Package 'tf'

May 8, 2026

Title S3 Classes and Methods for Tidy Functional Data

Version 0.4.1

Description Provides S3 vector types for functional data represented on grids, in spline bases, or via functional principal components. Supports arithmetic and summary methods, plotting, derivation, integration, smoothing, registration, and data import/export for these functional vectors. Includes data-wrangling tools for re-evaluation, subsetting, sub-assignment, zooming into sub-domains, and extracting functional features such as minima, maxima, and their locations. Enables joint analysis of functional and scalar variables by integrating functional vectors into standard data frames.

License AGPL (>= 3)

URL https://tidyfun.github.io/tf/, https://github.com/tidyfun/tf/

BugReports https://github.com/tidyfun/tf/issues

Depends R (>= 4.1)

Imports checkmate, cli, methods, mgcv, mvtnorm, pracma, purrr (>= 1.0.0), rlang, stats, vctrs (>= 0.2.4), zoo

Suggests covr, dplyr, fda, fdasrvf, pillar, refund, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Collate 'approx.R' 'assertions.R' 'bibentries.R' 'brackets.R' 'calculus.R' 'convert-construct-utils.R' 'convert.R' 'data.R' 'depth.R' 'evaluate.R' 'fda-connectors.R' 'fwise.R' 'globals.R' 'graphics.R' 'interpolate.R' 'landmarks.R' 'ops.R' 'math.R' 'methods.R' 'print-format.R' 'rebase.R' 'register-cc.R' 'register-utils.R' 'register.R' 'registration-class.R' 'rng.R' 'smooth.R' 'soft-impute-svd.R' 'split-combine.R' 'summarize.R' 'tf-package.R' 'tfb-fpc.R' 'tfb-spline.R' 'tfb-class.R'

'tfd-class.R' 'tf-s4.R' 'tfb-fpc-utils.R' 'tfb-spline-utils.R'
'utils.R' 'vctrs-cast.R' 'vctrs-ptype2.R' 'where.R' 'zoom.R'
'zzz.R'

NeedsCompilation no

Author Fabian Scheipl [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-8172-3603>),
Jeff Goldsmith [aut],
Maximilian Mücke [aut] (ORCID: <https://orcid.org/0009-0000-9432-9795>),
Julia Wrobel [ctb] (ORCID: <https://orcid.org/0000-0001-6783-1421>),
Sebastian Fischer [ctb] (ORCID:
<https://orcid.org/0000-0002-9609-3197>),
Trevor Hastie [ctb] (softImpute author),
Rahul Mazumder [ctb] (softImpute author),
Chen Meng [ctb] (mogsa author)

Maintainer Fabian Scheipl <fabian.scheipl@googlemail.com>

Repository CRAN

Date/Publication 2026-04-07 13:00:02 UTC

Contents

`as.data.frame.tf` *Convert functional data back to tabular data formats*

Description

Various converters to turn `tfb`- or `tfd`-vectors into data.frames or matrices, or even an actual R function.

Usage

```
## S3 method for class 'tf'
as.data.frame(x, row.names = NULL, optional = FALSE, unnest = FALSE, ...)

## S3 method for class 'tf'
as.matrix(x, arg, interpolate = FALSE, ...)

## S3 method for class 'tf'
as.function(x, ...)
```

Arguments

`x`	a `tf` object.
`row.names`	`NULL` or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	not used.
`unnest`	if `TRUE`, the function will return a data.frame with the evaluated functions.
`...`	additional arguments to be passed to or from methods.

arg	a vector of argument values / evaluation points for x. Defaults to tf_arg(x) (so for x on irregular grids, this will be the union of all observed arg-values by default).
interpolate	should functions be evaluated (i.e., inter-/extrapolated) for values in arg for which no original data is available? Only relevant for the raw data class tfd, for which it defaults to FALSE. Basis-represented functional data tfb are always "interpolated".

Value

for as.data.frame.tf**:** if unnest is FALSE (default), a one-column data.frame with a tf-column containing x. if unnest is TRUE, a 3-column data frame with columns id (containing (unique) names of x or a numeric identifier if x is unnamed), arg, and value, with each row containing one function evaluation at the original arg-values.

for as.matrix.tf**:** a matrix with one row per function and one column per arg.

for as.function.tf**:** an R function with argument arg that evaluates x on arg and returns the list of function values

Examples

```
f <- tfd(sin(seq(0, 2 * pi, length.out = 11)), arg = seq(0, 1, length.out = 11))
as.data.frame(f)
as.data.frame(f, unnest = TRUE)
as.matrix(f)
fun <- as.function(f)
fun(c(0, 0.5, 1))
```

---

ensure_list *Turns any object into a list*

Description

See above.

Usage

```
ensure_list(x)
```

Arguments

x	any input.

Value

x turned into a list.

See Also

Other tidyfun developer tools: prep_plotting_arg(), unique_id()

Examples

```
ensure_list(1:3)
ensure_list(list(1, 2))
```

---

fivenum *Tukey's Five Number Summary for* tf *vectors*

Description

Computes a depth-based five number summary for functional data: the observations with minimum, lower-hinge, median, upper-hinge, and maximum depth values.

Usage

```
fivenum(x, na.rm = FALSE, ...)

## Default S3 method:
fivenum(x, na.rm = FALSE, ...)

## S3 method for class 'tf'
fivenum(x, na.rm = FALSE, depth = "MHI", ...)
```

Arguments

x	a tf vector (or numeric for the default method).
na.rm	logical; if TRUE, NA observations are removed first.
...	passed to tf_depth().
depth	depth method for ordering. See tf_depth(). Defaults to "MHI" for an up-down ordering.

Value

fivenum.tf: a named tf vector of length 5.
fivenum.default: see stats::fivenum().

See Also

Other tidyfun summary functions: functionwise, tfsummaries

Examples

```
set.seed(1)
f <- tf_rgp(7)
fivenum(f)
```

`fpc_wsvd` *Eigenfunctions via weighted, regularized SVD*

Description

Compute (truncated) orthonormal eigenfunctions and scores for (partially missing) data on a common (potentially non-equidistant) grid.

Usage

```
fpc_wsvd(data, arg, pve = 0.995)

## S3 method for class 'matrix'
fpc_wsvd(data, arg, pve = 0.995)

## S3 method for class 'data.frame'
fpc_wsvd(data, arg, pve = 0.995)
```

Arguments

`data`	numeric matrix of function evaluations (each row is one curve, no NAs).
`arg`	numeric vector of argument values.
`pve`	percentage of variance explained.

Details

Performs a weighted SVD with trapezoidal quadrature weights s.t. returned vectors represent (evaluations of) orthonormal eigen*functions* $\phi_j(t)$, not eigen*vectors* $\phi_j = (\phi_j(t_1), \ldots, \phi_j(t_n))$, specifically:
$\int_T \phi_j(t)^2 dt \approx \sum_i \Delta_i \phi_j(t_i)^2 = 1$ given quadrature weights Δ_i, not $\phi_j' \phi_j = \sum_i \phi_j(t_i)^2 = 1$;
$\int_T \phi_j(t)\phi_k(t)dt = 0$ not $\phi_j' \phi_k = \sum_i \phi_j(t_i)\phi_k(t_i) = 0$, c.f. `mogsa::wsvd()`.
For incomplete data, this uses an adaptation of `softImpute::softImpute()`, see references. Note that will not work well for data on a common grid if more than a few percent of data points are missing, and it breaks down completely for truly irregular data with no/few common timepoints, even if observed very densely. For such data, either re-evaluate on a common grid first or use more advanced FPCA approaches like `refund::fpca_sc()`, see last example for `tfb_fpc()`

Value

a list with entries

- `mu` estimated mean function (numeric vector)
- `efunctions` estimated FPCs (numeric matrix, columns represent FPCs)
- `scores` estimated FPC scores (one row per observed curve)
- `npc` how many FPCs were returned for the given `pve` (integer)
- `scoring_function` a function that returns FPC scores for new data and given eigenfunctions, see `tf:::.fpc_wsvd_scores` for an example.

Author(s)

Trevor Hastie, Rahul Mazumder, Chen Meng, Fabian Scheipl

References

code adapted from / inspired by `mogsa::wsvd()` by Chen Meng and `softImpute::softImpute()` by Trevor Hastie and Rahul Mazumder.
Meng C (2023). *mogsa: Multiple omics data integrative clustering and gene set analysis*. doi:10.18129/B9.bioc.mogsa, https://bioconductor.org/packages/mogsa.

Mazumder, Rahul, Hastie, Trevor, Tibshirani, Robert (2010). "Spectral Regularization Algorithms for Learning Large Incomplete Matrices." *The Journal of Machine Learning Research*, **11**, 2287–2322.

Hastie T, Mazumder R (2021). *softImpute: Matrix Completion via Iterative Soft-Thresholded SVD*. doi:10.32614/CRAN.package.softImpute, R package version 1.4-1, https://CRAN.R-project.org/package=softImpute.

See Also

Other tfb-class: `tfb`, `tfb_fpc()`, `tfb_spline()`

Other tfb_fpc-class: `tfb_fpc()`

`functionwise` *Summarize each* `tf` *in a vector (function-wise)*

Description

These functions extract (user-specified) **function-wise** summary statistics from every entry in a `tf`-vector. To summarize a vector of functions at each argument value, see `?tfsummaries`. Note that most of these will tend to yield lots of `NA`s for irregular `tfd` unless you set a `tf_evaluator()`-function that does inter- and extrapolation for them beforehand.

Usage

```
tf_fwise(x, .f, arg = tf_arg(x), ...)

tf_fmax(x, arg = tf_arg(x), na.rm = FALSE)

tf_fmin(x, arg = tf_arg(x), na.rm = FALSE)

tf_fmedian(x, arg = tf_arg(x), na.rm = FALSE)

tf_frange(x, arg = tf_arg(x), na.rm = FALSE, finite = FALSE)

tf_fmean(x, arg = tf_arg(x))

tf_fvar(x, arg = tf_arg(x))
```

```
tf_fsd(x, arg = tf_arg(x))

tf_crosscov(x, y, arg = tf_arg(x))

tf_crosscor(x, y, arg = tf_arg(x))
```

Arguments

x	a tf object.
.f	a function or formula that is applied to each entry of x, see purrr::as_mapper() and details.
arg	defaults to standard argument values of x.
...	additional arguments for purrr::as_mapper().
na.rm	a logical (TRUE or FALSE) indicating whether missing values should be removed.
finite	logical, indicating if all non-finite elements should be omitted.
y	a tf object.

Details

tf_fwise turns x into a list of data.frames with columns arg and values internally, so the function/formula in .f gets a data.frame .x with these columns, see examples below or source code for tf_fmin(), tf_fmax(), etc.

Value

a list (or vector) of the same length as x with the respective summaries.

Functions

- tf_fwise(): User-specified function-wise summary statistics
- tf_fmax(): maximal value of each function
- tf_fmin(): minimal value of each function
- tf_fmedian(): median value of each function
- tf_frange(): range of values of each function
- tf_fmean(): mean of each function: $\frac{1}{|T|}\int_T x_i(t)dt$
- tf_fvar(): variance of each function: $\frac{1}{|T|}\int_T (x_i(t) - \bar{x}(t))^2 dt$
- tf_fsd(): standard deviation of each function: $\sqrt{\frac{1}{|T|}\int_T (x_i(t) - \bar{x}(t))^2 dt}$
- tf_crosscov(): cross-covariances between two functional vectors: $\frac{1}{|T|}\int_T (x_i(t) - \bar{x}(t))(y_i(t) - \bar{y}(t))dt$
- tf_crosscor(): cross-correlation between two functional vectors: tf_crosscov(x, y) / (tf_fsd(x) * tf_fsd(y))

See Also

Other tidyfun summary functions: `fivenum()`, `tfsummaries`

Examples

```
x <- tf_rgp(3)
layout(t(1:3))
plot(x, col = 1:3)
#  each function's values to [0,1]:
x_clamp <- (x - tf_fmin(x)) / (tf_fmax(x) - tf_fmin(x))
plot(x_clamp, col = 1:3)
# standardize each function to have mean / integral 0 and sd 1:
x_std <- (x - tf_fmean(x)) / tf_fsd(x)
tf_fvar(x_std) == c(1, 1, 1)
plot(x_std, col = 1:3)
# Custom functions:
# 80%tiles of each function's values:
tf_fwise(x, \(.x) quantile(.x$value, 0.8)) |> unlist()
# minimal value of each function for t > 0.5
tf_fwise(x, \(.x) min(.x$value[.x$arg > 0.5])) |> unlist()

tf_crosscor(x, -x)
tf_crosscov(x, x) == tf_fvar(x)
```

---

gait *Hip and knee angle while walking data*

Description

Hip and knee angle measurements in degrees through a 20-point movement cycle for 39 boys. The data represents the angular positions of hip and knee joints during normal walking gait, captured at evenly spaced time points throughout the gait cycle.

Usage

```
gait
```

Format

A data frame with 39 rows and 3 variables:

subject_id subject identifier

knee_angle knee joint angles in degrees

hip_angle hip joint angle in degrees

References

Olshen, A R, Biden, N E, Wyatt, P M, Sutherland, H D (1989). "Gait Analysis and the Bootstrap." *The Annals of Statistics*, **17**(4), 1419–1440.

