Package ‘tis’

December 19, 2017

Title     Time Indexes and Time Indexed Series
Version   1.34
Author    Jeff Hallman <jeffhallman@gmail.com>
Maintainer Brian Salzer <brian.m.salzer@frb.gov>
Depends   R (>= 2.3)
Enhances  zoo, ggplot2, graphics
Suggests  reshape, scales
Description Functions and S3 classes for time indexes and time indexed series, which are compatible with FAME frequencies.
License   Unlimited
RoxygenNote 6.0.1
NeedsCompilation yes
Repository  CRAN
Date/Publication 2017-12-19 20:24:10 UTC

R topics documented:

tis-package .......................................................... 3
aggregate.tis ....................................................... 5
as.data.frame.tis .................................................. 6
as.Date.jul .......................................................... 7
as.list.ti ............................................................ 8
as.matrix.tis ....................................................... 9
as.ts.tis ............................................................ 9
asClassyList ...................................................... 10
assignList .......................................................... 11
barplot.tis ........................................................ 12
barplot2 ........................................................... 13
basis ................................................................. 15
between ............................................................. 17
blanks ............................................................... 17
capitalize ........................................................... 18
### R topics documented:

- `cbind.tis` .................................................. 18
- `columns` ..................................................... 19
- `constantGrowthSeries` .................................... 20
- `convert` ...................................................... 21
- `csv` .......................................................... 23
- `cumsum.tis` .................................................. 24
- `currentMonday` .............................................. 25
- `currentPeriod` .............................................. 26
- `dateRange` ................................................... 27
- `dayOfPeriod` ................................................ 28
- `description` ................................................ 29
- `evalOrEcho` ................................................ 30
- `format.ti` ................................................... 31
- `fortify.tis` ................................................ 32
- `frColors` .................................................... 33
- `growth.rate` ............................................... 34
- `hms` .......................................................... 35
- `holidays` ..................................................... 36
- `inferTi` ....................................................... 38
- `Intraday` ..................................................... 39
- `isIntradayTif` .............................................. 40
- `isLeapYear` ................................................ 41
- `jul` .......................................................... 42
- `lags` .......................................................... 44
- `latestPeriod` ............................................... 45
- `linearSplineIntegration` .................................. 46
- `lines.tis` .................................................... 48
- `mergeSeries` ............................................... 49
- `naWindow` ................................................... 50
- `nberShade` .................................................. 50
- `nberShade.ggplot` ......................................... 52
- `plotWindow` ................................................ 53
- `POSIXct` ..................................................... 54
- `print.ti` ..................................................... 55
- `print.tis` ................................................... 56
- `RowMeans` ................................................... 57
- `scatterPlot` ............................................... 58
- `screenPage` ................................................ 62
- `setColors` ................................................... 64
- `setDefaultFrequencies` ................................... 64
- `solve.tridiag` ............................................. 66
- `ssDate` ....................................................... 66
- `start.tis` ................................................... 67
- `stripBlanks` ............................................... 68
- `stripClass` ................................................ 69
- `t.tis` ........................................................ 70
- `ti` ........................................................... 70
### Description

Functions and S3 classes for time indices and time indexed series, a flexible kind of time series compatible with series and frequencies understood by the FAME DBMS.

### Details

For a complete list of functions provided by this package, use `library(help="tis")`.

The `ti` (Time Index) and `tis` (Time Indexed Series) classes provide date arithmetic facilities and an alternative to the somewhat inflexible `ts` class in the standard R stats package.

#### Time Indexes (ti class)

A time index has two parts: a `tif` (Time Index Frequency) code and a period. `tif` codes all lie in the open interval (1000..5000) and the period is a nonnegative number less than 1e10. The `ti` encodes both, as for any `ti`:

\[
\text{unclass}(z) = (\text{tif}(z) \times 1e10) + \text{period}(z)
\]

Each `tif` has a particular base period (the time period where `period(z) == 0`). For example, the base period for an “anndecember” (annual December) `ti` is the year ending December 31, 1599. Periods before the base period cannot be represented by instances of the `ti` class.

If `x` and `y` are `ti` objects with the same `tif`, then

\[
x - y = \text{period}(x) - \text{period}(y)
\]

and

\[
x + (\text{period}(y) - \text{period}(x)) = y
\]

are both `TRUE`, so you can use `ti`'s for various calendrical calculations.

A `jul` class is also provided for Julian date-time objects. The `jul()` constructor can create a `jul` from an `ssDate` (spreadsheet date), a `POSIXct` or `POSIXlt` object, a `ti` object, a decimal time (like...
2007.5), a yyyymmdd number, a Date, or anything that can be coerced to a Date by as.Date. The ymd() function and its derivatives (year(), month(), day(), etc.) work on anything that jul() can handle.

Time Indexed Series (tis class)

The tis class maps very closely to the FAME (http://www.sungard.com/Fame/) database notion of what a time series is. A tis (Time Indexed Series) is vector or matrix indexed by a ti. If x is a tis, then start(x) gives the ti for the first observation, and [start(x) + k] is the ti for the k'th observation, while end(x) gives the ti for the last observation.

You can replace, say, the 5'th observation in a tis x by

```r
x[start(x) + 4] <- 42
```

and of course the [ operator also works with a ti. So if you want the value of the daily series x from July 3, 1998, you can get it with

```r
x[ti(19980703, "daily")]
```

provided, of course, that ymd(start(x)) <= 19980703 <= ymd(end(x)).

Numerous methods for tis objects are provided:

```r
> methods(class = "tis")
[1] aggregate.tis* as.data.frame.tis* as.matrix.tis* as.tis.tis*
[5] as.ts.tis* cbind.tis* cummax.tis* cummin.tis*
[9] cumprod.tis* cumsum.tis* cycle.tis* deltat.tis*
[13] diff.tis* edit.tis* end.tis* frequency.tis*
[17] lag.tis* lines.tis* Ops.tis* points.tis*
[21] print.tis* RowMeans.tis* RowSums.tis* start.tis*
[25] tifName.tis* tif.tis* time.tis* [.tis*
[29] [.tis* ti.tis* t.tis* window.tis*
```

as well as tisMerge and tisPlot functions. The convert function creates series of different frequencies from a given series in ways both more powerful and more flexible than aggregate can.

Setting Default Frequencies:

Like the FAME DBMS, the ti class has seven weekly frequencies for weeks ending on Sunday, Monday, and it has 12 annual frequencies, for years ending on the last day of each of the 12 months. There are also multiple biweekly, bimonthly and semiannual frequencies.

At any time you can use the function setDefaultFrequencies to change which actual frequencies the strings "weekly", "biweekly", "bimonthly", "quarterly", "semiannual" and "annual" refer to. Note (as shown by args(setDefaultFrequencies)) that if you don’t specify some other frequencies to setDefaultFrequencies, you’ll get default weeks ending on Monday and default biweeks ending on the first Wednesday. I chose these because they are the weeks and biweeks used most often (for money and reserves data) in my section at the Federal Reserve Board. You might want to use something like

```r
setHook(packageEvent("tis", "onLoad"), tis::setDefaultFrequencies(yourArgsGoHere))
```
in your .First() to automatically set the defaults up the way you want them whenever this package is loaded.

LAGS:
The stats::lag(x, k) function says that a series lagged by a positive k starts earlier. The opposite is true for the Lag function in this package, to maintain consistency with the common usage of 'first lag, second lag' and so on in econometrics.

tisFilter:

The stats::filter function coerces its argument to a ts time series and returns a ts. For tis and zoo series, this is not right. Both I and the author of the zoo package have requested that stats::filter be made an S3 generic with the current version renamed as filter.default. This would allow zoo and tis to define filter.zoo and filter.tis such that filter(aZooOrTis) would do the right thing. We are hopeful that this will happen soon. Meanwhile, tisFilter should be used to filter a tis.

Author(s)

Jeff Hallman <jhallman@frb.gov>
Maintainer: ditto

aggregate.tis

Compute Summary Statistics of Time Series Subsets

Description

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

Usage

## S3 method for class 'tis'
aggregate(x, FUN = sum, ...)

Arguments

x

a tis time series.

FUN

a scalar function to compute the summary statistics which can be applied to all data subsets.

... further arguments passed to aggregate.tis

Details

This is a method for the generic aggregate function.

The aggregate function was really designed for ts objects, not tis objects which may or may not meet the assumptions embedded in the function. The convert function is better suited for tis series.

aggregate.tis calls as.ts on its x argument, then passes that and all other arguments on to aggregate.ts and then turns the result back into a tis series. If there is a local version of aggregate.ts that can be found by exists("aggregate.ts", envir = globalenv()), it will be used in preference to the function in package:stats.
as.data.frame.tis

Value

the tis object returned by as.tis() called on the ts object returned by aggregate.ts.

See Also

apply, lapply, tapply, aggregate, and convert.

Examples

z <- tis(1:24, start = latestJanuary()) ## a monthly series
aggregate(z, nf = 4, FUN = mean) ## quarterly average
aggregate(z, nf = 1, FUN = function(x) x[length(x)]) ## December is annual level

---

as.data.frame.tis Coerce to a Data Frame

Description

Coerce a Time Indexed Series to a data frame.

Usage

## S3 method for class 'tis'
as.data.frame(x, ...)

Arguments

x a tis series
...
other args passed on to as.data.frame.matrix or as.data.frame.vector

Details

The function is very simple: it calls as.data.frame.matrix if x is a matrix, or as.data.frame.vector if it is not.

Value

a data frame.

See Also
data.frame
Description

Methods to convert ti and jul objects to class "Date" representing calendar dates.

Usage

```r
## S3 method for class 'ti'
as.Date(x, offset = 1, ...)
## S3 method for class 'jul'
as.Date(x, origin = "1970-01-01", ...)
```

Arguments

- `x`: A ti or jul object to be converted.
- `offset`: a number between 0 and 1 specifying where in the period represented by the ti object `x` the desired date falls. `offset = 1` gives the first second of the period and `offset = 1` the last second, `offset = 0.5` the middle second, and so on.
- `origin`: a Date object, or something which can be coerced by `as.Date(origin, ...)` to such an object.
- `...`: additional args passed on to `as.Date.numeric`

Value

An object of class "Date".

See Also

`as.Date` for the generic function, `as.Date.numeric` for the method eventually called, and `Date` for details of the date class.

Examples

```r
as.Date(today())       ## invokes as.Date.ti
as.Date(jul(today() - 7)) ## a week ago, uses as.Date.jul
```
Construct a List from a Time Index or Jul Object

Description

as.list.ti creates a list of one-element ti objects from the elements of its arguments.

as.list.jul creates a list of one-element jul objects from the elements of its arguments.

Usage

```r
## S3 method for class 'ti'
as.list(x, ...)
## S3 method for class 'jul'
as.list(x, ...)
```

Arguments

- `x` a ti or jul object
- `...` not used

Details

These are the ti and jul methods for the generic as.list.

Value

a list of one-element ti or jul objects.

See Also

asClassyList, as.list

Examples

```r
as.list(today() + 1:5)
as.list(jul(today()) + 1:5)
```
as.matrix.tis

Create a Matrix from a Time Indexed Series

Description

The function adds a dim attribute of c(length(x), 1) to its argument unless it already has a dim attribute of length 2.

Usage

```r
## S3 method for class 'tis'
as.matrix(x, ...)  
```

Arguments

- `x` a tis object
- `...` ignored

Value

A tis object with a dim attribute of length 2.

as.ts.tis

Convert a Time Indexed Series to a Time Series

Description

Constructs a ts object from a tis object. The tis object's starting year, starting cycle, and frequency, along with the object's data, in a call to the ts function.

Usage

```r
## S3 method for class 'tis'
as.ts(x, ...)  
```

Arguments

- `x` a tis object to be converted
- `...` Ignored

Details

The tis class covers more frequencies than the ts class does, so the conversion may not be accurate.
asClassyList

Convert a vector into a list of objects with the same class

Description

Turns its argument into a list of elements with the same class as the argument.

Usage

asClassyList(x, ...)

Arguments

x object to be coerced or tested.
...
objects, possibly named.

Value

A list L of one-element vectors with L[[i]] == x[i] for i in 1:length(x)

Note

The implementation of this function is identical to as.list.factor. It is used in as.list.ti and as.list.jul.

See Also

as.list

Examples

asClassyList(today() + 1:5)
assignList

Assign Values In a List to Names

Description

Assigns the values in a list to variables in an environment. The variable names are taken from the names of the list, so all of the elements of the list must have non-blank names.

Usage

assignList(alist, pos = -1, envir = as.environment(pos), inherits = FALSE)

Arguments

- **alist**: a list of values to be assigned to variables with names given by names(alist).
- **pos**: where to do the assignment. By default, assigns into the current environment.
- **envir**: the environment to use.
- **inherits**: should the enclosing frames of the environment be inspected?

Details

See *assign* for details on how R assignment works. This function simply uses the elements of names(alist) and alist itself to call the .Internal function used by *assign* once for each element of alist.

Value

This function is invoked for its side effect, which assigns values to the variables with names given by names(alist).

See Also

- *assign*

Examples

```r
myList <- list(a = 1, b = 2, c = 3)
assignList(myList) ## equivalent to a <- 1; b <- 2; c <- 3
```
barplot.tis  Barplot for Time Indexed Series

Description

This is barplot for tis objects. If the first argument to barplot is a tis object, this function is called. There may be more than one tis argument. See details below.

Usage

```r
## S3 method for class 'tis'
barplot(height, ...)
```

Arguments

- `height`: a tis object. There may, however, be more than just one tis argument in the call.
- `...`: arguments passed on to barplot.default.

Details

barplot.tis constructs a call to barplot2. The tis arguments, including but not limited to `height`, are pulled out and used to construct a `height` argument for the constructed call, and the `width` argument will be calculated as described below.

If the series sent in are multivariate, i.e., have multiple columns, the constructed `height` argument will also have multiple columns, which is how barplot does stacked bar charts. If you supply several multivariate series to barplot.tis, all of the series must have the same number of columns.

If the `beside` argument is supplied in the `...` list, the `width` argument will be set to $d/(NC + 0.5)$, where $d$ is the mean difference in decimal time units (where one year = 1, a quarter = 0.25, and so on) of the series observation times, $NC$ is the number of columns in the series arguments, and `space` will be set to `c(0, 0.5)`. The effect of all this will be to make the total width of the barplot match the length (in years) of the series plotted. When combined with the calculated `x.offset` described below, this will make the plot align correctly on the time axis. However, note that the alignment will only really be correct if all of the series plotted have the same frequency, as the underlying barplot.default forces each group of bars to have the same width when `beside = TRUE`.

If the series being plotted are of different frequencies, you should not set `beside`, leaving it at the default value of `FALSE`. This will cause the widths of the bars for each series to be inversely proportional to the series frequencies, and the individual observations will align correctly on the time axis.

barplot.tis finds the earliest starting date of the tis arguments and shifts the plot rightward along the x-axis by that amount, which aligns the first bar with its start date. That is, if gdp is the tis argument with the earliest start date of all the series being plotted, the plot is shifted rightward to make the first bar align with `time(start(gdp))`.

You can use tisPlot to draw axes, axis labels, and tick marks for a barplot. First call tisPlot with the series you want to plot, and other arguments set to create the range, axes, labels, tick marks
and so on that you want, but set color = 0 to make the series lines invisible. Then call barplot with the series and additional barplot arguments, but set the add argument to add = TRUE. This adds the barplot, without axes, to the existing tisPlot.

**Value**

same as barplot.default.

**See Also**

barplot, barplot2

---

**barplot2**

*Horizontally Shifted Bar Plots*

**Description**

This function is a modified version of barplot with an additional argument that allows the bars to be shifted left or right along the x-axis.

**Usage**

```r
barplot2(height, width = 1, space = NULL,
         names.arg = NULL, legend.text = NULL, beside = FALSE,
         horiz = FALSE, density = NULL, angle = 45,
         col = NULL, border = par("fg"),
         main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
         xlim = NULL, ylim = NULL, xpd = TRUE, log = ",",
         axes = TRUE, axisnames = TRUE,
         cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
         inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
         add = FALSE, args.legend = NULL, x.offset = 0, ...)  
```

**Arguments**

- `height`: either a vector or matrix of values describing the bars which make up the plot. If `height` is a vector, the plot consists of a sequence of rectangular bars with heights given by the values in the vector. If `height` is a matrix and `beside` is FALSE then each bar of the plot corresponds to a column of `height`, with the values in the column giving the heights of stacked sub-bars making up the bar. If `height` is a matrix and `beside` is TRUE, then the values in each column are juxtaposed rather than stacked.

- `width`: optional vector of bar widths. Re-cycled to length the number of bars drawn. Specifying a single value will have no visible effect unless `xlim` is specified.
space

the amount of space (as a fraction of the average bar width) left before each bar. May be given as a single number or one number per bar. If height is a matrix and beside is TRUE, space may be specified by two numbers, where the first is the space between bars in the same group, and the second the space between the groups. If not given explicitly, it defaults to c(0, 1) if height is a matrix and beside is TRUE, and to 0.2 otherwise.

names.arg

a vector of names to be plotted below each bar or group of bars. If this argument is omitted, then the names are taken from the names attribute of height if this is a vector, or the column names if it is a matrix.

legend.text

a vector of text used to construct a legend for the plot, or a logical indicating whether a legend should be included. This is only useful when height is a matrix. In that case given legend labels should correspond to the rows of height; if legend.text is true, the row names of height will be used as labels if they are non-null.

beside

a logical value. If FALSE, the columns of height are portrayed as stacked bars, and if TRUE the columns are portrayed as juxtaposed bars.

col

a vector of colors for the bars or bar components. By default, grey is used if height is a vector, and a gamma-corrected grey palette if height is a matrix.

border

the color to be used for the border of the bars. Use border = NA to omit borders. If there are shading lines, border = TRUE means use the same colour for the border as for the shading lines.

main, sub

overall and sub title for the plot.

xlab

a label for the x axis.

ylab

a label for the y axis.

xlim

limits for the x axis.

ylim

limits for the y axis.

xpd

logical. Should bars be allowed to go outside region?

log

string specifying if axis scales should be logarithmic; see plot.default.

axes

logical. If TRUE, a vertical (or horizontal, if horiz is true) axis is drawn.

axisnames

logical. If TRUE, and if there are names.arg (see above), the other axis is drawn (with lty = 0) and labeled.

cex.axis

expansion factor for numeric axis labels.

cex.names

expansion factor for axis names (bar labels).

inside

logical. If TRUE, the lines which divide adjacent (non-stacked!) bars will be drawn. Only applies when space = 0 (which it partly is when beside = TRUE).
plot: logical. If FALSE, nothing is plotted.
axis.lty: the graphics parameter lty applied to the axis and tick marks of the categorical (default horizontal) axis. Note that by default the axis is suppressed.
offset: a vector indicating how much the bars should be shifted relative to the x axis.
add: logical specifying if bars should be added to an already existing plot; defaults to FALSE.
args.legend: list of additional arguments to pass to legend(); names of the list are used as argument names. Only used if legend.text is supplied.
x.offset: shifts the plot left or right along the x-axis.
...: arguments to be passed to/from other methods. For the default method these can include further arguments (such as axes, asp and main) and graphical parameters (see par) which are passed to plot.window(), title() and axis.

