Package 'celltrackR'

October 12, 2022

Type Package

Title Motion Trajectory Analysis

Date 2022-02-18

Version 1.1.0

Author Johannes Textor [aut, cre],
Katharina Dannenberg [aut],
Jeffrey Berry [aut],
Gerhard Burger [aut],
Annie Liu [aut],
Inge Wortel [aut]

Maintainer Johannes Textor <johannes.textor@gmx.de>

Description Provides methodology to analyze cells that move in a two- or three-dimensional space. Available measures include displacement, confinement ratio, autocorrelation, straightness, turning angle, and fractal dimension. Measures can be applied to entire tracks, steps, or subtracks with varying length. While the methodology has been developed for cell trajectory analysis, it is applicable to anything that moves including animals, people, or vehicles.
Some of the methodology implemented in this packages was described by: Beauchemin, Dixit, and Perelson (2007) <doi:10.4049/jimmunol.178.9.5505>, Beltman, Maree, and de Boer (2009) <doi:10.1038/nri2638>, Gneiting and Schlather (2004) <doi:10.1137/S0036144501394387>, Mokhtari, Mech, Zitzmann, Hasenberg, Gunzer, and Figge (2013) <doi:10.1371/journal.pone.0080808>, Moreau, Lemaitre, Terriac, Azar, Piel, Lennon-Dumenil, and Bousso (2012) <doi:10.1016/j.immuni.2012.05.014>, Textor, Peixoto, Henrickson, Sinn, von Andrian, and Westermann (2011) <doi:10.1073/pnas.1102288108>, Textor, Sinn, and de Boer (2013) <doi:10.1186/1471-2105-14-S6-S10>, Textor, Henrickson, Mandl, von Andrian, Westermann, de Boer, and Beltman (2014) <doi:10.1371/journal.pcbi.1003752>.

Depends R (>= 3.5.0)

License GPL-2

Encoding UTF-8

LazyData true

URL http://www.motilitylab.net

Imports stats, grDevices, graphics, utils, ellipse, pracma

Suggests scatterplot3d, fractaldim, testthat, wordspace, knitr, rmarkdown, uwot, dendextend, ggplot2, ggbeeswarm, gridExtra, mvtnorm

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2022-03-21 15:50:19 UTC

R topics documented:

`aggregate.tracks` *Compute Summary Statistics of Subtracks*

Description

Computes a given measure on subtracks of a given track set, applies a summary statistic for each subtrack length, and returns the results in a convenient form. This important workhorse function facilitates many common motility analyses such as mean square displacement, turning angle, and autocorrelation plots.

Usage

```
## S3 method for class 'tracks'
aggregate(
  x,
  measure,
  by = "subtracks",
  FUN = mean,
  subtrack.length = seq(1, (maxTrackLength(x) - 1)),
  max.overlap = max(subtrack.length),
  na.rm = FALSE,
  filter.subtracks = NULL,
  count.subtracks = FALSE,
  ...
)
```

Arguments

x	the tracks object whose subtracks are to be considered. If a single track is given, it will be coerced to a tracks object using wrapTrack (but note that this requires an explicit call aggregate.tracks).
measure	the measure that is to be computed on the subtracks.
by	a string that indicates how grouping is performed. Currently, two kinds of grouping are supported: • "subtracks" Apply measure to all subtracks according to the parameters subtrack.length and max.overlap. • "prefixes" Apply measure to all prefixes (i.e., subtracks starting from a track's initial position) according to the parameter subtrack.length.
FUN	a summary statistic to be computed on the measures of subtracks with the same length. Can be a function or a string. If a string is given, it is first matched to the following builtin values: • "mean.sd" Outputs the mean and $mean - sd$ as lower and $mean + sd$ as upper bound • "mean.se" Outputs the mean and $mean - se$ as lower and $mean + se$ as upper bound • "mean.ci.95" Outputs the mean and upper and lower bound of a parametric 95 percent confidence intervall. • "mean.ci.99" Outputs the mean and upper and lower bound of a parametric 95 percent confidence intervall. • "iqr" Outputs the interquartile range, that is, the median, and the 25-percent-quartile as a lower and and the 75-percent-quartile as an upper bound If the string is not equal to any of these, it is passed on to match.fun.
subtrack.length	an integer or a vector of integers defining which subtrack lengths are considered. In particular, subtrack.length=1 corresponds to a "step-based analysis" (Beltman et al, 2009).

`max.overlap` an integer controlling what to do with overlapping subtracks. A maximum overlap of `max(subtrack.length)` will imply that all subtracks are considered. For a maximum overlap of 0, only non-overlapping subtracks are considered. A negative overlap can be used to ensure that only subtracks a certain distance apart are considered. In general, for non-Brownian motion there will be correlations between subsequent steps, such that a negative overlap may be necessary to get a proper error estimate.

`na.rm` logical. If `TRUE`, then `NA`'s and `NaN`'s are removed prior to computing the summary statistic.

`filter.subtracks`
a function that can be supplied to exclude certain subtracks from an analysis. For instance, one may wish to compute angles only between steps of a certain minimum length (see Examples).

`count.subtracks`
logical. If `TRUE`, the returned dataframe contains an extra column `ntracks` showing the number of subtracks of each length. This is useful to keep track of since the returned `value` estimates for high `i` are often based on very few subtracks.

`...` further arguments passed to or used by methods.

Details

For every number of segments i in the set defined by `subtrack.length`, all subtracks of any track in the input `tracks` object that consist of exactly i segments are considered. The input `measure` is applied to the subtracks individually, and the `statistic` is applied to the resulting values.

Value

A data frame with one row for every i specified by `subtrack.length`. The first column contains the values of i and the remaining columns contain the values of the summary statistic of the measure values of tracks having exactly i segments.

References

Joost B. Beltman, Athanasius F.M. Maree and Rob. J. de Boer (2009), Analysing immune cell migration. *Nature Reviews Immunology* **9**, 789–798. doi:10.1038/nri2638

Examples

```
## A mean square displacement plot with error bars.
dat <- aggregate(TCells, squareDisplacement, FUN="mean.se")
with( dat ,{
  plot( mean ~ i, xlab="time step",
   ylab="mean square displacement", type="l" )
  segments( i, lower, y1=upper )
} )

## Note that the values at high i are often based on very few subtracks:
msd <- aggregate( TCells, squareDisplacement, count.subtracks = TRUE )
```

```
tail( msd )

## Compute a turning angle plot for the B cell data, taking only steps of at least
## 1 micrometer length into account
check <- function(x) all( sapply( list(head(x,2),tail(x,2)), trackLength ) >= 1.0 )
plot( aggregate( BCells, overallAngle, subtrack.length=1:10,
  filter.subtracks=check )[,2], type='l' )

## Compare 3 different variants of a mean displacement plot
# 1. average over all subtracks
plot( aggregate( TCells, displacement ), type='l' )
# 2. average over all non-overlapping subtracks
lines( aggregate( TCells, displacement, max.overlap=0 ), col=2 )
# 3. average over all subtracks starting at 1st position
lines( aggregate( TCells, displacement, by="prefixes" ), col=3 )
```

`analyzeCellPairs` *Find Distances and Angles for all Pairs of Tracks*

Description

Find all pairs of cells and return the shortest distance between them at any point in time (if they share any time points), as well as the angle between their overall displacement vectors.

Usage

```
analyzeCellPairs(X, searchRadius = Inf, quietly = FALSE, ...)
```

Arguments

`X`	a tracks object
`searchRadius`	if specified, only return analysis for pairs of cells that are within distance searchRadius from each other at least at one point in time.
`quietly`	(default FALSE) if TRUE, suppress warnings
`...`	further arguments passed on to `angleCells`

Details

Analyzing track angles at different distances can be useful to detect directional bias or local crowding effects; see (Beltman et al, 2009).

Internally, the function uses `cellPairs`, `angleCells`, and `distanceCells`.

Value

A dataframe with four columns: two for the indices of cellpairs, one for the distance between them, and one for their angle. Note that the distance will be NA for pairs of tracks that do not share time points, but their angle will still be computed.

References

Joost B. Beltman, Athanasius F.M. Maree and Rob. J. de Boer (2009), Analysing immune cell migration. *Nature Reviews Immunology* **9**, 789–798. doi:10.1038/nri2638

See Also

`analyzeStepPairs` to do something similar for single steps rather than entire tracks.

Examples

```
## Plot distance versus angle for all cell pairs. Sample T-cell data here for speed.
pairs <- analyzeCellPairs( sample( TCells, 100 ) )
scatter.smooth( pairs$dist, pairs$angle )
```

`analyzeStepPairs` *Find Distances and Angles for all Pairs of Steps*

Description

Find cell indices and timepoints where these cells both have a step, then return angles and distances for each pair of steps.

Usage

```
analyzeStepPairs(
  X,
  filter.steps = NULL,
  searchRadius = Inf,
  quietly = FALSE,
  ...
)
```

Arguments

`X`	a tracks object
`filter.steps`	optional: a function used to filter steps on. See examples.
`searchRadius`	if specified, only return analysis for pairs of steps that start within distance searchRadius from each other
`quietly`	(default FALSE) if TRUE, suppress warnings
`...`	further arguments passed on to `angleSteps`

Details

Analyzing step angles at different distances can be useful to detect directional bias or local crowding effects; see (Beltman et al, 2009).

Internally, the function uses `stepPairs`, `angleSteps`, and `distanceSteps`.

Value

A dataframe with five columns: two for the indices of cellpairs that share a step, one for the timepoint at which they do so, one for the distance between them, and one for their angle.

References

Joost B. Beltman, Athanasius F.M. Maree and Rob. J. de Boer (2009), Analysing immune cell migration. *Nature Reviews Immunology* **9**, 789–798. doi:10.1038/nri2638

See Also

`analyzeCellPairs` to do something similar for entire tracks rather than single steps.

Examples

```
## Plot distance versus angle for all step pairs, filtering for those that
## displace at least 2 microns. Sample dataset in this example for speed.
pairs <- analyzeStepPairs( sample( TCells, 100), filter.steps = function(t) displacement(t) > 2 )
scatter.smooth( pairs$dist, pairs$angle )
```

`AngleAnalysis` *Angle Analysis*

Description

Analyzing angles to reference directions, points, or planes can be useful to detect artefacts and/or directionality in tracking datasets (Beltman et al, 2009). All these functions take a track and a reference (point/direction/plane) as input and return a distance or angle as output. Angles/distances are by default computed to the first step in the given track.

Details

`angleToPoint` and `distanceToPoint` return the angle/distance of the track to the reference point. The distance returned is between the first coordinate in the track and the reference point. The angle is between the overall displacement vector of the track and the vector from its first coordinate to the reference point. Angles are by default returned in degrees, use `degrees=FALSE` to obtain radians. These functions are useful to detect directional bias towards a point of interest, which would result in an average angle of less than 90 degrees with the reference point (especially for tracks at a small distance to the reference point).

`angleToPlane` and `distanceToPlane` return the angle/distance of the track to a plane of interest. This plane must be specified by three points lying on it. The distance returned is between the first coordinate in the track and the reference point. The angle is between the overall displacement vector of the track and the plane of interest. These functions are useful to detect tracking artefacts near the borders of the imaging volume. Use `boundingBox` to guess where those borders are. Angles are by default returned in degrees, use `degrees=FALSE` to obtain radians.

`angleToDir` returns the angle of a track's overall displacement vector to a direction of interest. This function is useful to detect directionality in cases where the direction of the bias is known in

advance (e.g. when cells are known to move up a chemotactic gradient): in that case, the average angle to the reference direction should be less than 90 degrees. Angles are by default returned in degrees, use `degrees=FALSE` to obtain radians.