Data is also include in the `datasets` package in another format.

Examples

```
head(gait)
```

`growth` *Berkeley growth study data*

Description

Heights of 39 boys and 54 girls measured from age 1 to 18 years as part of the Berkeley Growth Study. The data tracks physical development over time with measurements at 31 different ages that are not equally spaced.

Usage

```
growth
```

Format

A data frame with 93 rows and 2 variables:

gender sex of the subject (boy/girl)

height height in centimeters

Details

Data is also include in the **fda** package in another format.

References

Ramsay, O. J, Hooker, Giles, Graves, Spencer (2009). *Functional Data Analysis with R and MATLAB*, series Use R!, 1 edition. Springer New York, New York. ISBN 978-0-387-98184-0, doi:10.1007/9780387981857.

Ramsay, O. J, Silverman, W. B (2005). *Functional Data Analysis*, series Springer Series in Statistics, 2nd edition. Springer, New York. ISBN 978-0-387-40080-8.

Ramsay, O. J, Silverman, W. B (2002). *Applied Functional Data Analysis*. Springer.

Tuddenham, D R (1954). "Physical growth of California boys and girls from birth to eighteen years." *University of California Publications in Child Development*, **1**, 183–364.

Examples

```
head(growth)
```

in_range *Find out if values are inside given bounds*

Description

in_range and its infix-equivalent %inr% return TRUE for all values in the numeric vector f that are within the range of values in r.

Usage

```
in_range(f, r)

f %inr% r
```

Arguments

f	a numeric vector.
r	numeric vector used to specify a range, only the minimum and maximum of r are used.

Value

a logical vector of the same length as f.

See Also

Other tidyfun utility functions: tf_arg(), tf_zoom()

Examples

```
in_range(1:10, c(3, 7))
1:10 %inr% c(3, 7)
```

pinch *Pinch force data*

Description

Measurements of pinch force during 20 replications, with 151 observations recorded every 2 milliseconds over 300 milliseconds. The data captures the dynamics of finger pinch force applied during controlled motor tasks.

Usage

```
pinch
```

Format

An object of class `tfd_reg` (inherits from `tfd`, `tf`, `vctrs_vctr`, `list`) of length 20.

Details

Data is also include in the **fda** package in another format.

References

Ramsay, O. J, Hooker, Giles, Graves, Spencer (2009). *Functional Data Analysis with R and MATLAB*, series Use R!, 1 edition. Springer New York, New York. ISBN 978-0-387-98184-0, doi:10.1007/9780387981857.

Ramsay, O. J, Silverman, W. B (2005). *Functional Data Analysis*, series Springer Series in Statistics, 2nd edition. Springer, New York. ISBN 978-0-387-40080-8.

Ramsay, O. J, Silverman, W. B (2002). *Applied Functional Data Analysis*. Springer.

Examples

```
pinch
```

---

plot.tf base *plots for* tf*s*

Description

Some base functions for displaying functional data in spaghetti- (i.e., line plots) and lasagna- (i.e., heat map) flavors.

Usage

```
## S3 method for class 'tf'
plot(
  x,
  y,
  n_grid = 50,
  points = is_irreg(x),
  type = c("spaghetti", "lasagna"),
  alpha = min(1, max(0.05, 2/length(x))),
  ...
)

## S3 method for class 'tf'
lines(x, arg, n_grid = 50, alpha = min(1, max(0.05, 2/length(x))), ...)

## S3 method for class 'tf'
points(
```

```
  x,
  arg,
  n_grid = NA,
  alpha = min(1, max(0.05, 2/length(x))),
  interpolate = FALSE,
  ...
)
```

Arguments

x	an tf object.
y	(optional) numeric vector to be used as arg (i.e., for the **x**-axis...!).
n_grid	minimal size of equidistant grid used for plotting, defaults to 50. See details.
points	should the original evaluation points be marked by points? Defaults to TRUE for irregular tfd and FALSE for all others.
type	"spaghetti": line plots, "lasagna": heat maps.
alpha	alpha-value (see grDevices::rgb()) for noodle transparency. Defaults to 2/(no. of observations). Lower is more transparent.
...	additional arguments for graphics::matplot() ("spaghetti") or image() ("lasagna").
arg	evaluation grid (vector).
interpolate	should functions be evaluated (i.e., inter-/extrapolated) for arg for which no original data is available? Only relevant for tfd, defaults to FALSE.

Details

If no second argument y is given, evaluation points (arg) for the functions are given by the union of the tf's arg and an equidistant grid over its domain with n_grid points. If you want to only see the original data for tfd-objects without inter-/extrapolation, use n_grid < 1 or n_grid = NA.

Value

the plotted tf-object, invisibly.

References

Swihart, J B, Caffo, Brian, James, D B, Strand, Matthew, Schwartz, S B, Punjabi, M N (2010). "Lasagna plots: a saucy alternative to spaghetti plots." *Epidemiology (Cambridge, Mass.)*, **21**(5), 621–625.

Examples

```
f <- tfd(sin(seq(0, 2 * pi, length.out = 51)), arg = seq(0, 1, length.out = 51))
plot(f)
plot(c(f, 2 * f), type = "lasagna")
```

prep_plotting_arg *Preprocess evaluation grid for plotting*

Description

(internal function exported for re-use in upstream packages)

Usage

```
prep_plotting_arg(f, n_grid)
```

Arguments

f	a tf-object.
n_grid	length of evaluation grid.

Value

a semi-regular grid rounded down to appropriate resolution.

See Also

Other tidyfun developer tools: ensure_list(), unique_id()

Examples

```
f <- tfd(sin(seq(0, 2 * pi, length.out = 21)), arg = seq(0, 1, length.out = 21))
prep_plotting_arg(f, n_grid = 50)
```

print.tf *Pretty printing and formatting for functional data*

Description

Prints and formats tf-objects for display. See details / examples for options that give finer control.

Usage

```
## S3 method for class 'tf'
print(x, n = 6, ...)

## S3 method for class 'tfd_reg'
print(x, n = 6, ...)

## S3 method for class 'tfd_irreg'
print(x, n = 6, ...)
```

```
## S3 method for class 'tfb'
print(x, n = 5, ...)

## S3 method for class 'tf'
format(
  x,
  digits = 2,
  nsmall = 0,
  width = options()$width,
  sparkline = TRUE,
  prefix = FALSE,
  ...
)
```

Arguments

x	any R object (conceptually); typically numeric.
n	how many elements of x to print out at most, defaults to 6.
...	handed over to format.tf().
digits	a positive integer indicating how many significant digits are to be used for numeric and complex x. The default, NULL, uses getOption("digits"). This is a suggestion: enough decimal places will be used so that the smallest (in magnitude) number has this many significant digits, and also to satisfy nsmall. (For more, notably the interpretation for complex numbers see signif.)
nsmall	the minimum number of digits to the right of the decimal point in formatting real/complex numbers in non-scientific formats. Allowed values are 0 <= nsmall <= 20.
width	default method: the *minimum* field width or NULL or 0 for no restriction. AsIs method: the *maximum* field width for non-character objects. NULL corresponds to the default 12.
sparkline	use a sparkline representation? defaults to TRUE (not available for irregular data).
prefix	prefix with names / index positions? defaults to FALSE.

Details

By default, tf objects on regular grids are shown as "sparklines" (cli::spark_bar()), set sparkline = FALSE for a text representation.

Sparklines are based on running mean values of the function values, but these don't check for non-equidistant grids, so the visual impression will be misleading for very unequal grid distances.

Sparklines use options()$width/3 bins for printing/formatting by default, use bins argument to set the number of bins explicitly. For pillar::glimpse(), we use 8 bins by default for compact display.

Value

`print`: prints out `x` and returns it invisibly.

a character representation of `x`.

Examples

```
t <- seq(0, 1, l = 201)
cosine <- lapply(1:4, \(i) cos(i * pi * t)) |> tfd(arg = t)
cosine
tf_sparsify(cosine, dropout = .8)

format(cosine, sparkline = FALSE)
format(cosine, bins = 5)
format(cosine, bins = 40)

#! very non-equidistant grids --> sparklines can mislead about actual shapes:
tfd(cosine, arg = t^3)
```

---

tfb *Constructors for functional data in basis representation*

Description

Various constructors for `tfb`-vectors from different kinds of inputs.

Usage

```
tfb(data = data_frame0(), basis = c("spline", "fpc", "wavelet"), ...)

tfb_wavelet(data, ...)

as.tfb(data, basis = c("spline", "fpc"), ...)
```

Arguments

`data`	a `matrix`, `data.frame` or `list` of suitable shape, or another `tf`-object containing functional data.
`basis`	either `"spline"` (see `tfb_spline()`, the default) or `"fpc"` (see `tfb_fpc()`). (`wavelet` not implemented yet)
`...`	further arguments for `tfb_spline()` or `tfb_fpc()`.

Details

tfb is a wrapper for functions that set up spline-, principal component- or wavelet-based representations of functional data. For all three, the input data $x_i(t)$ are represented as weighted sums of a set of common basis functions $B_k(t); k = 1, \ldots, K$ identical for all observations and weight or coefficient vectors $b_i = (b_{i1}, \ldots, b_{iK})$ estimated for each observation: $x_i(t) \approx \sum_k B_k(t) b_{ik}$. Depending on the value of basis, the basis functions $B(t)$ will either be spline functions or the first few estimated eigenfunctions of the covariance operator of the $x(t)$ (fpc) or wavelets (wavelet).

See tfb_spline() for more details on spline basis representation (the default). See tfb_fpc() for using an functional principal component representation with an orthonormal basis estimated from the data instead.

Value

a tfb-object (or a data.frame/matrix for the conversion functions, obviously).

See Also

Other tfb-class: fpc_wsvd(), tfb_fpc(), tfb_spline()

Other tfb-class: fpc_wsvd(), tfb_fpc(), tfb_spline()

Examples

```
arg <- seq(0, 1, length.out = 21)
x <- tfd(rbind(sin(2 * pi * arg), cos(2 * pi * arg)), arg = arg)
xb <- tfb(x, k = 8, penalized = FALSE)
xb

as.tfb(x, basis = "spline", k = 8)
```

---

tfbrackets *Accessing, evaluating, subsetting and subassigning* tf *vectors*

Description

These functions access, subset, replace and evaluate tf objects. For more information on creating tf objects and converting them to/from list, data.frame or matrix, see tfd() and tfb(). See details.

Usage

```
## S3 method for class 'tf'
x[i, j, interpolate = TRUE, matrix = TRUE]

## S3 replacement method for class 'tf'
x[i] <- value
```

Arguments

x	an tf.
i	index of the observations (integerish, character or logical, usual R rules apply). Can also be a two-column matrix for extracting specific (function, arg-value) pairs: the first column gives the function indices, the second column gives the arg values at which to evaluate each function. Returns a numeric vector in that case. j must not be provided when i is a matrix.
j	The arg used to evaluate the functions. A (list of) numeric vectors. *NOT* interpreted as a column number but as the argument value of the respective functional datum. If j is missing but matrix is explicitly given, j defaults to tf_arg(x).
interpolate	should functions be evaluated (i.e., inter-/extrapolated) for values in arg for which no original data is available? Only relevant for the raw data class tfd, for which it defaults to TRUE. Basis-represented tfb are always "interpolated".
matrix	should the result be returned as a matrix or as a list of data.frames? If TRUE, j has to be a (list of a) single vector of arg. See return value.
value	tf object for subassignment. This is typed more strictly than concatenation: subassignment only happens if the common type of value and x is the same as the type of x, so subassignment never changes the type of x but may do a potentially lossy cast of value to the type of x (with a warning).

Details

Note that these break certain (terrible) R conventions for vector-like objects:

- no argument recycling,
- no indexing with NA,
- no indexing with names not present in x,
- no indexing with integers > length(x)

All of the above will trigger errors.

Value

If i is a two-column matrix, a numeric vector of pointwise evaluations (one per row of i).
If j is missing (and i is not a matrix), a subset of the functions in x as given by i.
If j is given and matrix == TRUE, a numeric matrix of function evaluations in which each row represents one function and each column represents one argval as given in argument j, with an attribute arg=j and row- and column-names derived from x[i] and j.
If j is given and matrix == FALSE, a list of tbl_dfs with columns arg = j and value = evaluations at j for each observation in i.

Examples

```
x <- 1:3 * tfd(data = 0:10, arg = 0:10)
plot(x)
# this operator's 2nd argument is quite overloaded -- you can:
```

```
# 1. simply extract elements from the vector if no second arg is given:
x[1]
x[c(TRUE, FALSE, FALSE)]
x[-(2:3)]
# 2. use the second argument and optional additional arguments to
#    extract specific function evaluations in a number of formats:
x[1:2, c(4.5, 9)] # returns a matrix of function evaluations
x[1:2, c(4.5, 9), interpolate = FALSE] # NA for arg-values not in the original data
x[-3, seq(1, 9, by = 2), matrix = FALSE] # list of data.frames for each function
# 3. use a 2-column matrix to extract specific (function, arg) pairs:
x[cbind(1:3, c(0, 5, 10))] # one value per function
# 4. use matrix= with a missing j to evaluate on the default arg grid:
x[1:2, , matrix = FALSE] # same as x[1:2, tf_arg(x), matrix = FALSE]
# in order to evaluate a set of observed functions on a new grid and
# save them as a functional data vector again, use `tfd` or `tfb` instead:
tfd(x, arg = seq(0, 10, by = 0.01))
```

---

tfb_fpc *Functional data in FPC-basis representation*

Description

These functions perform a (functional) principal component analysis (FPCA) of the input data and return an tfb_fpc tf-object that uses the empirical eigenfunctions as basis functions for representing the data. The default ("method = fpc_wsvd") uses a (truncated) weighted SVD for complete data on a common grid and a nuclear-norm regularized (truncated) weighted SVD for partially missing data on a common grid, see fpc_wsvd(). The latter is likely to break down for high PVE and/or high amounts of missingness.

