Package: squat (via r-universe)

February 17, 2025

Title Statistics for Quaternion Temporal Data

Version 0.3.0

Description An implementation of statistical tools for the analysis of rotation-valued time series and functional data. It relies on pre-existing quaternion data structure provided by the 'Eigen' 'C++' library.

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

LinkingTo Rcpp, RcppArmadillo, RcppEigen, fdacluster

Imports cli, dbscan, dtw, fdacluster, funData, furrr, ggplot2, ggrepel, MFPCA, progressr, purrr, Rcpp, rlang, roahd, scales, tibble, tidyr

Depends R (>= 4.1.0)

Suggests covr, future, gganimate, gghighlight, testthat (>= 3.0.0), vdiffr, withr

Config/testthat/edition 3

URL https://github.com/LMJL-Alea/squat, https://lmjl-alea.github.io/squat/

BugReports https://github.com/LMJL-Alea/squat/issues

NeedsCompilation yes

Author Lise Bellanger [aut], Pierre Drouin [aut], Aymeric Stamm [aut, cre] (<https://orcid.org/0000-0002-8725-3654>), Benjamin Martineau [ctb]

Maintainer Aymeric Stamm <aymeric.stamm@cnrs.fr>

Date/Publication 2024-01-10 15:40:02 UTC

Additional_repositories https://cranhaven.r-universe.dev

Config/pak/sysreqs cmake libfftw3-dev make libicu-dev libssl-dev

Repository https://cranhaven.r-universe.dev

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/squat

RemoteSha e36e878786caa778bad222d07e7d8ec1bf7e3a76

RemoteSubdir squat

Contents

`*.qts` *Operator* `*` *for* `qts` *Objects*

Description

This function implements the pointwise quaternion Hamilton multiplication between two quaternion time series.

Usage

```
## S3 method for class 'qts'
x * rhs
```

Arguments

`x`	An object of class `qts`.
`rhs`	Either an object of class `qts` or a numeric value.

Value

An object of class `qts` storing the multiplication of the two inputs.

Examples

```
vespa64$igp[[1]] * vespa64$igp[[2]]
```

`+.qts` *Operator* `+` *for* `qts` *Objects*

Description

This function implements the pointwise addition between two quaternion time series.

Usage

```
## S3 method for class 'qts'
x + rhs
```

Arguments

x	An object of class qts.
rhs	Either an object of class qts or a numeric value.

Value

An object of class qts storing the addition of the two inputs.

Examples

```
vespa64$igp[[1]] + vespa64$igp[[2]]
```

-.qts *Operator - for* qts *Objects*

Description

This function implements the pointwise subtraction between two quaternion time series.

Usage

```
## S3 method for class 'qts'
x - rhs
```

Arguments

x	An object of class qts.
rhs	Either an object of class qts or a numeric value.

Value

An object of class qts storing the subtraction of the two inputs.

Examples

```
vespa64$igp[[1]] - vespa64$igp[[2]]
```

append *QTS Sample Concatenation*

Description

QTS Sample Concatenation

Usage

```
append(x, ...)

## Default S3 method:
append(x, values, after = length(x), ...)

## S3 method for class 'qts_sample'
append(x, y, ...)
```

Arguments

x	Either a numeric vector or an object of class qts_sample.
...	Extra arguments to be passed on to next methods.
values	to be included in the modified vector.
after	a subscript, after which the values are to be appended.
y	Either a numeric vector or an object of class qts_sample or an object of class qts.

Value

If x is a numeric vector, the output is a numeric vector containing the values in x with the elements of values appended after the specified element of x. If x is of class qts_sample, the output is another object of class qts_sample containing the elements in x and the ones in y appended after the last element of x.

Examples

```
append(vespa64$igp, vespa64$igp[1])
append(vespa64$igp, vespa64$igp[[1]])
```

`autoplot.prcomp_qts` *Plot for* `prcomp_qts` *objects*

Description

This function creates a visualization of the results of the PCA applied on a sample of QTS and returns the corresponding ggplot2::ggplot object which enable further customization of the plot.

Usage

```
## S3 method for class 'prcomp_qts'
autoplot(object, what = "PC1", ...)
```

Arguments

`object`	An object of class `prcomp_qts` as produced by the `prcomp.qts_sample()` method.
`what`	A string specifying what kind of visualization the user wants to perform. Choices are words starting with `PC` and ending with a PC number (in which case the mean QTS is displayed along with its perturbations due to the required PC) or `scores` (in which case individuals are projected on the required plane). Defaults to `PC1`.
`...`	If `what = "PC?"`, the user can specify whether to plot the QTS in the tangent space or in the original space by providing a boolean argument `original_space` which defaults to `TRUE`. If `what = "scores"`, the user can specify the plane onto which the individuals will be projected by providing a length-2 integer vector argument `plane` which defaults to `1:2`.

Value

A ggplot2::ggplot object.

Examples

```
df <- as_qts_sample(vespa64$igp[1:16])
res_pca <- prcomp(df)

# Plot the data points in a PC plane
# And color points according to a categorical variable
p <- ggplot2::autoplot(res_pca, what = "scores")
p + ggplot2::geom_point(ggplot2::aes(color = vespa64$V[1:16]))
```

autoplot.qts *Plot for* qts *objects*

Description

This function creates a visualization of a QTS and returns the corresponding ggplot2::ggplot object which enable further customization of the plot.

Usage

```
## S3 method for class 'qts'
autoplot(object, highlighted_points = NULL, ...)
```

Arguments

object	An object of class qts.
highlighted_points	An integer vector specifying point indices to be highlighted. Defaults to NULL, in which case no point will be highlighted with respect to the others.
...	Further arguments to be passed on to next methods.