`angleSteps` and `distanceSteps` return the angle/distance between a pair of steps in the data that occur at the same timepoint. Angles are in degrees by default, use `degrees=FALSE` to obtain radians. Use `stepPairs` to extract all pairs of steps that occur at the same timepoint, and use `analyzeStepPairs` to do this and then also obtain the angles and distances for each of these pairs.

`angleCells` and `distanceCells` return the angle/distance between a pair of tracks in the data. The computed angles are between the overall displacement vectors of the tracks, the distance is the shortest distance between them at any timepoint they share. Angles are in degrees by default, use `degrees=FALSE` to obtain radians. Use `cellPairs` to extract all pairs of cells in the data, and use `analyzeCellPairs` to do this and then also obtain the angles and distances for each of these pairs.

Value

This page is for documentation only and provides an overview of angle analysis functions and their use cases. The return values of each of these functions are documented separately; please follow the link to the documentation page of that specific function.

References

Joost B. Beltman, Athanasius F.M. Maree and Rob. J. de Boer (2009), Analysing immune cell migration. *Nature Reviews Immunology* **9**, 789–798. doi:10.1038/nri2638

See Also

`TrackMeasures` for other measures that can be used to quantify tracks.

See the vignettes on Quality Control and Track Analysis for more detailed examples of angle analyses. `browseVignettes( package = "celltrackR" )`

Examples

```
## Plotting the angle versus the distance to a reference point can be informative to
## detect biased movement towards that point. We should be suspicious especially
## when small angles are more frequent at lower distances.
steps <- subtracks( sample( Neutrophils, 50 ), 1 )
bb <- boundingBox( Neutrophils )
angles <- sapply( steps, angleToPoint, p = bb["max",-1] )
distances <- sapply( steps, distanceToPoint, p = bb["max",-1] )
scatter.smooth( distances, angles )
abline( h = 90, col = "red" )

## Get a distribution of Neutrophil step angles with the reference direction
## in positive y direction. The histogram is enriched for low angles, suggesting
## directed movement:
hist( sapply( steps, angleToDir, dvec=c(1,-1) ) )

## Plotting the angle versus the distance to a reference plane can be informative to
## detect tracking artefacts near the border of the imaging volume.
## We should be suspicious especially when small angles are more frequent at low distances
```

```
## to the border planes; as is the case in the z-dimension for the raw data:
load( system.file("extdata", "TCellsRaw.rda", package="celltrackR" ) )
steps <- subtracks( sample( TCellsRaw, 50 ), 1 )
minz <- boundingBox( TCellsRaw )["min","z"]
## Compute angles and distances to the lower plane in z-dimension
angles <- sapply( steps, angleToPlane, p1 = c(0,0,minz), p2 = c(1,0,minz), p3 = c(0,1,minz) )
distances <- sapply( steps, distanceToPlane, p1 = c(0,0,minz), p2 = c(1,0,minz), p3 = c(0,1,minz) )
scatter.smooth( distances, angles )
abline( h = 32.7, col = "red" )

## Plot distance versus angle for all cell pairs (here in only a sample to speed things up)
pairs <- analyzeCellPairs( sample( TCells, 50 ) )
scatter.smooth( pairs$dist, pairs$angle )
abline( h = 90, col = "red" )

## Plot distance versus angle for all step pairs, filtering for those that
## displace at least 2 microns
pairs <- analyzeStepPairs( sample( TCells, 50 ), filter.steps = function(t) displacement(t) > 2 )
scatter.smooth( pairs$dist, pairs$angle )
abline( h = 90, col = "red" )
```

`angleCells` *Angle between Two Tracks*

Description

Compute the angle between the displacement vectors of two tracks in the dataset, or of several such pairs at once. Note that in contrast to `distanceCells`, this angle is computed even when the two tracks do not share any time points.

Usage

```
angleCells(X, cellids, degrees = TRUE)
```

Arguments

`X`	a tracks object
`cellids`	a vector of two indices specifying the tracks to get steps from, or a dataframe/matrix of two columns (where every row contains a pair of cellids to compute an angle for)
`degrees`	logical; should angle be returned in degrees instead of radians? (defaults to `TRUE`)

Value

A single angle (if two cellids given), or a vector of angles (if multiple pairs of cellids are supplied).

See Also

distanceCells to compute the minimum distance between the tracks, and AngleAnalysis for other methods to compute angles and distances.

Examples

```
## Find the angle between the tracks with ids 1 and 3
angleCells( TCells, c("1","3") )

## Find the angles of several cell pairs at once
pairs <- data.frame( cell1 = c("1","1"), cell2 = c( "3","4" ) )
angleCells( TCells, pairs )
```

---

angleSteps *Angle between Two Steps*

Description

Compute the angle between two steps in the dataset that occur at the same timepoint.

Usage

```
angleSteps(X, trackids, t, degrees = TRUE, quietly = FALSE)
```

Arguments

X	a tracks object
trackids	a vector of two indices specifying the tracks to get steps from, or a dataframe/matrix of two columns (where every row contains a pair of trackids to compute a step angle for)
t	the timepoint at which the steps should start, or a vector of timepoints if trackids is a matrix with multiple step pairs to compute angles for.
degrees	logical; should angle be returned in degrees instead of radians? (defaults to TRUE)
quietly	logical; should a warning be returned if one or both of the steps are missing in the data and the function returns NA?

Value

A single angle, or NA if the desired step is missing for one or both of the tracks. If trackids is a matrix with multiple step pairs to compute angles for, the output is a numeric vector of angles (or NA values).

See Also

distanceSteps to compute the distance between the step starting points, timePoints to list all timepoints in a dataset, and AngleAnalysis for other methods to compute angles and distances.

Examples

```
## Find the angle between the steps of the tracks with ids 1 and 2, at the 3rd
## timepoint in the dataset.
t <- timePoints( TCells )[3]
angleSteps( TCells, c("1","3"), t )

## Do this for multiple pairs and times at once: between cells 1 and 3 at the
## 3rd timepoint, and between 1 and 4 at the fourth timepoint.
pairs <- data.frame( cell1 = c("1","1"), cell2 = c("3","4"))
times <- timePoints(TCells)[3:4]
angleSteps( TCells, pairs, times )
```

`angleToDir` *Angle with a Reference Direction*

Description

Compute the angle between a track's overall displacement and a reference direction. Useful to detect biased movement when the directional bias is known (see examples).

Usage

```
angleToDir(x, dvec = c(1, 1, 1), from = 1, degrees = TRUE)
```

Arguments

`x`	a single input track; a matrix whose first column is time and whose remaining columns are a spatial coordinate.
`dvec`	numeric vector specifying a reference direction to compute angles to.
`from`	index, or vector of indices, of the first row of the track. If `from` is a vector, angles are returned for all steps starting at the indices in `from`.
`degrees`	logical; should angles be returned in degrees rather than radians? (default = TRUE).

Details

The average angle of steps to a reference direction should be 90 degrees if there is no bias towards movement in the direction of the reference point. If there is such a bias, there should be an enrichment of smaller angles. The expected distribution without bias is a uniform distribution in 2D or a sine distribution in 3D (Beltman et al, 2009).

Value

A single angle.

References

Joost B. Beltman, Athanasius F.M. Maree and Rob. J. de Boer (2009), Analysing immune cell migration. *Nature Reviews Immunology* **9**, 789–798. doi:10.1038/nri2638

See Also

`AngleAnalysis` for other methods to compute angles and distances.

Examples

```
## Get a distribution of Neutrophil step angles with the reference direction in positive
## y direction. The histogram is enriched for low angles, suggesting directed movement:
steps <- subtracks( Neutrophils, 1 )
hist( sapply( steps, angleToDir, dvec=c(1,-1) ) )
```

---

angleToPlane *Angle with a Reference Plane*

Description

Compute the angle between a track's overall displacement and a reference plane. Useful to detect directed movement and/or tracking artefacts.

Usage

```
angleToPlane(
  x,
  p1 = c(0, 0, 0),
  p2 = c(0, 1, 0),
  p3 = c(1, 0, 0),
  from = 1,
  degrees = TRUE
)
```

Arguments

`x`	a single input track; a matrix whose first column is time and whose remaining columns are a spatial coordinate.
`p1, p2, p3`	numeric vectors of coordinates of three points specifying a reference plane to compute distances to.
`from`	index, or vector of indices, of the first row of the track. If `from` is a vector, angles are returned for all steps starting at the indices in `from`.
`degrees`	logical; should angles be returned in degrees rather than radians? (default = TRUE).

Details

The average angle of steps to a reference plane should be roughly 32.7 degrees. Lower angles to the border planes of an imaging volume can be indicative of tracking artefacts, and systematic deviations from 32.7 can indicate a directional bias (Beltman et al, 2009).

Value

A single angle.

References

Joost B. Beltman, Athanasius F.M. Maree and Rob. J. de Boer (2009), Analysing immune cell migration. *Nature Reviews Immunology* **9**, 789–798. doi:10.1038/nri2638

See Also

`distanceToPlane` to compute the distance to the reference plane, and `AngleAnalysis` for other methods to compute angles and distances.

Examples

```
## Plotting the angle versus the distance to a reference plane can be informative to
## detect tracking artefacts near the border of the imaging volume.
## We should be suspicious especially when small angles are more frequent at low distances
## to the border planes.
load( system.file("extdata", "TCellsRaw.rda", package="celltrackR" ) )
steps <- subtracks( TCellsRaw, 1 )
minz <- boundingBox( TCellsRaw )["min","z"]
## Compute angles and distances to the lower plane in z-dimension
angles <- sapply( steps, angleToPlane, p1 = c(0,0,minz), p2 = c(1,0,minz), p3 = c(0,1,minz) )
distances <- sapply( steps, distanceToPlane, p1 = c(0,0,minz), p2 = c(1,0,minz), p3 = c(0,1,minz) )
scatter.smooth( distances, angles )
abline( h = 32.7, col = "red" )
```

---

angleToPoint *Angle with a Reference Point*

Description

Compute the angle between a track's overall displacement vector and the vector from it's first coordinate to a reference point. Useful to detect directed movement towards a point (see examples).

Usage

```
angleToPoint(x, p = c(1, 1, 1), from = 1, degrees = TRUE)
```

Arguments

x	a single input track; a matrix whose first column is time and whose remaining columns are a spatial coordinate.
p	numeric vector of coordinates of the reference point p to compute angles/distances to.
from	index, or vector of indices, of the first row of the track. If from is a vector, angles are returned for all steps starting at the indices in from.
degrees	logical; should angles be returned in degrees rather than radians? (default = TRUE).

Details

The average angle of steps to a reference point should be 90 degrees if there is no bias towards movement in the direction of the reference point. If there is such a bias, there should be an enrichment of smaller angles. The expected distribution without bias is a uniform distribution in 2D or a sine distribution in 3D (Beltman et al, 2009).

Value

A single angle.

References

Joost B. Beltman, Athanasius F.M. Maree and Rob. J. de Boer (2009), Analysing immune cell migration. *Nature Reviews Immunology* **9**, 789–798. doi:10.1038/nri2638

See Also

distanceToPoint to compute the distance to the reference point, and AngleAnalysis for other methods to compute angles and distances.

