"tS"`, `"h"`, or `"ts"`.
`by`	a character string, one of `"month"`, `"quarter"`.
`...`	arguments passed to the print method for the data part, which is a `"matrix"` or, in the case of `style = "ts"`, to the print method for class `"ts"`.

Details

show does not have additional arguments.

The print method allows to modify the way the object is shown by explicitly calling print.

The default for style is tS. For univariate time series the style = "h" causes the object to be printed as a vector with the time stamps as labels. Finally, style = "ts" prints like objects from base R class "ts", which is suitable for quarterly and monthly time series.

Value

Prints an object of class timeSeries.

Examples

```
## Load Micsrosoft Data -
   setRmetricsOptions(myFinCenter = "GMT")
   LPP <- MSFT[1:12, 1:4]

## Abbreviate Column Names -
   colnames(LPP) <- abbreviate(colnames(LPP), 6)

## Print Data Set -
   print(LPP)

## Alternative Use, Show Data Set -
   show(LPP)

## a short subseries to demo 'print'
   hC <- head(MSFT[ , "Close"])
   class(hC)
   print(hC)
   print(hC, style = "h")
```

---

rank *Sample ranks of a time series*

Description

Compute the sample ranks of the values of a 'timeSeries' object.

Usage

```
## S4 method for signature 'timeSeries'
rank(x, na.last = TRUE, ties.method = )
```

Arguments

x	an univariate object of class timeSeries.
na.last	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed; if "keep" they are kept with rank NA.
ties.method	a character string specifying how ties are treated; can be abbreviated.

Details

If all components are different (and no NAs), the ranks are well defined, with values in seq_len(x). With some values equal (called 'ties'), argument ties.method determines the result at the corresponding indices. The "first" method results a permutation with increasing values at each index set of ties. The "random" method puts these in random order, whereas the default, "average", replaces them by their mean, and "max" and "min" replace them with their maximum and minimum respectively, the latter being the typical sports ranking.

NA values are never considered to be equal: for na.last = TRUE and na.last = FALSE they are given distinct ranks in the order in which they occur in x.

Value

a "timeSeries" object

Examples

```
## Load Microsoft Data -
   X <- 100 * returns(MSFT)

## Compute the Ranks -
   head(rank(X[, "Open"]), 10)

## Only Interested in the Vector, then use -
   head(rank(series(X[, "Open"])), 10)
```

---

readSeries *Read a 'timeSeries' from a text file*

Description

Reads a file in table format and creates a "timeSeries" object from it. The first column of the table must hold the timestamps.

Usage

```
readSeries(file, header = TRUE, sep = ";", zone = "",
    FinCenter = "", format, ...)
```

Arguments

file	the filename of a spreadsheet dataset from which to import the data records.
header	a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: 'header' is set to 'TRUE' if and only if the first row contains one fewer fields than the number of columns.
sep	the field seperator used in the spreadsheet file to separate columns, by default ";". If sep = ";" and reading the series fails, then the reading is automatically repeated with sep=",".
zone	the time zone or financial center where the data were recorded. By default zone = "" which is short for GMT.
FinCenter	a character with the the location of the financial center named as "continent/city".
format	a character string with the format in POSIX notation specifying the timestamp format. The format has not to be specified if the first column in the file has the timestamp format specifier, e.g. "%Y-%m-%d" for the short ISO 8601 format.
...	Additional arguments passed to read.table() which is used to read the file.

Details

The file is imported with read.table. Note the different default for argument "sep".

The first column of the table must hold the timestamps. Format of the timestamps can be either specified in the header of the first column or by the format argument.

Value

an object of class "timeSeries"

See Also

as.timeSeries, timeSeries, dummyMonthlySeries, dummyDailySeries

Examples

```
## full path to an example file
fn <- system.file("extdata/msft.csv", package = "timeSeries")
## first few lines of the file
readLines(fn, n = 5)

## import the file
msft <- readSeries(fn)
head(msft)

## is msft the same as the data object MSFT?
all.equal(msft, MSFT)
## ... almost, except for slot 'documentation'
c(msft@documentation, MSFT@documentation)
## actually, all.equal() says 'attribute', not slot. this is ok too:
c(attr(MSFT, "documentation"), attr(msft, "documentation"))
```

```
## make 'documentation' equal, here "", and compare again:
msft@documentation <- ""
all.equal(msft, MSFT) # TRUE
```

`returns` *Financial returns*

Description

Compute financial returns from prices or indexes.

Usage

```
returns(x, ...)
returns0(x, ...)

## S4 method for signature 'ANY'
returns(x, method = c("continuous", "discrete",
    "compound", "simple"), percentage = FALSE, ...)
## S4 method for signature 'timeSeries'
returns(x, method = c("continuous", "discrete",
    "compound", "simple"), percentage = FALSE, na.rm = TRUE,
    trim = TRUE, ...)
```

Arguments

`x`	an object of class `timeSeries`.
`method`	a character string. Which method should be used to compute the returns, one of "continuous", "discrete", or "compound", "simple". The second pair of methods is a synonym for the first two methods.
`percentage`	a logical value. By default `FALSE`, if `TRUE` the series will be expressed in percentage changes.
`na.rm`	a logical value. Should NAs be removed? By default `TRUE`.
`trim`	a logical value. Should the time series be trimmed? By Default `TRUE`.
`...`	arguments to be passed.

Value

an object of class `timeSeries`.

`returns0` returns an untrimmed series with the first row of returns set to zero(s).

Note

The functions `returnSeries` and `getReturns` will be removed in the near future. They are synonyms for the function `returns` and their use was discouraged for many years. Just use `returns`.

