Package 'epiforecast'

September 6, 2019

Type Package

Title Tools for forecasting semi-regular seasonal epidemic curves and similar time series

Version 0.0.1

Date 2015-12-09

Author Logan C. Brooks, Sangwon Hyun, David C. Farrow, Aaron Rumack, Ryan J. Tibshirani, Roni Rosenfeld

Maintainer Logan C. Brooks <lcbrooks@cs.cmu.edu>

Imports stats,
utils,
lubridate,
splines,
glmgen,
glmnet,
httr,
Rcpp,
Matrix,
rlist,
Hmisc,
weights,
hexbin,
plotrix,
tibble,
dplyr,
pipeR,
matrixStats,
quantreg,
lpSolve,
parallel,
R.utils,
RCurl,
pbmcapply

Description Tools for forecasting semi-regular seasonal epidemic curves and similar time series. Includes an empirical Bayes approach that forms a prior

by transforming historical curves, a basis regression approach that balances matching observations from the current season and matching historical seasons' measurements for future weeks, and timestep-by-timestep weighted kernel density estimation on backward differences parameterized by both the time series measurements and the current time.

License GPL-2

RoxygenNote 6.1.1

Collate 'RcppExports.R'
'backcasters.R'
'match.R'
'utils.R'
'br.R'
'checks.R'
'cv.R'
'cv_apply.R'
'delphi_epidata.R'
'empirical.futures.R'
'empirical.trajectories.R'
'simclass.R'
'ensemble.R'
'namesp.R'
'map_join.R'
'epiproject.R'
'forecast_type.R'
'get_completed_fluview_state_df.R'
'holidays.R'
'weeks.R'
'loaders.R'
'read.R'
'simplify2arrayp.R'
'target_spec.R'
'todo-by-file.R'
'twkde.R'

LinkingTo Rcpp

SystemRequirements

Suggests testthat

R topics documented:

`augmentWeeklyDF` *Augment df with epiweek/year/week/season/model.week, make weekly, full seasons*

Description

Given a df with with either (a) $epiweek, (b) $year and $week, or (c) $date, fills in (if missing) $epiweek, $year, $week, $date, $season, and $model.week. Fills in missing weekly data from all seasons so that each season in `df$season` has all of its model weeks in `df$model.week`. Assumes epi week convention.

Usage

```
augmentWeeklyDF(df, first.week.of.season = NULL)
```

Arguments

df	data.frame with week numbers and other data
first.week.of.season	the first week number in each season or NULL (the default); if NULL, then the first week of the season is assumed to be the week of the first data point.

Details

Entries in data.frame are assumed without any checks to be sorted and weekly (potentially with some skipped weeks).

---

br.sim *Function for making forecasts with the basis regression method with output matching the format of distributional forecasting methods.*

Description

Function for making forecasts with the basis regression method with output matching the format of distributional forecasting methods.

Usage

```
br.sim(full.dat, max.n.sims = 100L, baseline = 0, bootstrap = TRUE,
  control.list = get_br_control_list(), ...)
```

Arguments

max.n.sims	single non-NA integer value or NULL: the number of curves to sample from the inferred distribution
...	arguments to forward to br.smoothedCurve.

Details

For the basis regression method, there is a single column per trajectory in new.dat containing the smoothed curve outputted by br.smoothedCurve, unless max.n.sims is non-NULL, in which case, it is a resampling of these smoothed curves.

Value

a sim object — a list with two components:

ys: a numeric matrix, typically with multiple columns; each column is a different possible trajectory for the current season, with NA's in the input for the current season filled in with random draws from the forecasted distribution, and non-NA's (observed data) filled in with an imagined resampling of noise based on the model (for some models, the non-NA values will remain unchanged).

weights: a numeric vector; assigns a weight to each column of ys, which is used by methods relying on importance sampling.

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file.prefix="fluview_nat_allfetch"),
           "wili", min.points.in.season=52L)
## Recent historical seasons + current season, minus 2009 (nonseasonal
## pandemic) season:
full.dat = split(fluview.nat.recent.df$wili, fluview.nat.recent.df$season)
names(full.dat) <- sprintf("S%s", names(full.dat))
full.dat <- full.dat[names(full.dat)!="S2009"]
sim = br.sim(full.dat, baseline = 2.1)
```

---

br.smoothedCurve *Function for making forecasts with the basis regression method*

Description

Estimates missing values in `dat.obj[[cur.season]]` by regressing the mean of "pseudo-trajectories" formed from non-NA observations from `dat.obj[[cur.season]]` and "pseudo-observations" formed from `dat.obj[-cur.season]` on a set of basis elements.

Usage

```
br.smoothedCurve(full.dat, dat.obj, cur.season,
  control.list = get.br.control.list())
```

Arguments

`dat.obj`	assumed to be a list, of length equal to number of past seasons. Each item here is itself a list, each component containing a vector of "signals" for that seasons.
`cur.season`	the number of the season to be forecast. Must be in between 1 and the length of dat.obj.
`control.list`	Contains simulation settings.

Details

First, constructs a pseudo-trajectory for each training trajectory (`dat.obj[-cur.season]`) by shifting the training trajectory so that the maximum of its observations at times where `dat.obj[[cur.season]]` is non-NA aligns more closely with the maximum of `dat.obj[[cur.season]]` (where it is non-NA); the alignment procedure consists of a time shift (so that the partial maximum of the training and test trajectories are the same) and a scale (controlled by `scale.method`, `baseline`, and

max.scale.factor). The pseudo-trajectory is formed by taking dat.obj[[cur.season]] where it is non-NA and the aligned training trajectory where dat.obj[[cur.season]] is NA.

Second, the mean of the pseudo-trajectories is regressed on a collection of basis elements to produce a single curve that provides estimates for dat.obj[[cur.season]] where it is NA.

Value

a numeric vector containing a smoothed version of the past observations and future "pseudo-observations" (predictions).

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

bw.SJnrd0 *bw.SJ with bw.nrd0 as fallback*

Description

At least sometimes when calling bw.SJ on backfill updates (lag.info$residual), an error is generated ("sample is too sparse to find TD"). bw.SJnrd0 uses bw.SJ if it succeeds, and falls back to bw.nrd0 if it generates any error.

Usage

```
bw.SJnrd0(x)
```

Arguments

x	numeric vector: the observations

Value

single numeric: the bandwidth selection

c.sim *Add information to a* sim *object*

Description

Uses recursive=FALSE and use.names=TRUE when forwarding to the list c method; any attempt to override these values will generate an error.

Usage

```
## S3 method for class 'sim'
c(my.sim, ...)
```

Arguments

my.sim	a sim object
...	list of components to add to the sim object; must lead to a resulting sim object with components that are all uniquely, nontrivially (!="") named

Value

sim object with the given components appended

c.target_forcast *Add information to a* target_forcast *object*

Description

Uses recursive=FALSE and use.names=TRUE when forwarding to the list c method; any attempt to override these values will generate an error.

Usage

```
## S3 method for class 'target_forcast'
c(target.forcast, ...)
```

Arguments

target.forcast	a target_forcast object
...	list of components to add to the target_forcast object; must lead to a resulting target_forcast object with components that are all uniquely, nontrivially (!="") named

Value

target_forcast object with the given components appended

c.uniform_forecast *Add information to a* uniform_forecast *object*

Description

Uses recursive=FALSE and use.names=TRUE when forwarding to the list c method; any attempt to override these values will generate an error.

Usage

```
## S3 method for class 'uniform_forecast'
c(classed.list, ..., recursive = FALSE,
  use.names = TRUE)
```

Arguments

...	list of components to add to the uniform_forecast object; must lead to a resulting uniform_forecast object with components that are all uniquely, non-trivially (!="") named
uniform.forecast	a uniform_forecast object

Value

uniform_forecast object with the given components appended

check.file.contents *Function to check if a data file is properly formatted (i.e. contains the headers, is filled with numeric values, etc.) Current implementation is very memory-inefficient.*

Description

Function to check if a data file is properly formatted (i.e. contains the headers, is filled with numeric values, etc.) Current implementation is very memory-inefficient.

Usage

```
check.file.contents(filename)
```

Arguments

filename	name of data filewith each column equal to each season, with the first (n-1) columns to be, and the n'th column with the current season.

check.list.format *Performs various checks on full.dat as a list of numeric vectors.*

Description

Performs various checks on full.dat as a list of numeric vectors.

Usage

```
check.list.format(full.dat)
```

check.table.format *Performs various checks on full.dat as a table (matrix/data.frame).*

Description

Performs various checks on full.dat as a table (matrix/data.frame).

Usage

```
check.table.format(full.dat)
```

cv.compare *Compare two cv.sim objects*

Description

Compare two cv.sim objects

Usage

```
cv.compare(cv.sim1, cv.sim2)
```

`cv.sim` *Class for cv objects; contains (1) control list (2) for each forecasting time and for each left-out season (fold), what are the densities in the 52 by n.grid block of the 2d plane?\ (3) several things pre-calculated; the prediction scores (negative log-likelihood) for 4 target forecasts. The idea is that, instead of 52 by nsim curves, we can store 52 by n.grid values that store the density estimates, where n.grid can be hundreds, while n.sim may be 10,000's. It should return /all/ quantities required for doing*

Description

Class for cv objects; contains (1) control list (2) for each forecasting time and for each left-out season (fold), what are the densities in the 52 by n.grid block of the 2d plane?\ (3) several things pre-calculated; the prediction scores (negative log-likelihood) for 4 target forecasts. The idea is that, instead of 52 by nsim curves, we can store 52 by n.grid values that store the density estimates, where n.grid can be hundreds, while n.sim may be 10,000's. It should return /all/ quantities required for doing

Usage

```
cv.sim()
```

`cv_apply` `apply`*-like function applying binary functions on training and test set selections*

Description

`apply`-like function applying binary functions on training and test set selections

Usage