Value

A ggplot2::ggplot object.

Examples

```
ggplot2::autoplot(vespa64$igp[[1]])
```

autoplot.qtsclust *Plot for* qtsclust *objects*

Description

This function creates a visualization of the clustering results obtained on a sample of QTS and returns the corresponding ggplot2::ggplot object which enable further customization of the plot.

Usage

```
## S3 method for class 'qtsclust'
autoplot(object, ...)
```

Arguments

object	An object of class qtsclust as produced by kmeans.qts_sample() or hclust.qts_sample().
...	Further arguments to be passed to other methods.

Value

A ggplot2::ggplot object.

Examples

```
out <- kmeans(vespa64$igp[1:10], n_clusters = 2)
ggplot2::autoplot(out)
```

---

autoplot.qts_sample *Plot for* qts_sample *objects*

Description

This function creates a visualization of a sample of QTS and returns the corresponding ggplot2::ggplot object which enable further customization of the plot.

Usage

```
## S3 method for class 'qts_sample'
autoplot(
  object,
  memberships = NULL,
  highlighted = NULL,
  with_animation = FALSE,
  ...
)
```

Arguments

object	An object of class qts_sample.
memberships	A vector coercible as factor specifying a group membership for each QTS in the sample. Defaults to NULL, in which case no grouping structure is displayed.
highlighted	A boolean vector specifying whether each QTS in the sample should be hightlighted. Defaults to NULL, in which case no QTS is hightlighted w.r.t. the others.
with_animation	A boolean value specifying whether to create a an animated plot or a static ggplot2::ggplot object. Defaults to FALSE which will create a static plot.
...	Further arguments to be passed to methods.

Value

A ggplot2::ggplot object.

Examples

```
ggplot2::autoplot(vespa64$igp)
```

`centring` *QTS Centering and Standardization*

Description

This function operates a centering of the QTS around the geometric mean of its quaternions. This is effectively achieved by left-multiplying each quaternion by the inverse of their geometric mean.

Usage

```
centring(x, standardize = FALSE, keep_summary_stats = FALSE)
```

Arguments

`x`	An object of class `qts`.
`standardize`	A boolean specifying whether to standardize the QTS in addition to centering it. Defaults to `FALSE`.
`keep_summary_stats`	A boolean specifying whether the mean and standard deviation used for standardizing the data should be stored in the output object. Defaults to `FALSE` in which case only the centered `qts` is returned.

Value

If `keep_summary_stats = FALSE`, an object of class `qts` in which quaternions have been centered (and possibly standardized) around their geometric mean. If `keep_summary_stats = TRUE`, a list with three components:

- `qts`: an object of class `qts` in which quaternions have been centered (and possibly standardized) around their geometric mean;
- `mean`: a numeric vector with the quaternion Fréchet mean;
- `sd`: a numeric value with the quaternion Fréchet standard deviation.

Examples

```
centring(vespa64$igp[[1]])
```

dbscan *QTS Nearest-Neighbor Clustering*

Description

This function massages the input quaternion time series to apply DBSCAN clustering on them, with the possibility of separating amplitude and phase variability and of choosing the source of variability through which clusters should be searched.

Usage

```
dbscan(x, ...)

## Default S3 method:
dbscan(x, eps, minPts = 5, weights = NULL, borderPoints = TRUE, ...)

## S3 method for class 'qts_sample'
dbscan(
  x,
  warping_class = c("affine", "dilation", "none", "shift", "srsf"),
  centroid_type = "mean",
  metric = c("l2", "pearson"),
  cluster_on_phase = FALSE,
  use_fence = FALSE,
  ...
)
```

Arguments

x	Either a numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns) or an object of class qts_sample.
...	additional arguments are passed on to the fixed-radius nearest neighbor search algorithm. See `frNN()` for details on how to control the search strategy.
eps	size (radius) of the epsilon neighborhood. Can be omitted if x is a frNN object.
minPts	number of minimum points required in the eps neighborhood for core points (including the point itself).
weights	numeric; weights for the data points. Only needed to perform weighted clustering.
borderPoints	logical; should border points be assigned to clusters. The default is `TRUE` for regular DBSCAN. If `FALSE` then border points are considered noise (see DBSCAN* in Campello et al, 2013).
warping_class	A string specifying the warping class Choices are `"affine"`, `"dilation"`, `"none"`, `"shift"` or `"srsf"`. Defaults to `"affine"`. The SRSF class is the only class which is boundary-preserving.

centroid_type A string specifying the type of centroid to compute. Choices are "mean", "median" "medoid", "lowess" or "poly". Defaults to "mean". If LOWESS appproximation is chosen, the user can append an integer between 0 and 100 as in "lowess20". This number will be used as the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. The default value is 10%. If polynomial approximation is chosen, the user can append an positive integer as in "poly3". This number will be used as the degree of the polynomial model. The default value is 4L.

metric A character string specifying the distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski" if x is not a QTS sample. Otherwise, it must be one of "l2", "pearson" or "dtw".

cluster_on_phase A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to FALSE which implies amplitude variation.

use_fence A boolean specifying whether the fence algorithm should be used to robustify the algorithm against outliers. Defaults to FALSE. This is used only when warping_class != "srsf".

Value

An object of class stats::kmeans or stats::hclust or dbscan_fast if the input x is NOT of class qts_sample. Otherwise, an object of class qtsclust which is effectively a list with four components:

- qts_aligned: An object of class qts_sample storing the sample of aligned QTS;
- qts_centers: A list of objects of class qts representing the centers of the clusters;
- best_clustering: An object of class fdacluster::caps storing the results of the best k-mean alignment result among all initialization that were tried.
- call_name: A string storing the name of the function that was used to produce the clustering structure;
- call_args: A list containing the exact arguments that were passed to the function call_name that produced this output.