Details

barplot2 is a slightly modified version of barplot.default with an additional parameter (x.offset) that can shift the plot left or right. It was originally written for use by barplot.tis, but it can now also be called on its own.

Value

same as barplot.default, i.e., A numeric vector (or matrix, when beside = TRUE), say mp, giving the coordinates of all the bar midpoints drawn, useful for adding to the graph.
If beside is true, use colMeans(mp) for the midpoints of each group of bars, see example.

See Also

barplot, barplot.tis

-------------------------------
basis  Optional tis attributes-------------------------------

Description

tis series have (sometimes implicit) basis and observed attributes, used when aggregating or disaggregating to different frequencies.

Usage

basis(x)
basis(x) <- value
observed(x)
obscerved(x) <- value
Arguments

- x: a tis series
- value: a character string, see the details

Details

These (optional) attributes of a tis series are used when converting a series from one frequency to another.

A series basis is "business" or "daily", indicating whether the data values in a series are associated with a 5-day business week or a 7-day calendar week.

The observed attribute of series is one of the following:

- annualized: Specifies that each time series value is the annualized sum of observations made throughout the associated time interval. For time scale conversion and totaling purposes, this attribute is the same as averaged.
- averaged: Specifies that each time series value is the average of the observations made throughout the associated time interval.
- beginning: Specifies that each time series value represents a single observation made at the beginning of the associated time interval.
- end: Specifies that each time series value represents a single observation made at the end of the associated time interval.
- formula: Specifies that the time series represents a transformation of other series. For time scale conversion and totaling purposes, this attribute is the same as averaged.
- high: Specifies that each time series value is the maximum value for the time interval.
- low: Specifies that each time series value is the minimum value for the time interval.
- summed: Specifies that each time series value is the sum of observations made throughout the associated time interval.

Value

basis and observed return a character string. The assignment forms invisibly return x.

References

The FAME documentation, available from Sungard.

See Also

convert
between

Check for Inclusion in a Closed Interval

Description

Returns a logical vector like \( y \) showing if each element lies in the closed interval \([\min(x_1, x_2), \max(x_1, x_2)]\).

Usage

\[ \text{between}(y, x_1, x_2) \]

Arguments

- \( y \) a numeric object
- \( x_1 \) a number
- \( x_2 \) a number

Value

A logical object like \( y \).

Examples

\[
\text{mat} \leftarrow \text{matrix(rnorm(16), 4, 4)}
\]
\[
\text{mat}
\]
\[
\text{between(mat, -2, 1)}
\]

blanks

Blanks

Description

Takes an integer argument \( n \) and return a string of \( n \) blanks

Usage

\[ \text{blanks}(n) \]

Arguments

- \( n \) an integer
**capitalize**  
*Capitalize strings*

**Description**

Capitalizes characters that begin strings, or that follow a character that is not digit, underscore, or letter.

**Usage**

```r
capitalize(strings)
```

**Arguments**

- `strings` character vector

**Value**

A character vector like `strings` but capitalized.

**Author(s)**

Jeff Hallman

**See Also**

`toupper` and `tolower`

---

**cbind.tis  
*Combine Series Into a Multivariate (Matrix) Time Indexed Series*

**Description**

This is `cbind` for `tis` objects. It binds several `ts` and `tis` objects together into a single matrix time indexed series.

**Usage**

```r
## S3 method for class 'tis'
cbind(..., union = F)
```
Arguments

\[\ldots\]

any number of univariate or multivariate ts or tis objects. All will be converted to tis objects by \texttt{as.tis}, and the result series all must have the same tif (time index frequency).

union

a logical. If union = \texttt{F}, a matrix created by the intersection of the time windows for the arguments will be created. If union = \texttt{T}, the union of the time windows will be used to create the matrix.

Details

If union is \texttt{TRUE} and the series in \ldots do not all start and end on the same time index, the missing observations are filled with \texttt{NA}.

The column names of the returned series are determined as follows:

If an argument was given a name in the call, the corresponding column has that name. If the argument was itself a matrix with column names, those will be used, otherwise the argument's name is expanded with digits denoting its respective columns.

Value

a multivariate tis object.

Note

Class "ts" has it's own \texttt{cbind} method which knows nothing about tis objects. R generic functions like \texttt{cbind} dispatch on the class of their first argument, so if you want to combine tis and ts objects by calling the generic \texttt{cbind}, be sure that the first argument is a tis, not a ts. You can always ensure this is the case by wrapping the first argument in \ldots in \texttt{as.tis()}. 

See Also

\texttt{cbind}
Value

rows returns a list of the rows of z. If z has row names, those will also be the names of the returned list.

columns does the same, but for columns. Note that if z is some kind of time series, so too will be the elements of the returned list.

constantGrowthSeries  Constant Growth Series

Description

Create tis time series that grow at constant rates.

Usage

fanSeries(startValue, start, end, rates)
tunnelSeries(startValue, start, end, rate, spreads)

Arguments

- startValue: starting value for the series at time start
- start: a ti (Time Index) for the first observation.
- end: a ti or something that can be turned into a ti giving the time index for the last observation.
- rates: annual growth rate(s) for the series to be created
- rate: annual growth rate for the series to be created
- spreads: vector of 2 numbers giving the percentage values by which the starting values of the 'tunnel' series should be offset from startValue

Value

fanSeries returns a multivariate series that starts on start and ends on end. There are length(rates) columns. Each column begins at startValue and grows at the rate given by its corresponding element in rates. These are not true growth rates, rather each column has a constant first difference such that over the course of the first year, column i will grow rates[i] percent. This yields series that plot as straight lines.

tunnelSeries first calls fanSeries to create a univariate series running from start to end with a starting value of startValue and growing rate percent over the first year. It returns a bivariate series with columns that are offset from that series by spreads[1] and spreads[2] percent of the startValue.

See Also

growth.rate
Description

Convert tis series from one frequency to another using a variety of algorithms.

Usage

convert(x, tif, method = "constant", observed = observed(x),
        basis = basis(x), ignore = F)

Arguments

x  a univariate or multivariate tis series. Missing values (NAs) are ignored.
tif a number or a string indicating the desired ti frequency of the return series. See help(ti) for details.
method method by which the conversion is done: one of "discrete", "constant", "linear", or "cubic". Note that this argument is effectively ignored if observed is "high" or "low", as the "discrete" method is the only one supported for that setting.
observed. "observed" attribute of the input series: one of "beginning", "end", "high", "low", "summed", "annualized", or "averaged". If this argument is not supplied and observed(x) != NULL it will be used. The output series will also have this "observed" attribute.
basis. "daily" or "business". If this argument is not supplied and basis(x) != NULL it will be used. The output series will also have this "basis" attribute.
ignore governs how missing (partial period) values at the beginning and/or end of the series are handled. For method == "discrete" or "constant" and ignore == T, input values that cover only part the first and/or last output time intervals will still result in output values for those intervals. This can be problematic, especially for observed == "summed", as it can lead to atypical values for the first and/or last periods of the output series.

Details

This function is a close imitation of the way FAME handles time scale conversions. See the chapter on "Time Scale Conversion" in the Users Guide to Fame if the explanation given here is not detailed enough.

Start with some definitions. Combining values of a higher frequency input series to create a lower frequency output series is known as aggregation. Doing the opposite is known as disaggregation. If observed == "high" or "low", the "discrete" method is always used.

Disaggregation for "discrete" series: (i) for observed == "beginning" ("end"), the first (last) output period that begins (ends) in a particular input period is assigned the value of that input period. All other output periods that begin (end) in that input period are NA. (ii) for observed == "high", "low", "summed" or "averaged", all output periods that end in a particular input period are assigned the
same value. For "summed", that value is the input period value divided by the number of output
periods that end in the input period, while for "high", "low" and "averaged" series, the output period
values are the same as the corresponding input period values.

Aggregation for "discrete" series: (i) for observed == "beginning" ("end"), the output period is
assigned the value of the first (last) input period that begins (ends) in the output period. (ii) for
observed == "high" ("low"), the output period is assigned the value of the maximum (minimum)
of all the input values for periods that end in the output period. (iii) for observed == "summed"
("averaged"), the output value is the sum (average) of all the input values for periods that end in the
output period.

Methods "constant", "linear", and "cubic" all work by constructing a continuous function F(t) and
then reading off the appropriate point-in-time values if observed == "beginning" or "end", or by
integrating F(t) over the output intervals when observed == "summed", or by integrating F(t) over
the output intervals and dividing by the lengths of those intervals when observed == "averaged".
The unit of time itself is given by the basis argument.

The form of F(t) is determined by the conversion method. For "constant" conversions, F(t) is a step
function with jumps at the boundaries of the input periods. If the first and/or last input periods only
partly cover an output period, F is linearly extended to cover the first and last output periods as well.
The heights of the steps are set such that F(t) aggregates over the input periods to the original input
series.

For "linear" ("cubic") conversions, F(t) is a linear (cubic) spline. The x-coordinates of the spline
knots are the beginnings or ends of the input periods if observed == "beginning" or "end", else
they are the centers of the input periods. The y-coordinates of the splines are chosen such that
aggregating the resulting F(t) over the input periods yields the original input series.

For "constant" conversions, if ignore == F, the first (last) output period is the first (last) one for
which complete input data is available. For observed == "beginning", for example, this means that
data for the first input period that begins in the first output period is available, while for observed
== "summed", this means that the first output period is completely contained within the available
input periods. If ignore == T, data for only a single input period is sufficient to create an output
period value. For example, if converting weekly data to monthly data, and the last observation is
June 14, the output series will end in June if ignore == T, or May if it is F.

Unlike the "constant" method, the domain of F(t) for "linear" and "cubic" conversions is NOT
extended beyond the input periods, even if the ignore option is T. The first (last) output period is
therefore the first (last) one that is completely covered by input periods.

Series with observed == "annualized" are handled the same as observed == "averaged".

Value

a t i s time series covering approximately the same time span as x, but with the frequency specified
by t i f.

BUGS

Method "cubic" is not currently implemented for observed "summed", "annualized", and "aver-
aged".
csv

References

Users Guide to Fame

See Also

aggregate, tif, ti

Examples

```r
wSeries <- tis(1:105, start = ti(19950107, tif = "wsaturday"))
observed(wSeries) <- "ending"  ## end of week values
mDiscrete <- convert(wSeries, "monthly", method = "discrete")
mConstant <- convert(wSeries, "monthly", method = "constant")
mLinear <- convert(wSeries, "monthly", method = "linear")
mCubic <- convert(wSeries, "monthly", method = "cubic")
```

```r
## linear and cubic are identical because wSeries is a pure linear trend
cbind(mDiscrete, mConstant, mLinear, mCubic)
```

```r
observed(wSeries) <- "averaged"  ## weekly averages
mDiscrete <- convert(wSeries, "monthly", method = "discrete")
mConstant <- convert(wSeries, "monthly", method = "constant")
mLinear <- convert(wSeries, "monthly", method = "linear")
```

```r
cbind(mDiscrete, mConstant, mLinear)
```

---

**csv**

Writes a CSV (comma separated values) file.

---

**Description**

Write a matrix or Time Indexed Series to a .csv file that can be imported into a spreadsheet.

**Usage**

```r
csv(z, file = "", noDates = FALSE, row.names = !is.tis(z), ...)
```

**Arguments**

- **z**: matrix or tis object
- **file**: either a character string naming a file or a connection. If "", a file name is constructed by deparsing z. The extension "csv" is appended to the file name if it is not already there.
- **noDates**: logical. If FALSE (the default) and z is a tis object, the first column of the output file will contain spreadsheet dates.
either a logical value indicating whether the row names of \texttt{z} are to be written along with \texttt{z}, or a character vector of row names to be written. If \texttt{FALSE} (the default) and \texttt{z} is a \texttt{tis} object, the first column of the output file will contain spreadsheet dates.

... other arguments passed on to \texttt{write.table}.

Details

csv is essentially a convenient way to call \texttt{write.table}. If \texttt{file} is not a connection, a file name with the ".csv" extension is constructed. Next, a column of spreadsheet dates is prepended to \texttt{z} if necessary, and then csv calls

\texttt{write.table(z, file = filename, sep = ",", row.names = !is.tis(z), ...)}

Value

csv returns whatever the call to \texttt{write.table} returned.

See Also

\texttt{write.table}

---

\texttt{cumsum.tis} \hspace{1cm} \textit{Cumulative Sums, Products, and Extremes}

Description

Return a \texttt{tis} whose elements are the cumulative sums, products, minima or maxima of the elements of the argument.

Usage

\begin{verbatim}
## S3 method for class 'tis'
cumsum(x)
## S3 method for class 'tis'
cumprod(x)
## S3 method for class 'tis'
cummax(x)
## S3 method for class 'tis'
cummin(x)
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{x} \hspace{1cm} a \texttt{tis} series.
\end{itemize}

Details

These are \texttt{tis} methods for generic functions.
**currentMonday**

**Value**

A `tis` like `x`. An NA value in `x` causes the corresponding and following elements of the return value to be NA, as does integer overflow in `cumsum` (with a warning).

**See Also**

`cumsum`, `cumprod`, `cummin`, `cummax`

---

**currentMonday**

*Day of Week Time Indexes*

**Description**

Return daily `ti`’s for particular days of the week

**Usage**

```r
currentMonday(xTi = today())
currentTuesday(xTi = today())
currentWednesday(xTi = today())
currentThursday(xTi = today())
currentFriday(xTi = today())
currentSaturday(xTi = today())
currentSunday(xTi = today())
latestMonday(xTi = today())
latestTuesday(xTi = today())
latestWednesday(xTi = today())
latestThursday(xTi = today())
latestFriday(xTi = today())
latestSaturday(xTi = today())
latestSunday(xTi = today())
```

**Arguments**

- `xTi` a `ti` object or something that the `ti()` function can turn into a `ti` object

**Value**

`currentMonday` returns the daily `ti` for the last day of the Monday-ending week that its argument falls into. `currentTuesday` returns the daily `ti` for the last day of the Tuesday-ending week that its argument falls into, and so on for the other weekdays.

`latestMonday` returns the daily `ti` for the last day of the most recent completed Monday-ending week that its argument falls into. Ditto for the other days of the week.

**See Also**

`ti`
**currentPeriod**

<table>
<thead>
<tr>
<th>Current Period Time Indexes</th>
</tr>
</thead>
</table>

**Description**

Return a current ti of the desired frequency

**Usage**

```r
currentWeek(xTi = today())
currentMonth(xTi = today())
currentQuarter(xTi = today())
currentHalf(xTi = today())
currentYear(xTi = today())
currentQ4(xTi = today())
currentQMonth(xTi = today())
currentJanuary(xTi = today())
currentFebruary(xTi = today())
currentMarch(xTi = today())
currentApril(xTi = today())
currentMay(xTi = today())
currentJune(xTi = today())
currentJuly(xTi = today())
currentAugust(xTi = today())
currentSeptember(xTi = today())
currentOctober(xTi = today())
currentNovember(xTi = today())
currentDecember(xTi = today())
```

**Arguments**

- `xTi`: a ti object or something that the ti() function can turn into a ti object

**Details**

currentWeek returns the weekly ti for the week that its argument falls into. If the argument is itself a ti, the returned week contains the last day of the argument’s period. The default weekly frequency is "wmonday" (Monday-ending weeks), so currentWeek always returns wmonday ti’s. This can be changed via the setDefaultFrequencies function.

All of the other current{SomeFreq} functions work the same way, returning the ti’s of tif SomeFreq that the last day of their arguments period falls into. The tif’s for currentHalf and currentQ4 are "semiannual" and "quarterly", respectively. Finally, currentQMonth returns the quarter-ending month of the currentQuarter of its argument.

currentJanuary returns the monthly ti for January of the January-ending year that the last day of its argument falls into. currentFebruary returns the monthly ti for February of the February-ending year that the last day of its argument falls into, and so on.
**dateRange**

**Description**

Returns the starting and ending times of a series in a `ti` object of length 2.

**Usage**

```r
dateRange(x)
```

**Arguments**

- `x` a `ts` or `tis` time series

**Value**

A `ti` (Time Index) object of length two. The first element is the starting time index, while the second is the ending time index.

**See Also**

- `start`, `end`, `ti`, `tis`

**Examples**

```r
t = ts(1:24, start = c(2001, 1), freq = 12)
ti = as.tis(t)
dateRange(t)
dateRange(ti)
```
Description

Return position within a ti period, or a particular day within the period.

Usage

dayOfPeriod(xTi = today(), tif = NULL)
dayOfWeek(xTi = today())
dayOfMonth(xTi = today())
dayOfYear(xTi = today())
firstDayOf(xTi)
lastDayOf(xTi)
firstBusinessDayOf(xTi)
lastBusinessDayOf(xTi)
firstBusinessDayOfMonth(xTi)
lastBusinessDayOfMonth(xTi)
currentMonthDay(xTi, daynum)
latestMonthDay(xTi, daynum)

Arguments

xTi a ti object or something that the ti() function can turn into a ti object
tif a time index frequency code or name. See tif.
daynum day number in month

Details

The dayOfXXXX functions all work the same way, returning the day number of the XXXXX that jul(xTi) falls on. For example, if today is Thursday, January 5, 2006, then dayOfWeek(), dayOfMonth() and dayOfYear() are all 5. All of these are implemented via dayOfPeriod, which converts its first argument to a Julian date (via jul(xTi)) and finds the ti with frequency tif that day falls into. It returns the day number of the period represented by that time index that the Julian date falls on.

firstDayOf and lastDayOf return a daily ti for the first or last day of the period represented by xTi. firstBusinessDayOf and lastBusinessDayOf do the same but the returned ti has business daily frequency.

firstBusinessDayOfMonth returns a business daily ti for the first business day of the month of xTi. lastBusinessDayOfMonth does the same but for the last business day of the month of xTi.

currentMonthDay returns a daily ti for the next upcoming daynum’th of the month. latestMonthDay does the same for the most recent daynum’th of the month.

currentMonday returns the daily ti for the last day of the Monday-ending week that its argument falls into. The other current{Weekday} functions work the same way.
Value

All of the functions except the `dayOfxxxxx` return ti objects as described in the details section above. The `dayOfxxxxx` functions return numbers.

Note

None of these business-day functions take account of holidays, so `firstBusinessDayOfMonth(20010101)`, for example, returns January 1, 2001 which was actually a holiday. To see how to handle holidays, look at the `holidays` and `nextBusinessDay` help pages.

See Also

ti, tif, jul, holidays, nextBusinessDay, previousBusinessDay
**evalOrEcho**

*Evaluate an Argument If Possible*

**Description**

This function parses, evaluates and returns the string given as its first argument. If it can't, the argument itself is returned. Use of `evalOrEcho` to process arguments inside a function can make for more flexible code.

**Usage**

```r
evalOrEcho(x, resultMode = NULL, n = 0)
```

**Arguments**

- `x`: a string or other object to attempt to parse and evaluate.
- `resultMode`: a string or NULL. If non-NULL, the evaluation of `x` is considered to have failed if the resulting object is not of this mode.
- `n`: parent generations to go back. The evaluation is attempted in the environment specified by `parent.frame(n)` (of the caller).