```
cv_apply(data, indexer_list, fn, parallel_dim_i = 0L, ...)
```

Arguments

`data`	an array
`indexer_list`	a named list with one entry per dimension of `data` specifying how to select training and test indices for that dimension; for the `i`th entry: • `each=NULL` slices both training and test data into `dim(data)[[i]]` pieces by selecting each index along the `i`th dimension; the corresponding output dimension width of `dim(data)[[i]]` • `all=NULL` performs no indexing along the `i`th dimension for either training or test data; the corresponding output dimension has width 1 and name "all"

- `smear=relative.indices` acts like `each=NULL`, but instead of each slices corresponding to a single index, allows for nearby indices to be included as well; for the `j`th slice, includes data for valid indices in `j+relative.indices`
- `ablation=NULL` acts like `each=NULL`, but for the `j`th "slice", excludes, rather than selects, data corresponding to index `j`
- `subsets=subset.list` indexes both training and test data based on the subsets in `subset.list`; each entry in `subset.list` should be a vector of indices (logical, integer, or character) into the `i`th dimension of `data`; the output dimension has width `length(subset.list)` and names `names(subset.list)`
- `loo=NULL` performs leave-one-out cross-validation indexing: the training set like `ablation=NULL`, while the test set is indexed like `each=NULL`
- `oneahead=test_start_ind` slices the test data by taking each index greater than or equal to the specified single *integer* index `test_start_ind`; the test data corresponding to index i is paired with training data from indices strictly preceding i
- `loo_oneahead=oneahead_start_ind` slices the test data by each index (similar to each); if the index i is less than the specified single *integer* index `oneahead_start_ind`, then it is paired with training data from indices strictly preceding `oneahead_start_ind`, excluding i; if i >= `oneahead_start_ind`, then the training data contains all indices strictly preceding i

`fn` a `function(training.slice, test.slice)` returning a scalar, vector, matrix, array, or list, with the same class and fixed structure for all inputs

Value

an array with dimensionality equal to the sum of the dimensionless of the output of `fn` and of `data`

`dat.to.matrix` *Numeric matrix of the first* `n` *elements of each numeric vector in* `dat`.

Description

A more efficient implementation of `sapply(dat, `[`, seq_len(n))`. Any vectors in `dat` with length less than `n` are extended with `NA_real_`'s at the end.

Usage

```
dat.to.matrix(dat, n)
```

Arguments

`dat`	a list of numeric vectors
`n`	a single integer: the number of elements to take from each vector

Value

a `n`-by-`length(dat)` numeric matrix

Examples

```
dat = list(11:15, 21:26)
dat.to.matrix(dat, 5) # (5x2: dat[[2]] is cut off)
dat.to.matrix(dat, 6) # (6x2: dat[[1]] is extended with NA_real_)
n = 3
identical(c(n, length(dat)), dim(dat.to.matrix(dat, n)))
```

`DatesOfSeason` *Get the first weekday of every week in the given seasons*

Description

Get the first weekday of every week in the given seasons

Usage

```
DatesOfSeason(season, first.week, first.wday, owning.wday)
```

Arguments

`season`	integer-valued vector: season numbers
`first.week`	integer-valued vector: first week number of each season
`first.wday`	integer-valued vector of weekday numbers: first weekday number in each week
`owning.wday`	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

$`length(season)`$-length named list of 52/53-length `Date` vectors

`DateToYearWeekWdayDF` *Convert dates to a data frame of year-week-wday according to a specified convention.*

Description

Convert dates to a data frame of year-week-wday according to a specified convention.

Usage

```
DateToYearWeekWdayDF(date, first.wday, owning.wday)
```

Arguments

date	object compatible with as.Date: the dates to convert
first.wday	weekday number(s) (0–7, Sunday can be 0 or 7): the weekday that is considered the beginning of the week; typically Sunday or Monday
owning.wday	weekday number(s) (0–7, Sunday can be 0 or 7): a week is assigned to a given year if the owning weekday of that week falls in that year; typically first.wday or first.wday+3

Value

a data frame (tbl_df) with three columns, $year, $week, and $wday, corresponding to the given dates using the convention specified by first.wday and owning.wday; each wday entry will be %in% 0:6.

---

downsample_sim *Resample a sim (if necessary) to get <= max.n.sims simulated curves*

Description

Resample a sim (if necessary) to get <= max.n.sims simulated curves

Usage

```
downsample_sim(sim.obj, max.n.sims)
```

Arguments

sim.obj	a sim object
max.n.sims	a single non-NA non-negative integer; the inclusive upper bound on the number of simulated curves in the result

Details

Resampling is only performed if the number of simulations needs to change. Resampling is done without replacement to prevent unnecessary reductions in the number of "effective particles".

Value

a sim object with <= max.n.sims simulated curves; the weights in the result are the same as the weights of the corresponding curves in sim.obj, implying that the sum of the weights in the result is less than or equal to the sum of the weights in the result (due to sampling without replacement), similar to reductions in "effective particles" in particle sampling contexts due to resampling.

dur *Calculate the duration of a trajectory*

Description

Calculate the duration of a trajectory

Usage

```
dur(trajectory, baseline, is.inseason, ...)
```

Arguments

trajectory	a vector
baseline	the onset threshold
is.inseason	logical vector length-compatible with trajectory, acting as a mask on possible output values
...	ignored

Value

a single non-NA integer: the number of indices which are part of the in-season and part of a run of at least three consecutive observations above the onset threshold

empirical.futures.sim *Simulate future in current trajectory with empirical (historical) distribution*

Description

Simulate future in current trajectory with empirical (historical) distribution

Usage

```
empirical.futures.sim(full.dat, baseline = NA_real_,
  max.n.sims = 2000L)
```

Arguments

max.n.sims	single non-NA integer value or NULL: the number of curves to sample from the inferred distribution

Value

a sim object — a list with two components:

ys: a numeric matrix, typically with multiple columns; each column is a different possible trajectory for the current season, with NA's in the input for the current season filled in with random draws from the forecasted distribution, and non-NA's (observed data) filled in with an imagined resampling of noise based on the model (for some models, the non-NA values will remain unchanged).

weights: a numeric vector; assigns a weight to each column of ys, which is used by methods relying on importance sampling.

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file.prefix="fluview_nat_allfetch"),
           "wili", min.points.in.season=52L)
## Recent historical seasons + current season, minus 2009 (nonseasonal
## pandemic) season:
full.dat = split(fluview.nat.recent.df$wili, fluview.nat.recent.df$season)
names(full.dat) <- sprintf("S%s", names(full.dat))
full.dat <- full.dat[names(full.dat)!="S2009"]
## Recent historical seasons minus 2009:
dat = head(full.dat, -1L)
## Current season:
new.dat = tail(full.dat, 1L)[[1]]
## Sample from conditional curve distribution estimate using CDC's 2015
## national %wILI onset threshold baseline of 2.1:
sim = empirical.futures.sim(dat, new.dat, 2.1, max.n.sims=50)
```

---

empirical.trajectories.sim

Simulate future in current trajectory with empirical (historical) distribution

---

Description

Simulate future in current trajectory with empirical (historical) distribution

Usage

```
empirical.trajectories.sim(full.dat, baseline = NA_real_,
  max.n.sims = 2000L)
```

Arguments

max.n.sims	single non-NA integer value or NULL: the number of curves to sample from the inferred distribution

Value

a sim object — a list with two components:

ys: a numeric matrix, typically with multiple columns; each column is a different possible trajectory for the current season, with NA's in the input for the current season filled in with random draws from the forecasted distribution, and non-NA's (observed data) filled in with an imagined resampling of noise based on the model (for some models, the non-NA values will remain unchanged).

weights: a numeric vector; assigns a weight to each column of ys, which is used by methods relying on importance sampling.

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file.prefix="fluview_nat_allfetch"),
           "wili", min.points.in.season=52L)
## Recent historical seasons + current season, minus 2009 (nonseasonal
## pandemic) season:
full.dat = split(fluview.nat.recent.df$wili, fluview.nat.recent.df$season)
names(full.dat) <- sprintf("S%s", names(full.dat))
full.dat <- full.dat[names(full.dat)!="S2009"]
## Recent historical seasons minus 2009:
dat = head(full.dat, -1L)
## Current season:
new.dat = tail(full.dat, 1L)[[1]]
## Sample from conditional curve distribution estimate using CDC's 2015
## national %wILI onset threshold baseline of 2.1:
sim = empirical.trajectories.sim(dat, new.dat, 2.1, max.n.sims=50)
```

---

epiweek_Seq *Compute a sequence of epiweeks with the specified range*

Description

Uses the rules of Seq; if end.epiweek lies before start.epiweek, an empty sequence will be returned.

Usage

```
epiweek_Seq(from.epiweek, to.epiweek)
```

Arguments

from.epiweek	the first epiweek to include in the sequence
to.epiweek	the first epiweek to include in the sequence

Value

a sequence of epiweeks

Examples

```
epiweek_Seq(201450L, 201503L)
epiweek_Seq(201550L, 201603L)
epiweek_Seq(201550L, 201203L)
epiweek_Seq(201250L, 201503L)
```

---

epi_week_to_model_week

Convert epi week numbers into model week numbers

Description

Convert epi week numbers into model week numbers

Usage

```
epi_week_to_model_week(epi.week, first.week.of.season, n.weeks.in.season)
```

Arguments

epi.week	integer or numeric vector; epi week numbers (wrapper versions of model week numbers in 1..52 + [0,1) or 1..53 + [0,1))
first.week.of.season	epi week number of the first element in a trajectory (i.e., corresponding to time 1L)
n.weeks.in.season	52L or 53L — the length of the indexed trajectory, corresponding to the number of weeks in the first year contained in the season

Value

integer or numeric vector with same length as `epi.week`; model weeks — epi weeks that don't (necessarily) wrap around to 1 at the start of a new year (after week 52 or 53 of the previous year) but rather wrap around when a season ends, starting back at `first.week.of.season`; values could be integers between `first.week.of.season` to `first.week.of.season+52L` or `first.week.of.season+53L` for integers, or a valid integer value plus a number in the clopen interval [0,1) for numeric vectors

---

`fetchEpidataDF` *Fetch & cache* Rhrefhttps://github.com/undefx/delphi-epidatadelphi-epidata *data, convert it to a* `data.frame`

Description

Fetch & cache delphi-epidata data, convert it to a `data.frame`

Usage