Examples

```
out <- dbscan(vespa64$igp[1:10])
plot(out)
```

differentiate *QTS Differentiation*

Description

This function computes the first derivative of quaternion time series with respect to time.

Usage

```
differentiate(x)

## S3 method for class 'qts'
differentiate(x)

## S3 method for class 'qts_sample'
differentiate(x)
```

Arguments

x	An object of class qts or qts_sample.

Value

An object of the same class as the input argument x in which quaternions measure the rotation to be applied to transform attitude at previous time point to attitude at current time point.

Examples

```
differentiate(vespa64$igp[[1]])
differentiate(vespa64$igp)
```

dist *QTS Distance Matrix Computation*

Description

This function massages an input sample of quaternion time series to turn it into a pairwise distance matrix.

Usage

```
dist(x, metric, ...)

## Default S3 method:
dist(
  x,
 metric = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  diag = FALSE,
  upper = FALSE,
  p = 2,
  ...
)

## S3 method for class 'qts_sample'
dist(
  x,
  metric = c("l2", "pearson", "dtw"),
  warping_class = c("affine", "dilation", "none", "shift", "srsf"),
  cluster_on_phase = FALSE,
  labels = NULL,
  ...
)
```

Arguments

x	A numeric matrix, data frame, stats::dist object or object of class qts_sample specifying the sample on which to compute the pairwise distance matrix.
metric	A character string specifying the distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski" if x is not a QTS sample. Otherwise, it must be one of "l2", "pearson" or "dtw".
...	not used.
diag	logical value indicating whether the diagonal of the distance matrix should be printed by print.dist.
upper	logical value indicating whether the upper triangle of the distance matrix should be printed by print.dist.
p	The power of the Minkowski distance.
warping_class	A string specifying the warping class Choices are "affine", "dilation", "none", "shift" or "srsf". Defaults to "affine". The SRSF class is the only class which is boundary-preserving.
cluster_on_phase	A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to FALSE which implies amplitude variation.
labels	A character vector specifying curve labels. Defaults to NULL which uses sequential numbers as labels.

Value

An object of class stats::dist.

Examples

```
D <- dist(vespa64$igp[1:5])
```

DTW *Dynamic Time Warping for Quaternion Time Series*

Description

This function evaluates the Dynamic Time Warping (DTW) distance between two quaternion time series (QTS).

Usage

```
DTW(
  qts1,
  qts2,
  resample = TRUE,
  disable_normalization = FALSE,
  distance_only = FALSE,
  step_pattern = dtw::symmetric2
)
```

Arguments

qts1	An object of class qts.
qts2	An object of class qts.
resample	A boolean specifying whether the QTS should be uniformly resampled on their domain before computing distances. Defaults to TRUE.
disable_normalization	A boolean specifying whether quaternion normalization should be disabled. Defaults to FALSE which ensures that we always deal with unit quaternions.
distance_only	A boolean specifying whether to only compute distance (no backtrack, faster). Defaults to FALSE.
step_pattern	A dtw::stepPattern specifying the local constraints on the warping path. Defaults to dtw::symmetric2 which uses symmetric and normalizable warping paths with no local slope constraints. See dtw::stepPattern for more information.

Details

If no evaluation grid is provided, the function assumes that the two input QTS are evaluated on the same grid.

Value

An object of class dtw::dtw storing the dynamic time warping results.

Examples

```
DTW(vespa64$igp[[1]], vespa64$igp[[2]])
```

`exp` *QTS Exponential*

Description

This function computes the exponential of quaternion time series as the time series of the quaternion exponentials.

Usage

```
## S3 method for class 'qts'
exp(x, ...)

## S3 method for class 'qts_sample'
exp(x, ...)
```

Arguments

`x`	An object of class qts or qts_sample.
`...`	Extra arguments to be passed on to next methods.

Value

An object of the same class as the input argument `x` in which quaternions have been replaced by their exponential.

Examples

```
x <- log(vespa64$igp[[1]])
exp(x)
y <- log(vespa64$igp)
exp(y)
```

hclust *QTS Hierarchical Agglomerative Clustering*

Description

This function massages the input quaternion time series to apply hierarchical agglomerative clustering on them, with the possibility of separating amplitude and phase variability and of choosing the source of variability through which clusters should be searched.

Usage

```
hclust(x, metric, linkage_criterion, ...)

## Default S3 method:
hclust(
  x,
 metric = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  linkage_criterion = c("complete", "average", "single", "ward.D2"),
  ...
)

## S3 method for class 'qts_sample'
hclust(
  x,
  metric = c("l2", "pearson"),
  linkage_criterion = c("complete", "average", "single", "ward.D2"),
  n_clusters = 1L,
  warping_class = c("affine", "dilation", "none", "shift", "srsf"),
  centroid_type = "mean",
  cluster_on_phase = FALSE,
  ...
)
```

Arguments

x	Either a numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns) or an object of class qts_sample.
metric	A character string specifying the distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski" if x is not a QTS sample. Otherwise, it must be one of "l2", "pearson" or "dtw".
linkage_criterion	A string specifying which linkage criterion should be used to compute distances between sets of curves. Choices are "complete" for complete linkage, "average" for average linkage and "single" for single linkage. See stats::hclust() for more details. Defaults to "complete".