The function `returnSeries` is no longer exported. `getReturns` is exported only because we are waiting for a package on CRAN to be updated.

See Also

cumulated, drawdowns, splits, spreads, midquotes, index2wealth

Examples

```
## Load Microsoft Data -
   setRmetricsOptions(myFinCenter = "GMT")
   data(MSFT)
   X = MSFT[1:10, 1:4]
   X

## Continuous Returns -
   returns(X)
   returns0(X)

## Discrete Returns:
   returns(X, method = "discrete")

## Don't trim:
   returns(X, trim = FALSE)

## Use Percentage Values:
   returns(X, percentage = TRUE, trim = FALSE)
```

---

rev *Reverse a 'timeSeries'*

Description

Reverses an uni- or multivariate "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
rev(x)
```

Arguments

x	an uni- or multivariate "timeSeries" object.

Value

a "timeSeries" object

Examples

```
## Create Dummy "timeSeries"  -
   tS <- dummyMonthlySeries()

## Reverse Series -
   rev(tS)
```

`rollMean` *Rolling statistics*

Description

Computes rolling mean, min, max and median for a `"timeSeries"` object.

Usage

```
rollStats(x, k, FUN = mean, na.pad = FALSE,
    align=c("center", "left", "right"), ...)

rollMean(x, k, na.pad = FALSE,
    align = c("center", "left", "right"), ...)
rollMin(x, k, na.pad = FALSE,
    align = c("center", "left", "right"), ...)
rollMax(x, k, na.pad = FALSE,
    align = c("center", "left", "right"), ...)
rollMedian(x, k, na.pad = FALSE,
    align = c("center", "left", "right"), ...)
```

Arguments

`x`	an uni- or multivariate `"timeSeries"` object.
`k`	an integer width of the rolling window. Must be odd for `rollMedian`.
`FUN`	the function to be rolled.
`na.pad`	a logical flag. Should NA padding be added at beginning? By default `FALSE`.
`align`	a character string specifying whether the index of the result should be left- or right-aligned or centered compared to the rolling window of observations. The default choice is set to `align="center"`.
`...`	optional arguments to be passed.

Details

The code in the core of the functions `rollMean`, `rollMin`, `rollMax`, and `rollMedian` was borrowed from the package `zoo` authored by Achim Zeileis, Gabor Grothendieck and Felix Andrews.

Value

an object of class "timeSeries"

Author(s)

Achim Zeileis, Gabor Grothendieck and Felix Andrews for code from the contributed R package zoo used in the functions rollMean, rollMin, rollMax, and rollMedian.

Examples

```
## Use Swiss Pension Fund Data Set of Returns -
   head(LPP2005REC)
   SPI <- LPP2005REC[, "SPI"]
   head(SPI)

## Plot Drawdowns -
   rmean <- rollMean(SPI, k = 10)
   plot(rmean)
```

---

rowCum *Cumulative row statistics*

Description

Compute cumulative row statistics.

Usage

```
## S4 method for signature 'ANY'
rowCumsums(x, na.rm = FALSE, ...)
## S4 method for signature 'timeSeries'
rowCumsums(x, na.rm = FALSE, ...)
```

Arguments

x	a time series, may be an object of class "matrix" or "timeSeries".
na.rm	a logical. Should missing values be removed?
...	arguments to be passed.

Value

for the default method, a matrix,

for the "timeSeries" method, an S4 object of class "timeSeries".

See Also

colCumXXX

Examples

```
## Simulated Monthly Return Data -
   X = matrix(rnorm(24), ncol = 2)

## Compute cumulated Sums -
   rowCumsums(X)
```

---

runlengths *Runlengths of a time series*

Description

Computes runlengths of an univariate "timeSeries" object.

Usage

```
runlengths(x, ...)
```

Arguments

x	an univariate time series of class "timeSeries".
...	arguments passed to the function na.omit.

Details

Runlengths are defined here as contiguous sequences of values having the same sign.

Zeroes are treated as NAs.

Value

an object of class "timeSeries"

Examples

```
## random time series -
   set.seed(4711)
   x <- rnorm(12)
   tS <- timeSeries(data = x, charvec = timeCalendar(), units = "x")
   tS

## return runlengths -
   runlengths(tS)

## replace the middle value of the negative stretch of 3 values
tS[5] <- NA
## the two negative values separated by NA are still one run
runlengths(tS)
```

sample *Resample 'timeSeries' objects*

Description

Takes a sample of the specified size from the elements of a "timeSeries".

Usage

```
## S4 method for signature 'timeSeries'
sample(x, size, replace = FALSE, prob = NULL)
```

Arguments

x	an object from class "timeSeries".
size	a non-negative integer giving the number of items to choose.
replace	sample with replacement if TRUE, otherwise without replacement.
prob	a vector of probability weights for obtaining the elements of the vector being sampled.

Details

The function takes a sample of size size from the elements of the time series with or without replacement depending on argument replace. The result is returned as a "timeSeries" object.

For details about the arguments see the documentation of base:sample.

Value

an object from class "timeSeries"

See Also

sample (sample in base R),

sample (the "timeDate" method)

Examples

```
## Monthly Calendar Series -
   x <- daily2monthly(LPP2005REC[, 1:2])[3:14, ]


## Resample the Series with respect to the time stamps -
   resampled <- sample(x)
   resampled
   is.unsorted(resampled)
```

scale *Center and scale 'timeSeries' objects*

Description

Center and scale a "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
scale(x, center = TRUE, scale = TRUE)
```

Arguments

x	an object from class "timeSeries".
center, scale	a numeric vector or a logical value, see 'Details'.