**Details**

Using this function inside another function to process some of its arguments can be very useful. For example, `tisPlot` has a number or arguments that specify text labels for headers, subheaders, footnotes, axis labels, and so on. One of those arguments is `sub`, which specifies the subheader. By doing this:

```r
sub <- evalOrEcho(sub, resultMode = "character")
```

`tisPlot` can handle the `sub` argument given in any of these forms:

1. `sub = "This is a simple subtitle"`.
2. `sub = c("this is a two", "line subtitle")`.
3. `sub = c("this is another", "two line subtitle")`.

**Value**

If `x` is successfully parsed and evaluated, and its mode matches `resultMode` (if supplied), the resulting object is returned. Otherwise, `x` itself is returned.

**See Also**

`try`
format.ti

Convert Time Index or Jul to Character

Description

format formats a jul or time index object for printing. as.character for a jul or ti object is essentially just an alias for format.

Usage

```r
## S3 method for class 'ti'
format(x, ..., tz = quote(UTC))
## S3 method for class 'jul'
format(x, ...)
## S3 method for class 'ti'
as.character(x, ...)
## S3 method for class 'jul'
as.character(x, ...)
```

Arguments

- `x` a jul or ti (time index) object
- `tz` A timezone specification to be used for the conversion if x has an intraday frequency. System-specific, but """ is the current time zone, and "GMT" is UTC.
- `...` other args passed on to format.POSIXlt.

Details

The as.character methods do nothing but call the corresponding format methods. x is converted to a POSIXlt object and then format.POSIXlt takes over.

Value

a character vector representing x

Note

format.POSIXlt has been modified to understand two additional format symbols in addition to those documented in link{strftime}: "%q" in the format string is replaced with the quarter number (1 thru 4) and "%N" is replaced with the first letter of the month name.

See Also

format.POSIXlt, strftime
Examples

format(today() + 0:9, "%x")
as.character(jul(today()))

fortify.tis  Fortify a tis object

Description

A fortify method for tis objects

Usage

## S3 method for class 'tis'
fortify(x, offset = 0.5, dfNames = NULL, timeName = "date")

Arguments

- **x**: A tis object of time series
- **offset**: A number between 0 and 1 specifying where in the period of time represented by the 'ti(x)' the points should eventually be plotted in ggplot2. 'offset = 0' gives the beginning of the period and 'offset = 1' the end of the period, 'offset = 0.5' the middle of the period, and so on. For example if x is a tis object of quarterly time series and offset = 0.5 then the resulting plotted points would fall in the middle of each quarter. offset is passed on to POSIXct(ti(x), offset=offset) and used to create the field date in the resulting data frame.
- **dfNames**: A character vector of the names for the tis objects contained in x. Defaults to the name of the tis object in the univariate case and the column names of the tis object in the multivariate case.
- **timeName**: A character vector of length one with the desired name for the column of dates that will be created from the tis object time index. Default name is "date".

Details

This function turns a tis object into a data frame containing the original time series plus a field of dates adjusted by an 'offset', so that the time series can be more easily plotted with ggplot2.

Author(s)

Trevor Davis

See Also

fortify
frColors

Examples

```r
if(require("ggplot2") && require("reshape")) {
  # Examples of plotting tis series with ggplot2
  require("datasets")
  require("scales")

  # univariate example
  num_discoveries <- as.tis(discoveries)
  ggplot(data = fortify(num_discoveries, offset=0)) +
    geom_line(aes(x=date, y=num_discoveries)) +
    scale_x_date(breaks = date_breaks("10 years"), labels = date_format("%Y"))

  # multivariate example using the "melt trick"
  Seatbelts.tis <- as.tis(Seatbelts[, c("drivers", "front", "rear")])
  Seatbelt.names <- c("Driver", "Front Seat Passenger", "Back Seat Passenger")
  Seatbelts.df <- fortify(Seatbelts.tis, dfNames = Seatbelt.names,
                          timeZone = "Time")
  Seatbelts.df.m <- melt(Seatbelts.df, id.var = "Time", variable_name="type")
  qplot( Time, value, data = Seatbelts.df.m, geom="line",
         group=type, colour=type, linetype=type ) +
         geom_vline(xintercept=as.numeric(as.Date("1983-01-31")),
                     colour="black", linetype="dashed") +
         ylab("Monthly Road Casualties in the UK")
}
```

frColors

**FRB Color Palettes**

**Description**

Returns a vector of 11 color names that can be used as a palette suitably for a light or dark background.

**Usage**

```r
frColors(dark = FALSE)
```

**Arguments**

- **dark**
  
  If TRUE return a set of colors suitable for drawing on a black background. The first color is "white". If FALSE (the default), return colors suitable for drawing on a white background, with the first color set to "black".

**Value**

A vector of 11 color names.

**Author(s)**

Jeff Hallman
growth.rate

See Also

setColors, colors

Examples

```r
plot.new()
for(i in 1:11) abline(h = (i-0.5)/11, lwd = 2, lty = 1, col = frColors()[i])
plot.new()
for(i in 1:11) abline(h = (i-0.5)/11, lwd = 2, lty = 1, col = frColors(dark=TRUE)[i])
```

---

growth.rate  Growth Rates of Time Series

Description

Get or set growth rates of a time series in annual percent terms.

Usage

```r
growth.rate(x, lag = 1, simple = T)
growth.rate(x, start = end(x) + 1, simple = T) <- value
```

Arguments

- `x`: a time series or something that can be turned into one by `as.tis`
- `lag`: number of lags to use in calculating the growth rate as outlined in the details below
- `simple`: simple growth rates if `TRUE`, compound growth rates if `FALSE`
- `start`: the first time index for which values of `x` should be replaced to make `growth.rate(x[start]) == value`
- `value`: desired growth rates

Details

An example: Suppose `x` is a quarterly series, then if `simple` is `TRUE`,

```r
growth.rate(x, lag = 3) == 100 * ((x[t]/x[t-3]) - 1) * (4/3)
```

while if `simple` is `FALSE`

```r
growth.rate(x, lag = 3) == 100 * ((x[t]/x[t-3])^(4/3) - 1).
```

Value

- `growth.rate(x)` returns a time series of growth rates in annual percentage terms.
- Beginning with the observation indexed by `start`,
  ```r
growth.rate(x) <- value
  ```

sets the values of `x` such that the growth rates in annual percentage terms will be equal to `value`. `x` is extended if necessary. The modified `x` is invisibly returned.
**hms**

*Hours, Minutes and Seconds from a Time Index or Jul*

**Description**

Extract the fractional part of a `ti` (time index) or `jul` (julian date) object as a normalized list of hours, minutes, and seconds.

**Usage**

```r
hms(x)
```

**Arguments**

- `x` a `jul` or something numeric that can be converted into a `jul` with a fractional part.

**Details**

The fractional part of `x` is multiplied by 86400 (the number of seconds in a day) and rounded to get the number of seconds. This is then divided by 3600 to get the number of hours, and the remainder of that is divided by 60 to get the normalized number of minutes. The remainder from the second division is the normalized number of seconds.

**Value**

A list with components:

- `hours` Normalized number of hours
- `minutes` Normalized number of minutes
- `seconds` Normalized number of seconds

See the details.

**Note**

Support for fractional days in `ti` and `jul` objects is relatively new and untested. There is probably code lurking about that assumes the numeric parts of `ti` and `jul` objects are integers, or even code that may round them to make sure they are integers. The fractional parts of `ti` and `jul` objects may not survive encounters with such code.

**See Also**

- `ti` and `jul`. Also see `hourly` for information on intraday frequencies
Examples

```r
hms(today() + 0.5)
hms(today())
hms(today()) + 43201/86400
```

---

**holidays**  
*Holidays*

**Description**

Functions that know about Federal and FRB (Federal Reserve Board) holidays.

**Usage**

```r
nextBusinessDay(x, holidays = NULL, goodFriday = F, board = F, inaug = board)
previousBusinessDay(x, holidays = NULL, goodFriday = F, board = F, inaug = board)
isHoliday(x, goodFriday = F, board = F, inaug = board, businessOnly = T)
isBusinessDay(x, ...)
isGoodFriday(x)
isEaster(x)
holidays(years, goodFriday = F, board = F, inaug = board, businessOnly = T)
federalHolidays(years, board = F, businessOnly = T)
goodFriday(years)
easter(years)
ingaurationDay(years)
holidaysBetween(startTi, endTi, goodFriday = F, board = F, inaug = board,
  businessOnly = T)
```

**Arguments**

- `x`  
a ti time index, or something that can be turned into one, such as a `yyyymmdd` number or a `Date` object.

- `holidays`  
a vector of holidays (in `yyyymmdd` form) to skip over, or `NULL`. In the latter case, the `holidays` function is used to determine days to skip over.

- `goodFriday`  
if TRUE, consider Good Friday as a holiday. Default is FALSE because Good Friday is not a federal holiday.

- `board`  
if TRUE, the Friday preceding a Saturday NewYear's, Independence, Veterans or Christmas Day is considered a holiday.

- `inaug`  
if TRUE, consider the U.S. Presidential Inauguration Day to be a holiday, but only if it falls on a weekday.

- `businessOnly`  
if TRUE (the default), ignore Saturday NewYear's, Independence, Veterans and Christmas Day holidays. Has no effect if `board` is TRUE, since that moves Saturday holidays to Friday.

- `...`  
arguments passed on to `isHoliday`

- `years`  
numeric vector of 4 digit years
**Details**

Federal law defines 10 holidays. Four of them, New Years, Independence, Veterans and Christmas, fall on the same date every year. The other six fall on particular days of the week and months (MLK, Presidents, Memorial, Labor, Columbus, and Thanksgiving).

If one of the four fixed-date holidays falls on a Sunday, the federal holiday is celebrated the next day (Monday). If it falls on a Saturday, the preceding day (Friday) is a holiday for the Federal Reserve Board, but not for the Reserve Banks and the banking system as a whole.

Presidential Inauguration day is a Federal holiday only in the DC area, and then only if it falls on a weekday, so it is not included in the holidays returned by `federalHolidays`, but it can be included in several of the other functions by setting the `inaug` argument to `TRUE`.

The function `isBusinessDay` returns `TRUE` for `x` if and only if `x` is not a holiday, a Saturday or a Sunday.

**Value**

`nextBusinessDay` and `previousBusinessDay` return "business" frequency `ti` objects.

`isHoliday`, `isGoodFriday`, `isEaster` and `isBusinessDay` return Boolean vectors as long as `x`.

`easter` and `goodFriday` return numeric vectors of `yyyymmdd` dates of the appropriate holidays for each year in the `years` argument.

`inaugurationDay` returns a numeric vector of `yyyymmdd` dates of U.S. Presidential Inauguration Days, if any, that fall in the years given in the `years` argument.

`federalHolidays` returns a numeric vector of `yyyymmdd` dates for the federal holidays for each year in `years`. The `names` attribute of the returned vector contains the holiday names.

`holidays` returns a vector like `federalHolidays` does. The only difference between the two functions is that `holidays` has the option of including Good Fridays.

`holidaysBetween` returns a vector of `yyyymmdd` dates for holidays that fall within the time spanned by `[startTi, endTi]`.

**Note**

The algorithm for finding Easter dates was found somewhere on the web (I don’t remember where) and is unbelievably complex. It would probably be simpler to just celebrate the home opener of the Cleveland Indians instead.
inferTi

*Create a Time Index to Match a Vector of DateTimes*

**Description**

Attempts to find a tif (Time Index Frequency) that “fits” the supplied dateTimes, and returns a ti object that hits the same dates and/or times.

**Usage**

`inferTi(dateTimes)`

**Arguments**

dateTimes a vector POSIXct object, or something that as.POSIXct can coerce into one.

**Value**

a ti object as long as the input

**Note**

May fail if there is no tif that closely matches the inputs.

**See Also**

`as.ti`

**Examples**

`inferTi(Sys.time()+c(1:5)*86400)`

interpNA

*Interpolate missing values in a Time Indexed Series*

**Description**

Calls `approxfun` or `splinefun` to interpolate missing values in a tis object.

**Usage**

`interpNA(x, method = "constant", useTimes = F, offset = 1, rule = 2, f = 0, ...)`
Arguments

x  
a tis time series

method  
One of c("constant", "linear", "fmm", "natural", "periodic"). Methods "constant" and "linear" call approxfun; the others call splinefun.

useTimes  
if TRUE, use time(x, offset) (the decimal times of x) as the 'x' part of the (x, y) pairs used for interpolation. If FALSE (the default), use ti(x) (the integer time indices of x) as the 'x' part of the (x, y) pairs.

offset  
if useTimes is TRUE, a number in the range [0,1] telling where in the periods represented by ti(x) to get the points for the 'x' parts of the (x, y) pairs. See the help for jul for a more detailed explanation of this parameter.

rule  
For methods "constant" and "linear": an integer describing how interpolation is to take place outside the interval [min(x), max(x)]. If rule is 1 then NAs are returned for such points and if it is 2, the value at the closest data extreme is used.

f  
For method="constant" a number between 0 and 1 inclusive, indicating a compromise between left- and right-continuous step functions. If y0 and y1 are the values to the left and right of the point then the value is y0*(1-f)+y1*f so that f=0 is right-continuous and f=1 is left-continuous.

...  
Other arguments passed along to approxfun for methods "constant" and "linear".

Details

Depending on the method specified, a call to either approxfun or splinefun is constructed with appropriate arguments and executed for each column of x. In the call to approxfun or splinefun, the time indices ti(x) (or the decimal times returned by time(x, offset), if useTimes is TRUE) serve as the 'x' argument and the column values as the 'y' argument.

Value

A tis object like x with NA values filled in by interpolated values.

See Also

approxfun, splinefun, ti

Description

create tif (TimeIndexFrequency) codes for hourly, minutely, and secondly ti’s.
Usage

hourly(n = 0)
minutely(n = 0)
secondly(n = 0)

Arguments

n number of base periods to skip. That is, hourly(2) gives a tif code for a series observed every 2nd hour, while both minutely(1) and minutely(1) are for a series observed once per minute, secondly(30) means every 30 seconds, and so on.

Details

The current implementation has hourly(n) \rightarrow 2000 + n, minutely(n) \rightarrow 3000 + n, and secondly(n) \rightarrow 4000 + n. If n divides evenly into 3600 for secondly(n), the return code will be the same as hourly(n/3600). For secondly(n) and minutely(n), if n divides evenly into 60, the return code will be as if minutely(n/60) or hourly(n/60) had been called, respectively.

For hourly(n), n must evenly divide into 24 and be less than 24, i.e., n is one of 1, 2, 3, 4, 6, 8, 12.

For minutely(n), n must be an even divisor of 1440, and less than 720. For secondly(n), n must divide evenly into 86400, and be no larger than 960.

Value

An integer tif code.

See Also

tif

isIntradayTif

Check for Intraday Time Index Frequency

Description

The intraday frequencies are hourly(n), minutely(n) and secondly(n), where n is an appropriate integer. Their numeric tif codes are between 2000 and 4900, and that is what is actually checked for.

Usage

isIntradayTif(tif)

Arguments

tif a character vector of tif names (see tifName) or a numeric vector of tif codes (see tif to be checked
isLeapYear

Value

A logical vector as long as the input indicating which elements are intraday Time Index frequencies.

Note

The function does not attempt to verify if the supplied tif is actually valid, intraday or not.

See Also

hourly, minutely, secondly

Examples

isIntradayTif(hourly(6))
isIntradayTif(tif(today()))
isIntradayTif(minutely(30))

isleapYear

Check Leap Year

Description

Checks whether or not the elements of its input are leap years.

Usage

isleapYear(y)

Arguments

y numeric vector of years

Details

y is a leap year if it is evenly divisible by 4 and either it is not evenly divisible by 100 or it is evenly divisible by 400, i.e., y%%4 == 0 & (y%%100 != 0 | y%%400 == 0).

Value

logical vector of same length as y indicating whether or not the given years are leap years.

Examples

isleapYear(c(1899:2004))
Description

The function `jul` is used to create `jul` (julian date) objects, which are useful for date calculations. `as.jul` and `asJul` coerce an object to class "jul", the difference being that `as.jul` calls the constructor `jul`, while `asJul` simply forces the class of its argument to be "jul" without any checking as to whether or not it makes sense to do so.

`is.jul` tests whether an object inherits from class "jul".

Usage

```r
jul(x, ...)  
## S3 method for class 'Date'
jul(x, ...)  
## S3 method for class 'IDate'
jul(x, ...)  
## S3 method for class 'ti'
jul(x, offset = 1, ...)  
## S3 method for class 'yearmon'
jul(x, offset = 0, ...)  
## S3 method for class 'yearqtr'
jul(x, offset = 0, ...)  
## Default S3 method:
jul(x, ...)  
as.jul(x, ...)  
asJul(x)  
is.jul(x)
```

Arguments

- `x` object to be tested (`is.jul`) or converted into a `jul` object. As described in the details below, the constructor function `jul` can deal with several different kinds of `x`.

- `...` other args to be passed to the method called by the generic function. `jul.default` may pass these args to `as.Date`.

- `offset` For `jul.ti`, a number in the range [0,1] telling where in the period represented by `x` to find the day. 0 returns the first day of the period, while the default value 1 returns the last day of the period. For example, if `x` has `tif` = "wmonday" so that `x` represents a week ending on Monday, than any `offset` in the range [0, 1/7] will return the Tuesday of that week, while `offset` in the range (1/7, 2/7] will return the Wednesday of that week, `offset` in the range (6/7, 1] will return the Monday that ends the week, and so on.

`jul.yearmon` and `jul.yearqtr` work on `yearmon` and `yearqtr` objects from `zoo`. Note that the default `offset` for these functions is 0, not 1, as that is how
the other index-to-date functions in zoo work, i.e., if \texttt{ym} is a yearmon object, then \texttt{as.Date(ym)} and \texttt{as.jul(ym)} should give the same date.

Details

The jul’s for any pair of valid dates differ by the number of days between them. \texttt{R}’s \texttt{Date} class defines a Date as a number of days elapsed since January 1, 1970, but jul uses the encoding from the \textit{Numerical Recipes} book, which has Jan 1, 1970 = 2440588, and the code for converting between ymd and jul representations is a straightforward port of the code from that tome.

Adding an integer to, or subtracting an integer from a jul results in another jul, and one jul can be subtracted from another. Two jul’s can also be compared with the operators (==, !=, <, >, <=, >=).

The jul class implements methods for a number of generic functions, including \texttt{[1"}, \texttt{as.Date}, \texttt{as.POSIXct, as.POSIXlt, c, format, max, min, print, rep, seq, ti, time, ymd}.

jul is a generic function with specialized methods to handle \texttt{Date} and \texttt{ti} objects. A recent addition is a method to handle \texttt{Date} objects as defined in the \textit{data.table} package.

The default method (\texttt{.default}) deals with character \texttt{x} by calling \texttt{as.Date} on it. Otherwise, it proceeds as follows:

If \texttt{x} is numeric, isYmd is used to see if it could be yyyymmdd date, then istime is called to see if \texttt{x} could be a decimal time (a number between 1799 and 2200). If all else fails, \texttt{as.Date(x)} is called to attempt to create a Date object that can then be used to construct a jul.