Usage

```
tfb_fpc(data, ...)

## S3 method for class 'data.frame'
tfb_fpc(
  data,
  id = 1,
  arg = 2,
  value = 3,
  domain = NULL,
  method = fpc_wsvd,
  ...
)

## S3 method for class 'matrix'
tfb_fpc(data, arg = NULL, domain = NULL, method = fpc_wsvd, ...)
```

```
## S3 method for class 'numeric'
tfb_fpc(data, arg = NULL, domain = NULL, method = fpc_wsvd, ...)

## S3 method for class 'tf'
tfb_fpc(data, arg = NULL, method = fpc_wsvd, ...)

## Default S3 method:
tfb_fpc(data, arg = NULL, domain = NULL, method = fpc_wsvd, ...)
```

Arguments

data	a matrix, data.frame or list of suitable shape, or another tf-object containing functional data.
...	arguments to the method which computes the (regularized/smoothed) FPCA - see e.g. fpc_wsvd(). Unless set by the user, uses proportion of variance explained pve = 0.995 to determine the truncation levels.
id	The name or number of the column defining which data belong to which function.
arg	For the list- and matrix-methods: numeric, or list of numerics. The evaluation grid. For the data.frame-method: the name/number of the column defining the evaluation grid. The matrix method will try to guess suitable arg-values from the column names of data if arg is not supplied. Other methods fall back on integer sequences (1:<length of data>) as the default if not provided.
value	The name or number of the column containing the function evaluations.
domain	range of the arg.
method	the function to use that computes eigenfunctions and scores. Defaults to fpc_wsvd(), which is quick and easy but returns completely unsmoothed eigenfunctions unlikely to be suited for noisy data. See details.

Details

For the FPC basis, any factorization method that accepts a data.frame with columns id, arg, value containing the functional data and returns a list with eigenfunctions and FPC scores structured like the return object of fpc_wsvd() can be used for the method argument, see example below. Note that the mean function, with a fixed "score" of 1 for all functions, is used as the first basis function for all FPC bases.

Value

an object of class tfb_fpc, inheriting from tfb. The basis used by tfb_fpc is a tfd-vector containing the estimated mean and eigenfunctions.

Methods (by class)

- tfb_fpc(default): convert tfb: default method, returning prototype when data is NULL

See Also

fpc_wsvd() for FPCA options.

Other tfb-class: fpc_wsvd(), tfb, tfb_spline()

Other tfb_fpc-class: fpc_wsvd()

Examples

```
set.seed(13121)
x <- tf_rgp(25, nugget = .02)
x_pc <- tfb_fpc(x, pve = .9)
x_pc
plot(x, lwd = 3)
lines(x_pc, col = 2, lty = 2)
x_pc_full <- tfb_fpc(x, pve = .995)
x_pc_full
lines(x_pc_full, col = 3, lty = 2)
# partially missing data on common grid:
x_mis <- x |> tf_sparsify(dropout = .05)
x_pc_mis <- tfb_fpc(x_mis, pve = .9)
x_pc_mis
plot(x_mis, lwd = 3)
lines(x_pc_mis, col = 4, lty = 2)
# extract FPC basis --
# first "eigenvector" in black is (always) the mean function
x_pc |> tf_basis(as_tfd = TRUE) |> plot(col = 1:5)

# Apply FPCA for sparse, irregular data using refund::fpca.sc:
set.seed(99290)
# create small, sparse, irregular data:
x_irreg <- x[1:8] |>
  tf_jiggle() |> tf_sparsify(dropout = 0.3)
plot(x_irreg)
x_df <- x_irreg |>
  as.data.frame(unnest = TRUE)
# wrap refund::fpca_sc for use as FPCA method in tfb_fpc --
# 1. define scoring function (simple weighted LS fit)
fpca_scores <- function(data_matrix, efunctions, mean, weights) {
  w_mat <- matrix(weights, ncol = length(weights), nrow = nrow(data_matrix),
                  byrow = TRUE)
  w_mat[is.na(data_matrix)] <- 0
  data_matrix[is.na(data_matrix)] <- 0
  data_wc <- t((t(data_matrix) - mean) * sqrt(t(w_mat)))
  t(qr.coef(qr(efunctions), t(data_wc) / sqrt(weights)))
}
# 2. define wrapper for fpca_sc:
fpca_sc_wrapper <- function(data, arg, pve = 0.995, ...) {
  data_mat <- tfd(data) |> as.matrix(interpolate = TRUE)
  fpca <- refund::fpca.sc(
    Y = data_mat, argvals = attr(data_mat, "arg"), pve = pve, ...
  )
  c(fpca[c("mu", "efunctions", "scores", "npc")],
```

```
    scoring_function = fpca_scores)
}
x_pc <- tfb_fpc(x_df, method = fpca_sc_wrapper)
lines(x_pc, col = 2, lty = 2)
```

`tfb_spline` *Spline-based representation of functional data*

Description

Represent curves as a weighted sum of spline basis functions.

Usage

```
tfb_spline(data, ...)

## S3 method for class 'data.frame'
tfb_spline(
  data,
  id = 1,
  arg = 2,
  value = 3,
  domain = NULL,
  penalized = TRUE,
  global = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'matrix'
tfb_spline(
  data,
  arg = NULL,
  domain = NULL,
  penalized = TRUE,
  global = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'numeric'
tfb_spline(
  data,
  arg = NULL,
  domain = NULL,
```

```
  penalized = TRUE,
  global = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'list'
tfb_spline(
  data,
  arg = NULL,
  domain = NULL,
  penalized = TRUE,
  global = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'fd'
tfb_spline(
  data,
  arg = NULL,
  domain = NULL,
  penalized = FALSE,
  global = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'fdSmooth'
tfb_spline(
  data,
  arg = NULL,
  domain = NULL,
  penalized = FALSE,
  global = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'tfd'
tfb_spline(
  data,
  arg = NULL,
  domain = NULL,
  penalized = TRUE,
  global = FALSE,
  verbose = TRUE,
  ...
```

```
)

## S3 method for class 'tfb'
tfb_spline(
  data,
  arg = NULL,
  domain = NULL,
  penalized = TRUE,
  global = FALSE,
  verbose = TRUE,
  ...
)

## Default S3 method:
tfb_spline(
  data,
  arg = NULL,
  domain = NULL,
  penalized = TRUE,
  global = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

data	a matrix, data.frame or list of suitable shape, or another tf-object containing functional data.
...	arguments to the calls to mgcv::s() setting up the basis (and to mgcv::magic() or mgcv::gam.fit() if penalized = TRUE). Uses k = 25 cubic regression spline basis functions (bs = "cr") by default, but should be set appropriately by the user. See details and examples in the vignettes.
id	The name or number of the column defining which data belong to which function.
arg	For the list- and matrix-methods: numeric, or list of numerics. The evaluation grid. For the data.frame-method: the name/number of the column defining the evaluation grid. The matrix method will try to guess suitable arg-values from the column names of data if arg is not supplied. Other methods fall back on integer sequences (1:<length of data>) as the default if not provided.
value	The name or number of the column containing the function evaluations.
domain	range of the arg.
penalized	TRUE (default) estimates regularized/penalized basis coefficients via mgcv::magic() or mgcv::gam.fit(), FALSE yields ordinary least squares / ML estimates for basis coefficients. FALSE is much faster but will overfit for noisy data if k is (too) large.
global	Defaults to FALSE. If TRUE and penalized = TRUE, all functions share the same smoothing parameter (see details).

verbose TRUE (default) outputs statistics about the fit achieved by the basis and other diagnostic messages.

Details

The basis to be used is set up via a call to mgcv::s() and all the spline bases discussed in mgcv::smooth.terms() are available, in principle. Depending on the value of the penalized- and global-flags, the coefficient vectors for each observation are then estimated via fitting a GAM (separately for each observation, if !global) via mgcv::magic() (least square error, the default) or mgcv::gam() (if a family argument was supplied) or unpenalized least squares / maximum likelihood.

After the "smoothed" representation is computed, the amount of smoothing that was performed is reported in terms of the "percentage of variability preserved", which is the variance (or the explained deviance, in the general case if family was specified) of the smoothed function values divided by the variance of the original values (the null deviance, in the general case). Reporting can be switched off with verbose = FALSE.

The ... arguments supplies arguments to both the spline basis (via mgcv::s()) and the estimation (via mgcv::magic() or mgcv::gam()), the most important arguments are:

- k: how many basis functions should the spline basis use, default is 25.
- bs: which type of spline basis should be used, the default is cubic regression splines (bs = "cr")
- family argument: use this if minimizing squared errors is not a reasonable criterion for the representation accuracy (see mgcv::family.mgcv() for what's available) and/or if function values are restricted to be e.g. positive (family = Gamma()/tw()/...), in $[0, 1]$ (family = betar()), etc.
- sp: numeric value for the smoothness penalty weight, for manually setting the amount of smoothing for all curves, see mgcv::s(). This (drastically) reduces computation time. Defaults to -1, i.e., automatic optimization of sp using mgcv::magic() (LS fits) or mgcv::gam() (GLM), source code in R/tfb-spline-utils.R.

If global == TRUE, this uses a small subset of curves (10% of curves, at least 5, at most 100; non-random sample using every j-th curve in the data) on which smoothing parameters per curve are estimated and then takes the mean of the log smoothing parameter of those as sp for all curves. This is much faster than optimizing for each curve on large data sets. For very sparse or noisy curves, estimating a common smoothing parameter based on the data for all curves simultaneously is likely to yield better results, this is *not* what's implemented here.

Value

a tfb-object

Methods (by class)

- tfb_spline(data.frame): convert data frames
- tfb_spline(matrix): convert matrices
- tfb_spline(numeric): convert matrices
- tfb_spline(list): convert lists

- `tfb_spline(fd)`: convert `fd` objects. Almost exact re-representation for objects using Fourier- or B-spline bases, other `fda`-style bases are not implemented here.
- `tfb_spline(fdSmooth)`: convert `fdSmooth` objects. Almost exact re-representation for objects using Fourier- or B-spline bases, other `fda`-style bases are not implemented here.
- `tfb_spline(tfd)`: convert `tfd` (raw functional data)
- `tfb_spline(tfb)`: convert `tfb`: modify basis representation, smoothing.
- `tfb_spline(default)`: convert `tfb`: default method, returning prototype when data is missing

See Also

`mgcv::smooth.terms()` for spline basis options.

Other tfb-class: `fpc_wsvd()`, `tfb`, `tfb_fpc()`

Examples

```
arg <- seq(0, 1, length.out = 21)
mat <- rbind(sin(2 * pi * arg), cos(2 * pi * arg))
fit <- tfb_spline(mat, arg = arg, k = 8, penalized = FALSE, verbose = FALSE)
fit
```

---

tfd *Constructors for vectors of "raw" functional data*

Description

Various constructor methods for `tfd`-objects.
`tfd` objects contain vectors of function evaluations at observed `arg`-values, either all at the same `arg`-values (`tfd_reg`) or at different `arg`-values (`tfd_irreg`). `NA`-functions are represented by `NULL`-entries in that list.

`tfd.matrix` accepts a numeric matrix with one function per *row* (!). If `arg` is not provided, it tries to guess `arg` from the column names and falls back on `1:ncol(data)` if that fails.

`tfd.data.frame` uses the first 3 columns of `data` for `id` (function ID), `arg` (argument value) and `value` (function value) by default.

`tfd.list` accepts a list of vectors of identical lengths containing evaluations or a list of 2-column matrices/data.frames with `arg` in the first and evaluations in the second column

`tfd.default` returns class prototype when argument to tfd() is `NULL` or not a recognised class.

`as.tfd_irreg` converts regular `tfd` or `tfb` objects into irregular ones. Mainly used internally for `tf_rebase` operations etc.

Usage

```
tfd(data, ...)