Details

scale centers and/or scales the columns of a "timeSeries" object.

The value of center determines how column centering is performed. If center is a numeric vector with length equal to the number of columns of x, then each column of x has the corresponding value from center subtracted from it. If center is TRUE then centering is done by subtracting the column means (omitting NAs) of x from their corresponding columns, and if center is FALSE, no centering is done.

The value of scale determines how column scaling is performed (after centering). If scale is a numeric vector with length equal to the number of columns of x, then each column of x is divided by the corresponding value from scale. If scale is TRUE then scaling is done by dividing the (centered) columns of x by their standard deviations if center is TRUE, and the root mean square otherwise. If scale is FALSE, no scaling is done.

Value

a centered and/or scaled "timeSeries" object

Examples

```
## Load Series:
   x <- 100* LPP2005REC[, c("SBI", "SPI")]

## Scale and Center -
   X <- scale(x)
   hist(X[, 1], prob=TRUE)
   s <- seq(-3, 3, length=201)
   lines(s, dnorm(s), col="red")
```

series-methods *Get and set the data component of a 'timeSeries'*

Description

Get and set the data component of a 'timeSeries'.

Usage

```
series(x)
series(x) <- value
```

Arguments

x	a timeSeries object.
value	a vector, a data.frame or a matrix object of numeric data.

Details

series returns the @.Data slot of a timeSeries object in matrix form.

The assignment version of series replaces the values of the time series with value. The row and column names of value are used if not NULL, otherwise they are left as in x. The most natural use is when value has the same dimensions as as.matrix(x), but if that is not the case the result is almost as if value was converted to "timeSeries" directly.

Methods for zoo::coredata and its assignment counterpart are defined, as well. Users who wish to use them should ensure that zoo::coredata is visible (e.g., by calling library('zoo') or library('xts')) or employ the zoo:: prefix in the calls. These methods are equivalent to series and `series<-`, respectively.

See Also

timeSeries

Examples

```
## A Dummy 'timeSeries' Object
   ts <- timeSeries()
   ts

## Get the Matrix Part -
   mat <- series(ts)
   class(mat)
   mat

## Assign a New Univariate Series -
   series(ts) <- rnorm(12)
   ts
```

```
## Assign a New Bivariate Series -
   series(ts) <- matrix(rnorm(12), ncol = 2)
   ts
```

---

smooth *Smooths time series objects*

Description

Smooths a "timeSeries" object.

Usage

```
smoothLowess(x, f = 0.5, ...)
smoothSpline(x, spar = NULL, ...)
smoothSupsmu(x, bass = 5, ...)
```

Arguments

x	an univariate "timeSeries" object.
f	the lowess smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness.
spar	smoothing parameter, typically (but not necessarily) in (0,1]. By default NULL, i.e. the value will be automatically selected.
bass	controls the smoothness of the fitted curve. Values of up to 10 indicate increasing smoothness.
...	optional arguments to be passed to the underlying smoothers.

Details

The functions smoothLowess, smoothSpline, smoothSupsmu allow to smooth timeSerie object. The are interfaces to the function lowess, supmsu. and smooth.spline in R's stats package.

The ... arguments allow to pass optional arguments to the underlying stats functions and tailor the smoothing process. We refer to the manual pages of these functions for a proper setting of these options.

Value

a bivariate "timeSeries" object, the first column holds the original time series data, the second the smoothed series.

Author(s)

The R core team for the underlying smoother functions.

Examples

```
## Use Close from MSFT's Price Series  -
   head(MSFT)
   MSFT.CLOSE <- MSFT[, "Close"]
   head(MSFT.CLOSE)

## Plot Original and Smoothed Series by Lowess -
   MSFT.LOWESS <- smoothLowess(MSFT.CLOSE, f = 0.1)
   head(MSFT.LOWESS)
   plot(MSFT.LOWESS)
   title(main = "Close - Lowess Smoothed")

## Plot Original and Smoothed Series by Splines -
   MSFT.SPLINE <- smoothSpline(MSFT.CLOSE, spar = 0.4)
   head(MSFT.SPLINE)
   plot(MSFT.SPLINE)
   title(main = "Close - Spline Smoothed")

## Plot Original and Smoothed Series by Supsmu -
   MSFT.SUPSMU <- smoothSupsmu(MSFT.CLOSE)
   head(MSFT.SUPSMU)
   plot(MSFT.SUPSMU)
   title(main = "Close - Spline Smoothed")
```

`sort` *Sort a 'timeSeries' by time stamps*

Description

Sort a `"timeSeries"` object with respect to its time stamps.

Usage

```
## S4 method for signature 'timeSeries'
sort(x, decreasing = FALSE, ...)

## S4 method for signature 'timeSeries'
is.unsorted(x, na.rm = FALSE, strictly = FALSE)
```

Arguments

`x`	a `"timeSeries"` object.
`decreasing`	a logical flag. Should we sort in increasing or decreasing order? By default `FALSE`.
`na.rm`	a logical value, should missing values be removed?
`strictly`	logical indicating if the check should be for strictly increasing values.
`...`	optional arguments passed to other methods.

Details

Sorts a time series either in increasing or decreasing time stamp order. Internally the function `order` from R's base package is used. `order` generates a permutation which rearranges the time stamps in ascending or descending order.

To find out if the series is unsorted, use `is.unsorted`.