Value

\texttt{is.jul} returns \texttt{TRUE} or \texttt{FALSE}.

\texttt{as.jul} and \texttt{asJul} return objects with class "jul".

\texttt{jul} constructs a jul object like \texttt{x}.

\texttt{jul} with no arguments returns the jul for the current day.

Note

The return value from \texttt{asJul} is not guaranteed to be a valid jul object. For example, \texttt{asJul("a")} will not throw an error, and it will return the string "a" with a class attribute "jul", but that’s not a valid julian date.

The Julian calendar adopted by the Roman Republic was not accurate with respect to the rotational position of the Earth around the sun. By 1582 it had drifted ten days off. To fix this, Pope Gregory XIII decreed that the day after October 4, 1582 would be October 15, and that thereafter, leap years would be omitted in years divisible by 100 but not divisible by 400. This modification became known as the Gregorian calendar. England and the colonies did not switch over until 1752, by which time the drift had worsened by another day, so that England had to skip over 11 days, rather than 10.

The algorithms used in \texttt{jul2ymd} and \texttt{ymd2jul} cut over at the end of October 1582.

References

See Also

`jul, ymd, ti, as.Date`

Examples

```r
dec31 <- jul(20041231)
jan30 <- jul("2005-1-30")
jan30 - dec31  # 30
feb28 <- jan30 + 29
jul()         # current date
```

lags

### Lag a Time Series

Description

`lag` creates a lagged version of a time series, shifting the time base forward by a given number of observations. Lag does exactly the opposite, shifting the time base backwards by the given number of observations. `lag` and `Lag` create a single lagged series, while `lags` and `Lags` can create a multivariate series with several lags at once.

Usage

```r
## S3 method for class 'tis'
lag(x, k = 1, ...)
```

```r
Lag(x, k = 1, ...)
lags(x, lags, name = "")
Lags(x, lags, name = "")
```

Arguments

- `x`: A vector or matrix or univariate or multivariate time series (including `tis` series)
- `k`: The number of lags. For `lag`, this is the number of time periods that the series is shifted `forward`, while for `Lag` it is the number of periods that the series is shifted `backwards`.
- `...`: further arguments to be passed to or from methods
- `lags`: vector of lag numbers. For code `lags`, each element gives a number of periods by which `x` is to be shifted `forward`, while for `Lags`, each element gives a number of periods by which `x` is to be shifted `backwards`.
- `name`: string or a character vector of names to be used in constructing column names for the returned series
Details

Vector or matrix arguments ‘x’ are coerced to time series.

For lags, column names are constructed as follows: If name is supplied and has as many elements as x has columns, those names are used as the base column names. Otherwise the column names of x comprise the base column names, or if those don’t exist, the first ncols(x) letters of the alphabet are used as base names. Each column of the returned series has a name consisting of the basename plus a suffix indicating the lag number for that column.

Value

Both functions return a time series (ts or tis) object. If the lags argument to the lags function argument has more than one element, the returned object will have a column for each lag, with NA’s filling in where appropriate.

latestPeriod

| Most Recent Period Time Indexes |

Description

Return a ti for the most recent period of the desired frequency.

Usage

latestWeek(xTi = today())
latestMonth(xTi = today())
latestQuarter(xTi = today())
latestHalf(xTi = today())
latestYear(xTi = today())
latestQ4(xTi = today())
latestJanuary(xTi = today())
latestFebruary(xTi = today())
latestMarch(xTi = today())
latestApril(xTi = today())
latestMay(xTi = today())
latestJune(xTi = today())
latestJuly(xTi = today())
latestAugust(xTi = today())
latestSeptember(xTi = today())
latestOctober(xTi = today())
latestNovember(xTi = today())
latestDecember(xTi = today())

Arguments

xTi a ti object or something that the ti() function can turn into a ti object
Details

The latest{whatever} functions are the same as the corresponding current{whatever} functions, except that they return the most recent completed ti of the desired frequency. A period is considered to be completed on its last day. For example, if today is Thursday, then latestWedweek() returns the week that ended yesterday. Yesterday it would have returned the same week, but the day before that (Tuesday) it would have returned the "wwednesday" ti for the week that had ended six days before.

latestWeek returns the weekly ti for the most recently completed week as of xTi. If the xTi is itself a ti, the returned week is the most recently completed week as of the last day of xTi. (Note that the default weekly frequency is "wmonday" (Monday-ending weeks), so latestWeek always returns "wmonday" ti's.) See setDefaultFrequencies to change this.

All of the other latest{SomeFreq} functions work the same way, returning the ti's for the most recently completed SomeFreq as of the last day of xTi. The ti's (frequencies) for latestHalf and latestQ4 are "semiannual" and "quarterly", respectively.

latestJanuary returns the monthly ti for January of the most recently completed January-ending year that the last day of its argument falls into. latestFebruary returns the monthly ti for February of the most recently completed February-ending year that the last day of its argument falls into, and so on.

Value

All return return ti objects as described in the details.

See Also

ti, tif, currentWeek setDefaultFrequencies

linearSplineIntegration

Linear Spline Integration

Description

lintegrate gives the values resulting from integrating a linear spline, while ilspline returns linear splines that integrate to given values.

Usage

lintegrate(x, y, xint, stepfun = F, rule = 0)
ilspline(xint, w)
Arguments

- **x**
  - x coordinates of the linear spline \( F \) defined by \((x, y)\)

- **y**
  - y coordinates of the linear spline \( F \) defined by \((x, y)\)

- **xint**
  - x intervals, i.e. \([x[1], x[2]]\) is the first interval, \([x[2], x[3]]\) is the second interval, and so on.

- **stepfun**
  - if TRUE, \( F \) is a left-continuous step function. The default (FALSE) says \( F \) is continuous.

- **rule**
  - one of \(\{0, 1, \text{NA}\}\) to specify the behavior of \( F \) outside the range of \( x \). Use zero if \( F \) is zero outside the range of \( x \), NA if \( F \) is NA outside the range of \( x \), and one if \( F \) is to be linearly extended outside the range of \( x \).

- **w**
  - values the linear spline must integrate to

Details

\texttt{lintegrate} integrates the linear spline \( F \) defined by \((x, y)\) over the \( xint \) intervals. The value of \( F \) outside the range of \( x \) is specified by the \( \text{rule} \) argument:

\[
\begin{align*}
\text{rule} &= 0 & \rightarrow & F(z) = 0 \text{ for } z \text{ outside the range of } x \\
\text{rule} &= \text{NA} & \rightarrow & F(z) = \text{NA} \text{ for } z \text{ outside the range of } x \\
\text{rule} &= 1 & \rightarrow & F(z) \text{ extended for } z \text{ outside the range of } x
\end{align*}
\]

If \( \text{stepfun} \) is TRUE, \( F(z) \) is assumed to be a left-continuous step function and the last value of \( y \) is never accessed.

\((x[i], y[i])\) pairs with NA values in either \( x[i] \) or \( y[i] \) NA are ignored in constructing \( F \).

\texttt{ilspline} finds linear splines that integrate over the \( N \) intervals specified by the monotonically increasing \( N+1 \) vector \( xint \) to the \( N \) values given in \( w \). The function finds \( N \)-vectors \( x \) and \( y \) such that:

\[
\begin{align*}
(i) & \quad x[j] = (xint[j-1] + xint[j])/2, \text{ i.e., the values of } x \text{ are the midpoints of the intervals specified by } xint, \text{ and} \\
(ii) & \quad \text{the linear spline that passes through the } (x[i], y[i]) \text{ pairs (and is extended to } xint[1] \text{ and } xint[N+1] \text{ by linear extrapolation) integrates over each interval } [xint[j], xint[j+1]] \text{ to } w[j] \text{.}
\end{align*}
\]

In fact, \( w \) can actually be a \( M \) by \( N \) matrix, in which case the \( y \) found by the function is also an \( M \) by \( N \) matrix, with each column of \( y \) giving the \( y \) coordinates of a linear spline that integrates to the corresponding column of \( w \).

Value

\texttt{lintegrate} returns a vector of length \( \text{length}(xint) - 1 \).

\texttt{ilspline} returns a list with components named 'x' and 'y'.
See Also

spline, approx

Examples

```r
w <- 10 + cumsum(rnorm(10))
blah <- ilspline(1:11, w)
ww <- lintegrate(blah$X, blah$Y, 1:11, rule = 1)
w - ww  ## should be all zeroes (or very close to zero)
```

Description

Plotting methods for tis objects

Usage

```r
## S3 method for class 'tis'
lines(x, offset = 0.5, dropNA = FALSE, ...)
## S3 method for class 'tis'
points(x, offset = 0.5, dropNA = FALSE, ...)
```

Arguments

- `x`: a tis (time indexed series) object
- `offset`: a number in the range [0,1] telling where in each period of `x` to plot the point. 0 means the first second of each period, 1 the last second of the period, and the default 0.5 plots each point in the middle of the time period in which it falls.
- `dropNA`: if TRUE, observations with NA values are dropped before calling `lines.default` or `points.default`. See the details for why you might or might not want to do this. The default is FALSE, to match the behavior of `lines.default` and `points.default`.
- `...`: other arguments to be passed on to `lines.default` or `points.default`.

Details

These are fairly simple wrappers around the `lines.default` and `points.default`. For example, `lines.tis` basically does this:

```r
lines.default(x = time(x, offset = offset), y = x, ...)
```

and `points.tis` is similar. If `dropNA` is TRUE, the observations in `x` that are NA are dropped from the `x` and `y` vectors sent to the .default functions. For `points`, this shouldn’t matter, since `points.tis` omits points with NA values from the plot.

For `lines` the `dropNA` parameter does make a difference. The help document for `lines` says:
"The coordinates can contain NA values. If a point contains NA in either its x or y value, it is omitted from the plot, and lines are not drawn to or from such points. Thus missing values can be used to achieve breaks in lines."

Note that if the type is one of c("p", "b", "o"), the non-NA points are still drawn, but line segments from those points to adjacent NA points are not drawn. If dropNA = TRUE, the NA points are dropped before calling lines.default, and all of the remaining points will be connected with line segments (unless suppressed by the type argument).

See Also

lines, points
Description

Windows a tis or ts time series to cut off leading and trailing NA (and optionally zero) observations.

Usage

naWindow(x, union = F, zero = F)

Arguments

x a tis or ts time series
union see details below
zero if TRUE, chop off leading and trailing zeroes also

details

For multivariate (multiple columns) series and union = TRUE, a row of x is considered to be NA if and only if all entries in that row are NA. If union = FALSE (the default), a row is considered to be NA if any of its entries is NA.

if zero is TRUE, the function chops off leading and trailing observations that are either NA or zero. Otherwise, it chops only NAs.

Value

A copy of x with leading and trailing NA observations deleted.

See Also

window

Description

nberDates returns a matrix with two columns of yyyyymmdd dates giving the Start and End dates of recessions fixed by the NBER.

nberShade is a generic method for shading recession areas on the current plot. The default version calls nberDates() to get a matrix of yyyyymmdd dates and then passes those dates and all other arguments along to ymdShade.

romerLines draws vertical lines on the current plot at the "Romer and Romer" dates when monetary policy is said to have become contractionary.
Usage

## Default S3 method:
nberShade(...)
nberDates()
romerLines()

Arguments

... args passed to `ymdShade`

Value

`nberDates` returns the two column matrix of recession date ranges described above. Nothing useful is returned by the other functions.

Note

Recessions are dated by the Business Cycle Dating Committee of the National Bureau of Economic Research.


References


See Also

`polygon, nberShade.ggplot`

Examples

```r
require("datasets")
plot(presidents, type='n', ylab="Presidents approval rating")
nberShade()
lines(presidents)
```
nberShade.ggplot

Plotting NBER Recessions in the ggplot graphics system

Description

nberShade.ggplot is a method for shading recession areas in a plot of class ggplot. It calls nberDates to get x coordinates for the areas to be shaded and then passes those coordinates along with its own arguments to geom_rect to do the shading. It may also draw a vertical line at the appropriate location if openshade is false and its xrange is of an appropriate size.

Usage

```r
## S3 method for class 'ggplot'
nberShade(gg = ggplot2::last_plot(), fill = "grey80", color = NA, size = 0.5,
          alpha = 0.5, xrange = NULL, openshade = TRUE, ...)
```

Arguments

- `gg`  
  A ggplot object that contains a mapping of y to its data. The default is to use the last ggplot object that was plotted.

- `fill`  
  color to shade recessionary periods, passed to geom_rect.

- `color`  
  The default (NA) omits borders on the shaded regions. Alternatively, specify a border color. Passed to geom_rect

- `size`  
  A number specifying how thick should the border lines be. Passed to geom_rect

- `alpha`  
  A number specifying how transparent the shaded regions should be. Passed to geom_rect

- `xrange`  
  Horizontal range over which recession shading should be drawn. The default value NULL uses the entire range of the recession dates. xlim and scale_x_date are convenient ggplot2 functions to afterwards restrict the plot to a more sensible range.

- `openshade`  
  governs how nberShade handles the case where the last row of the matrix returned by nberDates has an NA in the "End" column, indicating that the end date of the most recent recession is not yet known. If TRUE and xrange has been specified and extends far enough into the future it will replace the NA with xrange[2]. Else if openshade is TRUE and xrange is NULL then it will replace the NA with Sys.Date(). If openshade is FALSE then nberShade will draw a vertical line at the start of the most recent recession.

Value

Returns a ggplot object.
Note

Recessions are dated by the Business Cycle Dating Committee of the National Bureau of Economic Research.

References


See Also

`geom_rect`, `nberDates`, `nberShade`

Examples

```r
## Not run:
require("ggplot2")
ggp <- ggplot(aes(x=date, y=100*unemploy/pop), data=economics)

nberShade( ggp ) + xlim(as.Date("1967-01-01"), Sys.Date()) + 
  opts(legend.position="none") + geom_line() + theme_bw()

nberShade( ggp, fill = "yellow", color = "pink", 
  xrange = c("1969-01-01", "2008-02-01"), openShade = FALSE) + 
  opts(legend.position="none") + geom_line(size=1) + theme_bw()

## End(Not run)
```

---

**plotWindow**

*Set up Coordinates for Graphics Window*

**Description**

This is a shortcut call to `plot.window` that has some of the parameters set to different defaults than the usual values.

**Usage**

`plotWindow(xlim, ylim, log = "", asp = NA, xaxs = "i", yaxs = "i", ...)`

**Arguments**

- `xlim, ylim`: numeric of length 2, giving the x and y coordinates ranges.
- `log`: character; indicating which axes should be in log scale.
- `asp`: numeric, giving the aspect ratio y/x.
- `xaxs`: style of axis interval calculation for the x-axis
- `yaxs`: style of axis interval calculation for the y-axis
- `...`: further graphical parameters as in `par`.
POSIXct

Date-time Constructor Functions

Description

Functions to create objects of classes "POSIXlt" and "POSIXct" representing calendar dates and times.

Usage

POSIXct(x, ...)
POSIXlt(x, ...)
## S3 method for class 'jul'
as.POSIXct(x, tz = "", ...)
## S3 method for class 'ti'
as.POSIXct(x, tz = "", offset = 1, ...)
## S3 method for class 'jul'
POSIXct(x, ...)
## S3 method for class 'numeric'
POSIXct(x, tz = "", origin, ...)
## S3 method for class 'ti'
POSIXct(x, offset = 1, ...)
## Default S3 method:
POSIXct(x, ...)
## S3 method for class 'jul'
POSIXlt(x, ...)
## S3 method for class 'ti'
POSIXlt(x, ...)
## Default S3 method:
POSIXlt(x, ...)

Arguments

x
An object to be converted.

tz
A timezone specification to be used for the conversion, if one is required. System-specific (see time zones), but "" is the current timezone, and "GMT" is UTC (Universal Time, Coordinated).

origin
a date-time object, or something which can be coerced by as.POSIXct(tz="GMT") to such an object.

offset
a number between 0 and 1 specifying where in the period represented by the ti object x the desired time falls. offset = 1 gives the first second of the period and offset = 1 the last second, offset = 0.5 the middle second, and so on.

... other args passed to ISOdatetime (POSIXct.jul and POSIXct.ti), as.POSIXct or as.POSIXlt as appropriate. May include a tz argument as above.
Details

The default methods \texttt{POSIXct.default} and \texttt{POSIXlt.default} do nothing but call \texttt{as.POSIXct} and \texttt{as.POSIXlt}, respectively. The \texttt{POSIXct\_ti} method can take an offset argument as explained above, and the \texttt{POSIXct\_jul} method can handle \texttt{jul} objects with a fractional part. The \texttt{ti} and \texttt{jul} methods for \texttt{POSIXlt} just call the \texttt{POSIXct} constructor and then convert its value to a \texttt{POSIXlt} object.

Value

\texttt{as\_POSIXct}, \texttt{POSIXct} and \texttt{POSIXlt} return objects of the appropriate class. If \texttt{tz} was specified it will be reflected in the "tzone" attribute of the result.

See Also

\texttt{as\_POSIXct} and \texttt{link\{as\_POSIXlt\}} for the default conversion functions, and \texttt{DateTimeClasses} for details of the classes.

---

\textbf{print.ti} \hspace{1cm} \textit{Print a Time Index}

**Description**

Print method for time index.

**Usage**

\begin{verbatim}
## S3 method for class 'ti'
print(x, class = TRUE, ...)
\end{verbatim}

**Arguments**

\begin{itemize}
  \item \texttt{x} \hspace{1cm} a time index
  \item \texttt{class} \hspace{1cm} \texttt{if(TRUE} (the default), prints class: \texttt{ti} at the end.
  \item \texttt{...} \hspace{1cm} additional arguments that may be passed along to generic \texttt{print} function. See details.
\end{itemize}

**Details**

This simply calls \texttt{print(as.character(x), quote = FALSE, ...)}.

**See Also**

\texttt{format.ti}

**Examples**

\begin{verbatim}
print(today(), class = FALSE)
## compare to
today()
\end{verbatim}
print.tis

Printing Time Indexed Series

Description

Print method for time indexed series.

Usage

```r
## S3 method for class 'tis'
print(x, format = "%Y%m%d", matrix.format = FALSE, class = TRUE, ...)
```

Arguments

- `x`: a time indexed series
- `format`: a character string describing how to format the observation times if either `x` is printed in matrix form. Format strings are detailed in `format.ti`.
- `matrix.format`: TRUE or FALSE. See details.
- `class`: if(TRUE (the default), prints class: tis at the end.
- `...`: additional arguments that may be passed along to `print.ts`. See details.

Details

If `matrix.format` is F (the default) and `x` is a univariate monthly, quarterly or annual series, printing is accomplished by `print(as.ts(x), ...)`. Otherwise, `x` is printed as a matrix with rownames created by formatting the time indexes of the observations according to the format string.

See Also

`format.ti`, `print.ts`

Examples

```r
print(tis(1:31, start = today() - 30), format = "%b %d, %Y")
```
RowMeans

Form Row Sums and Means

Description

Form row sums and means for numeric arrays.