Examples

```
## Get a distribution of step angles with a reference point
## Use bb to get the corner with highest x,y (,z) value
## The histogram is enriched for low angles, suggesting directed movement:
steps <- subtracks( Neutrophils, 1 )
bb <- boundingBox( Neutrophils )
hist( sapply( steps, angleToPoint, p = bb["max",-1] ) )

## The same does not hold for movement of T cells towards the point (0,0)
steps <- subtracks( TCells, 1 )
hist( sapply( steps, angleToPoint, p = c(0,0) ) )

## Plotting the angle versus the distance to the reference point can also be informative,
## especially when small angles are more frequent at lower distances.
angles <- sapply( steps, angleToPoint, p = bb["max",-1] )
distances <- sapply( steps, distanceToPoint, p = bb["max",-1] )
scatter.smooth( distances, angles )
abline( h = 90, col = "red" )
```

applyStaggered *Compute a Measure on a Track in a Staggered Fashion*

Description

Computes a measure on all subtracks of a track and return them either as a matrix or return their mean.

Usage

```
applyStaggered(x, measure, matrix = FALSE, min.segments = 1)
```

Arguments

x	the track for which the measure is to be computed.
measure	the measure that is to be computed.
matrix	a logical indicating whether the whole matrix of values for the measure for each of the input track's subtracks is to be returned. Otherwise only the mean is returned.
min.segments	the number of segments that each regarded subtrack should at least consist of. Typically, this value would be set to the minimum number of segments that a (sub)track must have in order for the measure to be decently computed. For example, at least two segments are needed to compute the overallAngle.

Details

The measure is computed for each of the input track's subtracks of length at least min.segments, and the resulting values are either returned in a matrix (if matrix is set), or their mean is returned. The computed matrix is symmetric since the direction along which a track is traversed is assumed not to matter. The values at [i, i + j], where j is a nonnegative integer with $j <$ min.segments, (with the default value min.segments=1 this is exactly the main diagonal) are set to NA.

Value

If matrix is set, a matrix with the values of the measure for all the input track's subtracks is returned. The value of this matrix at position [i, j] corresponds to the subtrack that starts with the input track's jth point and ends at its ith. Thus, with increasing column number, the regarded subtrack's starting point is advanced on the original track, and the values for increasingly long subtracks starting from this point can be found columnwise below the main diagonal, respectively. If 'matrix' is not set, the mean over the values of the measure for all subtracks of at least 'min.segments' segments is retruned.

Examples

```
## Compute the staggered matrix for overallAngle applied to all long enough
## subtracks of the first T cell track
applyStaggered(TCells[[1]], overallAngle, matrix=TRUE, min.segments = 2)
```

`as.data.frame.tracks` *Convert Tracks to Data Frame*

Description

Converts tracks from the list-of-matrices format, which is good for efficient processing and therefore the default in this package, to a single dataframe which is convenient for plotting or saving the data.

Usage

```
## S3 method for class 'tracks'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  include.timepoint.column = FALSE,
  idsAsFactors = TRUE,
  ...
)
```

Arguments

`x`	the `tracks` object to be coerced to a data frame.
`row.names`	NULL or a character vector giving row names for the data frame. Missing values are not allowed.
`optional`	logical. Required for S3 consistency, but has no effect: column names are always assigned to the resulting data frame regardless of the setting of this option.
`include.timepoint.column`	logical. If set to `TRUE`, then the resulting dataframe will contain a column that consecutively numbers the positions according to their time. Note that this information is anyway implicitly present in the time information.
`idsAsFactors`	logical. If `TRUE`, then the id column of the resulting dataframe will be a factor column, otherwise a characeter column.
`...`	further arguments to be passed from or to other methods.

Value

A single data frame containing all individual tracks from the input with a prepended column named "id" containing each track's identifier in ‘x‘.

Examples

```
## Display overall average position of the T cell data
colMeans( as.data.frame( TCells )[-c(1,2)] )
```

as.list.tracks *Convert from Tracks to List*

Description

Coerces a tracks object to a list.

Usage

```
## S3 method for class 'tracks'
as.list(x, ...)
```

Arguments

x	the tracks object to be coerced to a list.
...	further arguments to be passed from or to other methods.

Value

A generic list of single tracks, where each track is a matrix with t/delta.t rows and 4 columns. This looks a lot like a tracks object, except that its class is not "tracks" anymore.

as.tracks.data.frame *Convert from Data Frame to Tracks*

Description

Get cell tracks from a data.frame. Data are expected to be organized as follows. One column contains a track identifier, which can be numeric or a string, and determines which points belong to the same track. Another column is expected to contain a time index or a time period (e.g. number of seconds elapsed since the beginning of the track, or since the beginning of the experiment). Input of dates is not (yet) supported, as absolute time information is frequently not available. Further columns contain the spatial coordinates. If there are three or less spatial coordinates, their names will by "x", "y", and "z" (depending on whether the tracks are 1D, 2D or 3D). If there are four or more spatial coordinates, their names will be "x1", "x2", and so on. The names or indices of these columns in the data.frame are given using the corresponding parameters (see below). Names and indices can be mixed, e.g. you can specify id.column="Parent" and pos.columns=1:3

Usage

```
## S3 method for class 'data.frame'
as.tracks(
  x,
  id.column = 1,
  time.column = 2,
```

```
  pos.columns = 3:ncol(x),
  scale.t = 1,
  scale.pos = 1,
  ...
)
```

Arguments

x	the data frame to be coerced to a tracks object.
id.column	index or name of the column that contains the track ID.
time.column	index or name of the column that contains elapsed time.
pos.columns	vector containing indices or names of the columns that contain the spatial coordinates. If this vector has two entries and the second entry is NA, e.g. c('x',NA) or c(5,NA) then all columns from the indicated column to the last column are used. This is useful when reading files where the exact number of spatial dimensions is not known beforehand.
scale.t	a value by which to multiply each time point. Useful for changing units, or for specifying the time between positions if this is not contained in the file itself.
scale.pos	a value, or a vector of values, by which to multiply each spatial position. Useful for changing units.
...	further arguments to be passed to read.csv, for instance sep="\t" can be useful for tab-separated files.

Value

A tracks object.

BCells *Two-Photon Data: B Cells in a Lymph Node*

Description

Labelled B cells were adoptively transfered and intravitally imaged (using two-photon microscopy) inside a peripheral lymph node of the recipient mouse. These data illustrate the characteristic "random-walk-like" motion pattern of B cells in lymph nodes.

Usage

```
data("BCells")
```

Format

An S3 object of class "tracks"; a list with 24 elements. Each element name identifies a cell track. Each element is a matrix containing the following four columns.

- `t` the time (in seconds)
- `x` The X coordinate (in micrometers)
- `y` The Y coordinate (in micrometers)
- `z` The Z coordinate (in micrometers)

Source

Data were generated in 2012 in the Mark J. Miller Lab, Department of Medicine, Washington University in St Louis, USA.

References

Zinselmeyer BH, Dempster J, Wokosin DL, Cannon JJ, Pless R, Parker I and Miller MJ (2009), Two-photon microscopy and multi-dimensional analysis of cell dynamics. *Methods in Enzymology*, **461**:349–78. doi:10.1016/S0076-6879(09)05416-0

Konjufca V and Miller MJ (2009), Imaging Listeria monocytogenes infection in vivo. *Current Topics in Microbiology and Immunology*, **334**:199–226. doi:10.1007/978-3-540-93864-4_9

Kreisel D, Nava RG, Li W, Zinselmeyer BH, Wang B, Lai J, Pless R, Gelman AE, Krupnick AS, and Miller MJ (2010), In vivo two-photon imaging reveals monocyte-dependent neutrophil extravasation during pulmonary inflammation. *PNAS*, **107**(42):18073–18078. doi:10.1073/pnas.1008737107

Examples

```
## load the tracks
data(BCells)
## visualize the tracks (calls function plot.tracks)
plot(BCells)
```

`beaucheminTrack` *Simulate a 3D Cell Track Using the Beauchemin Model*

Description

The Beauchemin model is a simple, particle-based description of T cell motion in lymph node in the absence of antigen, which is similar to a random walk (Beauchemin et al, 2007).

Usage

```
beaucheminTrack(
  sim.time = 10,
  delta.t = 1,
  p.persist = 0,
  p.bias = 0.9,
  bias.dir = c(0, 0, 0),
  taxis.mode = 1,
  t.free = 2,
  v.free = 18.8,
  t.pause = 0.5
)
```

Arguments

`sim.time`	specifies the duration of the track to be generated
`delta.t`	change in time between each timepoint.
`p.persist`	indicates how probable a change in direction is. With p.persist = 1, the direction never changes between steps and with p.persist = 0, a new direction is sampled at every step.
`p.bias`	strength of movement in the direction of `bias.dir`.
`bias.dir`	a 3D vector indicating the direction along which there is a preference for movement.
`taxis.mode`	specified mode of movement. 1 := orthotaxis, 2 := topotaxis, 3 := klinotaxis.
`t.free`	time interval for how long the cell is allowed to move between turns.
`v.free`	speed of the cell during the free motion.
`t.pause`	time that it takes the cell to adjust movement to new direction.

Details

In the Beauchemin model, cells move into a fixed direction for a fixed time `t.free` at a fixed speed `v.free`. They then switch to a different direction, which is sampled at uniform from a sphere. The change of direction takes a fixed time `t.pause`, during which the cell does not move. Thus, the Beauchemin model is identical to the freely jointed chain model of polymer physics, except for the explicit "pause phase" between subsequent steps.

The default parameters implemented in this function were found to most accurately describe 'default' T cell motion in lymph nodes using least-squares fitting to the mean displacement plot (Beauchemin et al, 2007).

This function implements an extended version of the Beauchemin model, which can also simulate directionally biased motion. For details, see Textor et al (2013).

Value

A track, i.e., a matrix with `t/delta.t` rows and 4 columns.

References

Catherine Beauchemin, Narendra M. Dixit and Alan S. Perelson (2007), Characterizing T cell movement within lymph nodes in the absence of antigen. *Journal of Immunology* **178**(9), 5505-5512. doi:10.4049/jimmunol.178.9.5505

Johannes Textor, Mathieu Sinn and Rob J. de Boer (2013), Analytical results on the Beauchemin model of lymphocyte migration. *BMC Bioinformatics* **14**(Suppl 6), S10. doi:10.1186/1471-2105-14-S6-S10

Examples

```
## Create track with model parameters and return matrix of positions
out <- beaucheminTrack(sim.time=20,p.persist = 0.3,taxis.mode = 1)
## Plot X-Y projection
plot( wrapTrack(out) )

## Create 20 tracks and plot them all
out <- simulateTracks( 20, beaucheminTrack(sim.time=10,
  bias.dir=c(-1,1,0),p.bias=10,taxis.mode = 2,
  p.persist = 0.1,delta.t = 1) )
plot( out )
```

`bootstrapTrack` *Simulate Tracks via Bootstrapping of Speed and Turning Angle from a Real Track Dataset*

Description

Returns a simulated dataset by sampling from the speed and turning angle distributions from an original track dataset (only in 2 or 3 dimensions)

Usage

```
bootstrapTrack(nsteps, trackdata)
```

Arguments

`nsteps`	desired number of steps (e.g. 10 steps generates a track with 11 positions).
`trackdata`	a tracks object to extract speeds and turning angles from.

Details

The number of dimensions is kept the same as in the original data (if data is 3D but simulated tracks should be 2D, consider calling `projectDimensions` on the input data before supplying it to `bootstrapTrack`). The time interval between "measurements" of the simulated track equals that in the real data and is found via `timeStep`. The first step starts at the origin in a random direction, with a speed sampled from the speed distribution to determine its displacement. All subsequent steps also have their turning angles sampled from the turning angle distribution in the data.

Value

A data frame containing in cell track with `nsteps` steps in the same number of dimensions as the original data is returned.

Generate bootstrapped tracks of the TCell data; compare its speed distribution to the ## original data (should be the same). T.bootstrap <- bootstrapTrack(100, TCells) step.speeds.real <- sapply(subtracks(TCells,1), speed) step.speeds.bootstrap <- sapply(subtracks(T.bootstrap, 1), speed) qqplot(step.speeds.real, step.speeds.bootstrap)

`boundingBox` *Bounding Box of a Tracks Object*

Description

Computes the minimum and maximum coordinates per dimension (including time) for all positions in a given list of tracks.