## S3 method for class 'matrix'
tfd(data, arg = NULL, domain = NULL, evaluator = tf_approx_linear, ...)

## S3 method for class 'numeric'
tfd(data, arg = NULL, domain = NULL, evaluator = tf_approx_linear, ...)

## S3 method for class 'data.frame'
tfd(
  data,
  id = 1,
  arg = 2,
  value = 3,
  domain = NULL,
  evaluator = tf_approx_linear,
  ...
)

## S3 method for class 'list'
tfd(data, arg = NULL, domain = NULL, evaluator = tf_approx_linear, ...)

## S3 method for class 'tf'
tfd(data, arg = NULL, domain = NULL, evaluator = NULL, ...)

## Default S3 method:
tfd(data, arg = NULL, domain = NULL, evaluator = tf_approx_linear, ...)

as.tfd(data, ...)

as.tfd_irreg(data, ...)
```

Arguments

data	a matrix, data.frame or list of suitable shape, or another tf-object. when this argument is NULL (i.e. when calling tfd()) this returns a prototype of class tfd.
...	not used in tfd, except for tfd.tf – specify arg and interpolate = TRUE to turn an irregular tfd into a regular one, see examples.
arg	For the list- and matrix-methods: numeric, or list of numerics. The evaluation grid. For the data.frame-method: the name/number of the column defining the evaluation grid. The matrix method will try to guess suitable arg-values from the column names of data if arg is not supplied. Other methods fall back on integer sequences (1:<length of data>) as the default if not provided.
domain	range of the arg.
evaluator	a function accepting arguments x, arg, evaluations. See details for tfd().

id	The name or number of the column defining which data belong to which function.
value	The name or number of the column containing the function evaluations.

Details

tfd-objects are list-vctrs of numeric vectors containing function evaluations.

evaluator: must be the (quoted or bare) name of a function with signature function(x, arg, evaluations) that returns the functions' (approximated/interpolated) values at locations x based on the function evaluations available at locations arg.
Available evaluator-functions:

- tf_approx_linear for linear interpolation without extrapolation (i.e., zoo::na.approx() with na.rm = FALSE) – this is the default,
- tf_approx_spline for cubic spline interpolation, (i.e., zoo::na.spline() with na.rm = FALSE),
- tf_approx_fill_extend for linear interpolation and constant extrapolation (i.e., zoo::na.fill() with fill = "extend")
- tf_approx_locf for "last observation carried forward" (i.e., zoo::na.locf() with na.rm = FALSE)
- tf_approx_nocb for "next observation carried backward" (i.e., zoo::na.locf() with na.rm = FALSE, fromLast = TR
  See tf:::zoo_wrapper and tf:::tf_approx_linear, which is simply zoo_wrapper(zoo::na.approx, na.rm = FALSE), for examples of implementations of this.

Value

a tfd-object (or a data.frame/matrix for the conversion functions, obviously).

Examples

```
# turn irregular to regular tfd by evaluating on a common grid:

f <- c(
  tf_rgp(1, arg = seq(0, 1, length.out = 11)),
  tf_rgp(1, arg = seq(0, 1, length.out = 21))
)
tfd(f, arg = seq(0, 1, length.out = 21))

set.seed(1213)
f <- tf_rgp(3, arg = seq(0, 1, length.out = 51)) |> tf_sparsify(0.9)
# does not yield regular data because linear extrapolation yields NAs
#   outside observed range:
tfd(f, arg = seq(0, 1, length.out = 101))
# this "works" (but may not yield sensible values..!!) for
#   e.g. constant extrapolation:
tfd(f, evaluator = tf_approx_fill_extend, arg = seq(0, 1, length.out = 101))
plot(f, col = 2)
tfd(f,
  arg = seq(0, 1, length.out = 151), evaluator = tf_approx_fill_extend
) |> lines()
```

`tfgroupgenerics` *Math, Summary and Ops Methods for* `tf`

Description

These methods and operators mostly work `arg`-value-wise on `tf` objects, see `vctrs::vec_arith()` etc. for implementation details.

Usage

```
## S3 method for class 'tfd'
e1 == e2

## S3 method for class 'tfd'
e1 != e2

## S3 method for class 'tfb'
e1 == e2

## S3 method for class 'tfb'
e1 != e2

## S3 method for class 'tfd'
vec_arith(op, x, y, ...)

## S3 method for class 'tfb'
vec_arith(op, x, y, ...)

## S3 method for class 'tfd'
Math(x, ...)

## S3 method for class 'tfb'
Math(x, ...)

## S3 method for class 'tf'
Summary(...)

## S3 method for class 'tfd'
cummax(...)

## S3 method for class 'tfd'
cummin(...)

## S3 method for class 'tfd'
cumsum(...)

## S3 method for class 'tfd'
```

```
cumprod(...)

## S3 method for class 'tfb'
cummax(...)

## S3 method for class 'tfb'
cummin(...)

## S3 method for class 'tfb'
cumsum(...)

## S3 method for class 'tfb'
cumprod(...)
```

Arguments

e1	an tf or a numeric vector.
e2	an tf or a numeric vector.
op	An arithmetic operator as a string.
x	a tf or numeric object.
y	a tf or numeric object.
...	tf-objects (not used for Math group generic).

Details

- Operations on tfd-objects do not extrapolate functions on a common grid first, they operate on the function at argument values that both objects have in common.
- With the exception of addition and multiplication, operations on tfb-objects first evaluate the data on their arg, perform computations on these evaluations and then convert back to an tfb-object, so a loss of precision should be expected – especially so for small spline bases and/or very wiggly data.
- Equality checks of functional objects are even more iffy than usual for computer math and not very reliable.
- Note that max and min are not guaranteed to be maximal/minimal over the entire domain, only at the argument values used for computation.

See examples below, many more are in tests/testthat/test-ops.R.

Value

a tf- or logical vector with the computed result.

See Also

tf_fwise() for scalar summaries of each function in a tf-vector

Examples

```
set.seed(1859)
f <- tf_rgp(4)
2 * f == f + f
sum(f) == f[1] + f[2] + f[3] + f[4]
log(exp(f)) == f
plot(f, points = FALSE)
lines(range(f), col = 2, lty = 2)

f2 <- tf_rgp(5) |> exp() |> tfb(k = 25)
layout(t(1:3))
plot(f2, col = gray.colors(5))
plot(cummin(f2), col = gray.colors(5))
plot(cumsum(f2), col = gray.colors(5))

# ?tf_integrate for integrals, ?tf_fwise for scalar summaries of each function
```

---

tfsummaries *Functions that summarize* tf *objects across argument values*

Description

These will return a `tf` object containing the respective *functional* statistic. See `tf_fwise()` for scalar summaries (e.g. `tf_fmean` for means, `tf_fmax` for max. values) of each entry in a `tf`-vector.

Usage

```
## S3 method for class 'tf'
mean(x, ...)

## S3 method for class 'tf'
median(x, na.rm = FALSE, depth = "MBD", ...)

sd(x, na.rm = FALSE)

## Default S3 method:
sd(x, na.rm = FALSE)

## S3 method for class 'tf'
sd(x, na.rm = FALSE)

var(x, y = NULL, na.rm = FALSE, use)

## Default S3 method:
var(x, y = NULL, na.rm = FALSE, use)

## S3 method for class 'tf'
var(x, y = NULL, na.rm = FALSE, use)
```

```
## S3 method for class 'tf'
summary(object, ..., depth = "MBD")
```

Arguments

x	a tf object.
...	optional additional arguments.
na.rm	logical. Should missing values be removed?
depth	depth method used for computing the median and central region. See tf_depth() for available methods, or pass a custom depth function. Defaults to "MBD".
y	NULL (default) or a vector, matrix or data frame with compatible dimensions to x. The default is equivalent to y = x (but more efficient).
use	an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".
object	a tfd object

Value

a tf object with the computed result.
summary.tf returns a tf-vector with the mean function, the functional median, the *pointwise* min and max of x, and the *pointwise* min and max of the central half of the functions in x, as defined by the chosen depth (default "MBD", see tf_depth()).

See Also

tf_fwise()

Other tidyfun summary functions: fivenum(), functionwise

Examples

```
set.seed(123)
x <- tf_rgp(1) * 1:5
mean(x)
median(x, depth = "pointwise")
sd(x)
var(x)
summary(x)
```

tf_align *Apply warping functions to align functional data*

Description

`tf_align()` applies the *inverse* warping function to unregistered data to obtain aligned (registered) functions.

Usage

```
tf_align(x, warp, ...)

## S3 method for class 'tfd'
tf_align(x, warp, ..., keep_new_arg = FALSE)

## S3 method for class 'tfb'
tf_align(x, warp, ...)
```

Arguments

x	`tf` vector of functions. For `tf_warp()`, these should be registered/aligned functions and unaligned functions for `tf_align()`.
warp	`tf` vector of warping functions used for transformation. See Details.
...	additional arguments passed to `tfd()`.
keep_new_arg	keep new `arg` values after (un)warping or return `tfd` vector on `arg` values of the input (default `FALSE` is the latter)? See Details.

Value

the aligned `tf` vector (registered functions)

See Also

Other registration functions: `tf_estimate_warps()`, `tf_landmarks_extrema()`, `tf_register()`, `tf_registration`, `tf_warp()`

Examples

```
# Estimate warps, then align manually:
t <- seq(0, 2 * pi, length.out = 101)
x <- tfd(t(sapply(c(-0.3, 0, 0.3), function(s) sin(t + s))), arg = t)
warps <- tf_estimate_warps(x, method = "affine", type = "shift")
aligned <- tf_align(x, warps)
plot(aligned, col = 1:3)
```

tf_approx_linear *Inter- and extrapolation functions for* tfd*-objects*

Description

These are exported evaluator callbacks for tfd objects. They control how function values are inter-/extrapolated to previously unseen arg values and are used by tf_evaluate().

In typical use, set an evaluator when constructing a tfd (tfd(..., evaluator = tf_approx_linear)) or replace it later via tf_evaluator(x) <- tf_approx_none.

These helpers are wrappers around zoo::na.fill(), zoo::na.approx(), etc. and all share the same signature (x, arg, evaluations), so they can also be called directly.

The list:

- tf_approx_linear for linear interpolation without extrapolation (i.e., zoo::na.approx() with na.rm = FALSE) – this is the default,
- tf_approx_spline for cubic spline interpolation, (i.e., zoo::na.spline() with na.rm = FALSE),
- tf_approx_none in order to not inter-/extrapolate ever (i.e., zoo::na.fill() with fill = NA)
- tf_approx_fill_extend for linear interpolation and constant extrapolation (i.e., zoo::na.fill() with fill = "extend")
- tf_approx_locf for "last observation carried forward" (i.e., zoo::na.locf() with na.rm = FALSE)
- tf_approx_nocb for "next observation carried backward" (i.e., zoo::na.locf() with na.rm = FALSE, fromLast = TR

For implementing your own, see source code of tf:::zoo_wrapper.

Usage

```
tf_approx_linear(x, arg, evaluations)

tf_approx_spline(x, arg, evaluations)

tf_approx_none(x, arg, evaluations)

tf_approx_fill_extend(x, arg, evaluations)

tf_approx_locf(x, arg, evaluations)

tf_approx_nocb(x, arg, evaluations)
```

Arguments

x	new arg values to approximate/interpolate/extrapolate the function for.
arg	the arg values of the evaluations.
evaluations	the function values at arg.

Value

a vector of values of the function defined by the given $(x_i, f(x_i))$=(arg, evaluations)-tuples at new argument values x.

See Also

tfd()

Other tidyfun inter/extrapolation functions: tf_evaluate(), tf_interpolate()

Other tidyfun inter/extrapolation functions: tf_evaluate(), tf_interpolate()

Other tidyfun inter/extrapolation functions: tf_evaluate(), tf_interpolate()

Other tidyfun inter/extrapolation functions: tf_evaluate(), tf_interpolate()

Other tidyfun inter/extrapolation functions: tf_evaluate(), tf_interpolate()

Other tidyfun inter/extrapolation functions: tf_evaluate(), tf_interpolate()

Examples

```
x <- tfd(matrix(c(0, 1), nrow = 1), arg = c(0, 1))
tf_evaluate(x, c(0, 0.5, 1))
tf_evaluator(x) <- tf_approx_none
tf_evaluate(x, c(0, 0.5, 1))

tf_approx_linear(
  x = c(0, 0.5, 1),
  arg = c(0, 1),
  evaluations = c(0, 1)
)
```

---

tf_arg *Utility functions for* tf*-objects*

Description

A bunch of methods & utilities that do what they say: get or set the respective attributes of a tf-object.

Usage

```
tf_arg(f)

tf_evaluations(f)

tf_count(f)

tf_domain(f)

tf_domain(x) <- value
```

```
tf_evaluator(f)

tf_evaluator(x) <- value

tf_basis(f, as_tfd = FALSE)

tf_arg(x) <- value

## S3 replacement method for class 'tfd_irreg'
tf_arg(x) <- value

## S3 replacement method for class 'tfd_reg'
tf_arg(x) <- value

## S3 replacement method for class 'tfb'
tf_arg(x) <- value

## S3 method for class 'tfb'
coef(object, ...)

## S3 method for class 'tf'
rev(x)

## S3 method for class 'tf'
is.na(x)

## S3 method for class 'tfd_irreg'
is.na(x)

is_tf(x)

is_tfd(x)

is_reg(x)

is_tfd_reg(x)

is_irreg(x)

is_tfd_irreg(x)

is_tfb(x)

is_tfb_spline(x)

is_tfb_fpc(x)
```

Arguments

f	an tf object.
x	an tf object.
value	**for** tf_evaluator<-**:** (bare or quoted) name of a function that can be used to interpolate an tfd. Needs to accept vector arguments x, arg, evaluations and return evaluations of the function defined by arg, evaluations at x. **for** tf_arg<-**:** (list of) new arg-values. **for** tf_domain<-**:** sorted numeric vector with the 2 new endpoints of the domain.
as_tfd	should the basis be returned as a tfd-vector evaluated on tf_arg(f)? Defaults to FALSE, which returns the matrix of basis functions (columns) evaluated on tf_arg(f) (rows).
object	as usual
...	dots

Value

either the respective attribute or, for setters (assignment functions), the input object with modified properties.

See Also

Other tidyfun utility functions: in_range(), tf_zoom()

Examples

```
x <- tf_rgp(3)
tf_arg(x)
tf_evaluations(x)
tf_count(x)
tf_domain(x)
tf_evaluator(x)
tf_evaluate(x, 0.25)
tf_evaluator(x) <- tf_approx_none
tf_evaluate(x, 0.25)
c(is_tf(x), is_tfd(x), is_reg(x), is_irreg(x))

xb <- tfb(x, k = 4, penalized = FALSE, verbose = FALSE)
tf_basis(xb)
tf_basis(xb)(c(0, .1, .2))
c(is_tfb(xb), is_tfb_spline(xb), is_tfb_fpc(xb))
```

tf_depth *Functional Data Depth*

Description

Data depths for functional data. All depths are scaled so that 1 means most central and 0 means most extreme. Available methods:

Usage

```
tf_depth(x, arg, depth = "MBD", na.rm = TRUE, ...)