Value

for `sort`, a `"timeSeries"` object,

for the `is.unsorted` method, `TRUE` or `FALSE`

Examples

```
## Monthly Calendar Series -
   x <- daily2monthly(LPP2005REC[, 1:2])[3:14, ]


## Resample the Series with respect to the time stamps -
   resampled <- sample(x)
   resampled
   is.unsorted(resampled)

## Now sort the serie in decreasing time order -
   sorted <- sort(resampled, , decreasing = TRUE)
   sorted
   is.unsorted(sorted)

## Is the reverted series ordered? -
   reverted <- rev(sorted)
   reverted
   is.unsorted(reverted)
```

`splits` *splits*

Description

Searches for outlier splits in a `"timeSeries"` object.

Usage

```
splits(x, sd = 3, complement = TRUE, ...)
```

Arguments

x	a "timeSeries" object.
sd	numeric(1); deviations of how many standard deviations to consider too big? Can be fractional. E.g., 5 means that values larger or smaller than five times the standard deviation of the series will be detected.
complement	a logical flag, should the outlier series or its complements be returned?
...	arguments to be passed.

Details

This function finds splits in financial price or index series. If a price or index is splitted we observe a big jump of several standard deviations in the returns, which is identified usually as an outlier.

Value

a "timeSeries" object

See Also

returns, cumulated, drawdowns,

spreads, midquotes, index2wealth

Examples

```
## Create a Return Series with a Split -
   data <- runif(12, -1, 1)
   data[6] <- 20
   x <- timeSeries(data, timeCalendar(), units="RUNIF")
   x

## Search for the Split:
   splits(x, sd=3, complement=TRUE)
   splits(x, sd=3, complement=FALSE)
```

---

spreads *Spreads and mid quotes*

Description

Compute spreads and midquotes from price streams.

Usage

```
spreads(x, which = c("Bid", "Ask"), tickSize = NULL)
midquotes(x, which = c("Bid", "Ask"))
```

Arguments

x	an object of class timeSeries.
which	a vector with two character strings naming the column names of the time series from which to compute the mid quotes and spreads. By default these are the bid and ask prices with column names c("Bid", "Ask").
tickSize	the default is NULL to simply compute price changes in original price levels. If ticksize is supplied, the price changes will be divided by the value of inTicksOfSize to compute price changes in ticks.

Value

all functions return an object of class timeSeries

See Also

returns, cumulated, drawdowns, splits,

midquotes, index2wealth

Examples

```
## Load the Microsoft Data -
   setRmetricsOptions(myFinCenter = "GMT")
   data(MSFT)
   X = MSFT[1:10, ]
   head(X)

## Compute Open/Close Midquotes -
   X.MID <- midquotes(X, which = c("Close", "Open"))
   colnames(X.MID) <- "X.MID"
   X.MID

## Compute Open/Close Spreads -
   X.SPREAD <- spreads(X, which = c("Close", "Open"))
   colnames(X.SPREAD) <- "X.SPREAD"
   X.SPREAD
```

---

start *Start and end of a 'timeSeries'*

Description

Returns start or end time stamp of a "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
start(x, ...)

## S4 method for signature 'timeSeries'
end(x, ...)
```

Arguments

x	an uni- or multivariate "timeSeries" object.
...	optional arguments passed to other methods.

Value

a "timeSeries" object

Examples

```
## Create a dummy \code{"timeSeries"} -
   tS <- dummyMonthlySeries()[, 1]
   tS

## Return start and end time stamp -
   c(start(tS), end(tS))
   range(time(tS))
```

---

str-methods *Display the structure of 'timeSeries' objects*

Description

Compactly display the structure of a "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
str(object, ...)
```

Arguments

object	an object of class timeSeries.
...	arguments passed to other methods.

Value

NULL, invisibly. The function is called for its side effect of printing a compact representation of the structure of the "timeSeries" object.

Examples

```
## Load Microsoft Data Set -
   data(MSFT)
   X <- MSFT[1:12, 1:4]
   colnames(X) <- abbreviate(colnames(X), 4)

## Display Structure -
   str(X)
```

---

t *Transpose 'timeSeries' objects*

Description

Returns the transpose of a "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
t(x)
```

Arguments

x	a 'timeSeries' object.

Value

a matrix

Examples

```
## Dummy 'timeSeries' with NAs entries
   data <- matrix(1:24, ncol = 2)
   s <- timeSeries(data, timeCalendar())
   s

## Transpose 'timeSeries' -
   t(s)
```

time *Get and set time stamps of a 'timeSeries'*

Description

Functions and methods extracting and modifying positions of 'timeSeries' objects.

Usage

```
## S4 method for signature 'timeSeries'
time(x, ...)
## S3 replacement method for class 'timeSeries'
time(x) <- value

getTime(x)
setTime(x) <- value
```

Arguments

value	a valid value for the time component of x.
x	an object of class timeSeries.
...	optional arguments passed to other methods.

Details

time and time<- are generic functions with methods for class "timeSeries". They get and set the time component of the object.

getTime and setTime are non-generic alternatives are non-generic wrappers of time and time<-, respectively.

There is another generic function time<- defined in package **zoo**. When that package is loaded its time<- gets the "timeSeries" method. Also, if "time<-" is called with an object from class other than "timeSeries", the call is dispatched to "zoo:time<-" to apply a suitable method.

Value

for time and getTime, a "timeDate" object,

for time<- and and setTime, the modified "timeSeries" object.

Examples

```
## Create Dummy 'timeSeries' -
   X <- timeSeries(matrix(rnorm(24), 12), timeCalendar())

## Return Series Positions -
   getTime(X)
```

```
   time(X)