...	Further graphical arguments. E.g., cex controls the size of the labels (if plotted) in the same way as text.
n_clusters	An integer value specifying the number of clusters. Defaults to 1L.
warping_class	A string specifying the warping class Choices are "affine", "dilation", "none", "shift" or "srsf". Defaults to "affine". The SRSF class is the only class which is boundary-preserving.
centroid_type	A string specifying the type of centroid to compute. Choices are "mean", "median" "medoid", "lowess" or "poly". Defaults to "mean". If LOWESS appproximation is chosen, the user can append an integer between 0 and 100 as in "lowess20". This number will be used as the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. The default value is 10%. If polynomial approximation is chosen, the user can append an positive integer as in "poly3". This number will be used as the degree of the polynomial model. The default value is 4L.
cluster_on_phase	A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to FALSE which implies amplitude variation.

Value

An object of class stats::kmeans or stats::hclust or dbscan_fast if the input x is NOT of class qts_sample. Otherwise, an object of class qtsclust which is effectively a list with four components:

- qts_aligned: An object of class qts_sample storing the sample of aligned QTS;
- qts_centers: A list of objects of class qts representing the centers of the clusters;
- best_clustering: An object of class fdacluster::caps storing the results of the best k-mean alignment result among all initialization that were tried.
- call_name: A string storing the name of the function that was used to produce the clustering structure;
- call_args: A list containing the exact arguments that were passed to the function call_name that produced this output.

Examples

```
out <- hclust(vespa64$igp[1:10], n_clusters = 2)
plot(out)
```

---

hemispherize *QTS Hemispherization*

Description

This function ensures that there are no discontinuities in QTS due to quaternion flips since two unit quaternions q and -q encode the same rotation.

Usage

```
hemispherize(x)

## S3 method for class 'qts'
hemispherize(x)

## S3 method for class 'qts_sample'
hemispherize(x)
```

Arguments

x	An object of class qts or qts_sample.

Value

An object of the same class as the input argument x with no quaternion flip discontinuities.

Examples

```
hemispherize(vespa64$igp[[1]])
hemispherize(vespa64$igp)
```

---

inverse_qts *Inverse Operator for* qts *Objects*

Description

This function implements the pointwise inverse of a quaternion time series.

Usage

```
inverse_qts(x)
```

Arguments

x	An object of class qts.

Value

An object of class qts storing the inverse of x.

Examples

```
inverse_qts(vespa64$igp[[1]])
```

kmeans *QTS K-Means Alignment Algorithm*

Description

This function massages the input quaternion time series to feed them into the k-means alignment algorithm for jointly clustering and aligning the input QTS.

Usage

```
kmeans(x, n_clusters, ...)

## Default S3 method:
kmeans(
  x,
  n_clusters = 1,
  iter_max = 10,
  nstart = 1,
  algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"),
  trace = FALSE,
  ...
)

## S3 method for class 'qts_sample'
kmeans(
  x,
  n_clusters = 1L,
  seeds = NULL,
 seeding_strategy = c("kmeans++", "exhaustive-kmeans++", "exhaustive", "hclust"),
  warping_class = c("affine", "dilation", "none", "shift", "srsf"),
  centroid_type = "mean",
  metric = c("l2", "pearson"),
  cluster_on_phase = FALSE,
  use_fence = FALSE,
  ...
)
```

Arguments

x	Either a numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns) or an object of class qts_sample.
n_clusters	An integer value specifying the number of clusters to be look for.
...	not used.
iter_max	An integer value specifying the maximum number of iterations for terminating the k-mean algorithm. Defaults to 10L.

nstart — if centers is a number, how many random sets should be chosen?

algorithm — character: may be abbreviated. Note that "Lloyd" and "Forgy" are alternative names for one algorithm.

trace — logical or integer number, currently only used in the default method ("Hartigan-Wong"): if positive (or true), tracing information on the progress of the algorithm is produced. Higher values may produce more tracing information.

seeds — An integer value or vector specifying the indices of the initial centroids. If an integer vector, it is interpreted as the indices of the intial centroids and should therefore be of length n_clusters. If an integer value, it is interpreted as the index of the first initial centroid and subsequent centroids are chosen according to the k-means++ strategy. It can be NULL in which case the argument seeding_strategy is used to automatically provide suitable indices. Defaults to NULL.

seeding_strategy — A character string specifying the strategy for choosing the initial centroids in case the argument seeds is set to NULL. Choices are "kmeans++", "exhaustive-kmeans++" which performs an exhaustive search over the choice of the first centroid, "exhaustive" which tries on all combinations of initial centroids or "hclust" which first performs hierarchical clustering using Ward's linkage criterion to identify initial centroids. Defaults to "kmeans++", which is the fastest strategy.

warping_class — A string specifying the warping class Choices are "affine", "dilation", "none", "shift" or "srsf". Defaults to "affine". The SRSF class is the only class which is boundary-preserving.

centroid_type — A string specifying the type of centroid to compute. Choices are "mean", "median" "medoid", "lowess" or "poly". Defaults to "mean". If LOWESS appproximation is chosen, the user can append an integer between 0 and 100 as in "lowess20". This number will be used as the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. The default value is 10%. If polynomial approximation is chosen, the user can append an positive integer as in "poly3". This number will be used as the degree of the polynomial model. The default value is 4L.

metric — A string specifying the metric used to compare curves. Choices are "l2" or "pearson". Defaults to "l2". Used only when warping_class != "srsf". For the boundary-preserving warping class, the L2 distance between the SRSFs of the original curves is used.

cluster_on_phase — A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to FALSE which implies amplitude variation.

use_fence — A boolean specifying whether the fence algorithm should be used to robustify the algorithm against outliers. Defaults to FALSE. This is used only when warping_class != "srsf".