```
fetchEpidataDF(source, area, lag = NULL, first.week.of.season = NULL,
  first.epiweek = NULL, last.epiweek = NULL,
  cache.file.prefix = NULL, cache.invalidation.period = as.difftime(1L,
  units = "days"), force.cache.invalidation = FALSE,
  ignore.no.results = FALSE, silent = FALSE)
```

Arguments

`source` length-1 character vector; name of data source; one of

- `"fluview"`: U.S. Outpatient Influenza-like Illness Surveillance Network (ILINet) data from CDC FluView reports; weekly, national or by HHS region; updated weekly
- `"ilinet"`: Estimates of ILINet data at state level; weekly; static
- `"gft"`: Google Flu Trends data
- `"ght"`: Google Health Trends data; currently restricted, not supported by this function
- `"twitter"`: Twitter data; currently restricted, not supported by this function
- `"wiki"`: Wikipedia access log data; currently not supported by this function
- `"nidss.flu"`: Taiwan National Infectious Disease Statistics System (NIDSS) outpatient ILI data
- `"nidss.dengue"`: Taiwan National Infectious Disease Statistics System (NIDSS) dengue incidence data

`area` length-1 character vector; name of area; possibilities by `source` are:

- `"fluview"`: `"hhs1"`, `"hhs2"`, ..., `"hhs10"`, `"nat"`
- `"ilinet"`: 2-character state abbreviation or `"DC"`
- `"gft"`: listed at https://github.com/undefx/delphi-epidata#gft-parameters

- "nidss.flu", "nidss.dengue": one of "nationwide", "central", "eastern", "kaoping", "northern", "southern", "taipei"

lag
: single integer value or NULL; for supported data sources for which observations for a particular time are revised at later times as more data is received, gives access to some of the older versions of the data. NULL gives the latest revision of the observation. An integer value gives the value for each observation lag weeks after its initial report; the value 0 corresponds to using the initial report for each data point. (This function does not reconstruct what a the data looked like at any particular point in the past, but can be used to do so.)

first.week.of.season
: single integer value giving the week number that seasons start with, or NULL to have seasons start with the week number of the first data point

first.epiweek
: year-epiweek as string in form "YYYYww" specifying a lower limit on times for which data will be fetched, or NULL to not impose a lower limit

last.epiweek
: year-epiweek as string in form "YYYYww" specifying an upper limit on times for which data will be fetched, or NULL to not impose an upper limit

cache.file.prefix
: length-1 character containing filepath prefix for files used to cache data from the delphi-epidata server, or NULL not to cache

cache.invalidation.period
: single difftime: time duration that must pass from last fetch for a new fetch to be performed instead of reading from the cache; default is one day

force.cache.invalidation
: single non-NA logical; if TRUE, then the cache.invalidation.period and a fetch will always be performed (the cache will not be read)

Value

data frame (tbl_df) with $date, $year, $week, and corresponding data (fields differ based on source)

Examples

```
## All fluview data at the national level:
fluview.nat.all.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file.prefix="fluview_nat_allfetch"),
           "wili", min.points.in.season=33L)
## Fluview data at the national level for more recent seasons for which data
## was reported for all weeks (not just those in the influenza season), plus
## the current season:
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file.prefix="fluview_nat_allfetch"),
           "wili", min.points.in.season=52L)
```

`fetchEpidataFullDat` *Fetch data from delphi-epidata, trim partial seasons, and convert to list of trajectories*

Description

Fetch data from delphi-epidata, trim partial seasons, and convert to list of trajectories

Usage

```
fetchEpidataFullDat(source, area, signal.ind, min.points.in.season = 52L,
  lag = NULL, first.week.of.season = NULL, first.epiweek = NULL,
  last.epiweek = NULL, cache.file.prefix = NULL,
  cache.invalidation.period = as.difftime(1L, units = "days"),
  force.cache.invalidation = FALSE)
```

Arguments

`source` length-1 character vector; name of data source; one of

- `"fluview"`: U.S. Outpatient Influenza-like Illness Surveillance Network (ILINet) data from CDC FluView reports; weekly, national or by HHS region; updated weekly
- `"ilinet"`: Estimates of ILINet data at state level; weekly; static
- `"gft"`: Google Flu Trends data
- `"ght"`: Google Health Trends data; currently restricted, not supported by this function
- `"twitter"`: Twitter data; currently restricted, not supported by this function
- `"wiki"`: Wikipedia access log data; currently not supported by this function
- `"nidss.flu"`: Taiwan National Infectious Disease Statistics System (NIDSS) outpatient ILI data
- `"nidss.dengue"`: Taiwan National Infectious Disease Statistics System (NIDSS) dengue incidence data

`area` length-1 character vector; name of area; possibilities by `source` are:

- `"fluview"`: `"hhs1"`, `"hhs2"`, ..., `"hhs10"`, `"nat"`
- `"ilinet"`: 2-character state abbreviation or `"DC"`
- `"gft"`: listed at https://github.com/undefx/delphi-epidata#gft-parameters
- `"nidss.flu"`, `"nidss.dengue"`: one of `"nationwide"`, `"central"`, `"eastern"`, `"kaoping"`, `"northern"`, `"southern"`, `"taipei"`

`signal.ind` single non-NA character/integer-valued index for column of `df`

`min.points.in.season` the minimum number of non-`NA` values for `signal.ind` that a season must have in order to be retained; all rows corresponding to seasons containing less observations will be removed from `df`

`lag` single integer value or `NULL`; for supported data sources for which observations for a particular time are revised at later times as more data is received, gives access to some of the older versions of the data. `NULL` gives the latest revision of the observation. An integer value gives the value for each observation `lag` weeks after its initial report; the value `0` corresponds to using the initial report for each data point. (This function does not reconstruct what a the data looked like at any particular point in the past, but can be used to do so.)

`first.week.of.season` single integer value giving the week number that seasons start with, or `NULL` to have seasons start with the week number of the first data point

`first.epiweek` year-epiweek as string in form `"YYYYww"` specifying a lower limit on times for which data will be fetched, or `NULL` to not impose a lower limit

`last.epiweek` year-epiweek as string in form `"YYYYww"` specifying an upper limit on times for which data will be fetched, or `NULL` to not impose an upper limit

`cache.file.prefix` length-1 `character` containing filepath prefix for files used to cache data from the delphi-epidata server, or `NULL` not to cache

`cache.invalidation.period` single `difftime`: time duration that must pass from last fetch for a new fetch to be performed instead of reading from the cache; default is one day

`force.cache.invalidation` single non-`NA` logical; if `TRUE`, then the `cache.invalidation.period` and a fetch will always be performed (the cache will not be read)

Value

named list of 52/53-length is.numeric vectors; each vector is the trajectory of the given signal for a single season, with NA's used to fill in for missing and future data; the names are of the form "SYYYY", where YYYY is the first year of the season, e.g., "S2003" corresponds to the 2003–2004 season.

Examples

```
## All fluview data at the national level:
fluview.nat.all.full.dat =
  fetchEpidataFullDat("fluview", "nat", "wili", 33L, first.week.of.season=21L,
                      cache.file.prefix="fluview_nat_allfetch")
## Fluview data at the national level for more recent seasons for which data was
## reported for all weeks (not just those in the influenza season), plus the
## current season:
fluview.nat.recent.full.dat =
  fetchEpidataFullDat("fluview", "nat", "wili", 52L, first.week.of.season=21L,
                      cache.file.prefix="fluview_nat_allfetch")
```

fetchEpidataHistoryDF *Combine current and lagged epidata into a data frame*

Description

Take the epidata data frames for the specified lags plus the current (unlagged) epidata data frame and combine them together into a single data frame (`tbl_df`). There are two major uses:

- Preparing an epidata.history.df for `mimicPastEpidataDF`: `lags` should include all possible `lag` values that could contain revisions; however, larger `lag` values may appear in the resulting data frame from the current epidata data frame, indicating that the corresponding data are finalized and there were no non-finalized versions recorded.
- Studying the errors for a set of `lags`: `lags` should include all `lags` of interest; the resulting data frame may include other lags from the current epidata data frame, which should be filtered out; this history data frame should be right-joined with a "ground truth" data frame.

Usage

```
fetchEpidataHistoryDF(source, area, lags, first.week.of.season = NULL,
  first.epiweek = NULL, last.epiweek = NULL, cache.file.prefix,
  cache.invalidation.period = as.difftime(1L, units = "days"),
  force.cache.invalidation = FALSE, ignore.no.results = FALSE,
  silent = FALSE)
```

See Also

mimicPastEpidataDF

fetchUpdatingResource *Fetch or read cache of a resource that potentially updates over time*

Description

Fetch or read cache of a resource that potentially updates over time

Usage

```
fetchUpdatingResource(fetch.thunk.fun, check.response.fun,
  cache.file.prefix = NULL, cache.invalidation.period = as.difftime(1L,
  units = "days"), force.cache.invalidation = FALSE, silent = FALSE)
```

Arguments

`fetch.thunk.fun`
: a function that takes no arguments that, when called, attempts to fetch the desired resource and returns the response

`check.response.fun`
: a function that takes a fetch response as input and either (a) stops if the response indicates an unsuccessful fetch or (b) returns nothing; this prevents invalid fetch responses from being saved to the cache.

`cache.file.prefix`
: length-1 `character` containing a filepath prefix to use when creating files to cache the fetch response, or `NULL` to disable caching

`cache.invalidation.period`
: length-1 `difftime`; the minimum amount of time that must pass between the last cache update and a call to this function for `fetch.thunk.fun` to be called again rather than using the cache result (unless `force.cache.invalidation=TRUE`)

`force.cache.invalidation`
: `TRUE` to force a cache update, even if the cache invalidation period has not passed; otherwise `FALSE`

`do.display.message`
: `TRUE` or `FALSE`; `TRUE` to enable messages about whether the cache is being used or not, or `FALSE` to disable these messages

---

`firstEpiweekOfUniverse` *Epiweek before which no data should exist*

Description

Assume that the universe was created in 1234 AD EW01, and that no data will be from times before this.

Usage