## Add / Subtract one Day from X
   setTime(X) <- time(X) - 24*3600 # sec
   X
   time(X) <- time(X) + 24*3600 # sec
   X
```

`timeSeries-class` *Class 'timeSeries' in package timeSeries*

Description

Class `"timeSeries"` in package timeSeries.

Objects from the Class

The main functions for creating objects from class `"timeSeries"` `timeSeries` and `as.timeSeries`.

Objects can also be created by calls of the form `new("timeSeries", .Data, units, positions, format, FinCenter, recordIDs, title, documentation)` but this is not recommended for routine work.

Slots

The structure of the `"timeSeries"` objects should, in general, be considered internal. The accessor functions to get and set the components should be used to get and set values of the slots.

`.Data`**:** Object of class `"matrix"` containing the data, one column for each variable.

`units`**:** Object of class `"character"`, the unit (or variable, or column) names of the time series object.

`positions`**:** Object of class `"numeric"`, the datetime stamps. If the time series doesn't have date-time stamps, then `positions` is of length zero.

`format`**:** Object of class `"character"`, a datetime format (such as `"%Y-%m-%d"`). if there are no time stamps `"format"` is equal to `"counts"`.

`FinCenter`**:** Object of class `"character"`, the financial center.

`recordIDs`**:** Object of class `"data.frame"` ~~

`title`**:** Object of class `"character"`, a title for printing.

`documentation`**:** Object of class `"character"`, by default it is set to the current date.

Extends

Class `"structure"`, from data part. Class `"vector"`, by class "structure", distance 2, with explicit coerce.

Methods

Below is a list of the methods that have "timeSeries" in their signature. It can be useful for technical purposes, for example in reporting bugs but most methods that need explanation are documented with the corresponding functions and looking at their help pages is recommended.

There are short explanations for Methods for functions that are not supposed to be called directly.

[signature(x = "timeSeries", i = "ANY", j = "index_timeSeries"): ...

[signature(x = "timeSeries", i = "character", j = "character"): ...

[signature(x = "timeSeries", i = "character", j = "index_timeSeries"): ...

[signature(x = "timeSeries", i = "character", j = "missing"): ...

[signature(x = "timeSeries", i = "index_timeSeries", j = "character"): ...

[signature(x = "timeSeries", i = "index_timeSeries", j = "index_timeSeries"): ...

[signature(x = "timeSeries", i = "index_timeSeries", j = "missing"): ...

[signature(x = "timeSeries", i = "matrix", j = "missing"): ...

[signature(x = "timeSeries", i = "missing", j = "character"): ...

[signature(x = "timeSeries", i = "missing", j = "index_timeSeries"): ...

[signature(x = "timeSeries", i = "missing", j = "missing"): ...

[signature(x = "timeSeries", i = "time_timeSeries", j = "ANY"): ...

[signature(x = "timeSeries", i = "time_timeSeries", j = "character"): ...

[signature(x = "timeSeries", i = "time_timeSeries", j = "index_timeSeries"): ...

[signature(x = "timeSeries", i = "time_timeSeries", j = "missing"): ...

[signature(x = "timeSeries", i = "timeDate", j = "character"): ...

[signature(x = "timeSeries", i = "timeDate", j = "index_timeSeries"): ...

[signature(x = "timeSeries", i = "timeDate", j = "missing"): ...

[signature(x = "timeSeries", i = "timeSeries", j = "index_timeSeries"): ...

[signature(x = "timeSeries", i = "timeSeries", j = "missing"): ...

[<- signature(x = "timeSeries", i = "character", j = "ANY"): ...

[<- signature(x = "timeSeries", i = "character", j = "missing"): ...

[<- signature(x = "timeSeries", i = "timeDate", j = "ANY"): ...

[<- signature(x = "timeSeries", i = "timeDate", j = "missing"): ...

$ signature(x = "timeSeries"): ...

$<- signature(x = "timeSeries", value = "ANY"): ...

$<- signature(x = "timeSeries", value = "factor"): ...

$<- signature(x = "timeSeries", value = "numeric"): ...

aggregate signature(x = "timeSeries"): ...

align signature(x = "timeSeries"): ...

apply signature(X = "timeSeries"): ...

as.data.frame signature(x = "timeSeries"): ...

as.list signature(x = "timeSeries"): ...

as.matrix signature(x = "timeSeries"): ...

as.ts signature(x = "timeSeries"): ...

attach signature(what = "timeSeries"): ...

cbind2 signature(x = "ANY", y = "timeSeries"): ...

cbind2 signature(x = "timeSeries", y = "ANY"): ...

cbind2 signature(x = "timeSeries", y = "missing"): ...

cbind2 signature(x = "timeSeries", y = "timeSeries"): ...

coerce signature(from = "ANY", to = "timeSeries")

coerce signature(from = "character", to = "timeSeries")

coerce signature(from = "data.frame", to = "timeSeries")

coerce signature(from = "timeSeries", to = "data.frame")