Value

An object of class stats::kmeans or stats::hclust or dbscan_fast if the input x is NOT of class qts_sample. Otherwise, an object of class qtsclust which is effectively a list with four components:

- qts_aligned: An object of class qts_sample storing the sample of aligned QTS;
- qts_centers: A list of objects of class qts representing the centers of the clusters;
- best_clustering: An object of class fdacluster::caps storing the results of the best k-mean alignment result among all initialization that were tried.
- call_name: A string storing the name of the function that was used to produce the clustering structure;
- call_args: A list containing the exact arguments that were passed to the function call_name that produced this output.

Examples

```
out <- kmeans(vespa64$igp[1:10], n_clusters = 2)
```

---

log *QTS Logarithm*

Description

This function computes the logarithm of quaternion time series as the time series of the quaternion logarithms.

Usage

```
## S3 method for class 'qts'
log(x, ...)

## S3 method for class 'qts_sample'
log(x, ...)
```

Arguments

x	An object of class qts or qts_sample.
...	Extra arguments to be passed on to next methods.

Value

An object of the same class as the input argument x in which quaternions have been replaced by their logarithm.

Examples

```
log(vespa64$igp[[1]])
log(vespa64$igp)
```

mean.qts_sample *QTS Geometric Mean*

Description

This function computes the pointwise geometric mean of a QTS sample.

Usage

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

Arguments

x	An object of class qts_sample.
...	Further arguments passed to or from other methods.

Value

An object of class qts in which quaternions are the pointwise geometric mean of the input QTS sample.

Examples

```
mean(vespa64$igp)
```

median.qts_sample *QTS Geometric Median*

Description

This function computes the pointwise geometric median of a QTS sample.

Usage

```
## S3 method for class 'qts_sample'
median(x, na.rm = FALSE, ...)
```

Arguments

x	An object of class qts_sample.
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
...	Further arguments passed to or from other methods.

Value

An object of class qts in which quaternions are the pointwise geometric median of the input QTS sample.

Examples

```
median(vespa64$igp)
```

`moving_average` *QTS Moving Average*

Description

This function performs QTS smoothing via moving average.

Usage

```
moving_average(x, window_size = 0)

## S3 method for class 'qts'
moving_average(x, window_size = 0)

## S3 method for class 'qts_sample'
moving_average(x, window_size = 0)
```

Arguments

`x`	An object of class qts or qts_sample.
`window_size`	An integer value specifying the size of the sliding window used to compute the median value. Defaults to `0L`.

Value

An object of the same class as the input argument `x` storing the smoothed QTS.

Examples

```
moving_average(vespa64$igp[[1]], window_size = 5)
moving_average(vespa64$igp, window_size = 5)
```

normalize *QTS Normalization*

Description

This function ensures that all quaternions in the time series are unit quaternions.

Usage

```
normalize(x)

## S3 method for class 'qts'
normalize(x)

## S3 method for class 'qts_sample'
normalize(x)
```

Arguments

x	An object of class qts or qts_sample.

Value

An object of the same class as the input argument x in which quaternions are unit quaternions.

Examples

```
normalize(vespa64$igp[[1]])
normalize(vespa64$igp)
```

plot.prcomp_qts *Plot for* prcomp_qts *objects*

Description

This function creates a visualization of the results of the PCA applied on a sample of QTS **without** returning the plot data as an object.

Usage

```
## S3 method for class 'prcomp_qts'
plot(x, what = "PC1", ...)

## S3 method for class 'prcomp_qts'
screeplot(x, ...)
```

Arguments

x	An object of class prcomp_qts as produced by the prcomp.qts_sample() method.
what	A string specifying what kind of visualization the user wants to perform. Choices are words starting with PC and ending with a PC number (in which case the mean QTS is displayed along with its perturbations due to the required PC) or scores (in which case individuals are projected on the required plane). Defaults to PC1.
...	If what = "PC?", the user can specify whether to plot the QTS in the tangent space or in the original space by providing a boolean argument original_space which defaults to TRUE. If what = "scores", the user can specify the plane onto which the individuals will be projected by providing a length-2 integer vector argument plane which defaults to 1:2.

Value

No return value, called for side effects.

Examples

```
df <- as_qts_sample(vespa64$igp[1:16])
res_pca <- prcomp(df)

# You can plot the effect of a PC on the mean
plot(res_pca, what = "PC1")

# You can plot the data points in a PC plane
plot(res_pca, what = "scores")
```

---

plot.qts *Plot for qts objects*

Description

This function creates a visualization of a QTS **without** returning the plot data as an object.

Usage

```
## S3 method for class 'qts'
plot(x, highlighted_points = NULL, ...)
```

Arguments

x	An object of class qts.
highlighted_points	An integer vector specifying point indices to be highlighted. Defaults to NULL, in which case no point will be highlighted with respect to the others.
...	Further arguments to be passed on to next methods.

Value

No return value, called for side effects.

Examples

```
plot(vespa64$igp[[1]])
```

`plot.qtsclust` *Plot for* `qtsclust` *objects*

Description

This function creates a visualization of the clustering results obtained on a sample of QTS **without** returning the plot data as an object.

Usage

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

Arguments

`x`	An object of class `qtsclust` as produced by `kmeans.qts_sample()` or `hclust.qts_sample()`.
`...`	Further arguments to be passed to other methods.

Value

No return value, called for side effects.