Usage

```
boundingBox(x)
```

Arguments

`x`	the input `tracks` object.

Value

Returns a matrix with two rows and $d + 1$ columns, where d is the number of spatial dimensions of the tracks. The first row contains the minimum and the second row the maximum value of any track in the dimension given by the column.

Examples

```
## Use bounding box to set up plot window
bb <- boundingBox(c(TCells,BCells,Neutrophils))
plot( Neutrophils, xlim=bb[,"x"], ylim=bb[,"y"], col=1 )
plot( BCells, col=2, add=TRUE )
plot( TCells, col=3, add=TRUE )
```

brownianTrack *Simulate an Uncorrelated Random Walk*

Description

Generates a random track with `nsteps` steps in `dim` dimensions.

Usage

```
brownianTrack(nsteps = 100, dim = 3, mean = 0, sd = 1)
```

Arguments

`nsteps`	desired number of steps (e.g. 10 steps generates a track with 11 positions).
`dim`	desired number of dimensions.
`mean`	stepwise mean drift per dimension; use 0 for an unbiased Brownian motion and other values for Brownian motion with drift.
`sd`	stepwise standard deviation per dimension.

Details

In in every step an for each dimension, a normally distributed value with mean `mean` and standard deviation `sd` is added to the previous cell position.

Value

A data frame containing in cell track with `nsteps` steps in `dim` dimensions is returned.

The Hurst exponent of a 1D Brownian track should be near 0.5 hurstExponent(brownianTrack(100, 1))

cellPairs *Find Pairs of Tracks*

Description

Get all unique combinations of two track ids.

Usage

```
cellPairs(X)
```

Arguments

`X`	a tracks object

Value

A dataframe with two columns: one for each of the track ids in the pair. Each row represents a pair.

Examples

```
## Find all pairs of cells in the T cell data
pairs <- cellPairs( TCells )
```

`celltrackR` *celltrackR: Quantitative analysis of motion.*

Description

The CelltrackR package is designed for analyzing cell tracks acquired by time-lapse microscopy (like those provided in the included datasets `TCells`, `BCells` and `Neutrophils`). But it can of course process any x-y-(z)-t data, and we hope that it may be useful for other purposes as well.

Details

For a complete list of functions, use `help( package="celltrackR" )`. A handy cheat sheet is available in pdf. You can open it by calling the function `cheatsheet`.

Data structure

The basic data structure that most functions in this package operate on is a set of *tracks*. A track is a list of spatial coordinates that are recorded at *fixed* time intervals; the function `timeStep` can be used to check for fluctuations of the recording intervals.

We expect tracks to be stored in a matrix (or data frame, but this is discouraged for efficiency reasons) whose first column denotes a time interval (e.g. seconds elapsed since the beginning of the experiment), and whose remaining columns denote a spatial coordinate. A set of tracks is stored as a `list` with S3 class `tracks`. CelltrackR provides some S3 methods for this class, which are explained in `tracks` as well as `plot.tracks`, `sort.tracks` and `as.list.tracks`.

Track analysis in celltrackR

A wide range of common track measures are included in the package. These are all functions that take a single track as an input, and output one or several numbers. For instance, the function `speed` estimates the average instantaneous speed of the track by linear interpolation, and `straightness` computes the start-to-end distance divided by the trajectory length (a number between 0 and 1, where 1 indicates a perfectly straight track). See `TrackMeasures` for an overview of measures that can be analyzed on tracks. Also see `AngleAnalysis` for an overview of functions that can help detect directional bias and tracking artefacts (see Beltman et al, 2009).

CelltrackR is designed to support various flavors of track analysis that have been suggested in the literature. The simplest kind is a *track-based* analysis, where we compute a single statistic for each track in a dataset (Beltman et al, 2009). Because track sets are lists, this is achieved simply by using `lapply` or `sapply` together with the track measure (see Examples).

In *step-based* analyses (Beltman et al, 2009), we chop each track up into segments of the same length and then apply our measures to those segments. This can help to avoid biases that arise from variations in track length (which are always present in cell tracking experiments). In CelltrackR, step-based analyses are performed by using the `subtracks` function. Often we want to perform such step-based analyses for all possible subtrack lengths simultaneously, and plot the result as a function of the subtrack length; a famous example is the *mean square displacement plot*. This can be achieved by using the `aggregate.tracks` function, which has options to control which subtrack lengths are considered and whether overlapping subtracks are considered.

In a *staggered staggered* analysis (Mokhtari et al, 2013), we analyse all subtracks (of any length) of a single track, and typically plot the result as a matrix. This can reveal dynamic patterns along a single track, e.g. turning behaviour or local slowdowns. Staggered analyses can be performed using the `applyStaggered` function.

Simulating tracks in celltrackR

Lastly, in addition to data analysis, the package contains some function to generate cell tracks by simulation. This is useful to develop and benchmark track analysis methodology (Textor et al, 2011), and for computational biology studies that try to extrapolate the long-term consequences of observed cell migration behaviour. Alongside a simple uncorrelated random walk (`brownianTrack`), this package implements a simulation model proposed by Beauchemin et al (2007) in the function `beaucheminTrack`. That model can also simulate directionally biased motion.

Author(s)

Johannes Textor, Katharina Dannenberg, Jeffrey Berry, Gerhard Burger, Inge Wortel Maintainer: Johannes Textor <johannes.textor@gmx.de>

References

Joost B. Beltman, Athanasius F.M. Maree and Rob. J. de Boer (2009), Analysing immune cell migration. *Nature Reviews Immunology* **9**, 789–798. doi:10.1038/nri2638

Zeinab Mokhtari, Franziska Mech, Carolin Zitzmann, Mike Hasenberg, Matthias Gunzer and Marc Thilo Figge (2013), Automated Characterization and Parameter–Free Classification of Cell Tracks Based on Local Migration Behavior. *PLoS ONE* **8**(12), e80808. doi:10.1371/journal.pone.0080808

Johannes Textor, Antonio Peixoto, Sarah E. Henrickson, Mathieu Sinn, Ulrich H. von Andrian and Juergen Westermann (2011), Defining the Quantitative Limits of Intravital Two-Photon Lymphocyte Tracking. *PNAS* **108**(30):12401–12406. doi:10.1073/pnas.1102288108

Catherine Beauchemin, Narendra M. Dixit and Alan S. Perelson (2007), Characterizing T cell movement within lymph nodes in the absence of antigen. *Journal of Immunology* **178**(9), 5505-5512. doi:10.4049/jimmunol.178.9.5505

See Also

The package vignettes, available from `browseVignettes( package="celltrackR" )`. Make sure you have installed the package with option `build_vignettes = TRUE`, or vignettes will not be visible. Also check out the package cheat sheet, which is available by calling the function `cheatsheet`.

Examples

```
 ## track-based speed comparison
 boxplot(sapply( Neutrophils, straightness ), sapply( BCells, straightness ))

 ## step-based turning angle comparison
 boxplot(sapply(subtracks(Neutrophils, 2), overallAngle),
   sapply(subtracks(BCells, 2), overallAngle))

## mean square displacement plot; a step-based displacement analysis for all step lengths
plot(aggregate(TCells, squareDisplacement)[,"value"])

## 'staggered' analysis of displacement over whole track. Reveals that this track
## slows down near its beginning and near its end.
filled.contour(applyStaggered(TCells[[4]], displacement, matrix=TRUE))

## a simple hierarchical clustering based on 2D asphericity

## tag track IDs so we can identify them later
names(TCells) <- paste0("T",names(TCells))
names(BCells) <- paste0("B",names(BCells))
names(Neutrophils) <- paste0("N",names(Neutrophils))
## project all tracks down to 2D
cells <-  projectDimensions(c(TCells,BCells,Neutrophils), c("x","y"))

## compute asphericity
asph <- lapply(cells, asphericity)

## plot clustering
plot(hclust(dist(asph)))
```

`cheatsheet` *Open the package cheat sheet*

Description

Running this function will open the package cheat sheet (a pdf) via a call to `system()`.

Usage

```
cheatsheet(opencmd = NULL)
```

Arguments

`opencmd`	The command used to open pdfs from the command line.

Value

None

clusterTracks *Cluster Tracks*

Description

Perform a quick clustering visualization of a set of tracks according to a given vector of track measures.

Usage

```
clusterTracks(
  tracks,
  measures,
  scale = TRUE,
  labels = NULL,
  method = "hclust",
  return.clust = FALSE,
  ...
)
```

Arguments

tracks	the tracks that are to be clustered.
measures	a function, or a vector of functions (see TrackMeasures). Each function is expected to return a single number given a single track.
scale	logical indicating whether the measures values shall be scaled using the function scale before the clustering.
labels	optional: a vector of labels of the same length as the track object. These are used to color points in the visualization.
method	"hclust" for hierarchical clustering, or "kmeans" for k-means clustering.
return.clust	logical: return the clustering object instead of only the plot? (defaults to FALSE).
...	additional parameters to be passed to the corresponding clustering function: hclust or kmeans.

Details

The measures are applied to each of the tracks in the given *tracks* object. According to the resulting values, the tracks are clustered using the chosen clustering method. If scale is TRUE, the measure values are scaled to mean value 0 and standard deviation 1 (per measure) before the clustering.

Method hclust plots a dendrogram of the clustering.

Method kmeans plots each computed cluster (x-axis) versus each of the track measures in the measures vector, producing one panel per measure. If labels are given, points are colored according to their "true" label.

Value

By default, only returns a plot. If `return.clust=TRUE`, also returns a clustering object as returned by `hclust` or `kmeans`. output object.

See Also

`getFeatureMatrix` to obtain a feature matrix that can be used for manual clustering and plotting, and `trackFeatureMap` to visualize high-dimensional track feature data via dimensionality reduction.

Examples

```
## Cluster tracks according to the mean of their Hust exponents along X and Y
## using hierarchical clustering

cells <- c(TCells,Neutrophils)
real.celltype <- rep(c("T","N"),c(length(TCells),length(Neutrophils)))
## Prefix each track ID with its cell class to evaluate the clustering visually
names(cells) <- paste0(real.celltype,seq_along(cells))
clust <- clusterTracks( cells, hurstExponent, method = "hclust",
 return.clust = TRUE  )

## How many cells are "correctly" clustered?
sum( real.celltype == c("T","N")[cutree(clust,2)] )
```

---

distanceCells *Minimum Distance between Two Cells*

Description

Compute the minimum distance between two cells in the dataset (minimum over all) the timepoints where they were both measured.

Usage

```
distanceCells(X, cellids, quietly = FALSE)
```

Arguments

X	a tracks object
cellids	a vector of two indices specifying the tracks to compute distance between, or a dataframe/matrix of two columns (where every row contains a pair of cellids to compute a distance for)
quietly	if TRUE, suppress warnings about returning NA distances.

Value

A single distance (NA if the the tracks do not have overlapping timepoints), or a vector of such distances if multiple pairs are supplied in `cellids`.

See Also

`angleCells` to compute the angle between the track displacement vectors, and `AngleAnalysis` for other methods to compute angles and distances.

Examples

```
## Find the minimum distance between the tracks with ids 1 and 3
distanceCells( TCells, c("1","3") )
```

`distanceSteps` *Distance between Two Steps*

Description

Compute the distance between two steps in the dataset that occur at the same timepoint. The distance is the distance between the step starting points.