coerce signature(from = "timeSeries", to = "list"):

coerce signature(from = "timeSeries", to = "matrix")

coerce signature(from = "timeSeries", to = "ts"):

coerce signature(from = "ts", to = "timeSeries"): coerce should not be called directly. Use as(object, "target_class") instead.

colCummaxs signature(x = "timeSeries"): ...

colCummins signature(x = "timeSeries"): ...

colCumprods signature(x = "timeSeries"): ...

colCumreturns signature(x = "timeSeries"): ...

colCumsums signature(x = "timeSeries"): ...

colMeans signature(x = "timeSeries"): ...

colnames signature(x = "timeSeries"): ...

colnames<- signature(x = "timeSeries"): ...

colSums signature(x = "timeSeries"): ...

comment signature(x = "timeSeries"): ...

comment<- signature(x = "timeSeries"): ...

coredata signature(x = "timeSeries"): ...

coredata<- signature(x = "timeSeries", value = "ANY"): ...

coredata<- signature(x = "timeSeries", value = "matrix"): ...

cummax signature(x = "timeSeries"): ...

cummin signature(x = "timeSeries"): ...

cumprod signature(x = "timeSeries"): ...

cumsum signature(x = "timeSeries"): ...

diff signature(x = "timeSeries"): ...

dim signature(x = "timeSeries"): ...

dim<- signature(x = "timeSeries"): ...

dimnames signature(x = "timeSeries"): ...

dimnames<- signature(x = "timeSeries", value = "list"): ...

end signature(x = "timeSeries"): ...

filter signature(x = "timeSeries"): ...

finCenter signature(x = "timeSeries"): ...

finCenter<- signature(x = "timeSeries"): ...

frequency signature(x = "timeSeries"): ...

getDataPart signature(object = "timeSeries"): ...

head signature(x = "timeSeries"): ...

initialize signature(.Object = "timeSeries"):
don't call "initialize", call new("timeSeries", ...) instead. Even better, call timeSeries.

is.na signature(x = "timeSeries"): ...

is.unsorted signature(x = "timeSeries"): ...

isDaily signature(x = "timeSeries"): ...

isMonthly signature(x = "timeSeries"): ...

isQuarterly signature(x = "timeSeries"): ...

isRegular signature(x = "timeSeries"): ...

lag signature(x = "timeSeries"): ...

lines signature(x = "timeSeries"): ...

median signature(x = "timeSeries"): ...

merge signature(x = "ANY", y = "timeSeries"): ...

merge signature(x = "matrix", y = "timeSeries"): ...

merge signature(x = "numeric", y = "timeSeries"): ...

merge signature(x = "timeSeries", y = "ANY"): ...

merge signature(x = "timeSeries", y = "matrix"): ...

merge signature(x = "timeSeries", y = "missing"): ...

merge signature(x = "timeSeries", y = "numeric"): ...

merge signature(x = "timeSeries", y = "timeSeries"): ...

na.contiguous signature(object = "timeSeries"): ...

na.omit signature(object = "timeSeries"): ...

names signature(x = "timeSeries"): ...

names<- signature(x = "timeSeries"): ...

Ops signature(e1 = "array", e2 = "timeSeries"): ...

Ops signature(e1 = "timeSeries", e2 = "array"): ...

Ops signature(e1 = "timeSeries", e2 = "timeSeries"): ...

Ops signature(e1 = "timeSeries", e2 = "ts"): ...

Ops signature(e1 = "timeSeries", e2 = "vector"): ...
Ops signature(e1 = "ts", e2 = "timeSeries"): ...
Ops signature(e1 = "vector", e2 = "timeSeries"): ...
outlier signature(x = "timeSeries"): ...
plot signature(x = "timeSeries"): ...
points signature(x = "timeSeries"): ...
print signature(x = "timeSeries"): ...
quantile signature(x = "timeSeries"): ...
rank signature(x = "timeSeries"): ...
rbind2 signature(x = "ANY", y = "timeSeries"): ...
rbind2 signature(x = "timeSeries", y = "ANY"): ...
rbind2 signature(x = "timeSeries", y = "missing"): ...
rbind2 signature(x = "timeSeries", y = "timeSeries"): ...
returns signature(x = "timeSeries"): ...
rev signature(x = "timeSeries"): ...
rowCumsums signature(x = "timeSeries"): ...
rownames signature(x = "timeSeries"): ...
rownames<- signature(x = "timeSeries", value = "ANY"): ...
rownames<- signature(x = "timeSeries", value = "timeDate"): ...
sample signature(x = "timeSeries"): ...
scale signature(x = "timeSeries"): ...
series signature(x = "timeSeries"): ...
series<- signature(x = "timeSeries", value = "ANY"): ...
series<- signature(x = "timeSeries", value = "matrix"): ...
setDataPart signature(object = "timeSeries"): ...
show signature(object = "timeSeries"): ...
sort signature(x = "timeSeries"): ...
start signature(x = "timeSeries"): ...
str signature(object = "timeSeries"): ...
t signature(x = "timeSeries"): ...
tail signature(x = "timeSeries"): ...
time signature(x = "timeSeries"): ...
window signature(x = "timeSeries"): ...

See Also

timeSeries and as.timeSeries for creating and converting to "timeSeries",

readSeries for importing from a text file,

dummyDailySeries for creation of dummy daily and monthly time series,

as.matrix, time, finCenter, getUnits, dim, start, etc., for accessing properties of the time series.

Examples