Usage

```r
RowSums(x, ...)
RowMeans(x, ...)
```

## Default S3 method:
RowSums(x, ...)
RowMeans(x, ...)

## S3 method for class 'tis'
RowSums(x, ...)
RowMeans(x, ...)

Arguments

- `x` an array of two or more dimensions, containing numeric, complex, integer or logical values, or a numeric data frame, or a tis time indexed series
- `...` arguments passed along to `rowSums` or `rowMeans`.

Value

The tis-specific methods return a tis.

For other types of `x`, see `rowMeans` or `rowSums`.

See Also

`rowMeans`, and `rowSums`

Examples

```r
mat <- tis(matrix(1:36, ncol = 3), start = latestJanuary())
cbind(mat, rowSums(mat), rowMeans(mat))
```
scatterPlot

Produce high-quality scatter plots

Description

Plotting function with scads of options for creating high quality scatter plots. Can be used with screenPage.

Usage

```r
scatterPlot(x, y,
  plotType = "p",
  lgtlineType = "solid", lineWidth = 1.5,
  plotChar = ".", dataCex = 1,
  color = "black",
  xAxisMin = NULL, xAxisMax = NULL, xExpandBy = 0.04,
  xTicks = 5, xTickLocations = NULL,
  labelXTicks = TRUE, xTickLabels = NULL,
  yAxisMin = NULL, yAxisMax = NULL, yExpandBy = 0.04,
  yTicks = 5, yTickLocations = NULL,
  yTickLabels = NULL, labelLeftTicks = FALSE, labelRightTicks = TRUE,
  xCex = 1, xAxisLabel = NULL, labelXAxis = TRUE, xSpace = 4,
  yCex = 1, yAxisLabel = NULL, labelYAxis = TRUE, ySpace = 4,
  cex = 1,
  head = NULL, headAlign = 0.5, headCex = 1.5,
  sub = NULL, subCex = 0.85,
  leftTopLabel = NULL, rightTopLabel = NULL, topLabelAlign = 0,
  labCex = 1, leftInsideLabel = NULL, rightInsideLabel = NULL,
  innerOffset = 0.05, innerCex = 0.8,
  foot = NULL, footAlign = 0, footCex = 0.8, footSpace = -1,
  tck = 0.03, axisWidth = 2, boxType = "u",
  leftMargin = -1, rightMargin = -1,
  topMargin = -1, bottomMargin = -1)
```

Arguments

- **x**: the x coordinates of points in the plot. If this is a string, the function eval0rEcho will attempt to evaluate the string to obtain the x coordinates.
- **y**: the y coordinates of points in the plot. If this is a string, the function eval0rEcho will attempt to evaluate the string to obtain the y coordinates.
- **plotType**: type of plot desired. Values are "p" for points (the default), "l" for lines, "b" for both points and lines (lines miss the points), and "o" for overlaid points and lines.
scatterPlot

- **lineType**: character or numeric vector specifying the line type if `plotType` calls for lines. Default is "solid". For most devices, type 1 is solid, 2 is dotted, 3 and up are a mix of dots and dashes.

- **lineWidth**: default is 1.5.

- **plotChar**: character (or number for plotting symbols – see the help for `points` for details) to be used for plotting points. Default is "*".

- **dataCex**: cex times this number gives the character expansion factor for the data points. Default is 1.

- **color**: string or number. Default is 1, the device default foreground color.

- **xAxisMin**: minimum value of the x axis. If non-NULL, this overrides the calculation described in `xExpandBy`.

- **xAxisMax**: maximum value of the x axis. If non-NULL, this overrides the calculation described in `xExpandBy`.

- **xExpandBy**: a single number or two numbers between 0 and 1. `xAxisMin` and `xAxisMax` will be calculated by multiplicatively extending the data range in both directions by these amounts. Default value .04 extends the data range by 4% in each direction.

- **xTicks**: number of ticks to draw on x axis at "pretty" locations. Default is 5. This argument is ignored if `xTickLocations` is non-NULL.

- **xTickLocations**: if non-NULL, a vector of desired tick locations or a string that evaluates to such a vector. The default value NULL lets the setting for `xTicks` take effect.

- **labelXTicks**: If TRUE, label ticks on the x axis. Default is FALSE.

- **xTickLabels**: character vector of tick labels or NULL (the default). If NULL and `labelXTicks` is TRUE, labels are constructed from the tick locations. This argument has no effect if `labelXTicks` is FALSE.

- **xCex**: cex times this number gives the character expansion factor for the x-axis labels. Default is 1.

- **xAxisLabel**: text to appear centered under the x axis. Default value NULL creates a string by deparsing the `x` argument. This argument is ignored if `labelXAxis` is FALSE.

- **labelXAxis**: if TRUE (the default), label the x axis according to `xAxisLabel`.

- **xSpace**: lines of space to set aside directly beneath the x-axis to hold tick, year and/or axis labels. Default is 4. The space created is `xSpace` times `labCex`.

- **yAxisMin**: minimum value of the y axis. If non-NULL, this overrides the calculation described in `yExpandBy`.

- **yAxisMax**: maximum value of the y axis. If non-NULL, this overrides the calculation described in `yExpandBy`.

- **yExpandBy**: a single number or two numbers between 0 and 1. `yAxisMin` and `yAxisMax` will be calculated by multiplicatively extending the data range in both directions by these amounts. Default value .04 extends the data range by 4% in each direction.

- **yTicks**: number of ticks to draw on y axis at "pretty" locations. Default is 5. This argument is ignored if `yTickLocations` is non-NULL.

- **yTickLocations**: if non-NULL, a vector of desired tick locations or a string that evaluates to such a vector. The default value NULL lets the setting for `yTicks` take effect.
yTickLabels  character vector of tick labels or NULL (the default). If NULL and labelLeftTicks or labelRightTicks is TRUE, labels are constructed from the tick locations. This argument has no effect if labelLeftTicks and labelRightTicks are both FALSE.

labelLeftTicks  If TRUE, label ticks on the left axis. Default is FALSE.
labelRightTicks  If TRUE, label ticks on the left axis. Default is TRUE.
yCex  cex times this number gives the character expansion factor for the left and right axis labels. Default is 1.
extendTopTick  if TRUE (the default) the top tick of the y axes encloses the panel. leftAxisMax and rightAxisMax are increased as necessary to include the top tick for enclosing the panel.
leftAxisLabel  text to appear centered outside the left axis. Default value NULL creates a string by deparsing the y argument. This argument is ignored if labelLeftAxis is FALSE.
labelLeftAxis  if TRUE (the default), label the left axis according to leftAxisLabel.
rightAxisLabel  text to appear centered outside the right axis. Default value NULL creates a string by deparsing the y argument. This argument is ignored if labelRightAxis is FALSE.
labelRightAxis  if TRUE label the right axis according to rightAxisLabel. Default is FALSE
cex  the base character expansion factor by which all of the ***cex parameters are scaled. The default setting is the value of par("cex").
head  text to appear at the top of the figure region, with alignment determined by headAlign. No default.
headAlign  number indicating justification for the strings in head and sub. 0 means left justify, 1 means right justify, 0.5 (the default) means to center the text. Other numbers are a corresponding distance between the extremes.
headCex  cex times this number gives the character expansion factor for head. Default is 1.5.
sub  text to appear just under head, with alignment determined by headAlign. No default.
subCex  cex times this number gives the character expansion factor for sub. Default is 0.85.
leftTopLabel  text to appear at the top of the left axis, with alignment determined by topLabelAlign. No default.
rightTopLabel  text to appear at the top of the right axis, with alignment determined by topLabelAlign. No default.
topLabelAlign  number indicating alignment for the strings in leftTopLabel. 0 (the default) means left justify, 1 means right justify, 0.5 means to center the text. rightTopLabel, if given, is aligned by 1 - topLabelAlign.
labCex  cex times this number gives the character expansion factor for leftTopLabel, rightTopLabel, and xAxisLabel.
leftInsideLabel
text to appear left justified and just inside the upper left corner of the plot region. No default.

rightInsideLabel
text to appear right justified and just inside the upper right corner of the plot region. No default.

innerOffset
number between 0 and 1, a fractional offset for the inside labels. The left edge of leftInsideLabel is offset by this fraction of the x range from the left edge of the plot, as is the right edge of rightInsideLabel from the right edge of the plot.

innerCex cex times this number gives the character expansion factor for leftInsideLabel and rightInsideLabel.

foot
text to appear at the bottom of the figure region, with alignment determined by footAlign. No default.

footAlign
number indicating justification for the strings in foot. 0 (the default) means left justify, 1 means right justify, 0.5 means to center the text. Other numbers are a corresponding distance between the extremes.

footCex cex times this number gives the character expansion factor for foot. Default is 0.8.

footSpace
lines of space to set aside directly beneath the space allocated by xSpace to hold footnotes. The space created is footSpace times footCex. Default is length(foot); using a higher value will result in extra space in the bottom figure margin.

tck
length of major tick marks in inches. Minor ticks are 2/3 as long. Default is 0.03.

axisWidth
line width for the axes and box (if any). Default is 2.

boxType
character representing the type of box. Characters "o", "l" (ell), "7", "c" will produce boxes which resemble the corresponding upper-case letters. The value "n" will suppress boxes. The default is "u".

leftMargin
lines of space for the left margin. Default value (-1) figures this out automatically.

rightMargin
lines of space for the right margin. Default value (-1) figures this out automatically.

topMargin
lines of space for the top margin. Default value (-1) figures this out automatically.

bottomMargin
lines of space for the bottom margin. Default value (-1) figures this out automatically.

Details
Each of the text items head, sub, leftTopLabel, rightTopLabel, leftInsideLabel, rightInsideLabel, foot, leftAxisLabel, rightAxisLabel and xAxisLabel can be given as a string, a collections of strings, or as a string that gets evaluated to one of the first two. Multiple strings are drawn on successive lines.
Value

scatterPlot invisibly returns a list of class "scatterPlot" and elements named xy (a matrix containing x and y in two columns), plotType, lineType, color, plotChar, lineWidth, x (x coordinate for legend), y (y coordinate for legend), xRange, yRange, innerCex and par. This list is useful mostly as an argument to legend.

Note

scatterPlot is a companion to tisPlot. Both are designed to be driven from a graphical user interface.

See Also

tisPlot, evalOrEcho

Description

Places header and footer text items in outer margin of page and splits the screen appropriately. Can also redraw header and footer.

Usage

screenPage(head = NULL, sub = NULL, foot = NULL, date = FALSE, dateFormat = "%x", time = FALSE, topleft = character(), topright = character(), headFont = par("font.main"), subFont = par("font.sub"), footFont = par("font"), cex = 1.001, headCex = 1.5, subCex = 0.85, footCex = 0.75, topleftCex = 0.85, toprightCex = 0.85, footAlign = 0, leftMargin = 0, rightMargin = leftMargin, topMargin = 0, bottomMargin = topMargin)

Arguments

head character string or strings to appear centered in the top outer margin of the page. If length(head) > 1, a multi-line main title results.
sub character string or strings to appear centered just under head.
foot character string or strings to appear in the bottom outer margin of the page.
date logical: if TRUE, put the current date in the upper right corner of the page.
dateFormat strptime-style format to use if date is TRUE. The default formats dates like 11/30/2006. What is actually being formatted is the value returned by Sys.time(), so you can also use time formats here. For example, setting dateFormat = "%c" will create a string like "Thu 30 Nov 2006 02:49:45 PM EST".
time logical: if TRUE, put the current time in the upper right corner of the page. If date is also true, the time string will be on the line below the date string.
topLeft character string or strings to appear at top left corner of the page
topRight character string or strings to appear at top right corner of the page
headFont font to use in writing the main title in head. The default uses whatever par("font.main") is set to.
subFont font to use in writing the sub title in sub. The default uses whatever par("font.sub") is set to.
footFont font to use in writing the footnotes in foot. The default uses whatever par("font") is set to.
cex number by which all of the other "cex" arguments are scaled.
headCex number: Character Expansion Factor (cex) for the string(s) in head. The actual cex used will be cex * headCex.
subCex number: cex for the string(s) in sub. The actual cex used will be cex * subCex.
footCex number: cex for the string(s) in foot. The actual cex used will be cex * footCex.
topLeftCex number: cex for the string(s) appearing in the top left corner of the page. The actual cex used for these strings will be cex * topLeftCex.
topRightCex number: cex for the string(s) appearing in the top right corner of the page, including the time and date stamps. The actual cex used for these strings will be cex * topRightCex.
footAlign number: justification for the strings in foot. 0 means left justify, 1 means right justify, 0.5 means to center the text. Other numbers are a corresponding distance between the extremes.
leftMargin left margin of page in inches.
rightMargin right margin of page in inches. Default is same as leftMargin
topMargin top margin of page in inches.
bottomMargin bottom margin of page in inches. Default is same as topMargin

Details
screenPage first sets aside space for the margins specified by topMargin, bottomMargin, leftMargin and rightMargin. Then it figures out how much additional space is needed for the top and bottom outer margin text elements, places them, and then splits the screen in 3, with screen 3 being the middle part of the page. The user is then free either to further subdivide screen 3 (using split.screen()) or to use it as is.
On exit, screen 3 is the active screen.

Value
This function returns a list of all of its arguments, including default values for arguments that were not supplied. The return is invisible if a graphics device is active.

See Also
split.screen
Examples

```r
screenPage(head = "Chart 1", date = TRUE, foot = rep(" ", 4),
  cex = 0.85, headCex = 1)

## then draw charts, possibly after further subdividing the screen
```

---

**setColors**  
*Set the Graphics Palette and Default Colors*

**Description**

Sets the color palette (used when a `col=` has a numeric index) to a given vector of colors, and also sets the graphics parameters (see `par`) `col`, `col.axis`, `col.lab`, `col.main`, `col.sub` and `fg` to `colors[1]`.

**Usage**

```r
setColors(colors)
```

**Arguments**

- `colors` a character vector

**Value**

The new color palette.

**See Also**

`palette`, `par`

---

**setDefaultFrequencies**  
*Return Known Time Index Frequencies, Change Default Frequencies*

**Description**

A tif (Time Index Frequency) can usually be set either by code (a number) or by name. `setDefaultFrequencies` sets particular frequencies for the tif names "weekly", "biweekly", "bimonthly" (also "bimonth"), "quarterly" (also "q"), "annual" (also "a"), and "semiannual" (also "sann").

tiflist returns the map of frequency names to frequency codes.
Usage

```
setDefaultFrequencies(weekly = "wmonday",
    biweekly = "b1wednesday",
    bimonthly = "bimonthdecember",
    quarterly = "qdecember",
    annual = "anndecember",
    semiannual = "sanndecember",
    setup = FALSE)

tifList()
```

Arguments

- `weekly`: A string giving the name of the particular frequency that frequency "weekly" will correspond to.
- `biweekly`: Ditto for "biweekly".
- `bimonthly`: Ditto for "bimonth" and "bimonthly".
- `quarterly`: Ditto for "q" and "quarterly".
- `annual`: Ditto for "a" and "annual".
- `semiannual`: Ditto for "sann" and "semiannual".
- `setup`: If TRUE, set all of the defaults, otherwise set only the defaults for which arguments were given. The default is FALSE, but see the details.

Details

The named vector `.tifList` (returned by the function of the same name) stored in the global environment contains the mapping of frequency names to frequency codes. Running this function modifies the `.tifList` vector and stores it back in the global environment. It gets run with `setup = TRUE` when the `tis` package is loaded. If you want different defaults, call the function sometime after that.

Value

A copy of the `.tifList` vector.

See Also

tifName
solve.tridiag: Solve a Tridiagonal System of Equations

Description

This function solves the equation \( \mathbf{a} \mathbf{x} = \mathbf{b} \) for \( \mathbf{x} \), where \( \mathbf{a} \) is tridiagonal and \( \mathbf{b} \) can be either a vector or a matrix.

Usage

```r
## S3 method for class 'tridiag'
solve(a, b, ...)  
```

Arguments

- `a`: a tridiag object: a square tridiagonal (all zeroes except for the main diagonal and the diagonals immediately above and below it) matrix containing the coefficients of the linear system.
- `b`: a vector or matrix giving the right-hand side(s) of the linear system. If missing, \( b \) is taken to be an identity matrix and the function will return the inverse of \( a \).
- `...`: ignored

Details

Uses the LINPACK \texttt{dgtsv} routine.

See Also

- `solve`

ssDate: ssDate Objects

Description

The function `ssDate` is used to create `ssDate` (spreadsheet date) objects, which are useful for reading and writing dates in spreadsheet form, i.e., as the number of days since December 30, 1899. `as.ssDate` and `is.ssDate` coerce an object to a `ssDate` and test whether an object is a `ssDate`.

Usage

```r
ssDate(x, ...)  
as.ssDate(x)  
is.ssDate(x)  
```
Arguments

x          object to be tested (is.ssDate) or converted into a ssDate object.
...        other args to be passed to jul function.

Details

an ssDate is essentially a rebased Julian date that represents a date as the number of days since December 30, 1899. The constructor function ssDate subtracts jul(18991230) from jul(x, ...) and coerces the result to class ssDate. Pretty much all of the stuff you can do with jul objects can also be done with ssDate objects.

Value

is.ssDate returns TRUE or FALSE.

as.ssDate coerces its argument to have class ssDate, without making any attempt to discern whether or not this is a sensible thing to do.

ssDate constructs a ssDate object like x.

ssDate with no arguments returns the ssDate for the current day.

See Also

jul

Examples

dec31 <- ssDate(20041231)
jan30 <- ssDate("2005-1-30")
jan30 - dec31               # 30
feb28 <- jan30 + 29         # current date

Description

Return the start or end time index for a tis object.

Usage

## S3 method for class 'tis'
start(x, ...)  
## S3 method for class 'tis'
end(x, ...)    
start(x) <- value
Arguments

- **x**: a tis object
- **value**: desired start attribute
- ... ignored

Value

- `start.tis` returns the start attribute of `x`, while `end.tis` returns `start(x) + nobs(x) - 1`.
- `start(x) <- value` returns the series `x` shifted such that its starting time is `value`.

Note

- `start` and `end` are generic functions with default methods that assume `x` has (or can be given) a `tsp` attribute. The default methods return a two vector as `c(year, period)`, while the methods described here return infinitely more useful `ti` objects.

See Also

- `start`, `end`

Examples

```r
x <- tis(numeric(8), start = c(2001, 1), freq = 4)
start(x)  ## --> ti object representing 2001Q1
start(as.ts(x))  ## --> c(2001, 1)
```

---

### stripBlanks

**Strip Blanks**

#### Description

Strip leading and trailing blanks from strings

#### Usage

```r
stripBlanks(strings)
```

#### Arguments

- **strings**: character vector

#### Value

An object like `strings` with no leading or trailing blanks.

#### See Also

- `blanks`, `gsub`
Description

An R object may have a class attribute that is a character vector giving the names of classes it inherits from. stripClass strips the class classString from that character vector. stripTis(x) is shorthand for stripClass(x, "tis").

Usage

stripClass(x, classString)

Arguments

x an object whose class character vector may or may not include classString

classString name of class to remove from the inheritance chain

Value

An object like x, but whose class attribute does not include classString. If the class attribute less classString is empty, unclass(x) is returned.