Usage

```
distanceSteps(X, trackids, t, quietly = FALSE)
```

Arguments

`X`	a tracks object
`trackids`	a vector of two indices specifying the tracks to get steps from, or a dataframe/matrix of two columns (where every row contains a pair of trackids to compute a step angle for)
`t`	the timepoint at which the steps should start, or a vector of such timepoints if multiple step pairs are supplied in `trackids`.
`quietly`	logical; should a warning be returned if one or both of the steps are missing in the data and the function returns NA?

Value

A single distance (NA if the desired timepoint is missing for one or both of the tracks), or a vector of such distances if multiple step pairs are supplied in `trackids`.

See Also

`angleSteps` to compute the angle between the steps, `timePoints` to list all timepoints in a dataset, and `AngleAnalysis` for other methods to compute angles and distances.

Examples

```
## Find the distance between the steps of the tracks with ids 1 and 3, at the 3rd
## timepoint in the dataset.
t <- timePoints( TCells )[3]
distanceSteps( TCells, c("1","3"), t )

## Do this for multiple pairs and times at once: between cells 1 and 3 at the
## 3rd timepoint, and between 1 and 4 at the fourth timepoint.
pairs <- data.frame( cell1 = c("1","1"), cell2 = c("3","4"))
times <- timePoints(TCells)[3:4]
distanceSteps( TCells, pairs, times )
```

`distanceToPlane` *Distance to a Reference Plane*

Description

Compute the (shortest) distance between the starting point of a track and a reference plane. Useful to detect directed movement and/or tracking artefacts.

Usage

```
distanceToPlane(x, p1 = c(0, 0, 0), p2 = c(0, 1, 0), p3 = c(1, 0, 0), from = 1)
```

Arguments

`x`	a single input track; a matrix whose first column is time and whose remaining columns are a spatial coordinate.
`p1, p2, p3`	numeric vectors of coordinates of three points specifying a reference plane to compute distances to.
`from`	index, or vector of indices, of the first row of the track. If `from` is a vector, distances are returned for all steps starting at the indices in `from`.

Value

A single distance.

See Also

`angleToPlane` to compute the angle to the plane, and `AngleAnalysis` for other methods to compute angles and distances.

Examples

```
## Plotting the angle versus the distance to a reference plane can be informative to
## detect tracking artefacts near the border of the imaging volume.
## We should be suspicious especially when small angles are more frequent at low distances
## to the border planes.
load( system.file("extdata", "TCellsRaw.rda", package="celltrackR" ) )
steps <- subtracks( TCellsRaw, 1 )
minz <- boundingBox( TCellsRaw )["min","z"]
## Compute angles and distances to the lower plane in z-dimension
angles <- sapply( steps, angleToPlane, p1 = c(0,0,minz), p2 = c(1,0,minz), p3 = c(0,1,minz) )
distances <- sapply( steps, distanceToPlane, p1 = c(0,0,minz), p2 = c(1,0,minz), p3 = c(0,1,minz) )
scatter.smooth( distances, angles )
abline( h = 32.7, col = "red" )
```

---

distanceToPoint *Distance to a Reference Point*

Description

Compute the distance between the starting point of a track and a reference point. Useful to detect directed movement towards a point (see examples).

Usage

```
distanceToPoint(x, p = c(0, 0, 0), from = 1)
```

Arguments

x	a single input track; a matrix whose first column is time and whose remaining columns are a spatial coordinate.
p	numeric vector of coordinates of the reference point p to compute distances to.
from	index, or vector of indices, of the first row of the track. If from is a vector, distances are returned for all steps starting at the indices in from.

Value

A single distance.

See Also

angleToPoint to compute the angle to the reference point, and AngleAnalysis for other methods to compute angles and distances.

Examples

```
## Plotting the angle versus the distance to a reference point can be informative to
## detect biased movement towards that point. We should be suspicious especially
## when small angles are more frequent at lower distances.
steps <- subtracks( Neutrophils, 1 )
bb <- boundingBox( Neutrophils )
angles <- sapply( steps, angleToPoint, p = bb["max",-1] )
distances <- sapply( steps, distanceToPoint, p = bb["max",-1] )
scatter.smooth( distances, angles )
abline( h = 90, col = "red" )
```

`filterTracks` *Filter Tracks*

Description

Extracts subtracks based on a given function.

Usage

```
filterTracks(f, x, ...)
```

Arguments

`f`	a function that accepts a single track as its first argument and returns a logical value (or a value that can be coerced to a locical).
`x`	a tracks object.
`...`	further arguments to be passed on to `f`.

Value

A `tracks` object containing only those tracks from `x` for which `f` evaluates to `TRUE`.

Examples

```
## Remove short tracks from the T cells data
plot( filterTracks( function(t) nrow(t)>10, TCells ) )
```

getFeatureMatrix *Obtaining A Feature Matrix*

Description

Applies a given vector of track measures directly on a set of tracks, returning output in a matrix with a column for each measure and a row for each track. Can also return a distance matrix, which some clustering methods require.

Usage

```
getFeatureMatrix(tracks, measures, dist = FALSE, ...)
```

Arguments

tracks	the tracks that are to be analyzed.
measures	a function, or a vector of functions (see TrackMeasures). Each function is expected to return a single number given a single track.
dist	should a distance matrix rather than a feature matrix be returned?
...	further arguments passed on to "dist"

Value

A matrix with a row for each track and a column for each measure.

See Also

clusterTracks for a quick method to compute the feature matrix and a clustering, and trackFeatureMap to perform dimensionality reduction methods on a set of track features.

Examples

```
## Get speed, meanTurningAngle, and straightness for T cell tracks
fm <- getFeatureMatrix( TCells, c(speed,meanTurningAngle,straightness))
str(fm)
```

`hotellingsTest` *Test Unbiasedness of Motion*

Description

Test the null hypothesis that a given set of tracks originates from an uncorrelated and unbiased type of motion (e.g., a random walk without drift). This is done by testing whether the mean step vector is equal to the null vector.

Usage

```
hotellingsTest(
  tracks,
  dim = c("x", "y"),
  step.spacing = 0,
  plot = FALSE,
  add = FALSE,
  ellipse.col = "blue",
  ellipse.border = "black",
  conf.level = 0.95,
  ...
)
```

Arguments

`tracks`	the tracks whose biasedness is to be determined.
`dim`	vector with the names of the track's dimensions that are to be considered. By default c("x", "y").
`step.spacing`	How many positions are to be left out between the steps that are considered for the test. For persistent motion, subsequent steps will be correlated, which leads to too low p-values because Hotelling's test assumes that the input data is independent. To avoid this, the resulting p-value should either be corrected for this dependence (e.g. by adjusting the degrees of freedom accordingly), or 'step.spacing' should be set to a value high enough to ensure that the considered steps are approximately independent.
`plot`	logical indicating whether the scatter of the step's directions, origin of ordinates (green circle) and the mean of the data points (green cross) are to be plotted. (In one dimension also the bounds of the condfidence interval are given.) Plot works only in one or two dimensions.
`add`	whether to add the plot to the current plot (`TRUE`) or create a
`ellipse.col`	color with which to draw the confidence ellipse of the mean (for 1D, this corresponds to the confidence interval of the mean). Use `NA` to omit the confidence ellipse.
`ellipse.border`	color of the confidence ellipse border. Use `NA` to omit the border.
`conf.level`	the desired confidence level for the confidence ellipse.
`...`	further arguments passed on to `plot`.

Details

Computes the displacement vectors of all segments in the tracks given in `tracks`, and performs Hotelling's T-square Test on that vector.

Value

A list with class `htest`.

References

Johannes Textor, Antonio Peixoto, Sarah E. Henrickson, Mathieu Sinn, Ulrich H. von Andrian and Juergen Westermann (2011), Defining the Quantitative Limits of Intravital Two-Photon Lymphocyte Tracking. *PNAS* **108**(30):12401–12406. doi:10.1073/pnas.1102288108

Examples

```
## Test H_0: T-cells migrate by uncorrelated random walk on x and y coordinates,
## and report the p-value.
hotellingsTest( TCells )$p.value
```

`interpolateTrack` *Interpolate Track Positions*

Description

Approximates the track positions at given time points using linear interpolation (via the `approx` function).

Usage

```
interpolateTrack(x, t, how = "linear")
```

Arguments

`x`	the input track (a matrix or data frame).
`t`	the times at which to approximate track positions. These must lie within the interval spanned by the track timepoints.
`how`	specifies how to perform the interpolation. Possible values are `"linear"` (which uses `approx` with default values) and `"spline"` (which uses `spline` with default values).

Value

The interpolated track (a matrix or data frame).

Examples

```
## Compare interpolated and non-interpolated versions of a track
bb <- boundingBox( TCells[2] )
plot( TCells[2] )
t2i <- interpolateTrack(TCells[[2]], seq(bb[1,"t"],bb[2,"t"],length.out=100),"spline")
plot( tracks( t2i ), add=TRUE, col=2 )
```

---

maxTrackLength *Length of Longest Track*

Description

Determines the maximum number of positions over the tracks in `x`.

Usage

```
maxTrackLength(x)
```

Arguments

`x`	the `tracks` object the tracks in which are to be considered.

Value

The maximum number of rows of a track in `x`

Neutrophils *Two-Photon Data: Neutrophils in an Infected Lung*

Description

Labelled neutrophils were adoptively transfered and intravitally imaged (using two-photon microscopy) inside the lung of the recipient mouse. These cells display a fairly directed kind of motion, as they move towards infection foci.

Usage

```
data("Neutrophils")
```

Format

An S3 object of class "tracks"; a list with 10 elements. Each element name identifies a cell track. Each element is a matrix containing the following four columns.

- `t` the time (in seconds)
- `x` The X coordinate (in micrometers)
- `y` The Y coordinate (in micrometers)
- `z` The Z coordinate (in micrometers)

Source

Data were generated in 2012 in the Mark J. Miller Lab, Department of Medicine, Washington University in St Louis, USA.

References

Zinselmeyer BH, Dempster J, Wokosin DL, Cannon JJ, Pless R, Parker I and Miller MJ (2009), Two-photon microscopy and multi-dimensional analysis of cell dynamics. *Methods in Enzymology*, **461**:349–78. doi:10.1016/S0076-6879(09)05416-0

Konjufca V and Miller MJ (2009), Imaging Listeria monocytogenes infection in vivo. *Current Topics in Microbiology and Immunology*, **334**:199–226. doi:10.1007/978-3-540-93864-4_9

Kreisel D, Nava RG, Li W, Zinselmeyer BH, Wang B, Lai J, Pless R, Gelman AE, Krupnick AS, and Miller MJ (2010), In vivo two-photon imaging reveals monocyte-dependent neutrophil extravasation during pulmonary inflammation. *PNAS*, **107**(42):18073–18078. doi:10.1073/pnas.1008737107

Examples

```
## load the tracks
data(Neutrophils)
## visualize the tracks (calls function plot.tracks)
plot(Neutrophils)
```

---

normalizeToDuration *Normalize a Measure to Track Duration*

Description

Returns a measure that divides the input measure by the duration of its input track.

Usage

```
normalizeToDuration(x)
```

Arguments

x	a track measure (see TrackMeasures).

Value

A function that computes the input measure for a given track and returns the result divided by the track's duration.

Examples

```
## normalizeToDuration(displacement) can be used as an indicator
## for the motion's efficiency
sapply(TCells, normalizeToDuration(displacement))
```

normalizeTracks *Normalize Tracks*

Description

Translates each track in a given set of tracks such that the first position is the origin.

Usage

```
normalizeTracks(x)
```

Arguments

x	the input tracks object.

Value

an output tracks object with all tracks shifted such that their starting position lies at the origin of the coordinate system.