```
firstEpiweekOfUniverse
```

Format

An object of class `integer` of length 1.

get.latest.time *Get index (time) of last non-NA in a vector or other new.dat.sim*

Description

Get index (time) of last non-NA in a vector or other new.dat.sim

Usage

```
get.latest.time(new.dat.sim)
```

get_br_control_list *Generates control list for BR*

Description

Generates control list for BR

Usage

```
get_br_control_list(parent = NULL, max.n.sims = 100L, n.out = 53L,
  model = "Basis Regression", max.match.length = NULL, df = 10,
  w = 1, smooth = TRUE, model.noisy = TRUE, baseline = 0,
  basis = "bs", max.scale.factor = 3, cv.rule = c("min", "1se"),
  scale.method = c("none", "max", "last"))
```

Arguments

max.match.length	the maximum number of past data points to which the spline is fitted. The default, NULL, indicates to use all past data points when fitting the spline.
df	the degrees of freedom for the basis. Default is 10.
w	the mixing weight between the two loss terms, as in: sum over obs times (yobs - f)^2 + w * sum over unobs times (ypast - f)^2, where yobs is the current season's observed data, ypast is the past season's data, suitably transformed, and f is the function to be estimated.
smooth	logical; if TRUE, past observations and future "pseudo-observations" (predictions) will be smoothed; if FALSE, the observations and pseudo-observations will be returned unsmoothed.
model.noisy	logical; if TRUE and bootstrapping is enabled, will inject noise into future values and pin past observations to the observed values; if FALSE or TRUE but used when bootstrapping is disabled, br.sim will return the "non-noisy" regression curve

`baseline` the anchoring point used for scaling past season's data; data above the baseline are scaled about the baseline. The default value, NA, indicates to scale about 0 (regardless of sign).

`basis` type of basis to use. So far only "bs" (B-splines) are implemented.

`max.scale.factor` single numeric: a limit on the amount of scaling performed by the scaling method: scale factors over `max.scale.factor` and under `1/max.scale.factor` will be clipped.

`cv.rule` one of "min" or "1se", where "min" gives the usual rule, and "1se" uses the 1-standard-error rule.

`scale.method` whether and how to scale past seasons to match data from the `cur.seasonth` trajectory: "none" performs no scaling; "max" scales the maximum of each other season's trajectory — restricted to times which correspond to non-`NA` values in the `cur.seasonth` trajectory — so that it matches the maximum of the `cur.seasonth` trajectory; and "last" performs the same scaling using data at the time corresponding to the latest observation in the `cur.seasonth` trajectory

---

`is_christmas` *Test if Date is Christmas Day (vectorized)*

Description

Test if Date is Christmas Day (vectorized)

Usage

```
is_christmas(Date)
```

Arguments

`Date` `Date` vector to test

Value

logical vector

is_newyear *Test if Date is (Gregorian) New Year's Day (vectorized)*

Description

Test if Date is (Gregorian) New Year's Day (vectorized)

Usage

```
is_newyear(Date)
```

Arguments

Date	Date vector to test

Value

logical vector

is_thanksgiving *Test if Date is Thanksgiving Day (vectorized)*

Description

Test if Date is Thanksgiving Day (vectorized)

Usage

```
is_thanksgiving(Date)
```

Arguments

Date	Date vector to test

Value

logical vector

lastWeekNumber *The number of weeks assigned to a given year or years.*

Description

The length of one input should be evenly divisible by the length of the other.

Usage

```
lastWeekNumber(year, owning.wday)
```

Arguments

year	integer/NA-valued vector of years
owning.wday	integer vector of wday numbers (0–7, 0 and 7 are both Sunday): a week is assigned to a given year based on whether this weekday is contained in that year

Value

the number of weeks assigned to each year in years

Examples

```
## The number of epi weeks in each year from 1990 to 2020:
lastWeekNumber(1990:2020, 3)
```

make_sim_ys_colors *Make transparent colors according to weights in (0,1).*

Description

Make transparent colors according to weights in (0,1).

Usage

```
make_sim_ys_colors(weights)
```

Arguments

weights	Numeric vector with elements in [0,1].

Value

Character vector containing color codes.

map_join_ *Map a function over the natural (Cartesian/other) join of array-like objects*

Description

Map a function over the natural (Cartesian/other) join of array-like objects

Usage

```
map_join_(f, arraylike.args, eltname.mismatch.behavior = c("stop",
  "intersect"), lapply_variant = parallel::mclapply, shuffle = TRUE,
  show.progress = TRUE, cache.prefix = NULL, use.proxy = FALSE)

map_join(f, ..., eltname.mismatch.behavior = c("stop", "intersect"),
  lapply_variant = parallel::mclapply, shuffle = TRUE,
  show.progress = TRUE, cache.prefix = NULL, use.proxy = FALSE)
```

Arguments

f	the function to map
arraylike.args	a list of array-like objects with named dimnames (including "scalars" with ndimp of 0) and/or no_join objects
...	array-like objects with named dimnames, converted into an arraylike.args parameter using list(...)

Details

The function f will be called with a number of arguments equal to the number of array-like arguments, with the argument names specified in the arraylike.args list; each argument will correspond to an element of one of the arraylike.args. dimnamesnamesp will be used on each array-like object to determine what indexing colnames it would have if melted; the result will have dimnamesnamesp containing all unique dimnamesnamesp from each of the arguments. For any two dimnames elements of with the same name selected from any of the arraylike arguments, either (a) the two should be identical, or (b) at least one should be trivial (NULL or repeated ""'s). Other types of objects can be wrapped in a list of class "no_join" using no_join and included in arraylike.args; they will be treated as scalars (constants), will not affect the number or naming of dimensions; the corresponding argument fed to f is always just the object wrapped inside the no_join object. This may be necessary if the same object should be used for all calls to f but the object appears to be array-like, to prevent new dimensions from being created or expected.

map_join is provided as a potentially more convenient interface, eliminating the need to explicitly form a list of arraylike args; it converts the ... arguments into a list and delegates to map_join_

Value

an object of class "array" and mode "list" containing outputs from f, or a single output from f if all arraylike.args are scalars or no_join's

See Also

no_join

Examples

```
library("pipeR")
map_join(`*`, 2L,3L, lapply_variant=lapply)
map_join(`*`,
         with_dimnamesnames(2:3,"A"),
         with_dimnamesnames(1:3,"B")) %>>%
  {mode(.) <- "numeric"; .}
map_join(`*`,
         vector_as_named_array(2:3,"A",letters[1:2]),
         with_dimnamesnames(array(1:6,2:3), c("A","B"))) %>>%
  {mode(.) <- "numeric"; .}
cache.dir = tempfile()
map_join(`*`,
         vector_as_named_array(2:3,"A",letters[1:2]),
         with_dimnamesnames(1:3,"B"),
         cache.prefix=file.path(cache.dir,"outer_product")) %>>%
{mode(.) <- "numeric"; .}
arraylike.args = list(NULL
  , A=array(1:24,2:4) %>>%
    {dimnames(.) <- list(DA=paste0("S",1:2),DB=1:3,DC=1:4); .}
  , B=vector_as_named_array_(c(2.0,2.1), "DA", c("S1","S2"))
  , C=matrix(1:4, 2L,2L) %>>%
      {dimnames(.) <- list(DA=paste0("S",1:2),DA=paste0("S",1:2)); .}
  , D=142
  , E=1:5
  , F=vector_as_named_array_(11:14, "DC", 1:4)
  , G=c(S1=1,S2=2)
  , CP=matrix(1:4, 2L,2L) %>>%
      {dimnames(.) <- list(DA=paste0("S",1:2),DA=paste0("S",1:2)); .} %>>%
      magrittr::extract(1:2,2:1)
  , FP=vector_as_named_array_(11:15, "DC", 1:5)[c(5,3,4,2,1)]
)[-1L]
map_join_(list, arraylike.args[c("A","B","C","D","F")])[[DA="S1",DB=1L,DC=1L]]
map_join_(list, arraylike.args[c("A","B","C","D","F","CP","FP")],
          eltname.mismatch.behavior="intersect")[[DA="S1",DB=1L,DC=1L]]
```

---

match.arg.else.default

`match.arg` *variant replacing unmatched args with* `choices[[1]]`*, allowing non-character choices*

Description

Assumes this usage: parent_fun = function(parent_arg[=parent_choices]) ... match.arg.forgiving(parent_arg) ... (with or without "=parent_choices").

Usage

```
match.arg.else.default(arg, choices)
```

Arguments

arg	the argument to match to a choice; should
choices	a positive-length vector; if it contains NULL, first choice should should be NULL to avoid ambiguity

Details

If arg is NULL, returns choices[[1]].

If choices is a character vector, this performs partial matches; otherwise, it checks for arg's that are all.equal with check.attributes=FALSE.

Value

arg, the corresponding match in choices, or choices[[1]] with a warning (when arg fails to match a choice)

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
library(testthat)
## author_header begin
## Copyright (C) 2016 Logan C. Brooks
##
## This file is part of epiforecast. Algorithms included in epiforecast were developed by Logan C. Brooks, David C.
##
## Research reported in this publication was supported by the National Institute Of General Medical Sciences of the
## author_header end
## license_header begin
## epiforecast is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 2 of the License, or
## (at your option) any later version.
##
## epiforecast is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
## GNU General Public License for more details.
##
## You should have received a copy of the GNU General Public License
## along with epiforecast.  If not, see <http://www.gnu.org/licenses/>.
## license_header end

parent_function = function(ch1=letters[1:5], ch2=c("AAA","AAB","BBB"),
```

```
                          num1=1:5, int1=6L:10L,
                          list1=list(1:2,3:4), list2=list(NULL,"a",c(two=2),1:5)) {
  return (list(
    ch1=match.arg.else.default(ch1),
    ch2=match.arg.else.default(ch2),
    num1=match.arg.else.default(num1),
    int1=match.arg.else.default(int1),
    list1=match.arg.else.default(list1),
    list2=match.arg.else.default(list2)
  ))
}

## Return default on missing:
expect_equal(parent_function(),
             list(ch1="a", ch2="AAA", num1=1, int1=6L, list1=1:2, list2=NULL))

## Return default on NULL:
expect_equal(parent_function(NULL, NULL, NULL, NULL, NULL, NULL),
             list(ch1="a", ch2="AAA", num1=1, int1=6L, list1=1:2, list2=NULL))

## Allow partial matches, all.equal ignoring attributes:
expect_equal(parent_function("b", c(extraneous.name="B"), 3L, 8.00000000001, c(p=3,q=4), 2),
             list(ch1="b", ch2="BBB", num1=3.0, int1=8L, list1=3:4, list2=c(two=2)))

## Return default with warning on mismatched inputs:
expect_equal(suppressWarnings(parent_function("q", "A", "nonnumeric", 11L, 1:4, c("A","B","C"))),
             list(ch1="a", ch2="AAA", num1=1, int1=6L, list1=1:2, list2=NULL))
expect_warning(parent_function("q"))
expect_warning(parent_function(,"A"))
expect_warning(parent_function(,,"nonnumeric"))
expect_warning(parent_function(,,,11L))
expect_warning(parent_function(,,,,1:4))
expect_warning(parent_function(,,,,,c("A","B","C")))

## Produce error on inappropriate inputs:
expect_error(parent_function(3), "length-1 character")
expect_error(parent_function(letters[1:2]), "length-1 character")

## todo produce error on inappropriate =choices=
```

`match.dat` *Match dat object input*

Description

Returns a list of possibly-named numeric-class vectors given a list of possibly-named (is.)numeric vectors as input, or generates an error if the input seems inappropriate.