## S3 method for class 'matrix'
tf_depth(
  x,
  arg,
  depth = c("MBD", "MHI", "FM", "FSD", "RPD"),
  na.rm = TRUE,
  ...
)

## S3 method for class 'tf'
tf_depth(x, arg, depth = "MBD", na.rm = TRUE, ...)
```

Arguments

x	tf (or a matrix of evaluations).
arg	grid of evaluation points.
depth	one of "MBD", "MHI", "FM", "FSD", or "RPD".
na.rm	remove missing observations? Defaults to TRUE.
...	for "RPD": u (regularization quantile, default 0.01), n_projections (M, default 5000), n_projections_beta (L, default 500).

Details

- "MBD": Modified Band-2 Depth (default). Scale of 0 (most extreme) to 1 (most central).
- "MHI": Modified Hypograph Index. **Ranks functions from lowest (0) to highest (1)** instead of most extreme to most central! For functions that never cross: $MBD = -2(MHI - 0.5)^2 + .5$.
- "FM": Fraiman-Muniz depth. Integrates pointwise univariate halfspace depths over the domain. Scale of 0 (most extreme) to 1 (most central).
- "FSD": Functional Spatial Depth. Based on spatial signs; robust to outliers. Scale of 0 (most extreme) to 1 (most central).

- "RPD": Regularized Projection Depth. Projects curves onto random directions and computes outlyingness. Especially useful for detecting shape outliers. Scale of 0 (most extreme) to 1 (most central). **Note:** results depend on the RNG state; set a seed (e.g. `set.seed(...)`) before calling for reproducibility. Accepts additional arguments via `...`: `u` (quantile level for regularization, default 0.01), `n_projections` (M, number of projection directions, default 5000), `n_projections_beta` (L, directions for estimating regularization parameter, default 500).

Value

vector of depth values

References

Sun, Ying, Genton, G M, Nychka, W D (2012). "Exact fast computation of band depth for large functional datasets: How quickly can one million curves be ranked?" *Stat*, **1**(1), 68–74.

López-Pintado, Sara, Romo, Juan (2009). "On the concept of depth for functional data." *Journal of the American Statistical Association*, **104**(486), 718–734.

López-Pintado, Sara, Romo, Juan (2011). "A half-region depth for functional data." *Computational Statistics & Data Analysis*, **55**(4), 1679–1695.

Fraiman, Ricardo, Muniz, Graciela (2001). "Trimmed means for functional data." *Test*, **10**(2), 419–440.

Chakraborty, Anirvan, Chaudhuri, Probal (2014). "The spatial distribution in infinite dimensional spaces and related quantiles and depths." *The Annals of Statistics*, **42**(3), 1203–1231.

Bočinec, Filip, Nagy, Stanislav, Yeon, Hyemin (2026). "Projection depth for functional data: Practical issues, computation and applications." *arXiv preprint arXiv:2602.22877*.

See Also

Other tidyfun ordering and ranking functions: `tf_minmax`, `tf_order`

Examples

```
x <- tf_rgp(3)/3 + 1:3
tf_depth(x, depth = "MBD")
tf_depth(x, depth = "MHI")
tf_depth(x, depth = "FM")
tf_depth(x, depth = "FSD")
```

---

tf_derive *Differentiating functional data: approximating derivative functions*

Description

Derivatives of `tf`-objects use finite differences of the evaluations for `tfd` and finite differences of the basis functions for `tfb`.

Usage

```
tf_derive(f, arg, order = 1, ...)

## S3 method for class 'matrix'
tf_derive(f, arg, order = 1, ...)

## S3 method for class 'tfd'
tf_derive(f, arg = tf_arg(f), order = 1, ...)

## S3 method for class 'tfd_irreg'
tf_derive(f, arg, order = 1, ...)

## S3 method for class 'tfb_spline'
tf_derive(f, arg = tf_arg(f), order = 1, ...)

## S3 method for class 'tfb_fpc'
tf_derive(f, arg = tf_arg(f), order = 1, ...)
```

Arguments

f	a tf-object
arg	grid to use for the finite differences.
order	order of differentiation. Maximal value for tfb_spline is 2. For tfb_spline-objects, order = -1 yields integrals (used internally).
...	not used

Details

The derivatives of tfd objects use second-order accurate central differences for interior points and second-order accurate one-sided differences at boundaries, following the non-uniform grid formulas from numpy.gradient with edge_order=2 (Fornberg, 1988). Domain and grid of the returned object are identical to the input. Unless the tfd has a rather fine and regular grid, representing the data in a suitable basis representation with tfb() and then computing the derivatives (or integrals) of those is usually preferable.

Note that, for spline bases like "cr" or "tp" which are constrained to begin/end linearly, computing *second* derivatives will produce artefacts at the outer limits of the functions' domain due to these boundary constraints. Basis "bs" does not have this problem for sufficiently high orders (but tends to yield slightly less stable fits).

Value

a tf (with the same arg for tfd-inputs, possibly different basis for tfb-inputs, see details).

Methods (by class)

- tf_derive(matrix): row-wise finite differences
- tf_derive(tfd): derivatives by finite differencing of function evaluations.

- `tf_derive(tfd_irreg)`: element-wise finite differencing for irregular grids. Falls back to `tf_derive.tfd` (interpolating to a common grid) if an explicit `arg` vector is supplied.
- `tf_derive(tfb_spline)`: derivatives by finite differencing of spline basis functions.
- `tf_derive(tfb_fpc)`: derivatives by finite differencing of FPC basis functions.

References

Fornberg, Bengt (1988). "Generation of Finite Difference Formulas on Arbitrarily Spaced Grids." *Mathematics of Computation*, **51**(184), 699–706.

See Also

Other tidyfun calculus functions: `tf_integrate()`

Examples

```
arg <- seq(0, 1, length.out = 31)
x <- tfd(rbind(arg^2, sin(2 * pi * arg)), arg = arg)
dx <- tf_derive(x)
x
dx
tf_arg(dx)
```

`tf_estimate_warps` *Estimate warping functions for registration*

Description

`tf_estimate_warps()` is the low-level workhorse for functional data registration. It estimates warping functions that align a set of functions to a template, but does *not* apply them. For a one-shot interface that also aligns the data, see `tf_register()`.

Usage

```
tf_estimate_warps(
  x,
  ...,
  template = NULL,
  method = c("srvf", "cc", "affine", "landmark"),
  max_iter = 3L,
  tol = 0.01
)
```

Arguments

x	a tf vector of functions to register.
...	additional method-specific arguments passed to backend routines (for example crit for method = "cc").
template	an optional tf vector of length 1 to use as the template. If NULL, a default template is computed (method-dependent). Not used for method = "landmark".
method	the registration method to use: • "srvf": Square Root Velocity Framework (elastic registration). For details, see fdasrvf::time_warping(). Default template is the Karcher mean. • "cc": continuous-criterion registration via a tf-native dense-grid optimizer with monotone spline warps. Default template is the arithmetic mean. • "affine": affine (linear) registration with warps of the form $h(t) = a \cdot t + b$. Simpler than elastic registration, appropriate when phase variability consists only of shifts and/or uniform speed-up/slow-down. Default template is the arithmetic mean. • "landmark": piecewise-linear warps that align user-specified landmark features. Requires landmarks argument.
max_iter	integer: maximum number of Procrustes-style template refinement iterations when template = NULL. The iteration cycle is: (1) estimate template as mean of (aligned) curves, (2) register all curves to current template, (3) update template as mean of newly aligned curves, (4) repeat until convergence or max_iter reached. Ignored when template is provided (no refinement needed) or for method = "landmark" (template not used). For method = "srvf" with template = NULL, the outer Procrustes loop is skipped regardless of max_iter because fdasrvf::time_warping() already computes the Karcher mean internally. Default is 3L.
tol	numeric: convergence tolerance for template refinement. For method = "cc", iteration stops when the relative improvement in the registration criterion becomes negligible; for the other iterative methods, iteration stops when the relative change in the template (L2 norm) falls below tol. Default is 1e-2.

Details

For method = "cc", tf uses a tf-native dense-grid optimizer with monotone spline warps. Each warp is represented as the normalized cumulative integral of exp(eta(t)), where eta(t) is a spline with nbasis coefficients. Registration is then carried out curve-by-curve by minimizing either an integrated squared-error criterion (crit = 1) or the first-eigenfunction variance criterion (crit = 2) plus an optional spline roughness penalty (lambda). The outer max_iter loop, when template = NULL, still performs the same Procrustes-style template refinement as the other methods.

Value

tfd vector of (forward) warping functions $h_i(s) = t$ with the same length as x. Apply with tf_align() to obtain registered functions, or use tf_invert() to obtain inverse warps $h_i^{-1}(t) = s$. The returned warps carry an attr(, "template") with the template used (NULL for landmark registration, which has no template).

Important method-specific arguments (passed via ...)

For `method = "srvf"`**:**

`lambda` non-negative number: penalty controlling the flexibility of warpings (default is `0` for unrestricted warps).

`penalty_method` cost function used to penalize warping functions. Defaults to `"roughness"` (norm of their second derivative), `"geodesic"` uses the geodesic distance to the identity and `"norm"` uses Euclidean distance to the identity.

For `method = "cc"`**:**

`nbasis` integer: number of B-spline basis functions for the monotone warp basis (default `6L`, minimum 2).

`lambda` non-negative number: roughness penalty for the warp basis (default `0` for unpenalized warping).

`crit` registration criterion. Defaults to `2` for the first-eigenfunction variance criterion; alternative is `1` for integrated squared error.

`conv` non-negative convergence tolerance for the inner optimizer. Default is `1e-4`.

`iterlim` maximum number of inner optimization iterations per curve. Default is `20L`.

For `method = "affine"`**:**

`type` character: `"shift"` (translation only), `"scale"` (scaling only), or `"shift_scale"` (both). Default is `"shift"`.

`shift_range` numeric(2): bounds for shift parameter. Default is `c(-range/2, range/2)` where range is the domain width. Larger bounds allow greater shifts but may result in more `NA` values.

`scale_range` numeric(2): bounds for scale parameter. Default is `c(0.5, 2)`. Must have `lower > 0`.

For `method = "landmark"`**:**

`landmarks` **(required)** numeric matrix of landmark positions with one row per function and one column per landmark. Use `tf_landmarks_extrema()` to find peaks/valleys automatically.

`template_landmarks` numeric vector of target landmark positions. Default is column-wise mean of `landmarks`.

Author(s)

Maximilian Muecke, Fabian Scheipl, Claude Opus 4.6

See Also

Other registration functions: `tf_align()`, `tf_landmarks_extrema()`, `tf_register()`, `tf_registration`, `tf_warp()`

Examples

```
# see tf_register() for full registration examples
set.seed(1)
f <- tf_rgp(5)
warps <- tf_estimate_warps(f, method = "srvf")
plot(warps)
```

`tf_evaluate` *Evaluate* `tf`*-vectors for given argument values*

Description

Also used internally by the `[`-operator for `tf` data (see `?tfbrackets`) to evaluate `object`, see examples.

Usage

```
tf_evaluate(object, arg, ...)

## Default S3 method:
tf_evaluate(object, arg, ...)

## S3 method for class 'tfd'
tf_evaluate(object, arg, evaluator = tf_evaluator(object), ...)

## S3 method for class 'tfb'
tf_evaluate(object, arg, ...)
```

Arguments

`object`	a `tf`, or a `data.frame`-like object with `tf` columns.
`arg`	optional evaluation grid (vector or list of vectors). Defaults to `tf_arg(object)`, implicitly.
`...`	not used.
`evaluator`	optional. The function to use for inter/extrapolating the `tfd`. Defaults to `tf_evaluator(object)`. See e.g. `tf_approx_linear()` for details.

Value

A list of numeric vectors containing the function evaluations on `arg`.

See Also

Other tidyfun inter/extrapolation functions: `tf_approx_linear()`, `tf_interpolate()`

Examples

```
f <- tf_rgp(3, arg = seq(0, 1, length.out = 11))
tf_evaluate(f) |> str()
tf_evaluate(f, arg = 0.5) |> str()
# equivalent, as matrix:
f[, 0.5]
new_grid <- seq(0, 1, length.out = 6)
tf_evaluate(f, arg = new_grid) |> str()
# equivalent, as matrix:
f[, new_grid]
```

`tf_integrate` *Integrals and anti-derivatives of functional data*

Description

Integrals of `tf`-objects are computed by simple quadrature (trapezoid rule). By default the scalar definite integral $\int_{lower}^{upper} f(s)ds$ is returned (option `definite = TRUE`), alternatively for `definite = FALSE` the *anti-derivative* on `[lower, upper]`, e.g. a `tfd` or `tfb` object representing $F(t) \approx \int_{lower}^{t} f(s)ds$, for $t \in$`[lower, upper]`, is returned.

Usage

```
tf_integrate(f, arg, lower, upper, ...)