Examples

```
## normalization of Neutrophil data reveals upward motion
plot( normalizeTracks( Neutrophils ) )
```

pairsByTime *Distance between pairs of tracks at every timepoint*

Description

For every timepoint in the dataset, compute pairwise distances between coordinates.

Usage

```
pairsByTime(X, searchRadius = Inf, times = timePoints(X), quietly = FALSE)
```

Arguments

X	a tracks object
searchRadius	if specified, return only pairs that are within this distance of each other. Defaults to Inf, so if left unspecified, all pairs are returned.
times	(optional) a vector of timePoints to check pairs at; by default this is just everything.
quietly	(default FALSE) if TRUE, suppress warnings when there are no tracks with overlapping timepoints and an empty dataframe is returned.

Value

a dataframe with the following columns:

cell1 the id of the track to which the first coordinate belongs

cell2 the id of the track to which the second coordinate belongs

t the time point at which their distance is assessed

dist the distance between the coordinates at this time

Examples

```
## compute find timepoints where two t cells are within 1 micron of each other.
pairsByTime( TCells, searchRadius = 1 )

## indeed, the following two cells nearly touch:
plot( TCells[ c("24","9258") ] )
```

---

plot.tracks *Plot Tracks in 2D*

Description

Plots tracks contained in a "tracks" object into a twodimensional space pallelel to the data's axes.

Usage

```
## S3 method for class 'tracks'
plot(
  x,
  dims = c("x", "y"),
  add = F,
  col = order(names(x)),
  pch.start = 1,
  pch.end = NULL,
  cex = 0.5,
  ...
)
```

Arguments

x	the tracks to be plotted.
dims	a vector giving the dimensions of the track data that shall be plotted, e.g. `c('x','y')` for the x and y dimension.
add	boolean value indicating whether the tracks are to be added to the current plot.
col	a specification of the color(s) to be used. This can be a vector of size `length(x)`, where each entry specififes the color for the corresponding track.

pch.start	point symbol with which to label the first position of the track (see points).
pch.end	point symbol with which to label the last position of the track
cex	point size for positions on the tracks.
...	additional parameters (e.g. xlab, ylab). to be passed to plot (for add=FALSE) or points (for add=TRUE), respectively.

Details

One dimension of the data (by default y) is plotted against another (by default x). The dimesions can be chosen by means of the parameter dims and the axes can be labeled accordingly with the aid of xlab and ylab. The color can be set through col. If the tracks should be added to an existing plot, add is to be set to TRUE.

Value

None

See Also

plot3d

---

plot3d *Plot Tracks in 3D*

Description

Takes an input tracks object and plots them in 3D using the scatterplot3d function.

Usage

```
plot3d(x, ...)
```

Arguments

x	the tracks which will be plotted in 3d
...	further arguments to be passed on to scatterplot3d

Value

None.

Examples

```
if( require("scatterplot3d",quietly=TRUE) ){
  plot3d( TCells )
}
```

`plotTrackMeasures` *Bivariate Scatterplot of Track Measures*

Description

Plots the values of two measures applied on the given tracks against each other.

Usage

```
plotTrackMeasures(
  x,
  measure.x,
  measure.y,
  add = FALSE,
  xlab = deparse(substitute(measure.x)),
  ylab = deparse(substitute(measure.y)),
  ellipse.col = "red",
  ellipse.border = "black",
  conf.level = 0.95,
  ...
)
```

Arguments

`x`	the input `tracks` object.
`measure.x`	the measure to be shown on the X axis (see TrackMeasures).
`measure.y`	the measure to be shown on the Y axis.
`add`	a logical indicating whether the tracks are to be added to an existing plot via `points`.
`xlab`	label of the x-axis. By default the name of the input function `measure.x`.
`ylab`	label of the y-axis. By default the name of the input function `measure.y`.
`ellipse.col`	color with which to draw the confidence ellipse of the mean (for 1D, this corresponds to the confidence interval of the mean). Use `NA` to omit the confidence ellipse.
`ellipse.border`	color of the confidence ellipse border. Use `NA` to omit the border.
`conf.level`	the desired confidence level for the confidence ellipse.
`...`	additional parameters to be passed to `plot` (in case `add=FALSE`) or `points` (`add=TRUE`).

Details

Plots the value of `measurey` applied to `x` against the value of `measurey` applied to `y`. This is useful for "FACS-like" motility analysis, where clusters of cell tracks are identified based on their motility parameters (Moreau et al, 2012; Textor et al, 2014).

Value

None

References

Moreau HD, Lemaitre F, Terriac E, Azar G, Piel M, Lennon-Dumenil AM, Bousso P (2012), Dynamic In Situ Cytometry Uncovers T Cell Receptor Signaling during Immunological Synapses and Kinapses In Vivo. *Immunity* **37**(2), 351–363. doi:10.1016/j.immuni.2012.05.014

Johannes Textor, Sarah E. Henrickson, Judith N. Mandl, Ulrich H. von Andrian, J\"urgen Westermann, Rob J. de Boer and Joost B. Beltman (2014), Random Migration and Signal Integration Promote Rapid and Robust T Cell Recruitment. *PLoS Computational Biology* **10**(8), e1003752. doi:10.1371/journal.pcbi.1003752

Examples

```
## Compare speed and straightness of 3 example population tracks.
## To make the comparison fair, analyze subtracks of fixed length.
plotTrackMeasures( subtracks(TCells,4,0), speed, straightness, ellipse.col="black" )
plotTrackMeasures( subtracks(BCells,4,0), speed, straightness,
  col=2, ellipse.col=2, pch=2, add=TRUE )
plotTrackMeasures( subtracks(Neutrophils,4,0), speed, straightness,
  col=3, ellipse.col=3, pch=3, add=TRUE )
```

---

prefixes *Get Track Prefixes*

Description

Creates a `tracks` object consisting of all prefixes (i.e., subtracks starting with the first position of a track) of ‘x‘ with ‘i‘ segments (i.e., ‘i‘+1 positions).

Usage

```
prefixes(x, i)
```

Arguments

`x`	a single track or a `tracks` object.
`i`	subtrack length. A single integer, lists are not supported.

Details

This function behaves exactly like `subtracks` except that only subtracks starting from the first position are considered.

Value

A *tracks* object is returned which contains all the subtracks of any track in the input *tracks* object that consist of exactly ‘i‘ segments and start at the first registered coordinate of the given track.

See Also

`subtracks` to extract all subtracks of a given length, `subtracksByTime` to extract all subtracks of a given length starting at some fixed timepoint, and `selectSteps` to extract single steps starting at a fixed timepoint from a subset of trackids.

`projectDimensions` *Extract Spatial Dimensions*

Description

Projects tracks onto the given spatial dimensions.

Usage

```
projectDimensions(x, dims = c("x", "y"))
```

Arguments

`x`	the input tracks object.
`dims`	a character vector (for column names) or an integer vector (for column indices) giving the dimensions to extract from each track. The time dimension (i.e., the first column of all tracks) is always included.

Value

A tracks object is returned that contains only those dimensions of the input `tracks` that are given in `dims`.

Examples

```
## Compare 2D and 3D speeds
load( system.file("extdata", "TCellsRaw.rda", package="celltrackR" ) )
speed.2D <- mean( sapply( subtracks( projectDimensions( TCellsRaw, c("x","z") ), 2 ), speed ) )
speed.3D <- mean( sapply( TCellsRaw, speed ) )
```

`read.tracks.csv` *Read Tracks from Text File*

Description

Reads cell tracks from a CSV or other text file. Data are expected to be organized as follows. One column contains a track identifier, which can be numeric or a string, and determines which points belong to the same track. Another column is expected to contain a time index or a time period (e.g. number of seconds elapsed since the beginning of the track, or since the beginning of the experiment). Input of dates is not (yet) supported, as absolute time information is frequently not available. Further columns contain the spatial coordinates. If there are three or less spatial coordinates, their names will by "x", "y", and "z" (depending on whether the tracks are 1D, 2D or 3D). If there are four or more spatial coordinates, their names will be "x1", "x2", and so on. The names or indices of these columns in the CSV files are given using the corresponding parameters (see below). Names and indices can be mixed, e.g. you can specify `id.column="Parent"` and `pos.columns=1:3`

Usage

```
read.tracks.csv(
  file,
  id.column = 1,
  time.column = 2,
  pos.columns = c(3, 4, 5),
  scale.t = 1,
  scale.pos = 1,
  header = TRUE,
  sep = "",
  track.sep.blankline = FALSE,
  ...
)
```

Arguments

`file`	the name of the file which the data are to be read from, a readable text-mode connection or a complete URL (see `read.table`).
`id.column`	index or name of the column that contains the track ID.
`time.column`	index or name of the column that contains elapsed time.
`pos.columns`	vector containing indices or names of the columns that contain the spatial coordinates. If this vector has two entries and the second entry is NA, e.g. `c('x',NA)` or `c(5,NA)` then all columns from the indicated column to the last column are used. This is useful when reading files where the exact number of spatial dimensions is not known beforehand.
`scale.t`	a value by which to multiply each time point. Useful for changing units, or for specifying the time between positions if this is not contained in the file itself.

`scale.pos` a value, or a vector of values, by which to multiply each spatial position. Useful for changing units.

`header` a logical value indicating whether the file contains the names of the variables as its first line. See `read.table`.

`sep` a character specifying how the colums of the data are separated. The default value `""` means columns are separated by tabs or other spaces. See `read.table`.

`track.sep.blankline`
logical. If set to `TRUE`, then tracks are expected to be separated by one or more blank lines in the input file instead of being designated by a track ID column. In this case, numerical track IDs are automatically generated.

`...` further arguments to be passed to `read.csv`, for instance `sep="\t"` can be useful for tab-separated files.

Details

The input file's first four fields are interpreted as id, pos, t and x, respectively, and, if available, the fifth as y and the sixth as z. The returned object has the class *tracks*, which is a list of data frames representing the single tracks and having columns t and x, plus y and z, if necessary. The tracks' ids are retained in their position in the list, while the field pos will be unmaintained.

Value

An object of class *tracks* is returned, which is a list of matrices, each containing the positions of one track. The matrices have a column t, followed by one column for each of the input track's coordinates.

`repairGaps` *Process Tracks Containing Gaps*

Description

Many common motility analyses, such as mean square displacement plots, assume that object positions are recorded at constant time intervals. For some application domains, such as intravital imaging, this may not always be the case. This function can be used to pre-process data imaged at nonconstant intervals, provided the deviations are not too extreme.

Usage

```
repairGaps(x, how = "split", tol = 0.05, split.min.length = 2)
```

Arguments

`x` the input tracks object.

`how` string specifying what do with tracks that contain gaps. Possible values are:

- "drop": the simplest option – discard all tracks that contain gaps.

- "split": split tracks around the gaps, e.g. a track for which the step between the 3rd and 4th positions is too long or too short is split into one track corresponding to positions 1 to 3 and another track corresponding to position 3 onwards.
- "interpolate": approximate the track positions using linear interpolation (see `interpolateTrack`). The result is a tracks object with constant step durations.

`tol` nonnegative number specifying by which fraction each step may deviate from the average step duration without being considered a gap. For instance, if the average step duration (see `timeStep`) is 100 seconds and `tol` is 0.05 (the default), then step durations between 95 and 105 seconds (both inclusive) are not considered gaps. This option is ignored for `how="interpolate"`.

`split.min.length` nonnegative integer. For `how="split"`, this discards all resulting tracks shorter than this many positions.

Value

A `tracks` object with gaps fixed according to the chosen method.

Examples

```
## The Neutrophil data are imaged at rather nonconstant intervals
print( length( Neutrophils ) )
print( length( repairGaps( Neutrophils, tol=0.01 ) ) )
```

`selectSteps` *Get Single Steps Starting at a Specific Time from a Subset of Tracks*

Description

Obtain all single steps starting at a given timepoint t from a subset of tracks of interest.