Note

This function can be useful in functions that return a modified version of one their arguments. For example, the format.ti method takes a ti (TimeIndex) as an argument and returns a character object object 'like' the original argument. The first thing format.ti(x) does internally is z <- stripClass(x, "ti"). This creates z as a copy of x but with the difference that z no longer inherits from class ti. The function then fills in the data elements of z with the appropriate strings and returns it. The beauty of this approach is that the returned z already has all of the attributes x had, except that it no longer inherits from class ti. In particular, if x was a matrix with dimnames, etc., z will also have those attributes.

See Also

class
t.tis  

**Matrix Transpose**

**Description**

Returns the transpose of `as.matrix(x)`

**Usage**

```r
## S3 method for class 'tis'
t(x)
```

**Arguments**

- `x`  
  a `tis` object. If `x` is univariate, it will be treated as if it were a single-column matrix, so its transpose will be a single-row matrix.

**Value**

A matrix, see `t`. Note that this is not a time series.

**See Also**

`t.tis`

**Examples**

```r
a <- tis(matrix(1:30, 5,6), start = latestMonth())
a
t(a)  ##i.e., a[i, j] == t(a)[j, i] for all i,j, and t(a) is NOT a time series
```

---

ti  

**Time Index Objects**

**Description**

The function `ti` is used to create time index objects, which are useful for date calculations and as indexes for `tis` (time indexed series).

`as.ti` and `asTi` coerce an object to a time index, the difference being that `as.ti` calls the constructor `ti`, while `asTi` simply forces the class of its argument to be "ti" without any checking as to whether or not it makes sense to do so.

`is.ti` tests whether an object is a time index.

`couldBeTi` tests whether or not `x` is numeric and has all elements within the range expected for a `ti` time index with the given `tif`. If `tif` is `NULL` (the default), the test is whether or not `x` could be a `ti` of *any* frequency. If so, it can be safely coerced to class `ti` by `as.ti`. 
Usage

\[
\begin{align*}
ti(x, \ldots) & \\
& \text{## S3 method for class 'Date'} \\
ti(x, \ldots) & \\
& \text{## Default S3 method:} \\
ti(x, \text{tif} = \text{NULL}, \text{freq} = \text{NULL}, \ldots) & \\
& \text{## S3 method for class 'jul'} \\
ti(x, \text{tif} = \text{NULL}, \text{freq} = \text{NULL}, \text{hour} = 0, \text{minute} = 0, \text{second} = 0, \ldots) & \\
& \text{## S3 method for class 'ssDate'} \\
ti(x, \ldots) & \\
& \text{## S3 method for class 'ti'} \\
ti(x, \text{tif} = \text{NULL}, \text{freq} = \text{NULL}, \ldots) & \\
& \text{## S3 method for class 'tis'} \\
ti(x, \ldots) & \\
& \text{## S3 method for class 'yearmon'} \\
ti(x, \ldots) & \\
& \text{## S3 method for class 'yearqtr'} \\
ti(x, \ldots) & \\
& \text{as.ti(x, \ldots)} \\
& \text{asTi(x)} \\
& \text{is.ti(x)} \\
& \text{couldBeTi(x, \text{tif} = \text{NULL})}
\end{align*}
\]

Arguments

- \(x\) object to be tested (is.ti) or converted into a ti object. As described in the details below, the constructor function ti can deal with several different kinds of \(x\).
- \(\text{hour}\) used if and only if \(\text{tif}\) is an intraday frequency
- \(\text{minute}\) used if and only if \(\text{tif}\) is an intraday frequency
- \(\text{second}\) used if and only if \(\text{tif}\) is an intraday frequency
- ... other args to be passed to the method called by the generic function.
- \(\text{tif}\) a ti Frequency, given as either a numerical code or a string. \(\text{tif()}\) with no arguments returns a list of the allowable numerical codes and names. Either \(\text{tif}\) or \(\text{freq}\) must be supplied for the variants of \(\text{ti()}\).
- \(\text{freq}\) some \(\text{tif}\)'s can alternatively be specified by their frequency, such as 1 (annual), 2 (semiannual), 4 (quarterly), 6 (bimonthly), 12 (monthly), 24 (semimonthly), 26 (biweekly), 36 (tenday), 52 (weekly), 262 (business) and 365 (daily). Either \(\text{tif}\) or \(\text{freq}\) must be supplied for the variants of \(\text{ti()}\).

Details

A ti has a tif (ti Frequency) and a period. The period represents the number of periods elapsed since the base period for that frequency. Adding or subtracting an integer to a ti gives another ti. Provided their corresponding element have matching tifs, the comparison operators \(<\), \(>, \leq, \geq, ==\) all work, and subtracting one ti from another gives the number of periods between them. See the examples section below.
The `ti` class implements methods for a number of generic functions, including "[", `as.Date`, `as.POSIXct`, `as.POSIXlt`, `c`, `cycle`, `edit`, `format`, `frequency`, `jul`, `max`, `min`, `print`, `rep`, `seq`, `tif`, `tifName`, `time`, `ymd`.

`ti` is a generic function with specialized methods to handle `jul`, `Date`, `ti`, `tis`, `yearmon` and `yearqtr` objects.

The default method (`ti.default`) deals with character `x` by calling `as.Date` on it. Otherwise, it proceeds as follows:

If `x` is numeric, a check is made to see if `x` could be a `ti` object that has somehow lost its class attribute. Failing that, `isYmd` is used to see if it could be `yyyymmdd` date, then `isTime` is called to see if `x` could be a decimal time (a number between 1799 and 2200). If `x` is of length 2, an attempt to interpret it as a `c(year, period)` pair is made. Finally, if all else fails, `as.Date(x)` is called to attempt to create a `Date` object that can then be used to construct a `ti`.

**Value**

`is.ti` and `couldBeTi` return `TRUE` or `FALSE`.

`as.ti` returns a `ti` object.

`asTi` returns its argument with the class attribute set to "ti".

`ti` constructs a `ti` object like `x`, except for two special cases:

1. If `x` is a `tis` series, the return value is a vector time index with elements corresponding to the observation periods of `x`.
2. If `x` is a numeric object of length 2 interpretable as `c(year, period)`, the return value is a single `ti`.

**Note**

The `as.Date(x)` call is not wrapped in a try-block, so it may be at the top of the stack when `ti` fails.

The return value from `asTi` is not guaranteed to be a valid `ti` object. For example, `asTi("a")` will not throw an error, and it will return the string "a" with a class attribute "ti", but that's not a valid time index.

**See Also**

`jul`, `ymd`, `tif`, `tifName`, `as.Date`

**Examples**

```r
z <- ti(19971231, "monthly")  ## monthly ti for Dec 97
is.ti(z)                       ## TRUE
is.ti(unclass(z))              ## FALSE
couldBeTi(unclass(z))          ## TRUE
ymd(z + 4)                     ## 19980430
z - ti(c(1997,6), freq = 12)   ## monthly ti for June 1997
ti(z, tif = "wmonday")         ## week ending Monday June 30, 1997
```
tiDaily

Daily and Business Day Time Indexes

Description

Return a daily or business day ti corresponding to a specified position within a time index.

Usage

`tiDaily(xTi, offset = 1)`
`tiBusiness(xTi, offset = 1)`

Arguments

- `xTi`: a ti object or something that the `ti()` function can turn into a ti object
- `offset`: for `xTi`, a number in the range [0,1] telling where in the period represented by `x` to find the day. 0 means the first day of the period, 1 the last day of the period, and fractional values for in-between day.

Value

`tiDaily` converts its first argument to a `jul` using the offset provided, and returns a daily `ti` for that day.

`tiBusiness` converts its first argument to a `jul` using the offset provided, and returns a "business" `ti` for that day.

See Also

`ti`, `jul`

tierChart

Create tier charts

Description

A tier chart plots several years’ observations of a series against the times of year in which they were observed. Useful for seeing seasonal patterns in daily, weekly and irregularly-spaced data.
Usage

tierChart(x,
    startMonth = latestJanuary(end(x)),
    nMonths = 4,
    nYears = 7,
    offsets = 0,
    padDays = 6,
    pch = "year",
    lty = "solid",
    lwd = 1.5,
    col = 1 + (n.y:1),
    type = "b",
    ylim = NULL,
    outlier.trim = 0,
    noTrimLastYear = TRUE,
    extendHorizontalTicks = TRUE,
    circles.ymd = NULL,
    circles.col = 6,
    circles.inches = 0.1,
    vlines.ymd = NULL,
    vlines.col = 2,
    vlines.lty = 4,
    vlines.lwd = 1.5,
    vlines2.ymd = NULL,
    vlines2.col = 3,
    vlines2.lty = "solid",
    vlines2.lwd = 2,
    hlines = NULL,
    hlines.col = 1,
    hlines.lty = 1,
    hlines.lwd = 1,
    tiPoints.1 = NULL,
    tiPoints.2 = NULL,
    pch.1 = "*",
    pch.2 = "+",
    col.1 = 2,
    col.2 = 3,
    nolegend = FALSE,
    main = deparse(substitute(x)),
    topleft.labels = NULL,
    topright.labels = NULL,
    legend.ncol = length(years),
    legend.bg = 0,
    timestamp = TRUE,
    topline = TRUE,
    vlines.periodEnd = TRUE,
    vlines.month = TRUE,
    midperiod = FALSE,
Arguments

x  
A monthly or higher frequency (such as weekly or daily) time indexed series (a
	tis object) or something that as.tis can convert to one.

startMonth  
a monthly time index (ti) object indicating first month to show on plot.

nMonths  
number of months to show on plot.

nYears  
number of years to include in the plot.

offsets  
vector of day offsets for the years in descending order. If offsets[2] is 3, for

element, the points for the second-to-last year will be shifted forward 3 days on

the plot. Negative numbers shift points back. If length(offsets) < nYears,

zeroes will be appended to offsets to make it nYears long.

padDays  
number of extra days to plot before and after the requested months.

pch  
plotting symbols to be drawn when plotting points. If pch is a character string,

such as "alb2", the first year's points will be labeled "a", the second year's with"1",

the third with "b", and so on. Alternatively, pch can be a numeric vector

giving the numbers of plotting symbols to use, as detailed in the documentation

for lines. The default string "year" uses the last digit of the year number for

each point in that year.

lty  
vector of line types. The first element is for the first year, the second element for

the second year, etc., even if lines are not plotted for all years. Line types will

be used cyclically until all years are drawn.

lwd  
number specifying line width

col  
vector of colors for the years, specified as numbers or color names.

type  
character string, telling which type of plot ("p", points; "l", lines; "b", both;

"n", none; or "h", high-density) should be done for each year. The first character

of type defines the first year, the second character the second, etc. Characters

in type are cycled through; e.g., "pl" alternately plots points and lines. The

default is "b".

ylim  
ylim is a vector of 2 numbers giving desired y-axis limits. The actual limits

on the plot will be the result of pretty(ylim). If ylim is not set explicitly, the

value of outlier.trim is used to calculate it such that the c(outlier.trim, 1 - outlier.trim)

quantiles of the plotted points fall within the calculated ylim before it is "pret-
tied".

outlier.trim  
see ylim above
noTrimLastYear  if TRUE (the default), outlier.trim is effectively set to zero for the most recent
year, i.e., the y-axis limits will be expanded, if necessary, to insure that all of the
points plotted for the most recent year fit on the plot.

extendHorizontalTicks  if TRUE (the default), extend the horizontal ticks with a dotted line all the way
across the chart.

circles.ymd  draws circles around the plotted points corresponding to these ymd dates. The
colors and sizes of the circles are given by circles.col and circles.inches.

circles.col  see circles.ymd above.
circles.inches  see circles.ymd above.

vlines.ymd  numeric vector of yyyymmdd dates, draws vertical lines of type vlines.lty
width vlines.lwd and colors vlines.col at the dates given.

vlines.col  see vlines.ymd above
vlines.lty  see vlines.ymd above
vlines.lwd  see vlines.ymd above

vlines2.ymd  numeric vector of yyyymmdd dates, draws vertical lines of type vlines2.lty
width vlines2.lwd and colors vlines2.col at the dates given.

vlines2.col  see vlines2.ymd above
vlines2.lty  see vlines2.ymd above
vlines2.lwd  see vlines2.ymd above

hlines  numeric vector, draws horizontal lines of type hlines.lty width hlines.lwd
and colors hlines.col at the locations given.

hlines.col  see hlines above
hlines.lty  see hlines above
hlines.lwd  see hlines above

tiPoints.1  a ti object specifying dates for which the corresponding points will be marked
with the characters in pch.1 in colors col.1.

pch.1  see tiPoints.1 above
col.1  see tiPoints.1 above

tiPoints.2  a ti object specifying dates for which the corresponding points will be marked
with the characters in pch.2 in colors col.2.

pch.2  see tiPoints.2 above
col.2  see tiPoints.2 above

nolegend  if T do not plot a legend

main  character string giving main title for the chart.

topleft.labels  strings to place in left corner of top margin
topright.labels  strings to place in right corner of top margin

legend.ncol  number of columns to use for legend. Has no effect if nolegend is T.

legend.bg  background color for legend
A tier chart shows seasonal patterns in data by superimposing the data from the same months of several years onto a single plot. Tier charts can be used both to present a view of a time series and to graphically edit its values by pointing and clicking.

For most purposes, adjustableTierChart is preferred to tierChart, since the former presents a chart that can be edited, scrolled and printed via mouse clicks, while the latter simply draws a single chart and returns. However, adjustableTierChart requires user interaction, which may make it unsuitable for some uses.

When adjustableTierChart is called, it draws on the current graphics device and then waits for mouse clicks to occur. A left mouse button click on one of the scroll arrows changes the display to show adjacent months, while a left mouse click on the PrintMe box causes the current plot to be copied to the printer. Left mouse clicks in the data area of the plot are used to edit the values of the time series. Arrows are drawn from the current data points to the mouse location to show where the new data values will be.

A middle mouse button click causes adjustableTierChart to return. Closing the graphics window via the windowing system (e.g., clicking on the window’s X button) has the same effect. Until adjustableTierChart is told to return, the entire R process will appear to be frozen. It isn’t actually frozen, it’s just waiting for mouse input. Use tierChart instead if no user interaction is desired.
Value

tierChart invisibly returns a list with the following components:

- **px**: a matrix with `nYears` columns containing the x coordinates of the points for each year.
- **py**: a matrix with `nYears` columns containing the y coordinates of the points for each year.
- **ymd**: matrix of `yyyymmdd` dates corresponding to x coordinates in `px`.
- **index**: matrix giving positions of the elements of the y matrix in the original x series, that is, `x[index[i,j]] == y[i,j]`.
- **lBox**: vector of 4 numbers giving the c(left, bottom, right, top) bounds of the scroll arrow box in the upper left corner of the plot.
- **rBox**: vector of 4 numbers giving the c(left, bottom, right, top) bounds of the scroll arrow box in the upper right corner of the plot.
- **printBox**: vector of 4 numbers giving the c(left, bottom, right, top) bounds of the print box in the adjoining the left scroll box.
- **startMonth**: the input argument of the same name.
- **nMonths**: number of months wide the plot is.
- **nYears**: number of years plotted.

If the input argument codeboxes is `F`, the `lBox`, `rBox` and `printBox` elements of the list will not be present.

`adjustableTierChart` returns the edited input series `x` as a `tis` object, with an additional `startMonth` attribute.

Side Effects

A tier chart is drawn on the current graphics device.

See Also

`monthplot` for a nice way to look at seasonality in monthly data.

Examples

```
## Not run:
tierChart(m1.w)  ## January - April of 7 most recent years

tierChart(m1.w, startMonth=1, nMonths = 12)  ## Tier chart for entire year

tierChart(m1.w, type="l", lty=1)  ## same as first example, but with solid lines and no plotting symbols

xe <- adjustableTierChart(x)  ## xe will be edited version of x

## End(Not run)
```
tif

Time Index Frequencies and Periods

tif

Description

Return the tif code of an object, the name associated with a tif code, the period number of a time index, or the first.

Usage

tif(x, ...)
## S3 method for class 'ti'
tif(x, ...)
## S3 method for class 'tis'
tif(x, ...)
## S3 method for class 'ts'
tif(x, ...)
## Default S3 method:
tif(x, freq = NULL, ...)
tifName(s)
## Default S3 method:
tifName(s)
## S3 method for class 'ti'
tifName(s)
## S3 method for class 'tis'
tifName(s)
period(z)
basePeriod(x)

Arguments

x a ti or tis object, or a string giving a tif name.
freq numeric. If x is missing, return the tif for this frequency, otherwise ignore.
... ignored
s a ti or tis object, or a tif code.
z a ti object.

Details

The tifList object associates tifNames with tif codes. Most functions that call for tif argument can take either a tif code or a tif name.

Both function are generic function with methods for ti and tis objects, as well as a default method. tif also has a method for ts objects.
Value

tif returns the tif code for x, while tifName returns a name for that code. Many of the codes have several names, but only the default one is returned.
tif or tifName called with no arguments returns a vector of all tif codes with names.
period returns a vector like z giving the number of periods elapsed since the first period defined for its argument’s frequency.
basePeriod returns the ti for the first period defined for tif(x).

See Also

ti.frequency

Examples

tif()                      ## returns a vector of all tif codes
tifName(today())          ## today() returns a ti
period(today())

tif2freq  

Periods Per Year for Time Index Frequencies

Description

Returns the frequency of a ti object constructed from the current date with the given tif.

Usage

tif2freq(tif)

Arguments

tif                        a tifName or tif code

Value

a number

See Also

tif, tifName, frequency

Examples

tif2freq("wmonday")
tif2freq("monthly")
tif2freq(tif(today()))
Description

The function `tis` is used to create time-indexed series objects.

`as.tis` and `is.tis` coerce an object to a time-indexed series and test whether an object is a time-indexed series.

Usage

```r
tis(data, start = 1, tif = NULL, frequency = NULL, end = NULL)
as.tis(x, ...)
## S3 method for class 'ts'
as.tis(x, ...)
## S3 method for class 'tis'
as.tis(x, ...)
## S3 method for class 'zoo'
as.tis(x, ...)
## Default S3 method:
as.tis(x, ...)
is.tis(x)
```

Arguments

data  
a numeric vector or matrix of the observed time-series values.

start  
the time of the first observation. This can be a ti object, or anything that `ti(start, tif = tif, freq = frequency)`, can turn into a ti object.