Examples

```
out <- kmeans(vespa64$igp[1:10], n_clusters = 2)
plot(out)
```

plot.qts_sample *Plot for* qts_sample *objects*

Description

This function creates a visualization of a sample of QTS **without** returning the corresponding ggplot2::ggplot object

Usage

```
## S3 method for class 'qts_sample'
plot(x, memberships = NULL, highlighted = NULL, with_animation = FALSE, ...)
```

Arguments

x	An object of class qts_sample.
memberships	A vector coercible as factor specifying a group membership for each QTS in the sample. Defaults to NULL, in which case no grouping structure is displayed.
highlighted	A boolean vector specifying whether each QTS in the sample should be hightlighted. Defaults to NULL, in which case no QTS is hightlighted w.r.t. the others.
with_animation	A boolean value specifying whether to create a an animated plot or a static ggplot2::ggplot object. Defaults to FALSE which will create a static plot.
...	Further arguments to be passed to methods.

Value

No return value, called for side effects.

Examples

```
plot(vespa64$igp)
```

prcomp.qts_sample *PCA for QTS Sample*

Description

This is the S3 specialization of the function stats::prcomp() for QTS samples.

Usage

```
## S3 method for class 'qts_sample'
prcomp(x, M = 5, fit = FALSE, ...)
```

Arguments

x	An object of class qts_sample.
M	An integer value specifying the number of principal component to compute. Defaults to 5L.
fit	A boolean specifying whether the resulting prcomp_qts object should store a reconstruction of the sample from the retained PCs. Defaults to FALSE.
...	Arguments passed to or from other methods.

Details

The mean_qts component of the resulting object is the QTS used for centering. It it part of the prcomp_qts object because it is needed to reconstruct the sample from the retained PCs. The prcomp_qts object also contains the total variance of the sample and the percentage of variance explained by each PC.

Value

An object of class prcomp_qts which is a list with the following components:

- tpca: An object of class MFPCAfit as produced by the function MFPCA::MFPCA(),
- var_props: A numeric vector storing the percentage of variance explained by each PC,
- total_variance: A numeric value storing the total variance of the sample,
- mean_qts: An object of class qts containing the mean QTS (used for centering the QTS sample before projecting it to the tangent space),
- principal_qts: A list of qtss containing the required principal components.

Examples

```
res_pca <- prcomp(vespa64$igp[1:16])
```

---

predict.prcomp_qts *Predict QTS from PCA decomposition*

Description

This function predicts the QTS of a new sample from the PCA decomposition of a previous sample.

Usage

```
## S3 method for class 'prcomp_qts'
predict(object, newdata, ...)
```

Arguments

object	An object of class prcomp_qts as produced by the prcomp.qts_sample() method.
newdata	An object of class qts or qts_sample specifying a QTS or a sample of QTS. The QTS should be evaluated on the same grid as the one used to fit the PCA model. If the evaluation grids map the same domain but with different sampling frequenciesa, the QTS will be linearly interpolated (in the Lie algebra) to the common grid used to fit the PCA model.
...	Additional arguments. Not used here.

Value

An object of class qts_sample containing the predicted QTS.

Examples

```
# Fit PCA model
pr <- prcomp(vespa64$igp, M = 5)

# Predict QTS
new_qts <- predict(pr)
```

---

qts *QTS Class*

Description

A collection of functions that implements the QTS class. It currently provides the as_qts() function for QTS coercion of tibble::tibbles and the is_qts() function for checking if an object is a QTS.

Usage

```
as_qts(x)

is_qts(x)

## S3 method for class 'qts'
format(x, digits = 5, ...)
```

Arguments

x	A tibble::tibble with columns time, w, x, y and z.
digits	An integer value specifying the number of digits to keep for printing. Defaults to 5L.
...	Further arguments passed to or from other methods.

Details

A quaternion time series (QTS) is stored as a `tibble::tibble` with 5 columns:

- `time`: A first column specifying the time points at which quaternions were collected;
- `w`: A second column specifying the first coordinate of the collected quaternions;
- `x`: A third column specifying the second coordinate of the collected quaternions;
- `y`: A fourth column specifying the third coordinate of the collected quaternions;
- `z`: A fifth column specifying the fourth coordinate of the collected quaternions.

Value

An object of class qts.

Examples

```
qts1 <- vespa64$igp[[1]]
qts2 <- as_qts(qts1)
is_qts(qts1)
is_qts(qts2)
```

`qts2aats` *QTS Transformation to Angle-Axis Time Series*

Description

This function converts a quaternion time series into its angle-axis representation.

Usage

```
qts2aats(x)
```

Arguments

`x`	An object of class qts.

Value

A time series stored as a tibble::tibble with columns `time`, `angle`, `ux`, `uy` and `uz` containing the angle-axis representation of the input quaternions.

Examples

```
qts2aats(vespa64$igp[[1]])
```

qts2ats *QTS Transformation To Angle Time Series*

Description

This function computes a univariate time series representing the angle between the first and other attitudes.

Usage

```
qts2ats(x, disable_normalization = FALSE)
```

Arguments

x	An object of class qts.
disable_normalization	A boolean specifying whether quaternion normalization should be disabled. Defaults to FALSE.

Value

A time series stored as a tibble::tibble with columns time and angle in which angle measures the angle between the current rotation and the first one.

Examples

```
qts2ats(vespa64$igp[[1]])
```

qts2avts *QTS Transformation to Angular Velocity Time Series*

Description

This function projects a quaternion time series into the space of angular velocities.

Usage

```
qts2avts(x, body_frame = FALSE)
```

Arguments

x	An object of class qts.
body_frame	A boolean specifying whether the fixed frame with respect to which coordinates of the angular velocity should be computed is the body frame or the global frame. Defaults to FALSE.

Value

A time series stored as a tibble::tibble with columns `time`, `x`, `y` and `z` containing the angular velocity at each time point.