## Default S3 method:
tf_integrate(f, arg, lower, upper, ...)

## S3 method for class 'tfd'
tf_integrate(
  f,
  arg = tf_arg(f),
  lower = tf_domain(f)[1],
  upper = tf_domain(f)[2],
  definite = TRUE,
  ...
)

## S3 method for class 'tfb'
tf_integrate(
  f,
  arg = tf_arg(f),
  lower = tf_domain(f)[1],
  upper = tf_domain(f)[2],
  definite = TRUE,
  ...
)
```

Arguments

f	a tf-object
arg	(optional) grid to use for the quadrature.
lower	lower limits of the integration range. For definite = TRUE, this can be a vector of the same length as f.
upper	upper limits of the integration range (but see definite arg / description). For definite = TRUE, this can be a vector of the same length as f.
...	not used
definite	should the definite integral be returned (default) or the antiderivative. See description.

Value

For definite = TRUE, the definite integrals of the functions in f. For definite = FALSE and tf-inputs, a tf object containing their anti-derivatives

See Also

Other tidyfun calculus functions: tf_derive()

Examples

```
arg <- seq(0, 1, length.out = 11)
x <- tfd(rbind(arg, arg^2), arg = arg)
tf_integrate(x)
anti <- tf_integrate(x, definite = FALSE)
tf_arg(anti)
```

---

tf_interpolate *Re-evaluate tf-objects on a new grid of argument values.*

Description

Change the internal representation of a tf-object so that it uses a different grid of argument values (arg). Useful for

- thinning out dense grids to make data smaller
- filling out sparse grids to make derivatives/integrals and locating extrema or zero crossings more accurate (... *if* the interpolation works well ...)
- making irregular functional data into (more) regular data.

For tfd-objects, this is just syntactic sugar for tfd(object, arg = arg). To inter/extrapolate more reliably and avoid NAs, call tf_interpolate with evaluator = tf_approx_fill_extend.
For tfb-objects, this re-evaluates basis functions on the new grid which can speed up subsequent computations if they all use that grid. NB: **To reliably impute very irregular data on a regular, common grid, you'll be better off doing FPCA-based imputation or other model-based approaches in most cases.**

Usage

```
tf_interpolate(object, arg, ...)

## S3 method for class 'tfb'
tf_interpolate(object, arg, ...)

## S3 method for class 'tfd'
tf_interpolate(object, arg, ...)
```

Arguments

object	an object inheriting from tf.
arg	a vector of argument values on which to evaluate the functions in object.
...	additional arguments handed over to tfd or tfb, for the construction of the returned object.

Value

a tfd or tfb object on the new grid given by arg.

See Also

tf_rebase(), which is more general.

Other tidyfun inter/extrapolation functions: tf_approx_linear(), tf_evaluate()

Examples

```
# thinning out a densely observed tfd
dense <- tf_rgp(10, arg = seq(0, 1, length.out = 1001))
less_dense <- tf_interpolate(dense, arg = seq(0, 1, length.out = 101))
dense
less_dense
# filling out sparse data (use a suitable evaluator-function!)
sparse <- tf_rgp(10, arg = seq(0, 5, length.out = 11))
plot(sparse, points = TRUE)
# change evaluator for better interpolation
tfd(sparse, evaluator = tf_approx_spline) |>
  tf_interpolate(arg = seq(0, 5, length.out = 201)) |>
  lines(col = 2, lty = 2)

set.seed(1860)
sparse_irregular <- tf_rgp(5) |>
  tf_sparsify(0.5) |>
  tf_jiggle()
tf_interpolate(sparse_irregular, arg = seq(0, 1, length.out = 51))
```

tf_invert *Invert a* tf *vector*

Description

Computes the functional inverse of each function in the tf vector, such that if $y = f(x)$, then $x = f^{-1}(y)$.

Usage

```
tf_invert(x, ...)
```

Arguments

x	a tf vector.
...	optional arguments for the returned object, see tfd() / tfb()

Value

a tf vector of the inverted functions.

Examples

```
arg <- seq(0, 2, length.out = 50)
x <- tfd(rbind(2 * arg, arg^2), arg = arg)
x_inv <- tf_invert(x)
layout(t(1:2))
plot(x, main = "original functions", ylab = "")
plot(x_inv, main = "inverted functions", ylab = "", points = FALSE)
```

tf_jiggle *Make a* tf *(more) irregular*

Description

Randomly create some irregular functional data from regular ones.

- **jiggle** it by randomly moving around its arg-values inside the intervals defined by its grid neighbors on the original argument grid.
- **sparsify** it by removing (100*dropout)% of the function values

Usage

```
tf_jiggle(f, amount = 0.4, ...)

tf_sparsify(f, dropout = 0.5)
```

Arguments

f	a tfd object.
amount	how far away from original grid points can the jiggled grid points lie, at most (relative to original distance to neighboring grid points). Defaults to at most 40% (0.4) of the original grid distances. Must be lower than 0.5.
...	additional args for the returned tfd in tf_jiggle.
dropout	what proportion of values of f to drop, on average. Defaults to half.

Value

an (irregular) tfd object.

See Also

Other tidyfun RNG functions: tf_rgp()

Examples

```
set.seed(1)
(x <- tf_rgp(2, arg = 21L))
(x_jig <- tf_jiggle(x, amount = 0.2))
(x_sp <- tf_sparsify(x, dropout = 0.3))
c(is_irreg(x_jig), is_irreg(x_sp))
```

---

tf_minmax *Depth-based minimum, maximum and range for* tf *vectors*

Description

By default, min, max, and range compute **pointwise** extremes (the existing behaviour). When a depth argument is supplied, they instead return the most extreme / most central observation according to the chosen depth. For the default "MHI" depth this gives the lowest / highest function in an up-down sense.

Usage

```
## S3 method for class 'tf'
min(..., na.rm = FALSE, depth = NULL)

## S3 method for class 'tf'
max(..., na.rm = FALSE, depth = NULL)

## S3 method for class 'tf'
range(..., na.rm = FALSE, depth = NULL)
```

Arguments

...	tf objects (and na.rm for the pointwise default).
na.rm	logical; passed on to the pointwise summary or used to filter NAs before computing depth.
depth	depth method to use. NULL (default) gives the pointwise min/max/range. Supply a depth name (e.g. "MHI") or a custom depth function for depth-based selection.

Value

a tf object.

See Also

tf_depth(), rank.tf()

Other tidyfun ordering and ranking functions: tf_depth(), tf_order

Examples

```
x <- tf_rgp(5) + 1:5
# pointwise (default):
min(x)
max(x)
# depth-based:
min(x, depth = "MHI")
max(x, depth = "MHI")
```

---

tf_order *Rank, order and sort* tf *vectors*

Description

These methods use tf_depth() to rank, order, and sort functional data. By default they use the modified hypograph index ("MHI") which provides an up-down ordering (lowest to highest). You can also use any of the other depth methods available via tf_depth(), or supply a custom depth function.

Usage

```
rank(
  x,
  na.last = TRUE,
  ties.method = c("average", "first", "last", "random", "max", "min"),
  ...
)

## Default S3 method:
rank(
```

```
  x,
  na.last = TRUE,
  ties.method = c("average", "first", "last", "random", "max", "min"),
  ...
)

## S3 method for class 'tf'
rank(
  x,
  na.last = TRUE,
  ties.method = c("average", "first", "last", "random", "max", "min"),
  depth = "MHI",
  ...
)

## S3 method for class 'tf'
xtfrm(x)

## S3 method for class 'tf'
sort(x, decreasing = FALSE, na.last = NA, depth = "MHI", ...)
```

Arguments

x	a tf vector.
na.last	for handling of NAs; see base::rank() and base::sort().
ties.method	a character string for handling ties; see base::rank().
...	passed to tf_depth() (e.g. arg).
depth	the depth function to use for ranking. One of the depths available via tf_depth() (default: "MHI") or a function that takes a tf vector and returns a numeric vector of depth values.
decreasing	logical. Should the sort be decreasing?

Details

rank assigns ranks based on depth values: lower depth values get lower ranks. For "MHI" this gives an ordering from lowest to highest function. For centrality-based depths ("MBD", "FM", "FSD", "RPD"), the most extreme function gets rank 1 and the most central gets the highest rank.

order returns the permutation which rearranges x into ascending order according to depth.

sort.tf returns the sorted tf vector.

xtfrm.tf returns a numeric vector of MHI depth values, enabling base::order and base::rank to work on tf vectors.

Value

rank: a numeric vector of ranks.
order: an integer vector of indices.
sort.tf: a sorted tf vector.
xtfrm.tf: a numeric vector of depth values.

See Also

tf_depth(), min.tf(), max.tf()

Other tidyfun ordering and ranking functions: tf_depth(), tf_minmax

Examples

```
x <- tf_rgp(5) + 1:5
rank(x)
order(x)
sort(x)
# use a centrality-based depth instead:
rank(x, depth = "MBD")
```

---

tf_rebase *Change (basis) representation of a* tf*-object*

Description

Apply the representation of one tf-object to another; i.e. re-express it in the other's basis, on its grid, etc.
Useful for making different functional data objects compatible so they can be combined, compared or computed with.

Usage

```
tf_rebase(object, basis_from, arg = tf_arg(basis_from), ...)

## S3 method for class 'tfd'
tf_rebase(object, basis_from, arg = tf_arg(basis_from), ...)

## S3 method for class 'tfb'
tf_rebase(object, basis_from, arg = tf_arg(basis_from), ...)
```

Arguments

object	a tf object whose representation should be changed.
basis_from	the tf object with the desired basis, arg, evaluator, etc.
arg	optional new arg values, defaults to those of basis_from.
...	forwarded to the tfb or tfd constructors.

Details

This uses double dispatch (S3) internally, so the methods defined below are themselves generics for methods tf_rebase.tfd.tfd, tf_rebase.tfd.tfb_spline, tf_rebase.tfd.tfb_fpc, tf_rebase.tfb.tfd, tf_rebase.tfb.tfb that dispatch on object_from.

Value

a tf-vector containing the data of object in the same representation as basis_from (potentially modified by the arguments given in ...).

Methods (by class)

- tf_rebase(tfd): re-express a tfd-vector in the same representation as some other tf-vector
- tf_rebase(tfb): re-express a tfb-vector in the same representation as some other tf-vector.

Examples

```
x <- tf_rgp(3)
xb <- tfb(x, k = 8, penalized = FALSE, verbose = FALSE)
tf_rebase(tf_rgp(3), xb)
```

---

tf_register *Register / align a tf vector against a template function*

Description

tf_register() is the high-level entry point for functional data registration. It estimates warping functions, applies them to align the data, and returns a tf_registration result object containing the aligned curves, inverse warping functions (observed to aligned time), and template. Use tf_aligned(), tf_inv_warps(), and tf_template() to extract components.

Usage

```
tf_register(
  x,
  ...,
  template = NULL,
  method = c("srvf", "cc", "affine", "landmark"),
  max_iter = 3L,
  tol = 0.01,
  store_x = TRUE
)
```

Arguments

x	a tf vector of functions to register.
...	additional method-specific arguments passed to backend routines (for example crit for method = "cc"). See tf_estimate_warps() for method-specific argument documentation.
template	an optional tf vector of length 1 to use as the template. If NULL, a default template is computed (method-dependent). Not used for method = "landmark".

method	the registration method to use: • "srvf": Square Root Velocity Framework (elastic registration). • "cc": continuous-criterion registration via a tf-native dense-grid optimizer with monotone spline warps. • "affine": affine (linear) registration. • "landmark": piecewise-linear warps aligning user-specified landmarks.
max_iter	integer: maximum Procrustes-style template refinement iterations. Default 3L.
tol	numeric: convergence tolerance for template refinement. Default 1e-2.
store_x	logical: store original data in the result object? Default TRUE. Set to FALSE to save memory.

Details

For a lower-level interface that returns only warping functions (without performing alignment), see tf_estimate_warps().

Value

A tf_registration object. Access components with tf_aligned(), tf_inv_warps(), tf_template().

Important method-specific arguments (passed via ...)

For method = "srvf":

lambda non-negative number: penalty controlling the flexibility of warpings (default is 0 for unrestricted warps).

penalty_method cost function used to penalize warping functions. Defaults to "roughness" (norm of their second derivative), "geodesic" uses the geodesic distance to the identity and "norm" uses Euclidean distance to the identity.

For method = "cc":

nbasis integer: number of B-spline basis functions for the monotone warp basis (default 6L, minimum 2).

lambda non-negative number: roughness penalty for the warp basis (default 0 for unpenalized warping).

crit registration criterion. Defaults to 2 for the first-eigenfunction variance criterion; alternative is 1 for integrated squared error.

conv non-negative convergence tolerance for the inner optimizer. Default is 1e-4.

iterlim maximum number of inner optimization iterations per curve. Default is 20L.

For method = "affine":

type character: "shift" (translation only), "scale" (scaling only), or "shift_scale" (both). Default is "shift".

shift_range numeric(2): bounds for shift parameter. Default is c(-range/2, range/2) where range is the domain width. Larger bounds allow greater shifts but may result in more NA values.

`scale_range` numeric(2): bounds for scale parameter. Default is `c(0.5, 2)`. Must have `lower > 0`.

For `method = "landmark"`:

`landmarks` **(required)** numeric matrix of landmark positions with one row per function and one column per landmark. Use `tf_landmarks_extrema()` to find peaks/valleys automatically.

`template_landmarks` numeric vector of target landmark positions. Default is column-wise mean of `landmarks`.

Author(s)

Maximilian Muecke, Fabian Scheipl, Claude Opus 4.6

References

Ramsay JO, Hooker G, Graves S (2009). *Functional Data Analysis with R and MATLAB*. Springer, New York. doi:10.1007/9780387981857.

Srivastava A, Wu W, Kurtek S, Klassen E, Marron JS (2011). "Registration of Functional Data Using Fisher-Rao Metric." *arXiv:1103.3817*.

Tucker JD, Wu W, Srivastava A (2013). "Generative models for functional data using phase and amplitude separation." *Computational Statistics & Data Analysis*, **61**, 50–66. doi:10.1016/j.csda.2012.12.001.

See Also

Other registration functions: `tf_align()`, `tf_estimate_warps()`, `tf_landmarks_extrema()`, `tf_registration`, `tf_warp()`

Examples