```
## see the help page for timeSeries()
showClass("timeSeries")
```

`timeSeries-method-stats` *Base R functions applied to 'timeSeries' objects*

Description

Many base R statistical functions work on (the data part of) `timeSeries` objects without the need for special methods, e.g., `var`, `sd`, `cov`, `cor`, probability densities, and others. This page gives some examples with such functions.

See Also

`colStats`, `colVars`, and other `colXXX` functions

Examples

```
## Load Microsoft Data Set -
   data(MSFT)
   X = MSFT[, 1:4]
   X = 100 * returns(X)

## Compute Covariance Matrix -
   cov(X[, "Open"], X[, "Close"])
   cov(X)

cor(X)
```

`TimeSeriesClass` *Create objects from class 'timeSeries'*

Description

`timeSeries` creates a `"timeSeries"` object from scratch.

Usage

```
timeSeries(data, charvec, units = NULL, format = NULL, zone = "",
           FinCenter = "", recordIDs = data.frame(), title = NULL,
           documentation = NULL, ...)
```

Arguments

data	a matrix object or any objects which can be coerced to a matrix.
charvec	a character vector of dates and times or any objects which can be coerced to a "timeDate" object.
units	an optional character string, which allows to overwrite the current column names of a "timeSeries" object. By default NULL which means that the column names are selected automatically.
format	the format specification of the input character vector, a character string with the format in POSIX notation.
zone	the time zone or financial center where the data were recorded.
FinCenter	a character with the the location of the financial center named as "continent/city".
recordIDs	for timeSeries, a data frame which can be used for record identification.
title	an optional title string, if not specified the input's data name is deparsed.
documentation	optional documentation string, or a vector of character strings.
...	arguments passed to other methods.

Details

Generation of Time Series Objects:

We have defined a "timeSeries" class which is in many aspects similar to the S-Plus class with the same name, but has also some important differences. The class has seven Slots, the 'Data' slot which holds the time series data in matrix form, the 'position' slot which holds the time/date as a character vector, the 'format' and 'FinCenter' slots which are the same as for the 'timeDate' object, the 'units' slot which holds the column names of the data matrix, and a 'title' and a 'documentation' slot which hold descriptive character strings. Date and time is managed in the same way as for timeDate objects.

as.timeSeries also creates "timeSeries" objects. as.timeSeries(x) is mostly equivalent to timeSeries(x) but the two functions have different methods. Beside that, the main difference between the two functions is that as.timeSeries doesn't accept additional arguments. The one argument call is naturally interpreted as 'convert to', so as.timeSeries is more expressive and is recommended in that case.

"timeSeries" methods are provided for many base R functions, including arithmetic operations, mathematical functions, print, summary, and time series functions. Not all are explicitly documented, since they can just be used.

Value

an S4 object of class "timeSeries"

See Also

as.timeSeries, class timeSeries,

Examples

```
## Load Microsoft data -
   # Microsoft Data:
   setRmetricsOptions(myFinCenter = "GMT")
   data(MSFT)
   head(MSFT)

## Create a 'timeSeries' object, the direct Way ...
   Close <- MSFT[, 5]
   head(Close)

## Create a 'timeSeries' object from scratch -
   data <- as.matrix(MSFT[, 4])
   charvec <- rownames(MSFT)
   Close <- timeSeries(data, charvec, units = "Close")
   head(Close)
   c(start(Close), end(Close))

## Cut out April data from 2001 -
   tsApril01 <- window(Close, "2001-04-01", "2001-04-30")
   tsApril01

## Compute Continuous Returns -
   returns(tsApril01)

## Compute Discrete Returns -
   returns(tsApril01, type = "discrete")

## Compute Discrete Returns, Don't trim -
   returns(tsApril01, trim = FALSE)

## Compute Discrete Returns, Use Percentage Values -
   tsRet <- returns(tsApril01, percentage = TRUE, trim = FALSE)
   tsRet

## Aggregate Weekly -
   GoodFriday(2001)
   to <- timeSequence(from = "2001-04-11", length.out = 3, by = "week")
   from <- to - 6*24*3600
   from
   to
   applySeries(tsRet, from, to, FUN = sum)

## Create large 'timeSeries' objects with different 'charvec' object classes -
   # charvec is a 'timeDate' object
   head(timeSeries(1:1e6L, timeSequence(length.out = 1e6L, by = "sec")))
   head(timeSeries(1:1e6L, seq(Sys.timeDate(), length.out = 1e6L, by = "sec")))
   # 'charvec' is a 'POSIXt' object
   head(timeSeries(1:1e6L, seq(Sys.time(), length.out = 1e6L, by = "sec")))
   # 'charvec' is a 'numeric' object
   head(timeSeries(1:1e6L, 1:1e6L))
```

TimeSeriesData *Time series data sets*

Description

Three data sets used in example files.

Details

The following datasets are available:

MSFT Daily Microsoft OHLC (Open-high-low-close) prices and volume from 2000-09-27 to 2001-09-27.

USDCHF USD/CHF intraday foreign exchange rates.

LPP2005REC Swiss pension fund assets returns benchmark from 2005-11-01 to 2007-04-11.

The datasets are objects from class "timeSeries".

Note

No further information about the LPP2005REC is available. The meaning of the columns?

See Also

readSeries, timeSeries

Examples

```
## LPP2005 example data set
data(LPP2005REC)
plot(LPP2005REC, type = "l")
class(LPP2005REC)
dim(LPP2005REC)
head(LPP2005REC)
LPP2005REC[1:5, 2:4]
range(time(LPP2005REC))
summary(LPP2005REC)

## MSFT example data set
data(MSFT)
plot(MSFT[, 1:4], type = "l")
plot(MSFT[, 5], type = "h")
class(MSFT)
range(time(MSFT))
head(MSFT)

## Plot USDCHF example data set
data(USDCHF)
plot(USDCHF)
```

```
range(time(USDCHF))
head(USDCHF)
```

TimeSeriesSubsettings *Subsetting time series*

Description

Objects from class "timeSeries" can be subsetted in different ways. Methods are defined for the subsetting operators "$", "[" and their assignment versions, as well as for some related functions from base R. A function to drop or extract outliers is also described here.

Usage

```
## S4 method for signature 'timeSeries'
head(x, n = 6, recordIDs = FALSE, ...)
## S4 method for signature 'timeSeries'
tail(x, n = 6, recordIDs = FALSE, ...)