...  
other args to be passed to the method called by the generic function. `as.tis.default` passes `x` and ... to the constructor function `tis`.

tif  
a ti Frequency, given as either a numerical code or a string. `tif()` with no arguments returns a list of the allowable numerical codes and names.

frequency  
As an alternative to supplying a tif, some tifs can alternatively be specified by their frequency, such as 1 (annual), 2 (semiannual), 4 (quarterly), 6 (bimonthly), 12 (monthly), 24 (semimonthly), 26 (biweekly), 36 (tenday), 52 (weekly), 262 (business) and 365 (daily). Many frequencies have multiple tifs associated with them. For example, all of the tifs (wsunday, wmonday, ..., wsaturday) have frequency 52. In this case, specifying `freq` gets you the default weekly tif `wmonday`.

data  
the time of the last observation, specified in the same way as `start`.

x  
object to be tested (`is.tis`) or converted into a `tis` object. As described in the details below, `as.tis` can deal with several different kinds of `x`. 
Details
The function `tis` is used to create `tis` objects, which are vectors or matrices with class of "tis" and a `start` attribute that is a `ti` (time index) object. Time-indexed series are a form of time series that is more flexible than the standard `ts` time series. While observations for a `ts` object are supposed to have been sampled at equispaced points in time, the observation times for a `tis` object are the times given by successive increments of the more flexible time index contained in the series start attribute. There is a close correspondence between Fame time series and `tis` objects, in that all of the Fame frequencies have corresponding `tif` codes.

`tis` objects operate much like vanilla R `ts` objects. Most of the methods implemented for `ts` objects have `tis` variants as well. Evaluate `methods(class = "tis")` to see a list of them.

One way or another, `tis` needs to figure out how to create a `start` attribute. If `start` is supplied, the function `ti` is called with it, `tif` and `frequency` as arguments. The same process is repeated for `end` if it was supplied. If only one of `start` and `end` was supplied, the other is inferred from it and the number of observations in `data`. If both `start` and `end` are supplied, the function `rep` is used to make `data` the length implied by `end - start + 1`.

`as.tis` is a generic function with specialized methods for other kinds of time series, including zoo series from `zoo`. The fallback default method calls `tis(x, ...)`.

Value
`tis` and `as.tis` return time-indexed series. `is.tis` returns TRUE or FALSE.

Note
If the index of a zoo series is a `ti`, the coercion `as.tis.zoo` does is trivial. For other kinds of zoo series, the function `inferTi` tries to figure out a time index that matches the times of the index of the zoo series. This may fail, as there are infinitely more possible kinds of zoo indexes than the finite number of time index frequencies.

See Also
Compare with `ts`. See `ti` for details on time indexes. `cbind.tis` combines several time indexed series into a multivariate `tis`, while `mergeSeries` merges series, and `convert` and `aggregate` convert series from one frequency to another. `start.tis` and `end.tis` return `ti` objects, while `ti.tis` returns a vector `ti`. There is a print method `print.tis` and several plotting methods, including `lines.tis` and `points.tis`. The `window.tis` method is also sufficiently different from the `ts` one to deserve its own documentation.

Examples
```r
  tis(1:48, start = c(2000, 1), freq = 12)
  tis(1:48, start = ti(20000101, tif = "monthly"))  # same result
  tis(0, start = c(2000,1), end = c(2000,52), tif = "weekly")
```
**Description**

Applies linear filtering to a univariate `tis` series or to each column separately of a multivariate `tis` series.

**Usage**

```r
tisFilter(x, ...)
```

**Arguments**

- `x`: a univariate or multivariate time series.
- `...`: arguments passed along to `filter`.

**Value**

A `tis` time indexed series with leading and trailing `NA` values stripped.

**Note**

If ever the `filter()` function is made generic, as it should be, this function could become the `tis` method for it.

**See Also**

`filter`

**Examples**

```r
x <- tis(1:100, start = c(2000,1), freq = 12)
tisFilter(x, rep(1, 3))
tisFilter(x, rep(1, 3), sides = 1)
tisFilter(x, rep(1, 3), sides = 1, circular = TRUE)
```
tisFromCsv

*Read time series from Comma Separated Values (.csv) file*

**Description**

Reads tis (Time Indexed Series) from a csv file, returning the series in a list, and optionally storing them in an environment.

**Usage**

```r
tisFromCsv(csvfile, dateCol = "date", dateFormat = "%Y%m%d", tz = "", tif = NULL, defaultTif = "business", save = F, envir = parent.frame(), naNumber = NULL, chopNAs = TRUE, tolerance = sqrt(.Machine$double.eps), ...)
```

**Arguments**

- `csvFile` A file name, connection, or URL acceptable to `read.csv`. Also see the the rest of this help entry for required attributes of this file.
- `dateCol` name of the column holding dates. This column must be present in the file.
- `dateFormat` format of the dates in `dateCol`. If the `dateCol` cells contain Excel dates, use `dateFormat = "excel"`. If they are strings, see `strptime` for date formats.
- `tz` the time zone to be used by `strptime` when converting date strings into `posixlt` timestamps. The default is to use the current time zone, which means it can change from, say, EST to EDT in the spring, and back to EST in the fall. If you have an "impossible" time in your csv file, like 2 am on March 13, 2011, this will result in an unexpected NA in the created ti dates, which will result in those rows in your csv being effectively ignored.
- `tif` time index frequency of the data. If this is NULL (the default), the function tries to infer the frequency from the dates in the `ymdCol` column.
- `defaultTif` If the frequency can't be inferred from the dates in the `ymdCol` column, this tif frequency will be used. This should be a rare occurrence.
- `save` If true, save the individual series in the environment given by the `envir` argument. Default is FALSE.
- `envir` if `save` == `TRUE`, the individual series (one per column) are saved in this environment. Default is the frame of the caller.
- `naNumber` if non-NULL, numbers within tolerance of this number are considered to be NA values. NA strings can be specified by including an `na.strings` argument as one of the...arguments that are passed along to `read.csv`.
- `chopNAs` if TRUE (the default), leading and trailing NA values are cut off of each column.
- `tolerance` Used to determine whether or not numbers in the file are close enough to `naNumber` to be regarded as equal to it. The default is about 1.48e-08.
- `...` Additional arguments passed along to the underlying `read.csv` function.
Details

**File Requirements:** The csv file must have column names across the top, and everything but the first row should be numeric. There must be as many column names (enclosed in quotes) as there are columns, and the column named by `dateCol` must have dates in the format indicated by `dateFormat`. The `dateCol` column must be present.

**Missing (NA) values:** Missing and NA values are the same thing. The underlying `read.csv` has ",," as its default separator and "NA" as its default na.string, so the rows

```
20051231,,13,,42,NA,
20060131,NA,14,,43,,NA
```

indicate NA values for both the Dec 2005 and Jan 2006 observations of the first, third, fifth and sixth series.

The values in the file are read into a single large tis series, with a tif (Time Index Frequency) inferred from the first six dates in the ymd column. The first date is converted to a ti (Time Index) of that frequency and becomes the start of the series. If `chopNAs` is TRUE, each individual column is then windowed via `nawindow` to strip off leading and trailing NA values, and the resulting series are put into a list with names given by lower-casing the column names from the csv file. If `save` is TRUE, the series are also stored in `envir` using those same names.

Value

A list of tis time series, one per column of the csv file. The list is returned invisibly if `save` is TRUE.

See Also

`tisPlot`, `tis.read.csv`, `read.table`

Usage

`tisLegend(..., xrel = 0.1, yrel = 0.1, xjust = 0, yjust = 1, boxType = "n", ncol = 1, cex = 1)`
Arguments

...  optional arguments to be passed on to `legend`. These can include x and y arguments to position the legend, or a list with components named x and y, such as the list returned by `locator(1)`.

xrel, yrel  Optional numbers between 0 and 1 to specify placement relative to the boundaries of the plot.

xjust, yjust, ncol  passed along to `legend`

boxType  passed along as bty to `legend`

cex  gets multiplied by the cex from `lastplot` and then passed on to `legend`

Details

This function is not strictly necessary, in that you could just call `legend` directly. `tisLegend` makes things a bit easier, however, by using the same argument names as `tisPlot` and `scatterPlot` to specify color, `lineType`, `plotChar` and `boxType`, rather than the less intuitive `col`, `lty`, `pch` and `bty` names. The `xrel` and `yrel` arguments provide an alternative way to specify legend placement, one that is used by the ChartMaker program.

Value

a list of the arguments that were sent on to `legend`, with class "tisLegend"

See Also

`legend`, `tisPlot`, `scatterPlot`

tisPlot  

Plot time indexed series (tis objects)

Description

tisPlot is a function with dozens of options for creating high quality time series plots. Can be used with `screenPage`.

Usage

tisPlot(...,
         leftAxis = TRUE, plotType = "l",
         lineType = "solid", lineWidth = 1.5,
         plotChar = "x", dataCex = 1, color = 1,
         midPoints = TRUE, dropNA = FALSE, xOffset = 0,
         xAxisMin = NULL, xAxisMax = NULL, xExpandBy = 0.04,
         xTickFreq = "Auto", xTickSkip = 0,
         xUnlabeledTickFreq = "None", xUnlabeledTickSkip = 0,
         xMinorTickFreq = "None", xMinorTickSkip = 0,
         ...)
Arguments

... any number of univariate or multivariate tis series to be plotted. Non-tis arguments will be converted by as.tis.

leftAxis logical. leftAxis[i] = TRUE means plot the i'th series against the left axis, otherwise plot it against the right axis.

plotType type of plot desired. Values are "p" for points, "l" for lines, "b" for both points and lines (lines miss the points), and "o" for overlaid points and lines.

lineType character or numeric vector specifying the line type for each series. The default is 1:nSeries, where nSeries is the number of series being plotted. Normally type 1 is solid, 2 is dotted, 3 and up are a mix of dots and dashes.

lineWidth numeric vector of line widths for the series. The default value is 1.5.

plotChar vector of characters (or numbers for plotting symbols – see the help for points for details) to be used for plotting points. Default is "*".

dataCex numeric vector. cex times these numbers give the character expansion factor for the data points. Default is 1.

color character or numeric vector specifies color for each series. Default is 1, the device default foreground color.
midPoints logical. midPoints[i] = TRUE aligns the data points of the i'th series with the middle day of the periods in which they fall, otherwise data points are aligned with the last day of their periods.

dropNA if TRUE, observations with NA values are dropped before calling lines.default to draw the lines and/or points on the plot. See the details section of the help for lines.tis for why you might or might not want to do this. The default is FALSE.

xOffset Shifts the data points for series[i] to the right by xOffset[i] days. This argument is replicated to be as long as the number of series.

xAxisMin minimum value of the x axis. If non-NULL, this overrides the calculation described in xExpandBy.

xAxisMax maximum value of the x axis. If non-NULL, this overrides the calculation described in xExpandBy.

xExpandBy a single number or two numbers between 0 and 1. xAxisMin and xAxisMax will be calculated by multiplicatively extending the data range in both directions by these amounts. Default value .04 extends the data range by 4% in each direction.

xTickFreq a string like the ones returned by tifName. This argument and xTickSkip jointly specify locations for labeled x axis ticks as follow: (i) find the ti's of the given frequency that correspond to xAxisMin and xAxisMax, then (ii) including those as endpoints, generate a sequence of every xTickSkip'th ti between them.

Two special strings can also be given. "none" means no labelled tick marks, while "auto" tries (not always successfully) to come up with reasonable tick locations automatically. "auto" also overrides any xTickSkip setting. The default is "auto".

xTickSkip a number used with xTickFreq to specify location of labeled tick marks. The default is zero.

xUnlabeledTickFreq same as xTickFreq, but for unlabeled major tick locations.

xUnlabeledTickSkip same as xTickSkip, but for unlabeled major tick locations.

xMinorTickFreq same as xTickFreq, but for minor tick locations.

xMinorTickSkip same as xTickSkip, but for minor tick locations.

dateFormat format string for x axis date labels. See strftime for formats. Default value "auto" tries to come up with a reasonable format automatically, for some bounded value of "reasonable".

cex times this number gives the character expansion factor for the x-axis labels. Default is 1.

midLabels if TRUE (the default) x axis label are centered between the ticks they label; if FALSE the label alignment depends on the setting of midPoints. If midPoints is TRUE, the labels (which reflect the last day of the period) are centered under the ticks, else they are (almost) right aligned with the ticks. If rotated labels are specified by nonzero xLabelRotationDegrees, tick labels are right aligned with their ticks.
yearLabels if TRUE place year labels centered under the x ticks they span. The default is FALSE.

xAxisLabel text to appear centered under the x axis. Must be a single character string, multi-line xAxisLabel is not supported. No default.

xSpace lines of space to set aside directly beneath the x-axis to hold tick, year and/or axis labels. Default is 4. The space created is xSpace times labCex.

log if TRUE use log scaling for y axes. Default is FALSE.

leftAxisMin minimum value of the left axis. If non-NULL, this overrides the calculation described in leftExpandBy.

leftAxisMax maximum value of the left axis. If non-NULL, this overrides the calculation described in leftExpandBy.

leftExpandBy a single number or two numbers between 0 and 1. leftAxisMin and leftAxisMax will be calculated by multiplicatively extending the data range in both directions by these amounts. Default value .04 extends the data range by 4% in each direction.

leftTicks number of ticks to draw on left axis at "pretty" locations. Default is 5. This argument is ignored if leftTickLocations is non-NULL.

leftTickLocations if non-NULL, a vector of desired tick locations or a string that evaluates to such a vector. The default value NULL lets the setting for leftTicks take effect.

labelLeftTicks If TRUE, label ticks on the left axis. Default is FALSE.

leftTickLabels character vector of tick labels or NULL (the default). If NULL and labelLeftTicks is TRUE, labels are constructed from the tick locations. This argument has no effect if labelLeftTicks is FALSE.

rightAxisMin minimum value of the right axis. If non-NULL, this overrides the calculation described in rightExpandBy.

rightAxisMax maximum value of the right axis. If non-NULL, this overrides the calculation described in rightExpandBy.

rightExpandBy a single number or two numbers between 0 and 1. rightAxisMin and rightAxisMax will be calculated by multiplicatively extending the data range in both directions by these amounts. Default value .04 extends the data range by 4% in each direction.

rightTicks number of ticks to draw on right axis at "pretty" locations. Default is 5. This argument is ignored if rightTickLocations is non-NULL.

rightTickLocations if non-NULL, a vector of desired tick locations or a string that evaluates to such a vector. The default value NULL lets the setting for rightTicks take effect.

labelRightTicks If TRUE, label ticks on the right axis. Default is FALSE.

rightTickLabels character vector of tick labels or NULL (the default). If NULL and labelRightTicks is TRUE, labels are constructed from the tick locations. This argument has no effect if labelRightTicks is FALSE.
ycex
cex times this number gives the character expansion factor for the left and right axis labels. Default is 1.

extendToptick
if TRUE (the default) the top tick of the y axes encloses the panel. leftAxisMax and rightAxisMax are increased as necessary to include the top tick for enclosing the panel.

cex
the base character expansion factor by which all of the **cex parameters are scaled. The default setting is the value of par("cex").

head
text to appear at the top of the figure region, with alignment determined by headAlign. No default.

headAlign
number indicating justification for the strings in head and sub. 0 means left justify, 1 means right justify, 0.5 (the default) means to center the text. Other numbers are a corresponding distance between the extremes.

headCex
cex times this number gives the character expansion factor for head. Default is 1.5.

sub
text to appear just under head, with alignment determined by headAlign. No default.

subCex
cex times this number gives the character expansion factor for sub. Default is 0.85.

leftTopLabel
text to appear at the top of the left axis, with alignment determined by toplabelAlign. No default.

rightTopLabel
text to appear at the top of the right axis, with alignment determined by toplabelAlign. No default.

toplabelAlign
number indicating alignment for the strings in leftTopLabel. 0 (the default) means left justify, 1 means right justify, 0.5 means to center the text. rightTopLabel, if given, is aligned by 1 - toplabelAlign.

labCex
cex times this number gives the character expansion factor for leftTopLabel, rightTopLabel, and xAxisLabel.

leftInsideLabel
text to appear left justified and just inside the upper left corner of the plot region. No default.

rightInsideLabel
text to appear right justified and just inside the upper right corner of the plot region. No default.

innerOffset
number between 0 and 1, a fractional offset for the inside labels. The left edge of leftInsideLabel is offset by this fraction of the x range from the left edge of the plot, as is the right edge of rightInsideLabel from the right edge of the plot.

innerLine
Number of lines in from the top edge of the plot to put the first line of the inside labels.

innerCex
cex times this number gives the character expansion factor for leftInsideLabel and rightInsideLabel.

foot
text to appear at the bottom of the figure region, with alignment determined by footAlign. Use a vector of character strings to get several footnotes. No default.
footAlign

number indicating justification for the strings in foot. 0 (the default) means left justify, 1 means right justify, 0.5 means to center the text. Other numbers are a corresponding distance between the extremes.

footCex
cex times this number gives the character expansion factor for foot. Default is 0.8.

footColor
character or numeric vector as long as foot, specifying the color for each footnote. Default is 1, the device default foreground color. The elements of this argument are cyclically repeated, if necessary, to make footColor the same length as foot.

footSpace
lines of space to set aside directly beneath the space allocated by xSpace to hold footnotes. The space created is footSpace times footCex. Default is length(foot); using a higher value will result in extra space in the bottom figure margin.

tck
The length of xTick, xUnlabeledTick and side tick marks as a fraction of the smaller of the width or height of the plotting region. Minor ticks (xMinorTicks) are 2/3 as long. If tck >= 0.5 it is interpreted as a fraction of the relevant side, so if tck = 1 grid lines are drawn. The default is tck = 0.03.

axisWidth
line width for the axes and box (if any). Default is 2.

start
starting date for the plot. The default is the earliest start time of all the series. This argument can be supplied in any of the forms understood by ti().

end
end date for the plot. The default is the latest end time of all the series. This argument can be supplied in any of the forms understood by ti().

boxType
character representing the type of box. Characters "o", "l" (ell), "7", "c" will produce boxes which resemble the corresponding upper-case letters. The value "n" will suppress boxes. The default is "u".

leftMargin
lines of space for the left margin. Default value (-1) figures this out automatically.

rightMargin
lines of space for the right margin. Default value (-1) figures this out automatically.

topMargin
lines of space for the top margin. Default value (-1) figures this out automatically.

bottomMargin
lines of space for the bottom margin. Default value (-1) figures this out automatically.

nberShade
if TRUE, use nberDates() to set the shadeDates. This will result in shading of NBER recession periods. The default is FALSE.

shadeDates
a matrix of yyyymmdd dates with two columns named 'Start' and 'End' that specifies the date ranges to be shaded. nberDates() produces a suitable matrix. If this argument is NULL and nberShade is not TRUE, no shading is done.

shadeColor
color to shade periods specified by shadeDates. Passed along to the ymdShade function.

shadeBorder
if shading is being done, this argument tells what color the border of the shaded areas should be drawn in. It can also be a logical, in which case TRUE draws borders in the foreground color and FALSE does not draw borders on the shaded areas. Default is FALSE.
polyArgs additional args that are passed along to the ymdShade function and eventually to polygon if shading of date ranges is being done.

Details

leftAxis, plotType, lineType, lineWidth, plotChar, dataCex, color and midPoints are all cyclically repeated to make them length nSeries, the number of series plotted.

Each of the text items head, sub, leftTopLabel, rightTopLabel, leftInsideLabel, rightInsideLabel, foot, and xAxisLabel can be given as a string, a collections of strings, or as a string that gets evaluated to one of the first two. (But xAxisLabel takes only a single string.) See the help details for evalOrEcho to see how this works.