Usage

```
match.dat(dat)
```

Arguments

dat	supposed to be a list of possibly-named numeric vectors

Value

dat as a list of numeric-class vectors

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

match.integer *Match integer-valued input*

Description

Returns a possibly-named integer-class vector version of the input, or produces an error if the input seems inappropriate.

Usage

```
match.integer(inp)
```

Arguments

inp	supposed to be a possibly-named numeric object with integer/NA values.

Value

inp as an integer vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

`match.new.dat.sim` *Convert trajectory, trajectory matrix, or sim to sim; error otherwise*

Description

*.sim methods should eventually all support sim objects as input for new.dat rather than just single trajectories, but, for the convenience of the user, allow other types of input as well. Specifically, we should accept:

- Trajectory: a numeric vector;
- Trajectory matrix: a numeric matrix with each column a trajectory; and
- Sim: a list with `$ys` a #times by #trajectories numeric matrix with #trajectories >= 1 and each row either all `NA` or all non-`NA`, and `$weights` a #trajectories-length numeric matrix with entries all >= 0.

Usage

```
match.new.dat.sim(new.dat.sim)
```

Arguments

`new.dat.sim`	trajectory / trajectory matrix / sim object

Details

This method checks that it receives such an input and outputs a corresponding sim object.

Value

sim object

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
match.new.dat.sim(1:5)
match.new.dat.sim(as.matrix(1:5)[,rep(1,10)])
match.new.dat.sim(list(ys=as.matrix(1:5)[,rep(1,10)], weights=0:9))
```

`match.nonnegative.numeric`
Match all non-negative — within `eps` — `is.numeric` *input*

Description

Returns the input with any nearly-non-negative entries replaced with 0 if the original input `is.numeric` and all of its entries are within `eps` of being non-negative. Otherwise raises an error.

Usage

```
match.nonnegative.numeric(x, eps = sqrt(.Machine[["double.eps"]]))
```

Arguments

`x`	object that `is.numeric`; object to check
`eps`	single non-NA non-negative `is.numeric`; threshold of tolerance for negative values

Value

`x` with negative (nearly non-negative) entries replaced with 0 if `x` meets the nearly non-negative criterion; otherwise an error is raised

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

`match.single.na.or.numeric`
Match length-1 numeric input

Description

Returns a possibly-named length-1 possibly-NA numeric vector version of the input, or produces an error if the input seems inappropriate.

Usage

```
match.single.na.or.numeric(x)
```

Arguments

`x`	supposed to be a possibly-named length-1 numeric object

Value

x as a length-1 numeric vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

match.single.nonna.integer

Match length-1 non-NA integer-valued input

Description

Returns a possibly-named length-1 non-NA integer-class vector version of the input, or produces an error if the input seems inappropriate.

Usage

```
match.single.nonna.integer(n)
```

Arguments

n	supposed to be a possibly-named length-1 non-NA integer-valued is.numeric object

Value

n as a length-1 integer-class vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

match.single.nonna.integer.or.null

Match length-1 non-NA integer-valued input or NULL

Description

Returns a possibly-named length-1 non-NA integer-class vector version of the input if non-NULL, NULL if input is NULL, or produces an error if the input seems inappropriate.

Usage

```
match.single.nonna.integer.or.null(n)
```

Arguments

n	supposed to be a possibly-named length-1 non-NA integer-valued numeric object or NULL

Value

n as a length-1 integer-class vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

---

match.single.nonna.numeric

Match length-1 non-NA numeric-valued input

Description

Returns a possibly-named length-1 non-NA numeric-class vector version of the input, or produces an error if the input seems inappropriate.

Returns a possibly-named length-1 non-NA numeric-class vector version of the input, or produces an error if the input seems inappropriate.

Usage

```
match.single.nonna.numeric(x)
```

```
match.single.nonna.numeric(x)
```

Arguments

x	supposed to be a possibly-named length-1 non-NA numeric-valued numeric object
x	supposed to be a possibly-named length-1 non-NA numeric-valued numeric object

Value

x as a length-1 numeric-class vector

x as a length-1 numeric-class vector

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

`match.single.wday.w` *Returns a possibly-named length-1 integer-class vector of weekday numbers in 0:6 (%w format) given an integer-valued length-1 numeric vector with values in 0:7 (%w or %u format), or throws an error if the input seems inappropriate.*

Description

Returns a possibly-named length-1 integer-class vector of weekday numbers in 0:6 (%w format) given an integer-valued length-1 numeric vector with values in 0:7 (%w or %u format), or throws an error if the input seems inappropriate.

Usage

```
match.single.wday.w(wday)
```

Arguments

`wday`	vector of weekday numbers (each `%in% 0:7`)

Value

integer-class %w format weekday numbers

`match.wday.w` *Returns a possibly-named integer-class vector of weekday numbers in 0:6 (%w format) given an integer-valued numeric vector with values in 0:7 (%w or %u format), or throws an error if the input seems inappropriate.*

Description

Returns a possibly-named integer-class vector of weekday numbers in 0:6 (%w format) given an integer-valued numeric vector with values in 0:7 (%w or %u format), or throws an error if the input seems inappropriate.

Usage

```
match.wday.w(wday)
```

Arguments

`wday`	vector of weekday numbers (each `%in% 0:7`)

Value

integer-class %w format weekday numbers

mimicPastDF *Mimic a past version of a dataset using (partial) history data*

Description

Given some potentially partial data about past issues, given by labeling and rbinding together (available sections of available) past issues, mimic what a particular past issue would have looked like. (NA issues are treated as larger than any non-NA issue.)

Usage

```
mimicPastDF(history.df, issue.colname, mimicked.issue,
  time.index.colnames = character(0), time.index.limits = list(),
  nontime.index.colnames = character(0))
```

Arguments

`history.df`	rbound partial past issues
`issue.colname`	name of the column containing issue(label)s, which should have some ordering
`mimicked.issue`	length-1; (label of) issue to mimic
`time.index.colnames`	character vector of colnames containing time indices for observations (should not include issue.colname); used for both filtering and grouping some data
`time.index.limits`	list with length matching that of `time.index.colnames`; data with time indices above the corresponding limits is filtered out this `time.index.limit`, using lexicographical ordering when multiple time indices are specified
`additional.group.colnames`	character vector; names of columns containing any non-time-related observation indices (e.g., locations)

mimicPastEpidataDF *Mimic what* `fetchEpidataDF` *would have given in the past*

Description

Attempt to reproduce what fetchEpidataDF would have produced given data through issue `forecast.epiweek` (i.e., data through the time when the initial report for `forecast.epiweek` would have been available). Historical data is not always complete. Try the following in order for each observation week: (1) look for data for the report corresponding to the forecast week, (2) look for data in earlier reports (taking the latest report with data), (3) look for data in later reports (taking the earliest report with data). For example, the ILINet (`source="fluview"`) US National (`area="nat"`) historical data skips issue 200352; when `forecast.epiweek` is chosen as 200352, observations for `epiweek` 200340 through 200351 are provided by the previous issue, 200351, while the observation

for epiweek 200352 uses data from the *next* issue, 200353. (Observations for earlier seasons also use "future" data from issue 201352, because issue 201352 is special, filling in data for epiweeks missing from all available past issues.) Isolated skipped weeks are uncommon; usually when backfill data is missing, it is for earlier seasons, during the off-season, or at the beginning of a season; for example, in HHS Region 1 (area="hhs1"), the first recorded issue was 200949, so mimicing any reports from the 2005/2006 season will use finalized data from issue 201352 instead, and mimicing report 200940 will fill in 200940 with data from issue 200949. Similarly, data is missing from the following off-season through issue 201040, inclusive, so mimicing report 201040 will fill in off-season data with finalized data from issue 201352, and the observation for 201040 with data from issue 201041.

Usage

```
mimicPastEpidataDF(epidata.history.df, forecast.epiweek)
```

Arguments

epidata.history.df
: output of fetchEpidataHistoryDF; the lags argument fed to these functions should include all lags that could possibly contain revisions.

forecast.epiweek
: length-1 integer: issue number to simulate the fetchEpidataDF for; epiweek format is YYYYww

Examples

```
## set up a cache directory:
epidata.cache.dir = "~/.epiforecast-cache"
if (!dir.exists(epidata.cache.dir)) {
  dir.create(epidata.cache.dir)
}
## fetch HHS Region 1 ILINet version history, assuming no significant
## revisions past lag 51:
fluview.area = "hhs1"
epidata.history.df = fetchEpidataHistoryDF(
  "fluview", fluview.area, 0:51,
  first.week.of.season = 31L,
  cache.file.prefix=file.path(epidata.cache.dir,paste0("fluview_",fluview.area))
)
## Simulate fetchEpidataHistoryDF when issue 201540 (the first ILINet report
## containing data for epiweek 201540) just came out:
mimicPastEpidataDF(epidata.history.df, 201540L)
```

model_week_to_epi_week *Convert model week numbers into epi week numbers*

Description

Convert model week numbers into epi week numbers

Usage