```
# Elastic registration (SRVF method)
height_female <- subset(growth, gender == "female", select = height, drop = TRUE)
growth_female <- tf_derive(height_female) |> tfd(arg = seq(1.125, 17.8), l = 101)
reg <- tf_register(growth_female)
layout(t(1:3))
plot(growth_female, xlab = "Chronological Age", ylab = "Growth Rate (cm/year)")
plot(tf_inv_warps(reg), xlab = "Chronological Age", ylab = "Biological Age")
plot(tf_aligned(reg), xlab = "Biological Age", ylab = "Growth Rate (cm/year)")

# Affine registration (shift only)
t <- seq(0, 2 * pi, length.out = 101)
x <- tfd(t(sapply(c(-0.3, 0, 0.3), function(s) sin(t + s))), arg = t)
reg <- tf_register(x, method = "affine", type = "shift")
plot(tf_aligned(reg), col = 1:3)

# Landmark registration
peaks <- tf_landmarks_extrema(x, "max")
reg <- tf_register(x, method = "landmark", landmarks = peaks)
plot(tf_aligned(reg), col = 1:3)
```

tf_registration *Registration Result Object*

Description

tf_registration objects store the result of tf_register(), including the aligned (registered) curves, estimated inverse warping functions h_i^{-1} (observed $\to$ aligned time), and the template used. Use accessors tf_aligned(), tf_inv_warps(), and tf_template() to extract components.

Usage

```
tf_aligned(x)

tf_inv_warps(x)

tf_template(x)

## S3 method for class 'tf_registration'
print(x, ...)

## S3 method for class 'tf_registration'
summary(object, ...)

## S3 method for class 'summary.tf_registration'
print(x, ...)

## S3 method for class 'tf_registration'
plot(x, ...)

## S3 method for class 'tf_registration'
x[i]

## S3 method for class 'tf_registration'
length(x)
```

Arguments

x	a tf_registration object
...	additional arguments (currently unused)
object	a tf_registration object
i	index for subsetting (integer, logical, or character)

Value

For tf_registration objects: a list with entries registered (tf-vector of aligned/registered functions from x), inv_warps (inverse warping functions aligning x to the template function), the

template function, the original data x (if store_x = TRUE was used in tf_register()), and the call to tf_register() that created the object. Accessors return the respective component.

Summary diagnostics

summary() computes per-curve diagnostics for assessing registration quality and prints their averages and/or deciles. The printed output contains four sections:

Amplitude variance reduction (only if store_x = TRUE): the proportion of pointwise variance removed by registration, computed as $1 - \bar{V}_{\text{registered}}/\bar{V}_{\text{original}}$ where $\bar{V}$ is the mean (across the domain) of the pointwise variance (across curves). Values near 1 indicate that registration removed most of the original variability; values near 0 indicate little change; negative values indicate that registration *increased* variability (a sign that something went wrong).

Warp deviation from identity (deciles across curves): each curve's inverse warping function h_i^{-1} is compared to the identity via the normalized integral $2/L^2 \int |h_i^{-1}(t) - t|\, dt$, where L is the domain length. The normalizing constant $L^2/2$ is the theoretical upper limit deviation for a monotone, domain-preserving warp that maps all timepoints to the first or last timepoint, so values range from 0 (identity warp, no time deformation) to 1 (maximal crazy warping). Values above ≈ 0.3 may suggest aggressive warping that could warrant inspection.

Warp slopes (deciles of per-curve min and max dh^{-1}/dt): a slope of 1 of the warp corresponds to no local time deformation (identity). Slopes > 1 indicate local time dilation (the warped curve is "stretched" relative to the template), slopes < 1 indicate local time compression, so slopes near 0 or very large slopes indicate extreme local deformation. For affine shift warps, all slopes are exactly 1.

Domain coverage loss (only printed if any loss occurs): the fraction of the original domain range that is lost per curve after alignment, computed as 1 - range(aligned_arg) / range(original_arg). This is only relevant for affine (non-domain-preserving) warps where alignment can shift parts of curves outside the original domain. Domain-preserving methods (srvf, cc, landmark) always have zero domain loss.

Accessors

- tf_aligned(x): extract the registered/aligned curves (tfd vector).
- tf_inv_warps(x): extract the estimated inverse warping functions $h_i^{-1}(t)$ that map observed time to aligned time (tfd vector). Use tf_invert() on the result to obtain forward warps if needed.
- tf_template(x): extract the template function (tf vector of length 1).

Author(s)

Fabian Scheipl, Claude Opus 4.6

See Also

Other registration functions: tf_align(), tf_estimate_warps(), tf_landmarks_extrema(), tf_register(), tf_warp()

Examples

```
reg <- tf_register(pinch[1:5], method = "affine", type = "shift_scale")
reg
summary(reg)
plot(reg)
```

`tf_rgp` *Gaussian Process random generator*

Description

Generates `n` realizations of a zero-mean Gaussian process. The function also accepts user-defined covariance functions (without "nugget" effect, see `cov`), The implemented defaults with `scale` parameter ϕ, `order` o and `nugget` effect variance σ^2 are:

- *squared exponential*: $Cov(x(t), x(t')) = \exp(-(t-t')^2)/\phi) + \sigma^2\delta_t(t')$.
- *Wiener* process: $Cov(x(t), x(t')) = \min(t', t)/\phi + \sigma^2\delta_t(t')$,
- *Matèrn* process: $Cov(x(t), x(t')) = \frac{2^{1-o}}{\Gamma(o)}(\frac{\sqrt{2o}|t-t'|}{\phi})^o \text{Bessel}_o(\frac{\sqrt{2o}|t-t'|}{s}) + \sigma^2\delta_t(t')$
- *Brownian Bridge* process for $t, t' \in [a, b]$: $Cov(x(t), x(t')) = \frac{(b-\max(s,t))(\min(s,t)-a)}{\phi(b-a)} + \sigma^2\delta_t(t')$

Usage

```
tf_rgp(
  n,
  arg = 51L,
  cov = c("squareexp", "wiener", "matern", "brown_bridge"),
  scale = diff(domain)/10,
  nugget = scale/200,
  order = 1.5,
  domain = NULL
)
```

Arguments

`n`	how many realizations to draw.
`arg`	vector of evaluation points (`arg` of the return object). Defaults to (0, 0.02, 0.04, ..., 1). If given as a single **integer** (don't forget the `L`...), creates a regular grid of that length over (0,1). If given as a `n`-long list of vectors, irregular functional data are created.
`cov`	type of covariance function to use. Implemented defaults are `"squareexp"`, `"wiener"`, `"matern"`, see description. Can also be any vectorized function returning $Cov(x(t), x(t'))$ *without nugget effect* for pairs of inputs t and t'.
`scale`	scale parameter (see description). Defaults to the width of the domain divided by 10.

nugget	nugget effect for additional white noise / unstructured variability. Defaults to scale/200 (so: very little white noise).
order	order of the Matèrn covariance (if used, must be >0), defaults to 1.5. The higher, the smoother the process. Evaluation of the covariance function becomes numerically unstable for large (>20) order, use "squareexp".
domain	of the generated functions. If not provided, the range of the supplied arg values.

Value

an tfd-vector of length n.

See Also

Other tidyfun RNG functions: tf_jiggle()

Examples

```
(x1 <- tf_rgp(10, cov = "squareexp", nugget = 0))
tf_rgp(2, arg = list(sort(runif(25)), sort(runif(34))))
```

---

tf_smooth *Simple smoothing of* tf *objects*

Description

Apply running means or medians, lowess or Savitzky-Golay filtering to smooth functional data. This does nothing for tfb-objects, which should be smoothed by using a smaller basis / stronger penalty.

Usage

```
tf_smooth(x, ...)

## S3 method for class 'tfb'
tf_smooth(x, verbose = TRUE, ...)

## S3 method for class 'tfd'
tf_smooth(
  x,
  method = c("lowess", "rollmean", "rollmedian", "savgol"),
  verbose = TRUE,
  ...
)
```

Arguments

x	a tf object containing functional data.
...	arguments for the respective method. See details.
verbose	give lots of diagnostic messages? Defaults to TRUE.
method	one of "lowess" (see stats::lowess()), "rollmean", "rollmedian" (see zoo::rollmean()) or "savgol" (see pracma::savgol()).

Details

tf_smooth.tfd overrides/automatically sets some defaults of the used methods:

- lowess uses a span parameter of f = 0.15 (instead of 0.75) by default.
- rollmean/median use a window size of k = $<$number of grid points$>$/20 (i.e., the nearest odd integer to that) and sets fill= "extend" (i.e., constant extrapolation to replace missing values at the extremes of the domain) by default. Use fill= NA for zoo's default behavior of shortening the smoothed series.
- savgol uses a window size of k = $<$number of grid points$>$/10 (i.e., the nearest odd integer to that).

Value

a smoothed version of the input. For some methods/options, the smoothed functions may be shorter than the original ones (at both ends).

Examples

```
library(zoo)
library(pracma)
f <- tf_sparsify(tf_jiggle(tf_rgp(4, 201, nugget = 0.05)))
f_lowess <- tf_smooth(f, "lowess")
# these methods ignore the distances between arg-values:
f_mean <- tf_smooth(f, "rollmean")
f_median <- tf_smooth(f, "rollmedian", k = 31)
f_sg <- tf_smooth(f, "savgol", fl = 31)
layout(t(1:4))
plot(f, points = FALSE, main = "original")
plot(f_lowess,
  points = FALSE, col = "blue", main = "lowess (default,\n span 0.9 in red)"
)
lines(tf_smooth(f, "lowess", f = 0.9), col = "red", alpha = 0.2)
plot(f_mean,
  points = FALSE, col = "blue", main = "rolling means &\n medians (red)"
)
lines(f_median, col = "red", alpha = 0.2) # note constant extrapolation at both ends!
plot(f, points = FALSE, main = "original and\n savgol (red)")
lines(f_sg, col = "red")
```

tf_split *Split / Combine functional fragments*

Description

tf_split separates each function into a vector of functions defined on a sub-interval of its domain, either with overlap at the cut points or without.

tf_combine joins functional fragments together to create longer (or more densely evaluated) functions.

Usage

```
tf_split(x, splits, include = c("both", "left", "right"))

tf_combine(..., strict = FALSE)
```

Arguments

x	a tf object.
splits	numeric vector containing arg-values at which to split.
include	which of the end points defined by splits to include in each of the resulting split functions. Defaults to "both", other options are "left" or "right". See examples.
...	tf-objects of identical lengths to combine
strict	only combine functions whose argument ranges do not overlap, are given in the correct order & contain no duplicate values at identical arguments? defaults to FALSE. If strict == FALSE, only the first function values at duplicate locations are used, the rest are discarded (with a warning).

Value

for tf_split: a list of tf objects.

for tf_combine: a tfd with the combined subfunctions on the union of the input tf_arg-values

Examples

```
x <- tfd(1:100, arg = 1:100)
tf_split(x, splits = c(20, 80))
tf_split(x, splits = c(20, 80), include = "left")
tf_split(x, splits = c(20, 80), include = "right")
  x <- tf_rgp(5)
  tfs <- tf_split(x, splits = c(.2, .6))
  x2 <- tf_combine(tfs[[1]], tfs[[2]], tfs[[3]])
 # tf_combine(tfs[[1]], tfs[[2]], tfs[[3]], strict = TRUE) # errors out - duplicate values!
  all.equal(x, x2)
  # combine works for different input types:
```

```
tfs2_sparse <- tf_sparsify(tfs[[2]])
tfs3_spline <- tfb(tfs[[3]])
tf_combine(tfs[[1]], tfs2_sparse, tfs3_spline)
# combine(.., strict = F) can be used to coalesce different measurements
# of the same process over different grids:
x1 <- tfd(x, arg = tf_arg(x)[seq(1, 51, by = 2)])
x2 <- tfd(x, arg = tf_arg(x)[seq(2, 50, by = 2)])
tf_combine(x2, x1, strict = FALSE) == x

plot(tf_combine(x2, x1, strict = FALSE))
points(x1, col = "blue", pch = "x")
points(x2, col = "red", pch = "o")
```

`tf_warp` *Elastic Deformation: warp and align* `tf` *vectors*

Description

These functions stretch and/or compress regions of the domain of functional data:

- `tf_warp()` applies warping functions to aligned (registered) functional data to recover the original unregistered curves: $x(s) \to x(h(s)) = x(t)$.
- `tf_align()` applies the *inverse* warping function to unregistered data to obtain aligned (registered) functions: $x(t) \to x(h^{-1}(t)) = x(s)$.

Usage

```
tf_warp(x, warp, ...)