Value

tisPlot invisibly returns a list of class "tisPlot" and elements named series, dateFormat, plotType, lineType, dataCex, color, plotChar, lineWidth, yLegendOffset, cex, xrange, leftRange, rightRange, midPoints and par. This list is useful mostly as an argument to tisLegend.

Note

The arguments for tisPlot and its sister function scatterPlot have more descriptive names than the corresponding arguments in plot. They are also all of unique types, unlike, for example, the lty argument in the usual R plotting functions, which can be either character or numeric. Limiting each argument to a single type was done to make it easier to design a user interface to drive the functions.

Use tisLegend to add legends to a plot created by tisPlot or scatterPlot.

See Also

evalOrEcho, scatterPlot, tisLegend, nbershade

Examples

firstTis <- tis(cumsum(rnorm(120)), start = c(1996,1), freq = 12)
secondTis <- tis(cumsum(rnorm(120)), start = c(1996,1), freq = 12)
tisPlot(firstTis, secondTis, color = c("red", "green"), lineType = "solid", head = "Two Random Walks")
tisLegend(legend = c("Random Walk 1", "Random Walk 2"))

series <- tis(cumsum(rnorm(200)), start = c(1960,1), tif = "quarterly")
tisPlot(series, xMinorTickFreq = "annual", nbershade = TRUE, head = "A Random Walk", sub = "Looks like an econ series", rightTopLabel = "$Billions")
romerLines()
**today**

*Time Index for the Current Date*

**Description**

Returns a ti for the current date.

**Usage**

`today(tif = "daily")`

**Arguments**

- **tif**: a ti Frequency, given as either a numerical code or a string. `tif()` with no arguments returns a list of the allowable numerical codes and names. The default "daily" returns a ti object for the current day.

**Value**

A ti object of the specified ti frequency that contains the current date in the time interval it represents. For example, if `tif` is "monthly", the returned ti object will be for the current month.

**See Also**

`ti, Sys.Date`

---

**updateColumns**

*Update lists and time series*

**Description**

`updateList` compares the names of `oldlist` and `newlist`, deletes the matching elements from a copy of `oldlist`, then returns the result of concatenating that list with `newlist`.

`updateColumns` updates columns of first series from same-named columns of second series using `mergeSeries()`. If second series has columns with names not found in `colnames` of first series, those columns are `cbind()`’ed onto first series.

**Usage**

`updateColumns(oldmat, newmat)`

`updateList(oldlist, newlist)`
window.tis

Arguments

oldmat  a multivariate tis series
newmat  a multivariate tis series
oldlist a list
newlist a list

Value

updateList returns the updated list.
updateColumns returns a multivariate tis series

See Also

tis, mergeSeries, cbind.tis

window.tis  Time windows for Time Indexed Series

Description

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

Usage

## S3 method for class 'tis'
window(x, start = NULL, end = NULL, extend = FALSE, noWarn = FALSE, ...)

Arguments

x  a tis object
start the start time of the period of interest.
end the end time of the period of interest.
extend logical. If TRUE, the start and end values are allowed to extend the series. If FALSE, attempts to extend the series are ignored and a warning is issued unless noWarn is FALSE.
noWarn logical. If FALSE (the default), warnings are generated if extend is FALSE and either (i) start is earlier than the start of the series or (ii) end is later than the end of the series.
... other arguments to this function are ignored.

Details

The start and end times can be ti objects, or anything that ti(z, tif = tif, freq = frequency), can turn into a ti object.
Value

A tis object that starts and ends at the given times.

Note

The replacement method window<-.tis has not been implemented. Use the subscript operator with a tti argument to replace values of a tis object.

Examples

```r
z <- tis(1:24, start = c(2001,1), freq = 12)
z2 <- window(z, start = 19991231, extend = TRUE) # z2 extends back with NA's
window(z, end = end(z) - 3)
```

---

**ymd**

*Extract parts of various Date-Time Objects*

Description

Extract the year, month or day, or all three (in yyyymmdd form), or the quarter, from a jul, ti, or from any object that jul() can handle.

Usage

```r
ymd(x, ...)
## S3 method for class 'jul'
ymd(x, ...)
## S3 method for class 'ssDate'
ymd(x, ...)
## S3 method for class 'ti'
ymd(x, offset = 1, ...)
## Default S3 method:
ymd(x, ...)
year(x, ...)
quarter(x, ...)
month(x, ...)
day(x, ...)
```

Arguments

- `x` a ti or jul, or something that jul() can create a jul object from.
- `...` other args to be passed to the method called by the generic function. year, quarter, month, day and ymd.default may pass these args to as.Date.
- `offset` for ti `x`, a number in the range [0,1] telling where in the period represented by `x` to find the day. 0 returns the first day of the period, while the default value 1 returns the last day of the period. For example, if `x` has tif = "wmonday" so that `x` represents a week ending on Monday, than any offset in the range [0,
ymdShade

Details

year, quarter, month and day call ymd, and thus understand the same arguments as it does. The
default implementation ymd.default passes it’s arguments to a call to the function jul, so all of
these functions work the same way that function does.

Value

ymd and it’s variants return numeric objects in yyyymmdd form.
year, quarter, month and day return numeric objects.
ymd() with no arguments returns today’s yyyymmdd.

See Also

jul, ti, as.Date

Examples

ymd() ## today’s date and time
weekFromNow <- ymd(today() + 7) ## today() returns a daily ti
year(jul(today()))
month(Sys.time())
## create a monthly tis (Time Indexed Series)
aTis <- tis(0, start = c(2000, 1), end = c(2004, 12), freq = 12)
ymd(ti(aTis)) ## the yyyymmdd dates of the observations

ymdShade

Shading Date Ranges

Description

ymdXY returns a list of x and y coordinates that can be fed to polygon to draw NBER shadings on
the current plot. If the last row of the ymds argument has a Start entry but an "NA" End entry and
openShade is FALSE, the returned list will not have coordinates for the last row, but will instead
include a vLine element that gives the x coordinate of the last Start. If openShade is TRUE (the
default), the list includes x and y coordinates for the last row of ymds, using the second element of
the horizontal range determined by the xrange parameter as its end time.

ymdShade shades date ranges on the current tisPlot. on the current plot. It calls ymdXY to get x
and y coordinates for the areas to be shaded and then passes those coordinates along with its own
arguments to polygon to do the shading. It also draws a vertical line at the appropriate location if
the list returned by ymdXY has a vLine element.
**ymdShade**

**Usage**

```r
ymdShade(ymds, col = grey(0.8), border = FALSE, xpd = FALSE, xrange = NULL, openShade = TRUE, ...)
```

```r
ymdXy(ymds, xrange = NULL, openShade = TRUE)
```

**Arguments**

All but `ymds` are passed along to `polygon`:

- a matrix of yyyymmdd dates with two columns named 'Start' and 'End' that specifies the date ranges to be shaded. `nberDates()` produces a suitable matrix.
- `ymds` - color to shade recessionary periods
- `border` - the default (FALSE) omits borders on the shaded regions. TRUE draws borders in the foreground color. Alternatively, specify a border color.
- `xpd` - should clipping take place?
- `...` - other args passed to `polygon`
- `xrange` - horizontal range over which recession shading should be drawn. The default value NULL uses the entire range of the plot. Note however the `tisPlot` uses the range of the data, which will generally differ the plot range unless the `tisPlot` parameters `leftExpandBy` and `rightExpandBy` are zero. You can force `tisPlot` to use the plot range by setting the parameter `nberArgs = list(xrange = NULL)`.
- `openShade` - governs how `ymdXy` and consequently `ymdShade` handle the case where the last row of the `ymds` matrix has an NA in the "End" column, indicating that the end date of the most recent shaded period is not known.

**Value**

As described above, `ymdXy` returns a list. `ymdShade` does not return anything useful.

**See Also**

`polygon`

**Examples**

```r
require("datasets")
plot(presidents, type='n', ylab="Presidents approval rating")
ymdShade(nberDates())
lines(presidents)
```
Index

*Topic algebra
  RowMeans, 57
  solve.tridiag, 66
*Topic aplot
  lines.tis, 48
  nberShade, 50
  nberShade.ggplot, 52
  plotWindow, 53
  tisLegend, 85
  ymdShade, 96
*Topic arith
  cumsum.tis, 24
*Topic array
  as.matrix.tis, 9
  RowMeans, 57
  t.tis, 70
*Topic attribute
  description, 29
*Topic character
  blanks, 17
  capitalize, 18
  stripBlanks, 68
*Topic chron
  as.Date, jul, 7
  currentMonday, 25
  currentPeriod, 26
  dayOfWeek, 28
  format.ti, 31
  hms, 35
  holidays, 36
  inferTi, 38
  Intraday, 39
  isEqualdayTif, 40
  isEqualYear, 41
  jul, 42
  latestPeriod, 45
  POSIXct, 54
  print.ti, 55
  setDefaultFrequencies, 64
  ssDate, 66
  ti, 70
  tiDaily, 73
  tif, 79
  tif2freq, 80
  today, 93
  ymd, 95
*Topic classes
  stripClass, 69
*Topic color
  frcolors, 33
  setColors, 64
*Topic connection
  tisFromCsv, 84
*Topic data
  assignList, 11
*Topic device
  frcolors, 33
*Topic dplot
  interpNA, 38
*Topic file
  csv, 23
  tisFromCsv, 84
*Topic hplot
  barplot2, 13
  scatterPlot, 58
  screenPage, 62
  tierChart, 73
  tisPlot, 86
*Topic list
  as.list.ti, 8
  asClassyList, 10
  columns, 19
  updateColumns, 93
*Topic manip
  as.list.ti, 8
  asClassyList, 10
  columns, 19
*Topic math
INDEX

between, 17
linearSplineIntegration, 46
*TG:Topic package
  tis-package, 3
*TG:Topic print
csv, 23
*TG:Topic programming
evalOrEcho, 30
*TG:Topic ts
  aggregate.tis, 5
  as.data.frame.tis, 6
  as.matrix.tis, 9
  as.ts.tis, 9
  barplot.tis, 12
  basis, 15
  cbind.tis, 18
  constantGrowthSeries, 20
  convert, 21
  cumsum.tis, 24
  currentMonday, 25
  currentPeriod, 26
  dateRange, 27
  dayOfWeek, 28
  growth.rate, 34
  interpNA, 38
  lags, 44
  latestPeriod, 45
  lines.tis, 48
  mergeSeries, 49
  naWindow, 50
  print.tis, 56
  RowMeans, 57
  start.tis, 67
  t.tis, 70
  ti, 70
  tiDaily, 73
  tierChart, 73
  tis, 81
  tisFilter, 83
  tisPlot, 86
  today, 93
  updateColumns, 93
  window.tis, 94
*TG:Topic utilities
  POSIXct, 54
  adjustableTierChart (tierChart), 73
  aggregate, 6, 23, 82
  aggregate.tis, 5
  apply, 6
  approx, 48
  approxfun, 38, 39
  as.character.jul (format.ti), 31
  as.character.ti (format.ti), 31
  as.data.frame.tis, 6
  as.Date, 7, 44, 72, 96
  as.Date.ti, 7
  as.Date.numeric, 7
  as.Date.ti (as.Date.ti), 7
  as.jul (jul), 42
  as.list, 8, 10
  as.list.jul (as.list.ti), 8
  as.list.ti, 8
  as.matrix.tis, 9
  as.POSIXct, 55
  as.POSIXct.jul (POSIXct), 54
  as.POSIXct.ti (POSIXct), 54
  as.ssDate (ssDate), 66
  as.ti, 38
  as.ti (ti), 70
  as.tis (tis), 81
  as.ts, 10
  as.ts.tis, 9
  asClassyList, 8, 10
  asJul (jul), 42
  assign, 11
  assignList, 11
  asTi (ti), 70
  axis, 15
  barplot, 13, 15
  barplot.tis, 12, 15
  barplot2, 12, 13, 13
  basePeriod (tif), 79
  basis, 15
  basis< (basis), 15
  between, 17
  blanks, 17, 68
  capitalize, 18
  cbind, 19
  cbind.tis, 18, 49, 82, 94
  class, 69
  colors, 34
  columns, 19
  constantGrowthSeries, 20
  convert, 6, 16, 21, 82
couldBeTi(ti), 70
csv, 23
cummax, 25
cummax.tis(cumsum.tis), 24
cummin, 25
cummin.tis(cumsum.tis), 24
cumprod, 25
cumprod.tis(cumsum.tis), 24
cumsum, 24
cumsum.tis, 24
currentApril (currentPeriod), 26
currentAugust (currentPeriod), 26
currentDecember (currentPeriod), 26
currentFebruary (currentPeriod), 26
currentFriday (currentMonday), 25
currentHalf (currentPeriod), 26
currentJanuary (currentPeriod), 26
currentJuly (currentPeriod), 26
currentJune (currentPeriod), 26
currentMarch (currentPeriod), 26
currentMay (currentPeriod), 26
currentMonday, 25
currentMonth (currentPeriod), 26
currentMonthDay (dayOfPeriod), 28
currentNovember (currentPeriod), 26
currentOctober (currentPeriod), 26
currentPeriod, 26
currentQ4 (currentPeriod), 26
currentQMonth (currentPeriod), 26
currentQuarter (currentPeriod), 26
currentSaturday (currentMonday), 25
currentSeptember (currentPeriod), 26
currentSunday (currentMonday), 25
currentThursday (currentMonday), 25
currentTuesday (currentMonday), 25
currentWednesday (currentMonday), 25
currentWeek, 46
currentWeek (currentPeriod), 26
currentYear (currentPeriod), 26
data.frame, 6
dateRange, 7
dateTimeClasses, 55
day (ymd), 95
dayOfMonth (dayOfWeek), 28
dayOfWeek (dayOfWeek), 28
dayOfYear (dayOfWeek), 28
description, 29
description<- (description), 29
documentation (description), 29
documentation<- (description), 29
easter (holidays), 36
day, 27, 68
day.tis, 82
day.tis (start.tis), 67
evaluation, 11
evalOrEcho, 30, 62, 92
fanSeries (constantGrowthSeries), 20
federalHolidays (holidays), 36
filter, 83
firstBusinessDayOf (dayOfPeriod), 28
firstBusinessDayOfMonth (dayOfPeriod), 28
firstDayOf (dayOfPeriod), 28
format.jul (format.ti), 31
format.POSIXlt, 31
format.ti, 31, 55, 56
fortify, 32
fortify.tis, 32
frColors, 33
frequency, 80
geom_rect, 52, 53
goodFriday (holidays), 36
graphical parameters, 15
growth.rate, 20, 34
growth.rate<- (growth.rate), 34
gsub, 68
hms, 35
holidays, 29, 36
holidaysBetween (holidays), 36
hourly, 35, 41
hourly (Intraday), 39
ilspline (linearSplineIntegration), 46
inaugurationDay (holidays), 36
inferTi, 38
interpNA, 38
Intraday, 39
is.jul (jul), 42
is.ssDate (ssDate), 66
is.ti (ti), 70
is.tis (tis), 81
isBusinessDay (holidays), 36
isEaster (holidays), 36
isGoodFriday (holidays), 36
isHoliday (holidays), 36
isIntradayTif, 40
isLeapYear, 41

jul, 29, 35, 39, 42, 44, 67, 72, 73, 96

lag (lags), 44
lag.tis (lags), 44
lags (lags), 44
lapply, 6
lastBusinessDayOf (dayOfPeriod), 28
lastBusinessDayOfMonth (dayOfPeriod), 28
lastDayOf (dayOfPeriod), 28
latestApril (latestPeriod), 45
latestAugust (latestPeriod), 45
latestDecember (latestPeriod), 45
latestFebruary (latestPeriod), 45
latestFriday (currentMonday), 25
latestHalf (latestPeriod), 45
latestJanuary (latestPeriod), 45
latestJuly (latestPeriod), 45
latestJune (latestPeriod), 45
latestMarch (latestPeriod), 45
latestMay (latestPeriod), 45
latestMonday (currentMonday), 25
latestMonth (latestPeriod), 45
latestMonthDay (dayOfPeriod), 28
latestNovember (latestPeriod), 45
latestOctober (latestPeriod), 45
latestPeriod, 45
latestQ4 (latestPeriod), 45
latestQuarter (latestPeriod), 45
latestSaturday (currentMonday), 25
latestSeptember (latestPeriod), 45
latestSunday (currentMonday), 25
latestThursday (currentMonday), 25
latestTuesday (currentMonday), 25
latestWednesday (currentMonday), 25
latestWeek, 27
latestWeek (latestPeriod), 45
latestYear (latestPeriod), 45
legend, 15, 86
linearSplineIntegration, 46
lines, 49
lines.tis, 48, 82, 88
lintegrate (linearSplineIntegration), 46
mergeSeries, 49, 82, 94
minutely, 41
minutely (Intraday), 39
month (ymd), 95
monthplot, 78

naWindow, 50
nberDates, 53
nberDates (nberShade), 50
nberShade, 50, 53, 92
nberShade, ggplot, 51, 52

nextBusinessDay, 29
nextBusinessDay (holidays), 36

observed (basis), 15
observed<- (basis), 15

palette, 64
par, 15, 53, 64
period (tif), 79
plot.default, 14
plot.window, 15, 54

points, 49, 59, 87
points.tis, 82
points.tis (lines.tis), 48

polygon, 51, 96, 97

POSIXct, 54
POSIXlt (POSIXct), 54

previousBusinessDay, 29
previousBusinessDay (holidays), 36
print.ti, 55
print.tis, 56, 82
print.tsf, 56

quarter (ymd), 95

read.csv, 84, 85
read.table, 85
romerLines (nberShade), 50
RowMeans, 57
rowMeans, 57
rows (columns), 19
RowSums (RowMeans), 57
rowSums, 57

scatterPlot, 58, 86, 92
screenPage, 62
secondly, 41
secondly (Intraday), 39
setColors, 34, 64
setDefaultFrequencies, 27, 46, 64
solve, 66
solve.tridiag, 66
spline, 48
splinefun, 38, 39
split.screen, 63
ssDate, 66
start, 27, 68
start.tis, 67, 82
start<- (start.tis), 67
strftime, 31, 88
stripBlanks, 68
stripClass, 69
stripTis (stripClass), 69
strptime, 84
Sys.Date, 93

t, 70
t.tis, 70
tapply, 6
ti, 23, 25, 27, 29, 35, 39, 44, 46, 70, 73, 80, 82, 85, 93, 96
ti.tis, 82
tiBusiness (tiDaily), 73
tiDaily, 73
tierChart, 73
tif, 23, 27–29, 40, 46, 72, 79, 80
tif2freq, 80
tifList (setDefaultFrequencies), 64
tifName, 40, 65, 72, 80
tifName (tif), 79
time zones, 54
tis, 23, 24, 27, 38, 70, 81, 85, 94
tis-package, 3
tisFilter, 83
tisFromCsv, 84
tisLegend, 85, 92
tisPlot, 62, 86, 86
title, 15
today, 93
tolower, 18
toupper, 18
try, 30
ts, 82
tunnelSeries (constantGrowthSeries), 20
updateColumns, 93
updateList (updateColumns), 93
window, 50
window.tis, 82, 94
write.table, 24
year (ymd), 95
ymd, 44, 72, 95
ymdShade, 51, 96
ymdXy (ymdShade), 96