```
model_week_to_epi_week(model.week, first.week.of.season, n.weeks.in.season)
```

Arguments

`model.week`	integer or numeric vector; model weeks — epi weeks that don't (necessarily) wrap around to 1 at the start of a new year (after week 52 or 53 of the previous year) but rather wrap around when a season ends, starting back at `first.week.of.season`; values could be integers between `first.week.of.season` to `first.week.of.season+52L` or `first.week.of.season+53L` for integers, or a valid integer value plus a number in the clopen interval [0,1) for numeric vectors
`first.week.of.season`	epi week number of the first element in a trajectory (i.e., corresponding to time `1L`)
`n.weeks.in.season`	`52L` or `53L` — the length of the indexed trajectory, corresponding to the number of weeks in the first year contained in the season

Value

integer or numeric vector with same length as `model.week`; epi week numbers (wrapper versions of model week numbers in 1..52 + [0,1) or 1..53 + [0,1)) corresponding to the given model week numbers

model_week_to_time *Convert model weeks (shifted indices) into times (indices)*

Description

Convert model weeks (shifted indices) into times (indices)

Usage

```
model_week_to_time(model.week, first.week.of.season)
```

Arguments

model.week	integer or numeric vector; model weeks — times shifted forward by first.week.of.season-1L
first.week.of.season	epi week number of the first element in a trajectory (i.e., corresponding to time 1L)

Value

integer or numeric vector with same length as model.week; times — integer indices into trajectories containing weekly data, or numeric numbers that could also refer to times between these indices

---

no_join *Mark an array-like or other argument to be used like a constant/scalar in map_join*

Description

Mark an array-like or other argument to be used like a constant/scalar in map_join

Usage

```
no_join(x)
```

Arguments

x	the argument to mark

Value

an object of class "map_join" wrapping x

Examples

```
map_join(`+`, with_dimnamesnames(1:3,"A"), with_dimnamesnames(1:3,"A"))
map_join(`+`, with_dimnamesnames(1:3,"A"), no_join(with_dimnamesnames(1:3,"A")))
```

ons — *Calculate the onset of a trajectory (post-rounding)*

Description

Calculate the onset of a trajectory (post-rounding)

Usage

```
ons(trajectory, baseline, is.inseason, ...)
```

Arguments

`trajectory`	a vector
`baseline`	the onset threshold
`is.inseason`	logical vector length-compatible with `trajectory`, acting as a mask on possible output values
`...`	ignored

Value

the first index which is part of the in-season and is part of a run of consecutive observations above the onset threshold that lasts at least for two additional indices

pht — *Calculate the peak height of a trajectory*

Description

Calculate the peak height of a trajectory

Usage

```
pht(trajectory, is.inseason = rep_len(TRUE, length(trajectory)), ...)
```

Arguments

`trajectory`	a vector
`is.inseason`	logical vector length-compatible with `trajectory`,
`...`	ignored

Value

a scalar — the maximum value of the vector

plot.sim *Plotting function for "sim" class*

Description

Plotting function for "sim" class

Usage

```
## S3 method for class 'sim'
plot(mysim, ylab = "Disease Intensity", xlab = "Time",
  lty = 1, nplot = min(100, ncol(mysim$ys)), type = c("lineplot",
  "density", "hexagonal"), overlay = FALSE, ...)
```

plot.target_forecast `plot` *method for* `target_forecast` *objects*

Description

`plot` method for `target_forecast` objects

Usage

```
## S3 method for class 'target_forecast'
plot(x, add = FALSE, ...)
```

Arguments

x	`target_forecast` object
add	logical, length 1, non-NA: whether to plot on top of the currently active plot (vs. creating a new plot)

print.sim *Printing function for "sim" class*

Description

Printing function for "sim" class

Usage

```
## S3 method for class 'sim'
print(x, verbose = TRUE, ...)
```

Arguments

x	Output from running an OO.sim() function.

print.target_forecast *Print a* forecast *object*

Description

Print a forecast object

Usage

```
## S3 method for class 'target_forecast'
print(x, sig.digit = 2L, ...)
```

Arguments

x	output of a forecast method

pwk *Calculate the peak week(s) in a vector of weekly observations*

Description

Calculate the peak week(s) in a vector of weekly observations

Usage

```
pwk(trajectory, is.inseason = rep_len(TRUE, length(trajectory)), ...)
```

Arguments

trajectory	a vector of weekly observations
is.inseason	logical vector length-compatible with trajectory,
...	ignored

Value

integer vector, typically of length 1; indices of elements that are >= all other elements

read.from.file *Function that reads from a csv file*

Description

Function that reads from a csv file

Usage

```
read.from.file(filename)
```

Arguments

filename	name of data file with each column equal to each season, with the first (n-1) columns to be, and the n'th column with the current season.

seasonModelWeekDFToYearWeekDF *Convert season-model.week in* data.frame *to year-week*

Description

Like yearWeekDFToSeasonModelWeekDF, but in opposite direction.

Usage

```
seasonModelWeekDFToYearWeekDF(seasonModelWeek, first.week, owning.wday)
```

Arguments

seasonModelWeek	data.frame (or other list) with columns $season: integer-valued vector: season numbers $model.week: integer-valued vector: model week numbers
first.week	integer-valued vector: week number that the seasons start on
owning.wday	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

data.frame with two columns giving the corresponding year-week breakdown:

$year: integer-class vector: years

$week: integer-class vector: week numbers

Examples

```
dates = as.Date("2015-01-01")+seq.int(0L, 1000L, 7L)
ywwd = DateToYearWeekWdayDF(dates, 0L, 3L) # epi week convention
dates.duplicate1 = yearWeekWdayDFToDate(ywwd, 0L, 3L)
identical(dates, dates.duplicate1)
smw = yearWeekDFToSeasonModelWeekDF(ywwd, 21L, 3L) # seasons starting on week number 21
yw = seasonModelWeekDFToYearWeekDF(smw, 21L, 3L)
identical(ywwd[,c("year","week")], yw)
dates.duplicate2 = seasonModelWeekWdayDFToDate(cbind(smw, wday=ywwd$wday), 21L, 0L, 3L)
identical(dates, dates.duplicate2)
```

`seasonModelWeekToYearWeekDF` *Convert season-model.week to year-week*

Description

Like `yearWeekToSeasonModelWeekDF`, but in opposite direction.

Usage

```
seasonModelWeekToYearWeekDF(season, model.week, first.week, owning.wday)
```

Arguments

`season`	integer-valued vector: season numbers
`model.week`	integer-valued vector: model week numbers
`first.week`	integer-valued vector: week number that the seasons start on
`owning.wday`	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

`data.frame` with two columns giving the corresponding year-week breakdown:

`$year`: integer-class vector: years

`$week`: integer-class vector: week numbers

seasonModelWeekWdayDFToDate
Convert season-model.week-wday data.frame *to* Date

Description

Convert season-model.week-wday data.frame to Date

Usage

```
seasonModelWeekWdayDFToDate(seasonModelWeekWday, first.week, first.wday,
  owning.wday, error.on.wrap = TRUE)
```

Arguments

seasonModelWeekWday	data.frame with $season, model.week, and wday columns
first.week	first week number of season
first.wday	first weekday number of week
owning.wday	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned
error.on.wrap	TRUE or FALSE: if TRUE, an error is generated if a given week falls outside the range of possible week numbers for the corresponding year (i.e., if it is nonpositive or greater than the number of weeks assigned to the corresponding year in year); if FALSE, the week number will wrap around to future or previous years (e.g., week 0 of 1997 will be considered the last week of 1996)

Value

Date vector with the corresponding dates

seasonModelWeekWdayToDate
Convert season-model.week-wday to Date

Description

Convert season-model.week-wday to Date

Usage

```
seasonModelWeekWdayToDate(season, model.week, wday, first.week, first.wday,
  owning.wday, error.on.wrap = TRUE)
```

Arguments

season	season number
model.week	model week number
wday	weekday number
first.week	first week number of season
first.wday	first weekday number of week
owning.wday	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned
error.on.wrap	TRUE or FALSE: if TRUE, an error is generated if a given week falls outside the range of possible week numbers for the corresponding year (i.e., if it is nonpositive or greater than the number of weeks assigned to the corresponding year in year); if FALSE, the week number will wrap around to future or previous years (e.g., week 0 of 1997 will be considered the last week of 1996)

Value

Date vector with the corresponding dates

---

seasonOfDate *Get the season number associated with a particular date*

Description

Seasons are 52/53-week-long time spans that start with a particular week number (from 1 to 52); years are a special case that start with week 1. Seasons contain weeks from two consecutive years except in the case that the season starts with week 1, in which case it coincides exactly with a single year. Seasons are numbered by the first distinct year from which they take weeks, and labeled with an "S" prefix; for example, a season containing weeks from 2015 and 2016 would be numbered 2015 and labeled "S2015". This function gives the season number associated with the dates.

Usage

```
seasonOfDate(date, first.week, first.wday, owning.wday)
```

Arguments

date	object convertible to Date
first.week	the week on which each season should start
first.wday	weekday number(s) (0–7, Sunday can be 0 or 7): the weekday that is considered the beginning of the week; typically Sunday or Monday
owning.wday	weekday number(s) (0–7, Sunday can be 0 or 7): a week is assigned to a given year if the owning weekday of that week falls in that year; typically first.wday or first.wday+3

Value

named integer-valued vector: season numbers associated with the inputted weeks with names giving the season labels

---

seasonOfYearWeek *Get the season number associated with a particular year and week*

Description

Seasons are 52/53-week-long time spans that start with a particular week number (from 1 to 52); years are a special case that start with week 1. Seasons contain weeks from two consecutive years except in the case that the season starts with week 1, in which case it coincides exactly with a single year. Each season ends the week before the next season (with the same starting week) begins. Seasons are numbered by the first distinct year from which they take weeks, and labeled with an "S" prefix; for example, a season containing weeks from 2015 and 2016 would be numbered 2015 and labeled "S2015". This function gives the season number associated with the inputted weeks specified as year-week combinations.

Usage