Examples

```
qts2avts(vespa64$igp[[1]])
```

`qts2dts` *QTS Transformation To Distance Time Series*

Description

This function computes a real-valued time series reporting the pointwise geodesic distance between the two input QTS at each time point.

Usage

```
qts2dts(x, y)
```

Arguments

`x`	An object of class qts.
`y`	An object of class qts.

Details

The function currently expects that the two input QTS are evaluated on the same time grid.

Value

A time series stored as a tibble::tibble with columns `time` and `distance` in which `distance` measures the angular distance between the quaternions of both input QTS at a given time point.

Examples

```
qts2dts(vespa64$igp[[1]], vespa64$igp[[2]])
```

qts2nts *QTS Transformation To Norm Time Series*

Description

This function computes a univariate time series representing the norm of the quaternions.

Usage

```
qts2nts(x, disable_normalization = FALSE)
```

Arguments

x	An object of class qts.
disable_normalization	A boolean specifying whether quaternion normalization should be disabled. Defaults to FALSE.

Value

A time series stored as a tibble::tibble with columns time and norm in which norm measures the angular distance between the current quaternion and the identity.

Examples

```
qts2nts(vespa64$igp[[1]])
```

qts_sample *QTS Sample Class*

Description

A collection of functions that implements the QTS sample class. It currently provides the as_qts_sample() function for QTS sample coercion of lists of qts objects, the is_qts_sample() function for checking if an object is a QTS sample and the subset operator.

Usage

```
as_qts_sample(x)

is_qts_sample(x)

## S3 method for class 'qts_sample'
x[i, simplify = FALSE]
```

Arguments

x	A list of tibble::tibbles, each of which with columns time, w, x, y and z.
i	A valid expression to subset observations from a QTS sample.
simplify	A boolean value specifying whether the resulting subset should be turned into a single QTS in case the subset is of size 1. Defaults to FALSE.

Details

A QTS sample is a collection of quaternion time series (QTS), each of which is stored as a tibble::tibble with 5 columns:

- time: A first column specifying the time points at which quaternions were collected;
- w: A second column specifying the first coordinate of the collected quaternions;
- x: A third column specifying the second coordinate of the collected quaternions;
- y: A fourth column specifying the third coordinate of the collected quaternions;
- z: A fifth column specifying the fourth coordinate of the collected quaternions.

Value

An object of class qts_sample.

Examples

```
x <- vespa64$igp
y <- as_qts_sample(x)
is_qts_sample(x)
is_qts_sample(y)
x[1]
x[1, simplify = TRUE]
```

---

reorient *QTS Reorientation*

Description

This function reorients the quaternions in a QTS for representing attitude with respect to the orientation of the sensor at the first time point.

Usage

```
reorient(x, disable_normalization = FALSE)

## S3 method for class 'qts'
reorient(x, disable_normalization = FALSE)

## S3 method for class 'qts_sample'
reorient(x, disable_normalization = FALSE)
```

Arguments

x	An object of class qts or qts_sample.
disable_normalization	A boolean specifying whether quaternion normalization should be disabled. Defaults to FALSE.

Value

An object of the same class as the input argument x in which quaternions measure attitude with respect to the orientation of the sensor at the first time point.

Examples

```
reorient(vespa64$igp[[1]])
reorient(vespa64$igp)
```

---

resample *QTS Resampling*

Description

This function performs uniform resampling using SLERP.

Usage

```
resample(x, tmin = NA, tmax = NA, nout = 0L, disable_normalization = FALSE)

## S3 method for class 'qts'
resample(x, tmin = NA, tmax = NA, nout = 0L, disable_normalization = FALSE)

## S3 method for class 'qts_sample'
resample(x, tmin = NA, tmax = NA, nout = 0L, disable_normalization = FALSE)
```

Arguments

x	An object of class qts or qts_sample.
tmin	A numeric value specifying the lower bound of the time interval over which uniform resampling should take place. It must satisfy tmin >= min(qts$time). Defaults to NA in which case it is set to min(qts$time).
tmax	A numeric value specifying the upper bound of the time interval over which uniform resampling should take place. It must satisfy tmax <= max(qts$time). Defaults to NA in which case it is set to max(qts$time).
nout	An integer specifying the size of the uniform grid for time resampling. Defaults to 0L in which case it uses the same grid size as the input QTS.
disable_normalization	A boolean specifying whether quaternion normalization should be disabled. Defaults to FALSE in which case the function makes sure that quaternions are normalized prior to performing SLERP interpolation.

Value

An object of the same class as the input argument `x` in which quaternions are uniformly sampled in the range `[tmin, tmax]`.

Examples

```
resample(vespa64$igp[[1]])
resample(vespa64$igp)
```

`rnorm_qts` *QTS Random Sampling*

Description

This function adds uncorrelated Gaussian noise to the logarithm QTS using an exponential covariance function.

Usage

```
rnorm_qts(n, mean_qts, alpha = 0.01, beta = 0.001)
```

Arguments

`n`	An integer specifying how many QTS should be generated.
`mean_qts`	An object of class `qts` specifying the mean QTS.
`alpha`	A positive scalar specifying the variance of each component of the log-QTS. Defaults to `0.01`.
`beta`	A positive scalar specifying the exponential weight. Defaults to `0.001`.

Details

See `exp_cov_function` for details about the roles of `alpha` and `beta` in the definition of the covariance operator.

Value

A list of `n` objects of class `qts` with added noise as specified by parameters `alpha` and `beta`.