outlier(x, sd = 5, complement = TRUE, ...)
```

Arguments

x	an object of class timeSeries.
n	an integer specifying the number of lines to be returned. By default n=6.
recordIDs	a logical value. Should the recordIDs be returned together with the data matrix and time series positions?
sd	a numeric value of standard deviations, e.g. 10 means that values larger or smaller than ten times the standard deviation will be removed from the series.
complement	a logical flag. If TRUE, the default, return the series free of outliers. If FALSE, return the outliers series.
...	arguments passed to other methods.

Details

The "timeSeries" methods for the subsetting operators "$", "[" and their assignment versions, as well as for the functions head and tail are meant to do what the user expects.

TODO: Further details are needed here, despite the above paragraph.

outlier drops the outliers if complement = TRUE and returns only them if complement = FALSE.

All functions described here return "timeSeries" objects.

See also window which extracts the sub-series between two datetimes.

Value

All functions return an object of class "timeSeries".

See Also

window

Examples

```
## Create an Artificial 'timeSeries' Object -
   setRmetricsOptions(myFinCenter = "GMT")
   charvec <- timeCalendar()
   set.seed(4711)
   data <- matrix(exp(cumsum(rnorm(12, sd = 0.1))))
   tS <- timeSeries(data, charvec, units = "tS")
   tS

## Subset Series by Counts "[" -
   tS[1:3, ]

## Subset the Head of the Series -
   head(tS, 6)
```

---

turns *Turning points of a time series*

Description

Extracts and analyzes turning points of an univariate "timeSeries" object.

Usage

```
turns(x, ...)

turnsStats(x, doplot = TRUE)
```

Arguments

x	an univariate "timeSeries" object of financial indices or prices.
...	optional arguments passed to the function na.omit.
doplot	a logical flag, should the results be plotted? By default TRUE.

Details

The function turns determines the number and the positions of extrema (turning points, either peaks or pits) in a regular time series.

The function turnsStats calculates the quantity of information associated with the observations in this series, according to Kendall's information theory.

The functions are borrowed from the contributed R package pastecs and made ready for working together with univariate timeSeries objects. You need not to load the R package pastecs, the code parts we need here are builtin in the timeSeries package.

We have renamed the function turnpoints to turns to distinguish between the original function in the contributed R package pastecs and our Rmetrics function wrapper.

For further details please consult the help page from the contributed R package pastecs.

Value

for turns, an object of class timeSeries.

for turnsStats, an object of class turnpoints with the following entries:

data	The dataset to which the calculation is done.
n	The number of observations.
points	The value of the points in the series, after elimination of ex-aequos.
pos	The position of the points on the time scale in the series (including ex-aequos).
exaequos	Location of exaequos (1), or not (0).
nturns	Total number of turning points in the whole time series.
firstispeak	Is the first turning point a peak (TRUE), or not (FALSE).
peaks	Logical vector. Location of the peaks in the time series without ex-aequos.
pits	Logical vector. Location of the pits in the time series without ex-aequos.
tppos	Position of the turning points in the initial series (with ex-aequos).
proba	Probability to find a turning point at this location.
info	Quantity of information associated with this point.

Author(s)

Frederic Ibanez and Philippe Grosjean for code from the contributed R package pastecs and Rmetrics for the function wrapper.

References

Ibanez, F., 1982, Sur une nouvelle application de la theorie de l'information a la description des series chronologiques planctoniques. J. Exp. Mar. Biol. Ecol., 4, 619–632

Kendall, M.G., 1976, Time Series, 2nd ed. Charles Griffin and Co, London.

Examples

```
## Load Swiss Equities Series -
   SPI.RET <- LPP2005REC[, "SPI"]
   head(SPI.RET)

## Cumulate and Smooth the Series -
   SPI <- smoothLowess(cumulated(SPI.RET), f=0.05)
   plot(SPI)

## Plot Turn Points Series -
   SPI.SMOOTH <- SPI[, 2]
```

```
   tP <- turns(SPI.SMOOTH)
   plot(tP)

## Compute Statistics -
   turnsStats(SPI.SMOOTH)
```

`units` *Get and set unit names of a 'timeSeries'*

Description

Gets and sets the column names of a `"timeSeries"` object. The column names are also called units or unit names.

Usage

```
getUnits(x)
setUnits(x) <- value
```

Arguments

`x`	a `"timeSeries"` object.
`value`	a character vector of unit names.

See Also

`timeSeries`

Examples

```
## A Dummy 'timeSeries' Object
   tS <- dummyMonthlySeries()
   tS

## Get the Units -
   getUnits(tS)

## Assign New Units to the Series -
   setUnits(tS) <- c("A", "B")
   head(tS)
```

wealth *Conversion of an index to wealth*

Description

Converts an index series to a wealth series normalizing the starting value to one.

Usage

```
index2wealth(x)
```

Arguments

x	an object of class 'timeSeries'.

Value

returns a time series object of the same class as the input argument x normalizing the starting value to one.

See Also

returns, cumulated, drawdowns, splits, spreads, midquotes,

Examples

```
## Load MSFT Open Prices  -
   INDEX <- MSFT[1:20, 1]
   INDEX

## Compute Wealth Normalized to 100 -
   100 * index2wealth(INDEX)
```

window *Methods for 'window' in package 'timeSeries'*

Description

Extract a part from a "timeSeries" object.

Usage

```
## S4 method for signature 'timeSeries'
window(x, start, end, ...)
```

Arguments

x	an object of class "timeSeries".
start, end	starting date and end date, end must be after start.
...	arguments passed to other methods.

Details

window extracts the subset of the "timeSeries" object x observed between the times start and end.

See Also

head, outlier

Examples

```
## Load LPP Benchmark Returns -
   x <- LPP2005REC[, 7:9]
   range(time(x))

## Extract Data for January 2006 -
   window(x, "2006-01-01", "2006-01-31")
```

Index