```
seasonOfYearWeek(year, week, first.week)
```

Arguments

year	integer-valued vector: year in which the weeks fall
week	integer-valued vector: associated week numbers (each `%in% 1:53`)
first.week	the week on which each season should start

Value

named integer-valued vector: season numbers associated with the inputted weeks with names giving the season labels

---

Seq *A* `seq` *variant that produces a 0-length vector when* `!(from <= to)`.

Description

A `seq` variant that produces a 0-length vector when `!(from <= to)`.

Usage

```
Seq(from, to, ...)
```

Arguments

from	starting number (or other object compatible with seq)
to	ending number (or other object compatible with seq)
...	arguments to forward to seq

---

show.sample.trajectories *Showing sample trajectories of ys.*

Description

Showing sample trajectories of ys.

Usage

```
show.sample.trajectories(ys, n.shown.rows = 100L, n.shown.cols = 100L)
```

Arguments

ys	A matrix whose columns contain simulated curves.
n.shown.rows	How many observations per curve to show.
n.shown.cols	How many curves to show.

---

table.to.list *Function to change full.dat from table to list.*

Description

Function to change full.dat from table to list.

Usage

```
table.to.list(full.dat)
```

target_forecast *Forecast distribution of targets such as peak height from model fits*

Description

This S3 method generates forecasts of various targets (e.g., peak height of a trajectory) derived from some type of model fit. For example, the fit model could be a collection of simulations of the trajectory, and this method will calculate the desired target for each simulation and aggregate these results into a distributional forecast. Refer to the appropriate S3 implementation for a given model fit for additional details.

Usage

```
target_forecast(fit.model, ...)
```

Arguments

fit.model	the fit model (trajectory forecast, simulations, regression fit, etc.), on which to base the target forecasts

See Also

target_forecast.sim

target_forecast.sim *Forecast a target (peak height, etc.) using a* sim *object*

Description

Forecast a target (peak height, etc.) using a sim object

Usage

```
## S3 method for class 'sim'
target_forecast(mysim, target = c("pwk", "pht", "ons",
  "dur"), target.name = target, target.fun = target,
  target_trajectory_preprocessor = function(trajectory) trajectory,
  target.spec = NULL, target_value_formatter = identity,
  target.multival.behavior = c("random.val", "closest.to.pred.val"),
  compute.estimates = TRUE, hist.bins = NULL, ...)
```

Arguments

mysim	Output from running an OO.sim() function.

time_to_model_week *Convert times (indices) into model weeks (shifted indices)*

Description

Convert times (indices) into model weeks (shifted indices)

Usage

```
time_to_model_week(time, first.week.of.season)
```

Arguments

time	integer or numeric vector; integer indices into trajectories containing weekly data, or numeric numbers that could also refer to times between these indices
first.week.of.season	epi week number of the first element in a trajectory (i.e., corresponding to time 1L)

Value

integer or numeric vector with same length as time; model weeks — times shifted forward by first.week.of.season-1L — corresponding to time

trimPartialPastSeasons *Trims incomplete past seasons from a* data.frame

Description

Removes rows from df corresponding to "past" seasons (i.e., all but the last season in df) for which df has less than min.points.in.season non-missing entries in df[[signal.ind]].

Usage

```
trimPartialPastSeasons(df, signal.ind, min.points.in.season)
```

Arguments

df	data frame with columns df$season, and df[[signal.ind]]
signal.ind	single non-NA character/integer-valued index for column of df
min.points.in.season	the minimum number of non-NA values for signal.ind that a season must have in order to be retained; all rows corresponding to seasons containing less observations will be removed from df

`twkde.markovian.sim` *Time-parameterized kernel density estimation sim method, Markovian version*

Description

Function for making forecasts with the basic time-parameterized kernel density estimation method. This method estimates `diff(new.dat)[t-1]` (used to produce `dat[t]`) for a trajectory `new.dat` based on weighted kernel density estimation using values of `dat` at the corresponding time of season (a Markov process). The weights are based on `new.dat[t-1]` and the corresponding values in `dat`; the weighting function is a Gaussian kernel with width determined by `bw.SJnrd0`.

Usage

```
twkde.markovian.sim(full.dat, baseline = NA_real_, max.n.sims = 1000L)
```

Arguments

`max.n.sims`	single non-`NA` integer value or `NULL`: the number of curves to sample from the inferred distribution

Value

a sim object — a list with two components:

`ys`: a numeric matrix, typically with multiple columns; each column is a different possible trajectory for the current season, with NA's in the input for the current season filled in with random draws from the forecasted distribution, and non-`NA`'s (observed data) filled in with an imagined resampling of noise based on the model (for some models, the non-`NA` values will remain unchanged).

`weights`: a numeric vector; assigns a weight to each column of `ys`, which is used by methods relying on importance sampling.

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file.prefix="fluview_nat_allfetch"),
           "wili", min.points.in.season=52L)
## Recent historical seasons + current season, minus 2009 (nonseasonal
## pandemic) season:
full.dat = split(fluview.nat.recent.df$wili, fluview.nat.recent.df$season)
names(full.dat) <- sprintf("S%s", names(full.dat))
full.dat <- full.dat[names(full.dat)!="S2009"]
## Recent historical seasons minus 2009:
```

```
dat = head(full.dat, -1L)
## Current season:
new.dat = tail(full.dat, 1L)[[1]]
## Sample from conditional curve distribution estimate using CDC's 2015
## national %wILI onset threshold baseline of 2.1:
sim = twkde.markovian.sim(dat, new.dat, 2.1, max.n.sims=100)
```

`twkde.sim` *Time-parameterized kernel density estimation sim method with heuristic adjustments*

Description

Function for making forecasts with the time-parameterized kernel density estimation method with tweaks. This method estimates `diff(new.dat)` based on weighted kernel density estimation. Weights are based on the time of season and four functions of `new.dat`: (a) the last observed value in `new.dat`, (b) the sum of observed values in `new.dat`, (c) an exponential moving average of the observed values in `new.dat`, and (d) an exponential moving average of the changes in observed values in `new.dat` (i.e., in `diff(new.dat)`). The weighting function is separable, and consists of two components: a highly weighted "base" weighting function and a lowly weighted boxcar weighting function. The base weighting function is the product of an integral Laplacian kernel with respect to time of season, and Gaussian kernels with respect to the four `new.dat`-based components (with bandwidths selected by the `bw.SJnrd0` method, and relative weighting controlled by `tradeoff.weights`). Each time a difference is drawn, simulating `diff(new.dat)[t-1]`, the corresponding result for `new.dat[t]` is linearly mixed with a randomly selected value from historical curves around that time.

Usage

```
twkde.sim(full.dat, baseline = NA_real_, max.n.sims = 1000L,
  decay.factor = 0.7, diff.decay.factor = 0.5,
  max.shifts = c(rep(10L, 10L), 10:1, rep(0L, 3L), 1:10, rep(10L, 20L)),
  shift.decay.factor = 0.7, tradeoff.weights = c(0.5, 0.25, 0.25, 0.5))
```

Arguments

`max.n.sims`	single non-`NA` integer value or `NULL`: the number of curves to sample from the inferred distribution
`decay.factor`	decay factor for the exponential moving average of covariate.
`diff.decay.factor`	decay factor for the exponential moving average of differences covariate.
`max.shifts`	numeric vector with length matching the trajectory length in `new.dat.sim`; specifies the width of the time-of-season kernel as a function of the time of season.
`shift.decay.factor`	decay factor for the time-of-season Laplacian kernel component.

`tradeoff.weights`
: log-scale weighting factors for the four non-time-based kernel components (last observed value, sum of observed values, exponential moving average of values, exponential moving average of differences).

Value

a sim object — a list with two components:

`ys`: a numeric matrix, typically with multiple columns; each column is a different possible trajectory for the current season, with NA's in the input for the current season filled in with random draws from the forecasted distribution, and non-`NA`'s (observed data) filled in with an imagined resampling of noise based on the model (for some models, the non-`NA` values will remain unchanged).

`weights`: a numeric vector; assigns a weight to each column of `ys`, which is used by methods relying on importance sampling.

Author(s)

Logan C. Brooks, David C. Farrow, Sangwon Hyun, Ryan J. Tibshirani, Roni Rosenfeld

Examples

```
fluview.nat.recent.df =
   trimPartialPastSeasons(fetchEpidataDF("fluview", "nat",
                          first.week.of.season=21L,
                          cache.file.prefix="fluview_nat_allfetch"),
           "wili", min.points.in.season=52L)
## Recent historical seasons + current season, minus 2009 (nonseasonal
## pandemic) season:
full.dat = split(fluview.nat.recent.df$wili, fluview.nat.recent.df$season)
names(full.dat) <- sprintf("S%s", names(full.dat))
full.dat <- full.dat[names(full.dat)!="S2009"]
## Recent historical seasons minus 2009:
dat = head(full.dat, -1L)
## Current season:
new.dat = tail(full.dat, 1L)[[1]]
## Sample from conditional curve distribution estimate using CDC's 2015
## national %wILI onset threshold baseline of 2.1:
sim = twkde.sim(dat, new.dat, 2.1, max.n.sims=50)
```

`unite_arraylike` *Reshape arraylike into fewer but larger dimensions and/or permute (and optionally rename) dimensions; like tidyr::unite on arrays using R.utils::wrap.array*

Description

Collapses specified array(like) object dimensions together to produce an array with fewer but larger dimensions and an equal number of entries; indices of each collapsed dimension refers to a Cartesian product over the indices of the dimensions it was made from. This operation should be similar or equivalent to melting the array, calling tidyr::unite, and casting back into an array.