## S3 method for class 'tfd'
tf_warp(x, warp, ..., keep_new_arg = FALSE)

## S3 method for class 'tfb'
tf_warp(x, warp, ...)
```

Arguments

`x`	`tf` vector of functions. For `tf_warp()`, these should be registered/aligned functions and unaligned functions for `tf_align()`.
`warp`	`tf` vector of warping functions used for transformation. See Details.
`...`	additional arguments passed to `tfd()`.
`keep_new_arg`	keep new `arg` values after (un)warping or return `tfd` vector on `arg` values of the input (default `FALSE` is the latter)? See Details.

Details

These functions will work best with functions evaluated on suitably dense and regular grids.

Warping functions $h(s) = t$ are strictly monotone increasing (no time travel backwards or infinite time dilation) with identical domain and co-domain: $h : T \to T$. Their input is the aligned "system" time s, their output is the unaligned "observed" time t.

By default (`keep_new_arg = FALSE`), the `tfd` methods will return function objects re-evaluated on the same grids as the original inputs, which will typically incur some additional interpolation error because (un)warping changes the underlying grids, which are then changed back. Set to `TRUE` to avoid. This option is not available for `tfb`-objects.

Value

- `tf_warp()`: the warped `tf` vector (un-registered functions)
- `tf_align()`: the aligned `tf` vector (registered functions)

Author(s)

Maximilian Muecke, Fabian Scheipl, Claude Opus 4.6

See Also

Other registration functions: `tf_align()`, `tf_estimate_warps()`, `tf_landmarks_extrema()`, `tf_register()`, `tf_registration`

Examples

```
# generate "template" function shape on [0, 1]:
set.seed(1351)
template <- tf_rgp(1, arg = 201L, nugget = 0)
# generate random warping functions (strictly monotone inc., [0, 1] -> [0, 1]):
warp <- {
  tmp <- tf_rgp(5)
  tmp <- exp(tmp - mean(tmp)) # centered at identity warping
  tf_integrate(tmp, definite = FALSE) / tf_integrate(tmp)
}
x <- tf_warp(rep(1, 5) * template, warp)
layout(t(1:3))
plot(template); plot(warp, col = 1:5); plot(x, col = 1:5)
# register the functions:
if (requireNamespace("fdasrvf", quietly = TRUE)) {
  reg <- tf_register(x)
} else {
  reg <- tf_register(x, method = "affine", type = "shift_scale")
}
layout(t(1:3))
plot(x, col = 1:5)
plot(tf_inv_warps(reg), col = 1:5); lines(tf_invert(warp), lty = 3, lwd = 1.5, col = 1:5)
plot(tf_aligned(reg), col = 1:5, points = FALSE); lines(template, lty = 2)
```

`tf_where` *Find out where functional data fulfills certain conditions.*

Description

`tf_where` allows to define a logical expression about the function values and returns the argument values for which that condition is true.
`tf_anywhere` is syntactic sugar for `tf_where` with `return = "any"` to get a logical flag for each function if the condition is `TRUE` *anywhere*, see below.

Usage

```
tf_where(
  f,
  cond,
  return = c("all", "first", "last", "range", "any"),
  arg = tf_arg(f)
)

tf_anywhere(f, cond, arg = tf_arg(f))
```

Arguments

`f`	a `tf` object.
`cond`	a logical expression about `value` (and/or `arg`) that defines a condition about the functions, see examples and details.
`return`	for each entry in `f`, `tf_where` either returns *all* `arg` for which `cond` is true, the *first*, the *last* or their *range* or logical flags whether the functions fulfill the condition *any*where. For `"range"`, note that `cond` may not be true for all `arg` values in this range, though, this is not checked.
`arg`	optional `arg`-values on which to evaluate `f` and check `cond`, defaults to `tf_arg(f)`.

Details

Entries in `f` that do not fulfill `cond` anywhere yield `numeric(0)`.
`cond` is evaluated as a `base::subset()`-statement on a `data.frame` containing a single entry in `f` with columns `arg` and `value`, so most of the usual `dplyr` tricks are available as well, see examples. Any condition evaluates to `NA` on `NA`-entries in `f`.

Value

depends on `return`:

- `return = "any"`, i.e, `anywhere`: a logical vector of the same length as `f`.
- `return = "all"`: a list of vectors of the same length as `f`, with empty vectors for the functions that never fulfill the condition.

- return = "range": a data frame with columns "begin" and "end".
- else, a numeric vector of the same length as f with NA for entries of f that nowhere fulfill the condition.

Examples

```
lin <- 1:4 * tfd(seq(-1, 1, length.out = 11), seq(-1, 1, length.out = 11))
tf_where(lin, value %inr% c(-1, 0.5))
tf_where(lin, value %inr% c(-1, 0.5), "range")
a <- 1
tf_where(lin, value > a, "first")
tf_where(lin, value < a, "last")
tf_where(lin, value > 2, "any")
tf_anywhere(lin, value > 2)

set.seed(4353)
f <- tf_rgp(5, 11)
plot(f, pch = as.character(1:5), points = TRUE)
tf_where(f, value == max(value))
# where is the function increasing/decreasing?
tf_where(f, value > dplyr::lag(value, 1, value[1]))
tf_where(f, value < dplyr::lead(value, 1, tail(value, 1)))
# where are the (interior) extreme points (sign changes of `diff(value)`)?
tf_where(
  f,
  sign(c(diff(value)[1], diff(value))) !=
    sign(c(diff(value), tail(diff(value), 1)))
)
# where in its second half is the function positive?
tf_where(f, arg > 0.5 & value > 0)
# does the function ever exceed?
tf_anywhere(f, value > 1)
```

---

tf_zoom *Functions to zoom in/out on functions*

Description

These are used to redefine or restrict the domain of tf objects.

Usage

```
tf_zoom(f, begin, end, ...)

## S3 method for class 'tfd'
tf_zoom(f, begin = tf_domain(f)[1], end = tf_domain(f)[2], ...)

## S3 method for class 'tfb'
```

```
tf_zoom(f, begin = tf_domain(f)[1], end = tf_domain(f)[2], ...)

## S3 method for class 'tfb_fpc'
tf_zoom(f, begin = tf_domain(f)[1], end = tf_domain(f)[2], ...)
```

Arguments

f	a tf-object.
begin	numeric vector of length 1 or length(f). Defaults to the lower limit of the domain of f.
end	numeric vector of length 1 or length(f). Defaults to the upper limit of the domain of f.
...	not used

Value

an object like f on a new domain (potentially). Note that regular functional data and functions in basis representation will be turned into irregular tfd-objects if begin or end are not scalar.

See Also

Other tidyfun utility functions: in_range(), tf_arg()

Examples

```
x <- tf_rgp(10)
plot(x)
tf_zoom(x, 0.5, 0.9)
tf_zoom(x, 0.5, 0.9) |> lines(col = "red")
tf_zoom(x, seq(0, 0.5, length.out = 10), seq(0.5, 1, length.out = 10)) |>
  lines(col = "blue", lty = 3)
```

---

unique_id *Make syntactically valid unique names*

Description

See above.

Usage

```
unique_id(x)
```

Arguments

x	any input.

Value

x turned into a list.

See Also

Other tidyfun developer tools: ensure_list(), prep_plotting_arg()

Examples

```
unique_id(c("a", "b", "a"))
```

---

vctrs vctrs *methods for* tf *objects*

Description

These functions are the extensions that allow tf vectors to work with vctrs.

Usage

```
## S3 method for class 'tfd_reg.tfd_reg'
vec_cast(x, to, ...)

## S3 method for class 'tfd_reg.tfd_irreg'
vec_cast(x, to, ...)

## S3 method for class 'tfd_reg.tfb_spline'
vec_cast(x, to, ...)

## S3 method for class 'tfd_reg.tfb_fpc'
vec_cast(x, to, ...)

## S3 method for class 'tfd_irreg.tfd_reg'
vec_cast(x, to, ...)

## S3 method for class 'tfd_irreg.tfd_irreg'
vec_cast(x, to, ...)

## S3 method for class 'tfd_irreg.tfb_spline'
vec_cast(x, to, ...)

## S3 method for class 'tfd_irreg.tfb_fpc'
vec_cast(x, to, ...)

## S3 method for class 'tfb_spline.tfb_spline'
vec_cast(x, to, ...)
```

```
## S3 method for class 'tfb_spline.tfb_fpc'
vec_cast(x, to, ...)

## S3 method for class 'tfb_fpc.tfb_spline'
vec_cast(x, to, ...)

## S3 method for class 'tfb_fpc.tfb_fpc'
vec_cast(x, to, ...)

## S3 method for class 'tfb_spline.tfd_reg'
vec_cast(x, to, ...)

## S3 method for class 'tfb_spline.tfd_irreg'
vec_cast(x, to, ...)

## S3 method for class 'tfb_fpc.tfd_reg'
vec_cast(x, to, ...)

## S3 method for class 'tfb_fpc.tfd_irreg'
vec_cast(x, to, ...)

## S3 method for class 'tfd_reg.tfd_reg'
vec_ptype2(x, y, ...)

## S3 method for class 'tfd_reg.tfd_irreg'
vec_ptype2(x, y, ...)

## S3 method for class 'tfd_reg.tfb_spline'
vec_ptype2(x, y, ...)

## S3 method for class 'tfd_reg.tfb_fpc'
vec_ptype2(x, y, ...)

## S3 method for class 'tfd_irreg.tfd_reg'
vec_ptype2(x, y, ...)

## S3 method for class 'tfd_irreg.tfd_irreg'
vec_ptype2(x, y, ...)

## S3 method for class 'tfd_irreg.tfb_spline'
vec_ptype2(x, y, ...)

## S3 method for class 'tfd_irreg.tfb_fpc'
vec_ptype2(x, y, ...)

## S3 method for class 'tfb_spline.tfb_spline'
vec_ptype2(x, y, ...)
```

```
## S3 method for class 'tfb_spline.tfb_fpc'
vec_ptype2(x, y, ...)

## S3 method for class 'tfb_spline.tfd_reg'
vec_ptype2(x, y, ...)

## S3 method for class 'tfb_spline.tfd_irreg'
vec_ptype2(x, y, ...)

## S3 method for class 'tfb_fpc.tfb_spline'
vec_ptype2(x, y, ...)

## S3 method for class 'tfb_fpc.tfb_fpc'
vec_ptype2(x, y, ...)

## S3 method for class 'tfb_fpc.tfd_reg'
vec_ptype2(x, y, ...)

## S3 method for class 'tfb_fpc.tfd_irreg'
vec_ptype2(x, y, ...)
```

Arguments

x	Vectors to cast.
to	Type to cast to. If NULL, x will be returned as is.
...	For vec_cast_common(), vectors to cast. For vec_cast(), vec_cast_default(), and vec_restore(), these dots are only for future extensions and should be empty.
y	vectors to cast.

Details

Notes on `vec_cast`**:** Use `tf_rebase()` to change the representations of `tf`-vectors, these methods are only for internal use – automatic/implicit casting of `tf` objects is tricky because it's hard to determine automatically whether such an operation would lose precision (different bases with different expressivity? different argument grids?), and it's not generally clear which instances of which `tf`-subclasses should be considered the "richer" objects. Rules for casting:

- If the casted object's `domain` would not contain the entire original `domain`, no casting is possible (would lose data).
- Every cast that evaluates (basis) functions on different `arg` values is a *lossy* cast, since it might lose precision (`vctrs::maybe_lossy_cast`).
- As long as the casted object's `domain` contains the entire original `domain`:
  - every `tfd_reg`, `tfd_irreg` or `tfb` can always be cast into an equivalent `tfd_irreg` (which may also change its `evaluator` and `domain`).
  - every `tfd_reg` can always be cast to `tfd_reg` (which may change its `evaluator` and `domain`)

  - every `tfb` can be cast *losslessly* to `tfd` (regular or irregular, note it's lossless only on the *original* `arg`-grid)
- Any cast of a `tfd` into `tfb` is potentially *lossy* (because we don't know how expressive the chosen basis is)
- Only `tfb` with identical bases and domains can be cast into one another *losslessly*

Value

for `vec_cast`: the casted `tf`-vector, for `vec_ptype2`: the common prototype

See Also

`vctrs::vec_cast()`, `vctrs::vec_ptype2()`

Index