Usage

```
selectSteps(X, trackids, t)
```

Arguments

`X`	Tracks object to obtain subtracks from
`trackids`	Character vector with the ids of tracks of interest
`t`	Timepoint at which the subtracks should start

Value

A *tracks* object is returned which contains all the extracted steps.

See Also

subtracks to extract all subtracks of a given length, prefixes to extract all subtracks of a given length starting from the first coordinate in each track, subtracksByTime to extract all subtracks of a given length starting at some fixed timepoint, and timePoints to return all timepoints occurring in the dataset.

Examples

```
## Get and plot all steps starting at the third timepoint in tracks 1 and 3 of
## the T cell dataset
subT <- selectSteps( TCells, c("1","5"), timePoints(TCells)[3] )
plot( subT )
```

---

selectTracks *Select Tracks by Measure Values*

Description

Given a tracks object, extract a subset based on upper and lower bounds of a certain measure. For instance, extract all tracks with a certain minimum length.

Usage

```
selectTracks(x, measure, lower, upper)
```

Arguments

x	the input tracks.
measure	measure on which the selection is based (see TrackMeasures).
lower	specifies the lower bound (inclusive) of the allowable measure.
upper	specifies the upper bound (inclusive) of the allowable measure.

Value

A tracks object with the selected subset of tracks.

Examples

```
## Slower half of T cells
slow.tcells <- selectTracks( TCells, speed, -Inf, median( sapply(TCells,speed) ) )
```

simulateTracks *Generate Tracks by Simulation*

Description

Generic function that executes `expr`, which is expected to return a track, `n` times and stores the output in a `tracks` object. Basically, this works like `replicate` but for tracks.

Usage

```
simulateTracks(n, expr)
```

Arguments

`n`	number of tracks to be generated.
`expr`	the expression, usually a call, that generates a single track.

Value

A `tracks` object containing `n` tracks.

Examples

```
## Generate 10 tracks, 100 steps each, from a random walk with standard normally
## distributed increments and plot them
plot( simulateTracks( 10, brownianTrack(100,3) ) )
```

sort.tracks *Sort Track Positions by Time*

Description

Sorts the positions in each track in a *tracks* object by time.

Usage

```
## S3 method for class 'tracks'
sort(x, decreasing = FALSE, ...)
```

Arguments

`x`	the *tracks* object whose tracks are to be sorted by time.
`decreasing`	logical. Should the sort be increasing or decreasing? Provided only for consistency with the generic sort method. The positions in each track should be sorted in increasing time order.
`...`	further arguments to be passed on to `order`.

Details

Sorts the positions of each track (represented as a data frame) in the *tracks* object by time (given in the column t).

Value

A *tracks object* that contains the tracks from the input object sorted by time is returned.

---

splitTrack *Split Track into Multiple Tracks*

Description

Split Track into Multiple Tracks

Usage

```
splitTrack(x, positions, id = NULL, min.length = 2)
```

Arguments

x	the input track (a data frame or a matrix).
positions	a vector of positive integers, given in ascending order.
id	a string used to identify the resulting tracks; for instance, if id="X", then the resulting tracks are named X_1, X_2 and so forth. Otherwise, they are simply labelled with integer numbers.
min.length	nonnegative integer. Resulting tracks that have fewer positions than the value of this parameter are dropped.

Value

An object of class *tracks* with the resulting splitted tracks.

---

staggered *Staggered Version of a Function*

Description

Returns the "staggered" version of a track measure. That is, instead of computing the measure on the whole track, the measure is averaged over all subtracks (of any length) of the track.

Usage

```
staggered(measure, ...)
```

Arguments

measure	a track measure (see TrackMeasures).
...	further parameters passed on to applyStaggered.

Details

This is a wrapper mainly designed to provide a convenient interface for track-based staggered computations with lapply, see example.

Value

Returns a function that computes the given measure in a staggered fashion on that track.

References

Zeinab Mokhtari, Franziska Mech, Carolin Zitzmann, Mike Hasenberg, Matthias Gunzer and Marc Thilo Figge (2013), Automated Characterization and Parameter–Free Classification of Cell Tracks Based on Local Migration Behavior. *PLoS ONE* **8**(12), e80808. doi:10.1371/journal.pone.0080808

Examples

```
hist( sapply( TCells, staggered( displacement ) ) )
```

---

stepPairs *Find Pairs of Steps Occurring at the Same Time*

Description

Find cell indices and timepoints where these cells both have a step.

Usage

```
stepPairs(X, filter.steps = NULL)
```

Arguments

X	a tracks object
filter.steps	optional: a function used to filter steps on. See examples.

Value

A dataframe with three columns: two for the indices of cellpairs that share a step, and one for the timepoint at which they do so.

Examples

```
## Find all pairs of steps in the T cell data that displace at least 2 microns.
pairs <- stepPairs( TCells, filter.steps = function(t) displacement(t) > 2 )
```

---

subsample *Subsample Track by Constant Factor*

Description

Make tracks more coarse-grained by keeping only every kth position.

Usage

```
subsample(x, k = 2)
```

Arguments

x	an input track or tracks object.
k	a positive integer. Every kth position of each input track is kept.

Value

A tracks object with the new, more coarse-grained tracks.

See Also

interpolateTrack, which can be used for more flexible track coarse-graining.

Examples

```
## Compare original and subsampled versions of the T cell tracks
plot( TCells, col=1 )
plot( subsample( TCells, 3 ), col=2, add=TRUE, pch.start=NULL )
```

subtracks *Decompose Track(s) into Subtracks*

Description

Creates a *tracks* object consisting of all subtracks of ‘x‘ with ‘i‘ segments (i.e., ‘i‘+1 positions).

Usage

```
subtracks(x, i, overlap = i - 1)
```

Arguments

x	a single track or a tracks object.
i	subtrack length. A single integer, lists are not supported.
overlap	the number of segments in which each subtrack shall overlap with the previous and next subtrack. The default i - 1 returns all subtracks. Can be negative, which means that space will be left between subtracks.

Details

The output is always a single *tracks* object, which is convenient for many common analyses. If subtracks are to be considered separately for each track, use the function staggered together with lapply. Subtrack extraction always starts at the first position of the input track.

Value

A *tracks* object is returned which contains all the subtracks of any track in the input *tracks* object that consist of exactly ‘i‘ segments and overlap adjacent subtracks in ‘overlap‘ segments.

See Also

prefixes to extract all subtracks of a given length starting from the first coordinate in each track, subtracksByTime to extract all subtracks of a given length starting at some fixed timepoint, and selectSteps to extract single steps starting at a fixed timepoint from a subset of trackids.

`subtracksByTime` *Extract Subtracks Starting at a Specific Time*

Description

Obtain all subtracks of i steps (i+1 positions) starting at a given timepoint t.

Usage

```
subtracksByTime(X, t, i = 1, epsilon = 1e-04, tlo = t, thi = t)
```

Arguments

`X`	Tracks object to obtain subtracks from.
`t`	Timepoint at which the subtracks should start. This value is ignored if tlo and thi are specified, see below.
`i`	Subtrack length (in number of steps). Set this to NULL to obtain subtracks of varying length but within a specified interval [tlo, thi] (see below).
`epsilon`	Small error allowed when comparing timepoints because of numerical inaccuracies, see details. Timepoints in tracks are returned if they are within [tlo-epsilon, thi+epsilon].
`tlo, thi`	Interval specifying the timepoints to be returned. By default, these are not used and tracks starting at timepoint t with exactly i steps are returned; see details.

Details

If i is specified, the given t is retrieved for all tracks in X that contain that timepoint, and any subtracks starting from that time that have exactly i steps are returned. For numerical reasons, timepoints in the data are allowed to deviate a small amount epsilon from t (because otherwise, equal timepoints can seem unequal because of very small deviations).

If i is set to NULL, subtracks are returned with all timepoints lying in the interval [tlo - epsilon, thi + epsilon]. These subtracks do NOT have to be of equal length.

Value

A *tracks* object is returned which contains all the subtracks of any track in the input *tracks* object that consist of exactly ‘i‘ segments and start at the given timepoint t, OR a *tracks* object with all the timepoints of any track in the input *tracks* object that are between tlo and thi.

See Also

`subtracks` to extract all subtracks of a given length, `prefixes` to extract all subtracks of a given length starting from the first coordinate in each track, `selectSteps` to extract single steps starting at a fixed timepoint from a subset of trackids, and `timePoints` to return all timepoints occurring in the dataset.

Examples

```
## Get all the single steps (i=1) starting at the third timepoint in the T cell tracks.
subT <- subtracksByTime( TCells, timePoints(TCells)[3], 1 )

## These all have the same number of steps:
sapply( subT, nrow )

## Or set i to NULL and return all subtracks within the five first timepoints:
subT2 <- subtracksByTime( TCells, NULL, i = NULL,
  tlo = timePoints( TCells )[1], thi = timePoints( TCells )[5] )

## These are not all the same length:
sapply( subT2, nrow )
```

TCells *Two-Photon Data: T Cells in a Lymph Node*

Description

Labelled T cells were adoptively transfered and intravitally imaged (using two-photon microscopy) inside a peripheral lymph node of the recipient mouse. These data represent the characteristic "random-walk-like" motion pattern of T cells in lymph nodes.

Usage

```
data("TCells")
```

Format

An S3 object of class "tracks"; a list with 22 elements. Each element name identifies a cell track. Each element is a matrix containing the following four columns.

- `t` the time (in seconds)
- `x` The X coordinate (in micrometers)
- `y` The Y coordinate (in micrometers)
- `z` The Z coordinate (in micrometers)

Source

Data were generated in 2012 in the Mark J. Miller Lab, Department of Medicine, Washington University in St Louis, USA.

References

Zinselmeyer BH, Dempster J, Wokosin DL, Cannon JJ, Pless R, Parker I and Miller MJ (2009), Two-photon microscopy and multi-dimensional analysis of cell dynamics. *Methods in Enzymology*, **461**:349–78. doi:10.1016/S0076-6879(09)05416-0

Konjufca V and Miller MJ (2009), Imaging Listeria monocytogenes infection in vivo. *Current Topics in Microbiology and Immunology*, **334**:199–226. doi:10.1007/978-3-540-93864-4_9

Kreisel D, Nava RG, Li W, Zinselmeyer BH, Wang B, Lai J, Pless R, Gelman AE, Krupnick AS, and Miller MJ (2010), In vivo two-photon imaging reveals monocyte-dependent neutrophil extravasation during pulmonary inflammation. *PNAS*, **107**(42):18073–18078. doi:10.1073/pnas.1008737107

Examples

```
## load the tracks
data(TCells)
## visualize the tracks (calls function plot.tracks)
plot(TCells)
```

`timePoints` *Find All Unique Time Points in a Track Dataset*

Description

Return a vector of all the timepoints t found in any of the matrices in the tracks object.

Usage

```
timePoints(X)
```

Arguments

`X`	a tracks object.

Value

A numeric vector of unique timepoints.

Examples

```
## Get all timepoints in the T cell dataset
tp <- timePoints( TCells )
```

timeStep *Compute Time Step of Tracks*

Description

Applies a summary statistics on the time intervals between pairs of consecutive positions in a track dataset.

Usage

```
timeStep(x, FUN = median, na.rm = FALSE)
```

Arguments

x	the input tracks.
FUN	the summary statistic to be applied.
na.rm	logical, indicates whether to remove missing values before applying FUN.

Details

Most track quantification depends on the assumption that track positions are recorded at constant time intervals. If this is not the case, then most of the statistics in this package (except for some very simple ones like duration) will not work. In reality, at least small fluctuations of the time steps can be expected. This function provides a means for quality control with respect to the tracking time.