Usage

```
unite_arraylike(arraylike, dnn.sets, sep = ".")
```

Arguments

arraylike	the arraylike object to reshape; must have the dimnames names referred to in {dnn.sets}
dnn.sets	a list of character vectors; each character vector specifies some dimensions to unite in the array; ordering within each vector determines the ordering of indices in the resulting collapsed dimension (as in R.utils::wrap.array); ordering between the vectors determines the ordering of the collapsed dimensions in the result; names of vector entries (e.g., names(dnn.sets[[1]])) are ignored; names(dnn.sets), if specified and nonblank, override automatic names of the collapsed dimensions
sep	length-1 character vector; string used to combine dimnames for collapsed dimensions, as well as automatic dimnames names for collapsed dimensions

Details

Collapsed/united dimensions are ordered after all untouched dimensions. To tweak the ordering or use this function to simply permute dimensions, individual dimnames names can be included as their own singleton "sets" in dnn.sets.

Examples

```
arraylike = array(1:2^5, rep(2,5))
dimnames(arraylike) <- list(A=c("a1","a2"),B=c("b1","b2"),C=c("c1","c2"),D=c("d1","d2"),E=c("e1","e2"))
## Collapsing A&B, C&D:
dimnames(unite_arraylike(arraylike, list(c("A","B"), c("C","D"))))
## Adjusting `sep` changes resulting dimnames and dimnames names:
dimnames(unite_arraylike(arraylike, list(c("A","B"), c("C","D")), sep="__"))
## Result dimnames names can be manually specified:
names(dimnames(unite_arraylike(arraylike, list(c("A","B"), DVD=c("C","D")), sep="__")))
## Singleton sets can be used to permute dimensions, optionally renaming them (changing the dimnames names):
##  Place dimension A at end:
names(dimnames(unite_arraylike(arraylike, list("A"))))
##  Permute all dims:
names(dimnames(unite_arraylike(arraylike, list("E","D","C","B","A"))))
##  Place some dimensions at end and rename one:
names(dimnames(unite_arraylike(arraylike, list("A","EEE"="E"))))
## Collapsing and permuting are actually the same operation and can be mixed:
names(dimnames(unite_arraylike(arraylike, list(c("C","D"),"A","EEE"="E"))))
```

upsample_sim *Resample a sim (if necessary) to get >= min.n.sims simulated curves*

Description

Resample a sim (if necessary) to get >= min.n.sims simulated curves

Usage

```
upsample_sim(sim.obj, min.n.sims, inflate.weights)
```

Arguments

sim.obj	a sim object
inflate.weights	a single non-NA logical; TRUE indicated that the weights be inflated proportionally with the increase in the number of simulations; FALSE indicates that the total weight should be preserved instead.
max.n.sims	a single non-NA non-negative integer; the inclusive lower bound on the number of simulated curves in the result

Details

Resampling is only performed if the number of simulations needs to change. Any resampling is (necessarily) done with replacement.

Value

a sim object with >= min.n.sims simulated curves; the sum of weights in the result will equal the sum of the results in the input.

usa.flu.first.week.of.season *First epi week in weekly USA flu-related trajectories used by some methods*

Description

First epi week in weekly USA flu-related trajectories used by some methods

Usage

```
usa.flu.first.week.of.season
```

Format

An object of class integer of length 1.

usa_flu_inseason_flags *Get* logical *marking in-season times in a USA flu-related trajectory*

Description

Given that a trajectory starts with the epi week usa.flu.first.week.of.season and is n.weeks.in.season long, marks parts of the trajectory that are part of the USA flu "in-season" (epi week 40 of the first year of a season to epi week 20 of the following year)

Usage

```
usa_flu_inseason_flags(n.weeks.in.season)
```

Arguments

n.weeks.in.season	52L or 53L; the number of weeks in the first year of a season

Value

logical vector of length equal to n.weeks.in.season; entries are TRUE at times corresponding to weeks that are part of the in-season, and FALSE otherwise (i.e., in the off-season)

weekConventions first.wday *and* owning.wday *for some week numbering conventions*

Description

Covers four common week numbering conventions:

- "epi": Epidemiological weeks or "epi weeks": weeks begin on Sunday, and are assigned to years based on what year the majority of days fall in (i.e., what year Wednesday falls in)
- "iso": ISO 8601 weeks: weeks begin on Monday, and are assigned to years based on what year the majority of days fall in (i.e., what year Thursday falls in)
- "usa": USA convention: weeks begin on Sunday, and are assigned to years based on what year Sunday falls in
- "uk": UK convention: weeks begin on Monday, and are assigned to years based on what year Monday falls in

Usage

```
weekConventions
```

Format

An object of class `matrix` with 2 rows and 4 columns.

Details

There are two rows, named `"first.wday"` and `"owning.wday"`. There are four columns, corresponding to the four conventions above.

---

weighted_tabulate *Weighted, more* nbins*-restrictive version of* base::tabulate

---

Description

Weighted, more nbins-restrictive version of `base::tabulate`

Usage

```
weighted_tabulate(bin, nbins, w)
```

Arguments

bin	integer-compatible vector; entries must be non-NA and between 1 and nbins; these indices denote entries in the result vector to which the corresponding weights in w should be added
nbins	single non-NA, non-negative integer; length of the vector to return
w	numeric-compatible vector of the same length as bin; weights corresponding to the indices in bin

Value

numeric vector of length nbins; the ith entry is like `sum(w[bin==i])`, but with a naive summation algorithm

---

yearWeekDFToSeasonModelWeekDF

Convert year-week in a data.frame *to season-model.week*

Description

Delegates to `yearWeekToSeasonModelWeekDF`.

Usage

```
yearWeekDFToSeasonModelWeekDF(yearWeek, first.week, owning.wday)
```

Arguments

yearWeek	data.frame (or other list) with columns $year and $week: $year integer-valued vector: year from the year-week numbering $week integer-valued vector: week from the year-week numbering
first.week	integer-valued vector: week number that the seasons start on
owning.wday	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

data.frame with two columns giving the corresponding season-model.week breakdown:

$season: integer-class vector: season numbers

$model.week: integer-class vector: model week numbers

---

yearWeekToSeasonModelWeekDF

Convert year-week to season-model.week

Description

Seasons starting with a week number n other than 1 contain week numbers n to 52/53 from the season's first year and 1 to n-1 of the season's second year. Sometimes we would like a numbering of weeks within a season that coincides with the week number when possible, but does not have a jump down from 52/53 to 1. Model weeks fulfill this purpose: they begin with the starting week of a season and increase by 1 for each subsequent week; they coincide with week numbers in the starting year, and are 52/53 plus the week number in the second year.

Usage

```
yearWeekToSeasonModelWeekDF(year, week, first.week, owning.wday)
```

Arguments

year	integer-valued vector: year from the year-week numbering
week	integer-valued vector: week from the year-week numbering
first.week	integer-valued vector: week number that the seasons start on
owning.wday	integer-valued vector of weekday numbers: weekday that determines the year to which a week will be assigned

Value

a data frame (tbl_df) with two columns giving the corresponding season-model.week breakdown:

$season: integer-class vector: season numbers

$model.week: integer-class vector: model week numbers

`yearWeekWdayDFToDate` *Convert a structure containing year, week, and weekday numbers into the corresponding dates under a specified week numbering convention. Reverses* `DateToYearWeekWdayDF`.

Description

Convert a structure containing year, week, and weekday numbers into the corresponding dates under a specified week numbering convention. Reverses `DateToYearWeekWdayDF`.

Usage

```
yearWeekWdayDFToDate(ywwd, first.wday, owning.wday, error.on.wrap = TRUE)
```

Arguments

`ywwd`	data frame with columns `$year`, `$week`, and `$wday` containing the year, week, and weekday numbers respectively (weekday numbers are 0–7; Sunday can be inputted either as 0 or as 7)
`first.wday`	wday number corresponding to the first weekday of any week
`owning.wday`	wday number; a week is assigned to a given year if the owning weekday of that week falls in the given year
`error.on.wrap`	TRUE or FALSE: if TRUE, an error is generated if a given week falls outside the range of possible week numbers for the corresponding year (i.e., if it is nonpositive or greater than the number of weeks assigned to the corresponding year in `year`); if FALSE, the week number will wrap around to future or previous years (e.g., week 0 of 1997 will be considered the last week of 1996)

Value

a `Date` vector: the dates corresponding to the year-week-wdays and week numbering convention

---

`yearWeekWdayListsToDate` *Like* `yearWeekWdayDFToDate`*, but allows parameters to be provided in several (is.)list structures and vectors.*

Description

Like `yearWeekWdayDFToDate`, but allows parameters to be provided in several (is.)list structures and vectors.

Usage

```
yearWeekWdayListsToDate(...)
```

Arguments

...	vectors and (is.)lists to feed to yearWeekWdayVecsToDate; each entry in a list (e.g., column in a data.frame) is treated as a vector. Names can be provided by the names attribute of list arguments or optional parameter names for the vector arguments. (With no names, parameters of yearWeekWdayVecsToDate are matched sequentially.): year integer-valued vector: the year numbers week integer-valued vector: the week numbers wday integer-valued vector: the wday numbers (0–7, either 0 or 7 can be used for Sunday) first.wday wday number of the first day of each week owning.wday if the owning weekday of a week falls in a particular year, the entire week is assigned to that year error.on.wrap if TRUE, throws errors when week numbers do not fall in the specified years; when FALSE, wraps them around into other years

Value

the corresponding dates

---

yearWeekWdayVecsToDate

Like yearWeekWdayDFToDate*, but with each column of* ywwd *provided as separate parameters.*

Description

Like yearWeekWdayDFToDate, but with each column of ywwd provided as separate parameters.

Usage

```
yearWeekWdayVecsToDate(year, week, wday, first.wday, owning.wday,
  error.on.wrap = TRUE)
```

Arguments

year	integer-valued vector: the year numbers
week	integer-valued vector: the week numbers
wday	integer-valued vector: the wday numbers (0–7, either 0 or 7 can be used for Sunday)
first.wday	wday number of the first day of each week
owning.wday	if the owning weekday of a week falls in a particular year, the entire week is assigned to that year
error.on.wrap	if TRUE, throws errors when week numbers do not fall in the specified years; when FALSE, wraps them around into other years

Value

the corresponding dates

Index