Examples

```
rnorm_qts(1, vespa64$igp[[1]])
```

scale *QTS Sample Centering and Standardization*

Description

QTS Sample Centering and Standardization

Usage

```
scale(x, center = TRUE, scale = TRUE, ...)

## Default S3 method:
scale(x, center = TRUE, scale = TRUE, ...)

## S3 method for class 'qts_sample'
scale(
  x,
  center = TRUE,
  scale = TRUE,
  by_row = FALSE,
  keep_summary_stats = FALSE,
  ...
)
```

Arguments

x	An object coercible into a numeric matrix or an object of class qts_sample representing a sample of observed QTS.
center	A boolean specifying whether to center the sample. If set to FALSE, the original sample is returned, meaning that no standardization is performed regardless of whether argument scale was set to TRUE or not. Defaults to TRUE.
scale	A boolean specifying whether to standardize the sample once it has been centered. Defaults to TRUE.
...	Extra arguments passed on to next methods.
by_row	A boolean specifying whether the QTS scaling should happen for each data point (by_row = TRUE) or for each time point (by_row = FALSE). Defaults to FALSE.
keep_summary_stats	A boolean specifying whether the mean and standard deviation used for standardizing the data should be stored in the output object. Defaults to FALSE in which case only the list of properly rescaled QTS is returned.

Value

A list of properly rescaled QTS stored as an object of class qts_sample when keep_summary_stats = FALSE. Otherwise a list with three components:

- `rescaled_sample`: a list of properly rescaled QTS stored as an object of class `qts_sample`;
- `mean`: a list of numeric vectors storing the corresponding quaternion Fréchet means;
- `sd`: a numeric vector storing the corresponding quaternion Fréchet standard deviations.

Examples

```
x <- scale(vespa64$igp)
x[[1]]
```

`smooth` *QTS Smoothing via SLERP Interpolation*

Description

This function performs a smoothing of a QTS by SLERP interpolation.

Usage

```
smooth(x, ...)

## Default S3 method:
smooth(
  x,
  kind = c("3RS3R", "3RSS", "3RSR", "3R", "3", "S"),
  twiceit = FALSE,
  endrule = c("Tukey", "copy"),
  do.ends = FALSE,
  ...
)

## S3 method for class 'qts'
smooth(x, alpha = 0.5, ...)

## S3 method for class 'qts_sample'
smooth(x, alpha = 0.5, ...)
```

Arguments

`x`	An object of class qts or qts_sample.
`...`	Extra arguments passed on to next methods.
`kind`	a character string indicating the kind of smoother required; defaults to `"3RS3R"`.
`twiceit`	logical, indicating if the result should be 'twiced'. Twicing a smoother $S(y)$ means $S(y) + S(y - S(y))$, i.e., adding smoothed residuals to the smoothed values. This decreases bias (increasing variance).
`endrule`	a character string indicating the rule for smoothing at the boundary. Either `"Tukey"` (default) or `"copy"`.

do.ends	logical, indicating if the 3-splitting of ties should also happen at the boundaries (ends). This is only used for kind = "S".
alpha	A numeric value in [0,1] specifying the amount of smoothing. The closer to one, the smoother the resulting QTS. Defaults to 0.5.

Value

An object of the same class as the input argument x which is a smooth version of the input QTS.

Examples

```
smooth(vespa64$igp[[1]])
smooth(vespa64$igp)
```

---

straighten *QTS Straightening*

Description

This function straightens QTS so that the last point equals the first point.

Usage

```
straighten(x)

## S3 method for class 'qts'
straighten(x)

## S3 method for class 'qts_sample'
straighten(x)
```

Arguments

x	An object of class qts or qts_sample.

Value

An object of the same class as the input argument x storing the straightened QTS.

Examples

```
straighten(vespa64$igp[[1]])
straighten(vespa64$igp)
```

vespa *The VESPA dataset*

Description

A set of QTS representing individual gait patterns (IGPs) of individuals collected under a number of varying factors.

Usage

```
vespa
```

Format

A `tibble` with 320 rows and 7 columns:

- `V`: a categorical variable with two levels specifying the ID of the `Volunteer`;
- `E`: a categorical variable with two levels specifying the ID of the `Experimenter`;
- `S`: a categorical variable with four levels specifying the type of `Sensor`;
- `P`: a categorical variable with four levels specifying the `Position` of the sensor;
- `A`: a categorical variable with two levels specifying the ID of the `Acquisition` pathway;
- `R`: a categorical variable with 5 levels specifying the ID of the `Repetition`;
- `igp`: A 101x5 `tibble` storing a QTS which represents the IGP of the individual under a specific set of VESPA conditions.

Details

The IGP measures the hip rotation during a typical gait cycle. Each rotation is expressed with respect to the mean position of the sensor during the gait cycle. Each IGP is then straightened so that it is periodic with a last point matching the first one.

vespa64 *The VESPA64 dataset*

Description

A set of QTS representing individual gait patterns (IGPs) of individuals collected under a number of varying factors.

Usage

```
vespa64
```

Format

A tibble with 320 rows and 7 columns:

- V: a categorical variable with two levels specifying the ID of the Volunteer;
- E: a categorical variable with two levels specifying the ID of the Experimenter;
- S: a categorical variable with four levels specifying the type of Sensor;
- P: a categorical variable with four levels specifying the Position of the sensor;
- A: a categorical variable with two levels specifying the ID of the Acquisition pathway;
- igp: A 101x5 tibble storing a QTS which represents the IGP of the individual under a specific set of VESPA conditions.

Details

The IGP measures the hip rotation during a typical gait cycle. Each rotation is expressed with respect to the mean position of the sensor during the gait cycle. Each IGP is then straightened so that it is periodic with a last point matching the first one.

It is essentially a reduced version of the VESPA data set where IGPs have been averaged over the repetition for each set of conditions.

Index