Value

Summary statistic of the time intervals between two consecutive positions in a track dataset.

Examples

```
## Show tracking time fluctuations for the T cell data
d <- timeStep( TCells )
plot( sapply( subtracks( TCells, 1 ) , duration ) - d, ylim=c(-d,d) )
```

trackFeatureMap *Dimensionality Reduction on Track Features*

Description

Perform a quick dimensionality reduction visualization of a set of tracks according to a given vector of track measures.

Usage

```
trackFeatureMap(
  tracks,
  measures,
  scale = TRUE,
  labels = NULL,
  method = "PCA",
  return.mapping = FALSE,
  ...
)
```

Arguments

tracks	the tracks that are to be clustered.
measures	a function, or a vector of functions (see TrackMeasures). Each function is expected to return a single number given a single track.
scale	logical indicating whether the measures values shall be scaled using the function scale before the mapping is performed.
labels	optional: a vector of labels of the same length as the track object. These are used to color points in the visualization.
method	"PCA" for a scatterplot along principal components, "MDS" for multidimensional scaling, "UMAP" for a UMAP. Note that for "UMAP", the uwot package must be installed.
return.mapping	logical: return the mapping object instead of only the plot? (defaults to FALSE).
...	additional parameters to be passed to the corresponding function: prcomp (for method="PCA"), cmdscale (for method="MDS"), or umap (for method="UMAP").

Details

The measures are applied to each of the tracks in the given *tracks* object. According to the resulting values, the tracks are mapped to fewer dimensions using the chosen method. If scale is TRUE, the measure values are scaled to mean value 0 and standard deviation 1 (per measure) before the mapping.

The dimensionality reduction methods PCA, MDS, and UMAP each produce a scatterplot of all tracks as points, plotted along the principal component axes generated by the corresponding method.

Value

By default, only returns a plot. If return.clust=TRUE, also returns a clustering object as returned by hclust, kmeans, prcomp (returns $x), cmdscale, or umap (returns $layout). See the documentation of those functions for details on the output object.

See Also

getFeatureMatrix to obtain a feature matrix that can be used for manual clustering and plotting, and clusterTracks to perform hierarchical or k-means clustering on a tracks dataset.

Examples

```
## Map tracks according to speed, mean turning angle, straightness, and asphericity
## using multidimensional scaling, and store output.

cells <- c(TCells,Neutrophils)
real.celltype <- rep(c("T","N"),c(length(TCells),length(Neutrophils)))
## Prefix each track ID with its cell class to evaluate the clustering visually
names(cells) <- paste0(real.celltype,seq_along(cells))
map <- trackFeatureMap( cells, c(speed,meanTurningAngle,straightness, asphericity),
 method = "MDS",  return.mapping = TRUE  )
```

TrackMeasures *Track Measures*

Description

Statistics that can be used to quantify tracks. All of these functions take a single track as input and give a single number as output.

Usage

```
trackLength(x)

duration(x)

speed(x)

displacement(x, from = 1, to = nrow(x))

squareDisplacement(x, from = 1, to = nrow(x))

displacementVector(x)

maxDisplacement(x)

displacementRatio(x)

outreachRatio(x)

straightness(x)

overallAngle(x, from = 1, to = nrow(x), xdiff = diff(x), degrees = FALSE)

meanTurningAngle(x, degrees = FALSE)

overallDot(x, from = 1, to = nrow(x), xdiff = diff(x))
```

```
overallNormDot(x, from = 1, to = nrow(x), xdiff = diff(x))

asphericity(x)

hurstExponent(x)

fractalDimension(x)
```

Arguments

x	a single input track; a matrix whose first column is time and whose remaining columns are a spatial coordinate.
from	index, or vector of indices, of the first row of the track.
to	index, or vector of indices, of last row of the track.
xdiff	row differences of x.
degrees	logical; should angles be returned in degrees rather than radians?

Details

Some track measures consider only the first and last position (or steps) of a track, and are most useful in conjunction with aggregate.tracks; for instance, squareDisplacement combined with aggregate.tracks gives a mean square displacement plot, and overallAngle combined with aggregate.tracks gives a turning angle plot (see the examples for aggregate.tracks). To speed up computation of these measures on subtracks of the same track, the arguments from, to and possibly xdiff are exploited by aggregate.tracks.

Value

trackLength sums up the distances between subsequent positsion; in other words, it estimates the length of the underlying track by linear interpolation (usually an underestimation). The estimation could be improved in some circumstances by using interpolateTrack. The function returns a single, non-negative number.

duration returns the time elapsed between x's first and last positions (a single, non-negative number).

speed simply divides trackLength by duration

displacement returns the Euclidean distance between the track endpoints and squareDisplacement returns the squared Euclidean distance.

displacementVector returns the vector between the track endpoints. This vector has an element (can be negative) for each (x,y,z) dimension of the coordinates in the track.

maxDisplacement computes the maximal Euclidean distance of any position on the track from the first position.

displacementRatio divides the displacement by the maxDisplacement; outreachRatio divides the maxDisplacement by the trackLength (Mokhtari et al, 2013). Both measures return values between 0 and 1, where 1 means a perfectly straight track. If the track has trackLength 0, then NaN is returned.

`straightness` divides the `displacement` by the `trackLength`. This gives a number between 0 and 1, with 1 meaning a perfectly straight track. If the track has `trackLength` 0, then `NaN` is returned.

`asphericity` is a different appraoch to measure straightness (Mokhtari et al, 2013): it computes the asphericity of the set of positions on the track _via_ the length of its principal components. Again this gives a number between 0 and 1, with higher values indicating straighter tracks. Unlike `straightness`, however, asphericity ignores back-and-forth motion of the object, so something that bounces between two positions will have low `straightness` but high `asphericity`. We define the asphericity of every track with two or fewer positions to be 1. For one-dimensional tracks with one or more positions, `NA` is returned.

`overallAngle` Computes the angle (in radians) between the first and the last segment of the given track. Angles are measured symmetrically, thus the return values range from 0 to pi; for instance, both a 90 degrees left and right turns yield the value pi/2. This function is useful to generate autocorrelation plots (together with `aggregate.tracks`). Angles can also be returned in degrees, in that case: set `degrees = TRUE`.

`meanTurningAngle` averages the `overallAngle` over all adjacent segments of a given track; a low `meanTurningAngle` indicates high persistence of orientation, whereas for an uncorrelated random walk we expect 90 degrees. Note that angle measurements will yield `NA` values for tracks in which two subsequent positions are identical. By default returns angles in radians; use `degrees = TRUE` to return angles in degrees instead.

`overallDot` computes the dot product between the first and the last segment of the given track. This function is useful to generate autocovariance plots (together with `aggregate.tracks`).

`overallNormDot` computes the dot product between the unit vectors along the first and the last segment of the given track. This function is useful to generate autocorrelation plots (together with `aggregate.tracks`).

`hurstExponent` computes the corrected empirical Hurst exponent of the track. This uses the function `hurstexp` from the 'pracma' package. If the track has less than two positions, NA is returned. `fractalDimension` estimates the fractal dimension of a track using the function `fd.estim.boxcount` from the 'fractaldim' package. For self-affine processes in n dimensions, fractal dimension and Hurst exponent are related by the formula $H = n + 1 - D$. For non-Brownian motion, however, this relationship need not hold. Intuitively, while the Hurst exponent takes a global approach to the track's properties, fractal dimension is a local approach to the track's properties (Gneiting and Schlather, 2004).

References

Zeinab Mokhtari, Franziska Mech, Carolin Zitzmann, Mike Hasenberg, Matthias Gunzer and Marc Thilo Figge (2013), Automated Characterization and Parameter–Free Classification of Cell Tracks Based on Local Migration Behavior. *PLoS ONE* **8**(12), e80808. doi:10.1371/journal.pone.0080808

Tillmann Gneiting and Martin Schlather (2004), Stochastic Models That Separate Fractal Dimension and the Hurst Effect. *SIAM Review* **46**(2), 269–282. doi:10.1137/S0036144501394387

See Also

`AngleAnalysis` for methods to compute angles and distances between pairs of tracks, or of tracks to a reference point, direction, or plane.

Examples

```
## show a turning angle plot with error bars for the T cell data.
with( (aggregate(BCells,overallDot,FUN="mean.se",na.rm=TRUE)),{
  plot( mean ~ i, xlab="time step",
   ylab="turning angle (rad)", type="l" )
  segments( i, lower, y1=upper )
} )
```

`tracks` *Tracks Objects*

Description

The function `tracks` is used to create tracks objects. `as.tracks` coerces its argument to a tracks object, and `is.tracks` tests for tracks objects. `c` can be used to combine (concatenate) tracks objects.

Usage

```
as.tracks(x, ...)

## S3 method for class 'list'
as.tracks(x, ...)

is.tracks(x)

## S3 method for class 'tracks'
c(...)

tracks(...)
```

Arguments

`x`	an object to be coerced or tested.
`...`	for `tracks`, numeric matrices or objects that can be coerced to numeric matrices. Each matrix contains the data of one track. The first column is the time, and the remaining columns define a spatial position. Every given matrix has to contain the same number of columns, and at least two columns are necessary. For `c`, tracks objects to be combined. For `as.tracks`, further arguments passed to methods (currently not used).

Details

Tracks objects are lists of matrices. Each matrix contains at least two columns; the first column is time, and the remaining columns are a spatial coordinate. The following naming conventions are used (and enforced by `tracks`): The time column has the name 't', and spatial coordinate columns have names 'x','y','z' if there are three or less coordinates, and 'x1',...,'xk' if there are $k \geq 4$ coordinates. All tracks in an object must have the same number of dimensions. The positions in a track are expected to be sorted by time (and the constructor `tracks` enforces this).

Value

A `tracks` object.

Examples

```
## A single 1D track
x <- tracks( matrix(c(0, 8,
10, 9,
20, 7,
30, 7,
40, 6,
50, 5), ncol=2, byrow=TRUE ) )

## Three 3D tracks
x2 <- tracks( rbind(
 c(0,5,0), c(1,5,3), c(2,1,3), c(3,5,6) ),
 rbind( c(0,1,1),c(1,1,4),c(2,5,4),c(3,5,1),c(4,-3,1) ),
 rbind( c(0,7,0),c(1,7,2),c(2,7,4),c(3,7,7) ) )
```

---

vecAngle *Angle Between Two Vectors*

Description

Compute the angle between two vectors a and b, which can be numeric vectors or matrices in which each row represents a numeric vector. In the last case, one angle is returned for each row. By default, angles are returned in degrees – set `degrees = TRUE` to return radians.

Usage

```
vecAngle(a, b, degrees = TRUE)
```

Arguments

a	the first vector or set of vectors. Must be a numeric vector or a matrix where each row represents a numeric vector.
b	the second vector or set of vectors, for which angles with the vector (set) a must be computed. Must have the same dimensions as a.
degrees	logical: if `TRUE` (default), return angles in degrees instead of radians.

Value

A single angle (if a and b are single vectors) or a numeric vector of angles (if a and b are matrices; in that case, the output vector contains one angle for each row in matrices a and b).

Examples

```
## The angle between the vectors [0,1] and [1,0] is 90 degrees:
vecAngle( c(0,1), c(1,0) )
## The same holds for 3D angles:
vecAngle( c(0,1,0), c(1,0,0) )
```

`wrapTrack` *Create Track Object from Single Track*

Description

Makes a `tracks` object containing the given track.

Usage

```
wrapTrack(x)
```

Arguments

`x`	the input track.

Value

A list of class `tracks` containing only the input track `x`, which is assigned the name "1".

Index